EFFICIENT AND EFFECTIVE MINING OF TIME SERIES

by

Xiaosheng Li
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Fall 2020
George Mason University
Fairfax, VA

Efficient and Effective Mining of Time Series

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Xiaosheng Li
Master of Engineering
Tianjin University, 2014
Bachelor of Engineering
Tianjin University, 2011

Director: Dr. Jessica Lin, Professor
Department of Computer Science

Fall 2020
George Mason University
Fairfax, VA

Table of Contents

List of Tables

List of Figures

Abstract

EFFICIENT AND EFFECTIVE MINING OF TIME SERIES

Xiaosheng Li, PhD

George Mason University, 2020

Dissertation Director: Dr. Jessica Lin

The mining of time series data has attracted much attention in the past two decades due to the ubiquity of time series in our daily lives. With the advance of technologies, the volume of available time series data becomes huge and the content is changing rapidly. This requires time series data mining methods to have low computational complexities. However, there is little work focuses on providing linear time complexity solutions for the time series data mining tasks. In this dissertation, we present three linear time methods for the classification, motif discovery, and clustering of time series respectively.

Time series classification can be formulated as: given a set of labeled time series, predict the class labels of previously unknown instances. The goal of motif discovery is to find the most similar non-overlapping pair of subsequences in a given time series. Time series clustering can be formulated as to place the given unlabeled time series instances into homogeneous and separated groups.

For time series classification, we propose to transform the subsequences in time series into symbolic patterns, extract symbolic pattern counts in time series as feature values by an incremental process, and use these features for efficient classification.

For motif discovery, we design a special data structure called Grids. The subsequences of the time series are used to update the Grids in a random order to discover the motif. We prove that this process can guarantee to find the optimal solution and the whole process takes linear time. For time series clustering, we propose to partitions the data by checking some randomly selected symbolic patterns in the time series. Theoretical analysis is provided to show that group structures in the data can be revealed from this process.

Besides these linear time complexity methods, this dissertation also presents a non-linear time method for time series classification that uses different idea from the state-of-the-arts. The idea behind the state-of-the-art time series classification techniques is to identify and extract patterns or characteristics from the labeled time series and use these patterns to classify new unlabeled data. We find that sequences of values that are very different from the patterns in the labeled time series can be used as references to classify time series effectively. We propose an evolution process to generate these sequences of values, which we call separating references, from the training data. The proposed method is robust to over-fitting and is especially suitable for the situation where little labeled data is available.

We evaluate the proposed methods on time series data mining extensively with various datasets, and compare with the state-of-the-art approaches with statistical analysis. The results demonstrate the effectiveness and efficiency of the proposed methods. The research in the dissertation shows that we can have linear time complexity methods for the time series data mining tasks under study with effectiveness comparable or superior to the state-of-the-arts. These methods can facilitate the handling of large size data and fast processing of time series in a wide variety of disciplines.

Chapter 1: Introduction

This chapter introduces the background of time series data mining as well as the motivation and objectives of our research. Section 1.1 introduces time series data mining and existing problems in current research as well as the motivation and objectives of our research. Three time series data mining tasks that we are interested are introduced. Section 1.2 provides the definitions and notations to precisely describe the problems under investigation and the proposed algorithms. Section 1.3 provides the techniques and models used in the dissertation.

1.1 Mining of Time Series

A time series is a sequence of ordered values. Time series data widely exists in various domains of our lives. It can come from the sensors in various devices and from records in different disciplines. Thus the mining of time series has drawn much attention in the past two decades. The goal of time series data mining is to discover and apply the information and knowledge in the time series data. Examples of time series data mining tasks include pattern discovery, clustering, classification, rule discovery, summarization, visualization and so on [1].

With the advance of technologies for example, the sensors are becoming cheaper and smaller, thus widely embedded in various devices and machines; we have much more storage than before; everyone is using devices like smartphones which can generate a lot of time series data there is a huge amount of time series data available and the content is changing rapidly. Sometimes we want to process the time series data very fast in real time, which requires the time series data mining methods to have low time complexity, i.e. the running time of the methods should grow slowly as the data size increases.

A desirable property for the time series data mining methods is to have linear time complexity, which means the running time of the algorithms grows linearly with the input data size. With this property, the methods can run very fast and is able to handle large size datasets.

Although there have been a wealth of work on time series data mining, little work is on providing linear time solutions. So the objective is to explore linear time solutions for the time series data mining tasks. With the knowledge and information about the problem structure of the given tasks, we would like to see if we could develop linear time complexity algorithms that are comparable to the state-of-the-art methods in effectiveness.

Our research focuses on three time series data mining tasks: classification, motif discovery, and clustering. These tasks have many applications in various fields including entomology [2–4], music [5, 6], weather prediction [7], seismology [8], astronomy [9], finance [10], gene biology [11] and so on. So the study of these tasks will have great impact in a wide variety of disciplines.

Time series classification can be formulated as follows: given a set of labeled time series, predict the class labels of previously unknown instances. The goal of time series motif discovery is to find the most similar non-overlapping pair of subsequences in a given time series. Time series clustering problem can be formulated as, given a set of unlabeled time series instances, to place them into homogeneous and separated groups.

In this dissertation, we present three linear time complexity methods for the three tasks of time series data mining respectively. For time series classification, we propose a Bag-Of-Pattern-Features (BOPF) method. The approach transforms the time series into the Bag-Of-Patterns representation [12] and considers the patterns in the representation as new features. Then a one-sweep incremental cross validation process is designed to select a subset of the features for effective classification. The centroids and term frequency-inverse document frequency (tf-idf) [13] weights of the new selected features are generated for fast classification.

For time series motif discovery, the proposed algorithm constructs and maintains a data structure called the Grids. The subsequences of a given time series are inserted into the Grids in a random order, and the Grids holds an invariant that the origin offset in the Grids always equals the closest distance of the subsequences already inserted. After inserting all the subsequences, the origin offset of the Grids is the closest subsequence pair distance of the given time series.

For time series clustering, we present a Symbolic Pattern Forest (SPF) method. The approach checks if some randomly selected symbolic patterns exist in the time series to partition the data instances. This partition process is executed multiple times, and the partitions are combined by ensemble to generate the final partition. We demonstrate that group structures in the data can emerge from the random partition process.

For each of the proposed methods, we provide theoretical analysis and perform experiments to demonstrate the linear time complexity. We also conduct extensive experiments, compare with the state-of-the-art methods with statistical analysis to show the effectiveness of the proposed algorithms. It is shown that the proposed methods for the three time series data mining tasks have linear time complexity and they are superior to or competitive with the state-of-the-art methods in effectiveness.

Besides these linear time complexity methods, this dissertation also presents a non-linear time method for time series classification that uses different idea from the state-of-the-arts. The idea behind the state-of-the-art time series classification techniques is to identify and extract patterns or characteristics from the labeled time series and use these patterns to classify new unlabeled data. We find that sequences of values that are very different from the patterns in the labeled time series can be used as references to classify time series effectively. We propose an evolution process to generate these sequences of values, which we call separating references, from the training data. This method is called Evolving Separating References or ESR. It is shown that the proposed ESR method is competitive with the state-of-the-arts in general in accuracy and is superior to other approaches when the training dataset size is small.

1.2 Definitions and Notations

Definition 1.2.1. *A time series T is a sequence of ordered real values $[t_1, t_2, \ldots, t_m]$, where m is the length of the time series.*

Definition 1.2.2. *A subsequence S of length l of time series T is a sequence of contiguous values taken from T: $S = [t_i, t_{i+1}, \ldots, t_{i+l-1}]$, where $1 \leq i \leq m - l + 1$ and $1 \leq l \leq m$. All the subsequences of a certain length from a time series can be extracted by sliding a window of the same length from the start point of the time series to the $(m - l + 1)$th point.*

1.3 Background

1.3.1 Symbolic Aggregate Approximation

Since our methods for classification and clustering use Symbolic Aggregate approXimation (SAX) [14] to transform a time series or subsequence to a symbolic pattern, we briefly introduce this technique. The symbolization can capture important shape features in the time series while filter out the noises, making the methods robust.

Figure 1.1 shows an example of transforming a subsequence to a symbolic pattern (SAX word). The subsequence is z-normalized and divided into ω segments (ω is 2 in this example). The mean value for each segment is computed (the green line and yellow line in the figure for the two segments respectively). These mean values are mapped to symbols according to a set of break points (the gray lines in the figure). These break points divide the value space in equal-probable regions. In this example the alphabet size of SAX is 4 (with an alphabet of 'a', 'b', 'c' and 'd'). The subsequence in the figure is transformed to the symbolic pattern "da". The alphabet size γ, number of segments (world length) ω, and subsequence length l are supplied by the users.

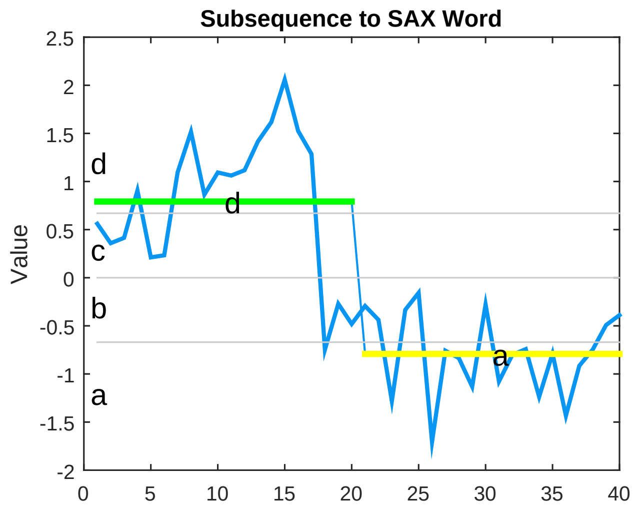


Figure 1.1: Transforming a subsequence to a symbolic pattern with Symbolic Aggregate approXimation (SAX).

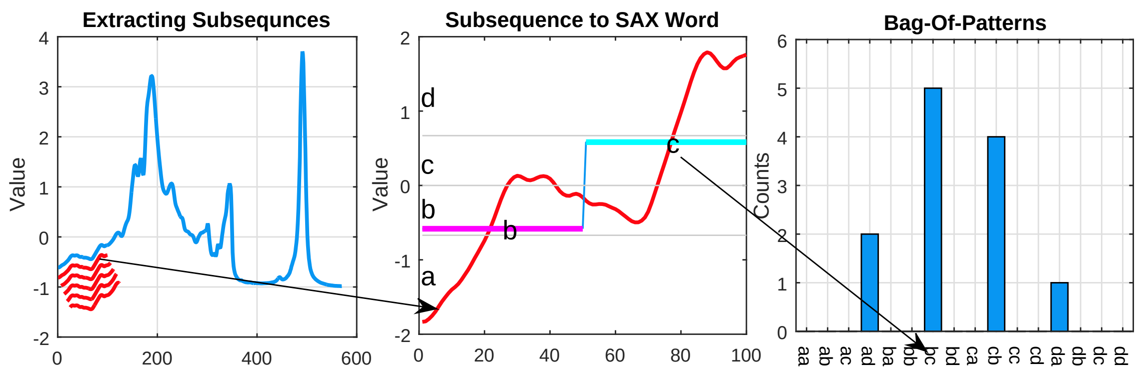


Figure 1.2: (Left) Extracting subsequences (red) from a time series (blue). (Center) Transforming a subsequence to a symbolic pattern (word). (Right) Building a histogram of the words from the time series.

1.3.2 Bag-Of-Patterns (BOP) Representation

Since our BOPF method in Chapter 2 is based on the BOP representation, this subsection briefly introduces this model. The idea of the model is to build histograms of all symbolic patterns in the time series and use these histograms as the new representations. Figure 1.2 provides an example of transforming a real-valued time series into the BOP representation.

In Figure 1.2 (Left), first the subsequences (red) are extracted from the time series (blue) using a sliding window. In Figure 1.2 (Center) each of the subsequences is transformed into a symbolic pattern (SAX word) using the Symbolic Aggregation approXimation (SAX) technique [14]. Concretely, the subsequence is z-normalized and divided into ω segments (2 in our case). The mean values of each segment (the magenta and cyan horizontal line in the figure respectively) are calculated.

These mean values are mapped to symbols according to a set of break points (the grey lines in the figure) that divide the values into equal-probable regions. In our case the alphabet size γ of the SAX transformation is 4 (with an alphabet of 'a', 'b', 'c' and 'd') and the subsequence is thus transformed to the word "bc".

In Figure 1.2 (Right) the counting of all the words forms a histogram that is the BOP representation for the time series. In BOP, as the window slides across the time series, a sequence of words is generated. Only the first encounter of a certain word in the sequence is counted and the following duplicate words are ignored. This process is called numerosity reduction [12]. The sliding window length (subsequence length) l, the number of segments (word length) ω and the alphabet size γ are parameters supplied by the user.

There are several advantages of using the BOP model over the raw time series. The averaging and symbolization process in BOP can filter out the noises in the time series, making the model robust. The original positions of the symbolic patterns are not preserved in the model, thus the model provides invariance to the phase shifts in the time series.

1.4 Outline

The rest of the dissertation is organized as follows:

- **Chapter 2** presents the Bag-Of-Pattern-Features method for time series classification.
- **Chapter 3** proposes linear time algorithms for time series motif discovery.
- **Chapter 4** introduces the Symbolic Pattern Forest approach for time series clustering.

- **Chapter 5** discusses the Evolving Separating References method for time series classification.
- **Chapter 6** gives the conclusion of the dissertation.

Chapter 2: Time Series Classification with Bag-of-Pattern-Features

There has been a lot of work on time series classification and most of the effort focuses on improving the classification accuracy. Recently in [15], the authors conduct a comprehensive experimental comparison of the state-of-the-art time series classification methods, and all of them have super-linear time complexities. The best performing method in accuracy is the Collective of Transformation-based Ensembles (COTE) [16]. This method is an ensemble of 35 classifiers on different data transformations and has a time complexity of $O(n^2m^4)$, where n is the number of training time series instances and m is the length of the time series. This high time complexity makes it inapplicable to large data or real-time analytics where the model may need to be changed frequently.

Although there exists a wealth of work on time series classification, little work focuses on providing a linear time complexity solution. In this chapter, we propose a Bag-Of-Pattern-Features (BOPF) method for time series classification that has a linear time complexity on the number of training instances and on the length of time series ($O(nm)$). The new approach transforms the time series into the Bag-Of-Patterns representation [12] and considers the patterns in the representation as new features. Then a one-sweep incremental cross validation process is designed to select a subset of the features for effective classification. The centroids and term frequency-inverse document frequency (tf-idf) [13] weights of the new selected features are generated for fast classification.

The new BOPF method is evaluated on all the 85 datasets in the well-known UCR time series classification archive [17], and the results are compared with those of a widely used benchmark method, i.e. 1-nearest neighbor with Dynamic Time Warping (DTW) [18], which has a super-linear time complexity. For DTW, we use the state-of-the-art implementation, UCR-Suite, which is optimized for speed [19].

The experimental results show that the new approach has better overall accuracy performance than DTW while achieving orders of magnitude speedup in actual running time. Further experiments are carried out to verify the effectiveness of the designed mechanisms in the new method.

Xeno-canto [20] is a website that enables sharing of bird sounds across the world. People around the world record the bird sounds in the wild and upload the data to the website. Currently, the website has 359960 recordings covering 9742 species. We download 15 gigabytes of audio files from the website and use the Mel-Frequency Cepstral Coefficients (MFCCs) [21] to transform the audio files into time series data. The resulting time series dataset is much larger than any of the datasets in the UCR time series classification archive, and it is one of the largest datasets for time series classification to date. Using the proposed BOPF method, we achieve competitive classification results in very short time on this large dataset. The work presented in this chapter has been published in [22].

2.1 Definition and Related Work

Definition 2.1.1. *A time series dataset D is a set of time series T_i with their respective class labels y_i, where $i = 1, 2, \ldots, n$ and n is the number of instances in the set. Let r denote the class value so $y_i = r_j$, $j \in \{1, \ldots, c\}$. c is the number of classes and usually $c \ll n$. For convenience let St_C denote the set of time series having the same class label C.*

There has been a great amount of work on time series classification. The state-of-the-art techniques can be categorized in two groups. One group consists of whole series based methods. Most of the methods in this group adopt the 1-nearest neighbor classifier with a certain distance measure for the comparison of time series. Existing research focuses on developing and applying alternative distance measures. Euclidean Distance (ED) is a classic distance metric for time series that has a linear computational time complexity. However, this measure does not consider the misalignment and phase-shift in the data, which are common phenomena in real-world time series. Thus, its classification performance is often inferior compared with other, elastic distance measures.

A widely used elastic distance measure is Dynamic Time Warping (DTW) [18]. It applies a dynamic programming process on the two time series under comparison to find an optimal alignment between the values in the two series. The 1-nearest neighbor classifier combined with DTW is regarded as a hard-to-beat standard baseline [23]. The drawback of DTW is its quadratic time complexity, thus some speed-up techniques are developed. The UCR-Suite [19] leverages early abandoning and cascading lower bounds to speedup the computation of DTW. Other alternative distance measures include Weighted DTW [24], Time Warp Edit [25], and Longest Common SubSequence (LCSS) [26].

Another group of time series classification approaches are short-pattern based. This type of methods extract some short patterns from the time series and use them for classification. Some methods in this group are based on the concept of time series shapelets [27–31].

A shapelet is a subsequence in the time series that can best discriminate different classes [27]. In [27], all subsequences in the time series dataset are extracted as candidates and the distances from a candidate to all the time series are calculated. The subsequence with a maximum information gain on splitting the distance values of different classes is regarded as the shapelet. The shapelet is then embedded in a decision tree classifier as the split test for classification. In [29], shapelets are selected using ANOVA F values and in [30], shapelets are learned via optimizing a classification objective function.

Some other methods in the short-pattern group are based on the Bag-Of-Patterns (BOP) [12] representations. The BOP employs a sliding window to extract all the subsequences in the time series and transforms them into symbolic patterns (words). A histogram of the symbolic patterns is built to represent the high-level structure of the time series. Methods based on BOP include BOSS [32] and WEASEL [33].

Some approaches are proposed to promote the scalability of time series classification. Fast Shapelet [34] method transforms the real-value time series into a low-dimension symbolic representation using Symbolic Aggregation approXimation (SAX) [14]. A random projection and hashing process is devised to find the promising shapelet candidates in the symbolic space. In SAX-VSM [35] and BOSS VS [36], the term frequency-inverse document frequency (tf-idf) weighting scheme is used to speed up the classification process. Instead of building a BOP model for each time series, they construct one model for one class of time series, thus improving the speed. These methods show quite significant speed-up, but still none of them has a linear time complexity.

For clarity we first present the idea of our Bag-Of-Pattern-Features (BOPF) method with a concrete small example. Then the formal description of the algorithm will be given.

2.2 A Concrete Illustrative Example

Figure 2.1 shows the time series classification process on a dataset of 4 instances from 2 different classes. The time series are taken from the CBF dataset from the UCR time series classification archive. Two of the time series (blue) are from the class Bell (Figure 2.1 Left, the first two rows) and two (red) are from the class Funnel (Figure 2.1 Left, the last two rows).

In BOPF, the time series are first transformed to the BOP representations (Figure 2.1 Center). Each of the word in the BOP is regarded as a feature and the respective word count becomes the feature value. Then the centroids and term frequency-inverse document frequency (tf-idf) weights can be calculated (Figure 2.1 Right).

Let $Ct_T(w)$ denote the count of a word "w" in the BOP model of a time series T. For a given class C, $Ct_C(w)$ is the sum of $Ct_T(w)$ for all $T \in St_C$, St_C is the set of time series with class C. Then the centroid value of a word "w" of a class C is $centroid(w, C)$:

$$centroid(w, C) = \frac{Ct_C(w)}{|St_C|} \tag{2.1}$$

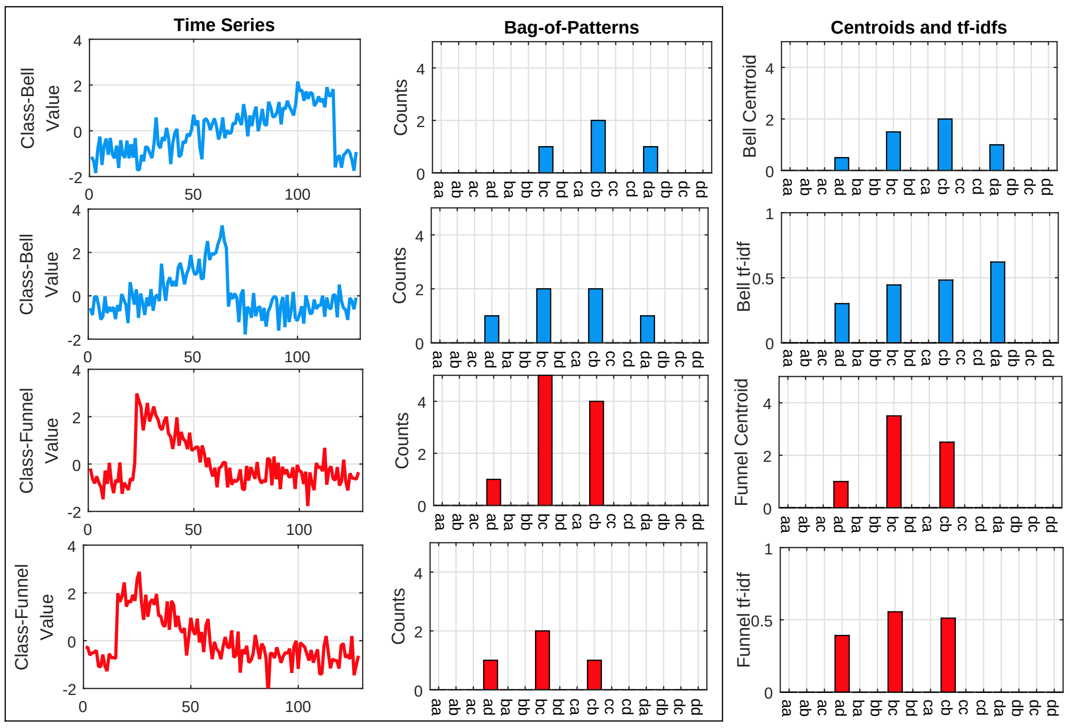


Figure 2.1: An illustrative example of the BOPF classification process. (Left) Time series from 2 classes. (Center) BOP representations of the time series on the left. (Right) The centroids and tf-idf weights of the two classes generated from the BOP representations.

The class centroid of a class C is $centroid(C)$:

$$centroid(C) = (centroid(w_1, C), \ldots, centroid(w_p, C)) \tag{2.2}$$

where "w_p" is the pth word in the BOP representation. The approach in [35][36] is adopted to calculate the tf-idf weights. The term frequency $tf(w, C)$ of a word "w" of a class C is given by:

$$tf(w, C) = \begin{cases} 1 + \log(Ct_C(w)), & \text{if } Ct_C(w) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

The inverse document frequency $idf(w)$ measures the uniqueness of the word "w" belonging to a certain class:

$$idf(w) = \log(1 + \frac{c}{wc}) \tag{2.4}$$

where c is the total number of classes and wc is the number of classes that contain the word "w". The tf-idf weight $tfidf(w, C)$ of a word "w" of a class C is defined as:

$$tfidf(w, C) = tf(w, C) \cdot idf(w) \tag{2.5}$$

The tf-idf weight of a class C is $tfidf(C)$:

$$tfidf(C) = (tfidf(w_1, C), \ldots, tfidf(w_p, C)) \tag{2.6}$$

where "w_p" is the pth word in the BOP representation.

In the 1-nearest neighbor method, an unlabeled test instance is compared with all the n labeled instances in the training set, thus having a complexity of $O(n)$ (assuming length is not considered here), whereas in BOPF, a test instance is compared with all the c class centroids and tf-idf weights to determine its predicted labels. Since $c \ll n$, the complexity is reduced to $O(c) = O(1)$.

In the comparison between a BOP vector Q and a class centroid vector S of length p, the squared Euclidean distance is applied to measure their distance $D(Q, S)$:

$$D(Q, S) = \sum_{i=1}^{p} (q_i - s_i)^2 \tag{2.7}$$

The Squared Cosine similarity is adopted to compare a vector Q and a tf-idf weight S:

$$SC(Q, S) = (\frac{Q \cdot S}{|Q||S|})^2 = \frac{(\sum_{i=1}^{p} q_i s_i)^2}{\sum_{i=1}^{p} q_i \sum_{i=1}^{p} s_i} \tag{2.8}$$

In BOPF, instead of using all the features for classification, only a subset of features

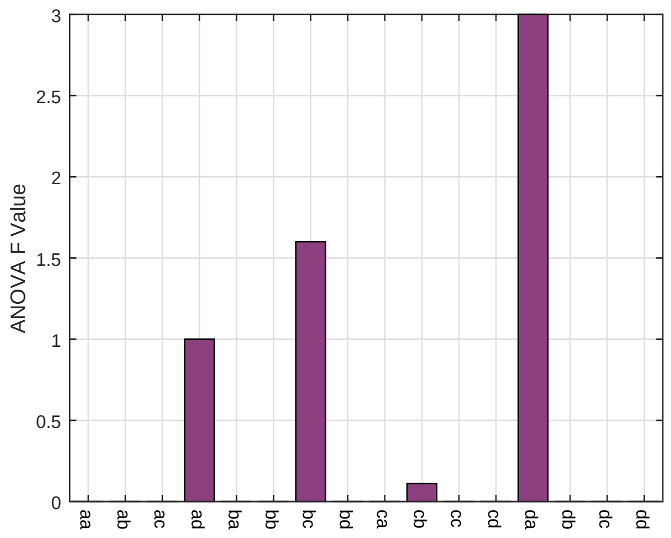


Figure 2.2: ANOVA F values for the features in the illustrative example.

are selected. A one-sweep cross validation process is designed to select the features. First for each of the features, the ANOVA F value [37] is calculated. The result for our example is shown in Figure 2.2.

The ANOVA F value of a feature equals the mean squared variance of the feature values among different classes divided by the mean squared variance within the same class. Thus a large F value indicates that the respective feature can discriminate different classes well. In our example the feature “da” only appears in the Bell class so it receives a large F value. Note that the F values for the features whose values are zero in all the instances are set to 0. We can store only the non-all-zero features to save space.

Assume there are k features with non-zero F values, then k times leave-one-out cross validation are performed. In the ith cross validation, all the training instances with their i best features (by F values we have calculated) are used. In our example, in the first round, only “da” is used to conduct the cross validation to receive a validation accuracy. Then the feature sets {“da”, “bc”}, {“da”, “bc”, “ad”}, {“da”, “bc”, “ad”, “cb”} are used sequentially. The feature set that generates the best validation accuracy is chosen as the Bag-Of-Pattern-Features. Larger size set is preferred when there is a tie.

In the cross validation process, the distance between an instance and the centroid vectors are calculated to decide the validation label of the instance. From equation (2.7), the distance can be computed incrementally. For instance, in our example and in the third round of cross validation, to calculate the distance between two vectors Q and S with feature set {"da", "bc", "ad"}, we can reuse the distance between Q and S with feature set {"da", "bc"} that we got from the previous round. We can just calculate the squared difference of the two "ad" feature values and add it to the previous distance to get the new distance for the third round. Thus in this way, each of the feature value needs to be accessed only once during the k times cross validation, which greatly reduces the running time. The same incremental calculation can also be applied when using the tf-idf weight according to equation (2.8).

When calculating the distance between an instance and the centroid that has the same label, as the instance's feature values contributed to the calculation of the centroid values, the direct distance between them may contain bias. To eliminate this bias, the instance's contribution part is deducted from the centroid before calculating their distance.

Figure 2.3 shows the result of the above cross validation process on our illustrative example. The centroid-based leave-one-out cross validation accuracies on the feature sets {"da"}, {"da", "bc"}, {"da", "bc", "ad"}, {"da", "bc", "ad", "cb"} are 100%, 75%, 75% and 50% respectively. So only "da" is selected as the Bag-Of-Pattern-Feature (Figure 2.3 Left). Figure 2.3 (Center) gives the results when using tf-idf and Figure 2.3 (Right) shows the final feature centroids and tf-idfs for the two classes.

When classifying a new unlabeled test time series, first it is transformed to the BOP representation. Only the features we selected during the training phase are used. The test feature vector is compared to the learned class centroids or tf-idfs to determine its predicted label. Between centroid and tf-idf, we can pick the one that generated better cross validation accuracy in the training phase for testing.

The effectiveness of using a subset of features from the above process instead of using all the features is verified experimentally in Section 2.4.3. The results on 85 datasets show that

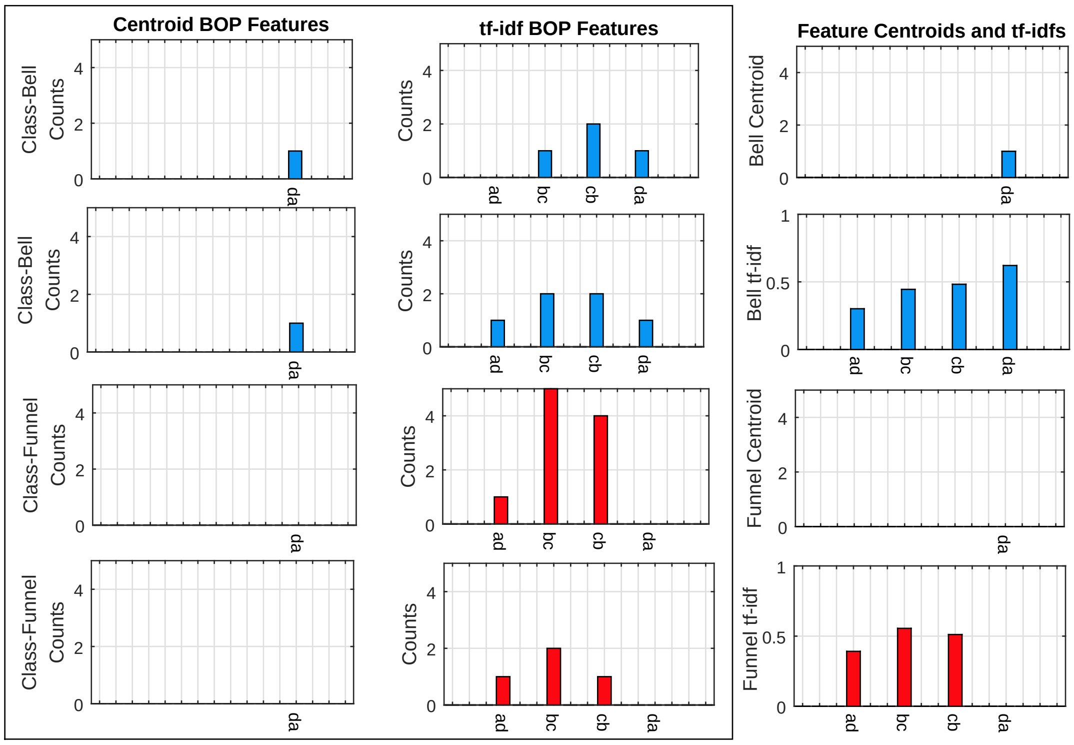


Figure 2.3: The Bag-Of-Pattern-Features generated from the illustrative example. (Left) features generated according to centroid (each row for the respective instance). (Center) features generated according to tf-idf. (Right) the feature centroids and tf-idfs.

just using the subset significantly outperforms employing all the features. Employing the subset can also reduce the classification time as fewer features are used in the classification computation.

2.3 BOPF Algorithm

2.3.1 Efficient SAX Calculation

To transform a time series to the BOP representation, the SAX word of each subsequence needs to be computed. In our algorithm, the cumulative sum technique in [28] is applied to calculate the SAX word of an arbitrary subsequence in $O(1)$ time (given the cumulative sums).

Given a time series T, its cumulative sum series U and cumulative squared sum series V are computed as:

$$u_j = \sum_{i=1}^{j} t_i \tag{2.9}$$

$$v_j = \sum_{i=1}^{j} t_i^2 \tag{2.10}$$

where $j = 1, \dots, m$ and u_0, v_0 are set to 0.

Given a subsequence $x = [t_i, \dots, t_{i+l-1}]$, the mean μ_x and standard deviation σ_x of x can be computed as:

$$\mu_x = \frac{u_{i+l} - u_{i-1}}{l} \tag{2.11}$$

$$\sigma_x = \sqrt{\frac{v_{i+l} - v_{i-1}}{l} - \mu_x^2} \tag{2.12}$$

x is divided into ω segments and each segment is transformed to a symbol. The normalized mean value of a segment $y = [t_i, \dots, t_j]$ is μ_y:

$$\mu_y = \frac{\frac{u_j - u_{i-1}}{j-i+1} - \mu_x}{\sigma_x} \tag{2.13}$$

Then μ_y is compared to the fixed break points to receive the respective symbol [14].

2.3.2 Parameter Selection

In the BOP method, the subsequence length l, SAX word length ω and SAX alphabet size γ are user-set parameters. In BOPF, γ is set to 4 as the previous research suggests this value is suitable for most of the datasets [14][12]. A grid search on the combinations of l and ω is performed to select the best values. The ω candidate set is $wd = \{3, 4, 5, 6, 7\}$ and the l candidate set is $wl = \{0.025, 0.05, 0.075, \dots, 1\}m$, i.e. an arithmetic sequence from $0.025m$ to m with a common difference of $0.025m$, m is the length of the time series. Duplicate values and values less than 10 are removed from wl. In total at most 200 ω and l combinations are tested.

The pseudo-code of BOPF is given in Algorithm 1. Given a training time series dataset D and a test time series Ts, the algorithm returns the predicted label of Ts as output. Lines 1-13 are for the training phase and Lines 14-19 are for the testing phase.

Algorithm 1 Bag-Of-Pattern-Features (BOPF)

```
Input:
    $D$: training time series dataset
    $Ts$: time series to be classified
Output:
    $h$: predicted label of $Ts$
 1: $[T, Label] = loadData(D)$
 2: $[U, V] = CumulativeSum(T)$
 3: for each $\omega \in wd$ do
 4:     for each $l \in wl$ do
 5:         $T_{BOP} = BOP(T, \omega, l, U, V)$
 6:         $T_F = ANOVA(T_{BOP}, Label)$
 7:         $[F_1, T_{cen}, AC_1] = CV_{cen}(T_{BOP}, Label, T_F)$
 8:         $[F_2, T_{tfidf}, AC_2] = CV_{tfidf}(T_{BOP}, Label, T_F)$
 9:         $Co_{cen}.add(\omega, l, F_1, T_{cen}, AC_1)$
10:         $Co_{tfidf}.add(\omega, l, F_2, T_{tfidf}, AC_2)$
11:     end for
12: end for
13: $Co = Selection(Co_{cen}, Co_{tfidf})$
14: for each $co \in Co$ do
15:     $[\omega, l, F, T_{cen}(T_{tfidf})] = co.extract()$
16:     $Ts_{BOP} = BOP(Ts, \omega, l)$
17:     $H_{co} = Classify(Ts_{BOP}, F, T_{cen}(T_{tfidf}))$
18: end for
19: $h = majorityVote(H)$
```

In Line 1, the dataset is loaded and the time series and labels are stored in T and $Label$ respectively. Line 2 computes the cumulative sums U and V for calculating SAX words. Line 3-4 perform the grid search on ω and l discussed before. Line 5 transforms time series T to its BOP representation T_{BOP}. The ANOVA F values for each word are calculated to T_F in Line 6. In Line 7 the cross validation based on centroids is conducted. The selected features F_1, feature centroids T_{cen}, cross validation accuracy AC_1 and the respective ω and l are stored to a combination set Co_{cen} in Line 9. The same process with tf-idf is carried out in Lines 8 and 10.

In Line 13, the best (in validation accuracy) 30 (15% out of 200) parameter combinations (denoted as Co_1) from Co_{cen} are selected. The average accuracy (denoted as $AACo_1$) of the 30 validation accuracies is computed. Similarly Co_2 and $AACo_2$ from Co_{tfidf} are received. If $AACo_1 > 0.7 \max(AACo_1, AACo_2)$, then Co_1 is selected for testing. Similarly if $AACo_2 > 0.7 \max(AACo_1, AACo_2)$, Co_2 is selected and added to Co in Line 13.

In Lines 14-18, each of the combinations co selected in Line 13 is used to transform the test series Ts to its BOP representations Ts_{BOP}. The respective features F, centroid T_{cen}, and tf-idf weight T_{tfidf} are used to predict the label H_{co} of Ts. Finally in Line 19 the majority vote of H is the output h of the algorithm.

The design of taking the top performing parameter combinations and using them to produce a majority vote predication aims at improving the test accuracy. The experiments in Section 2.4.3 verifies the effectiveness of this strategy.

2.3.3 Time Complexity

In Algorithm 1, assume the number of instances is n and the length of time series is m in dataset D, the computation of cumulative sum is in linear time complexity ($O(nm)$). The number of grid search combinations is at most 200. Transforming the time series to BOP models takes $O(nm)$ time. Computing the ANOVA F values takes $O(n)$ time as the number of total features is fixed. With the incremental computation process introduced in Section 2.2, the cross validation takes $O(n)$ time. So the total training time of BOPF is $O(nm)$.

To verify this theoretical result, BOPF is run on training sets with different numbers of instances and different lengths. The time series are taken from the StarLightCurves dataset in the UCR time series classification archive. The training time of fixing the length to 1000 and changing the instance number from 50 to 1000 is presented in Figure 2.4.

In Figure 2.4 the x-axis value of a red dot is the number of training instances and the y-axis value is the average training time of 10 runs on the respective training size. Linear regression curve fitting is performed on the data and the black line in the figure is the fit

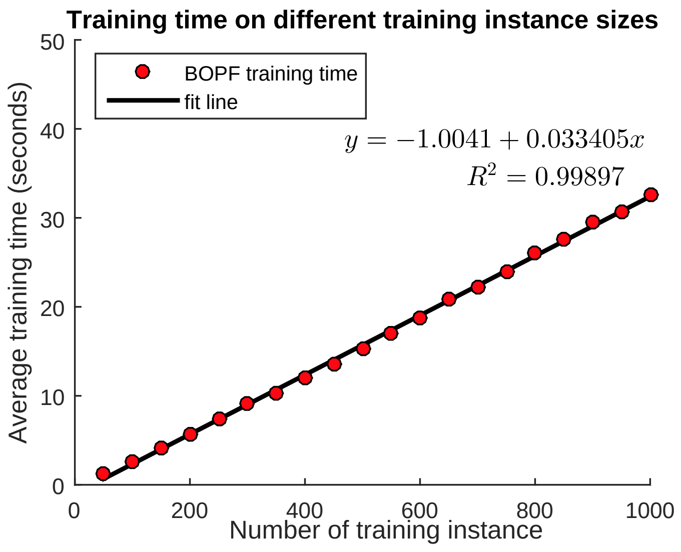


Figure 2.4: Training time of BOPF on different numbers of training instances.

line. The R^2 value of the fitting, which is the coefficient of determination, is 0.99897. This value is very close to 1, indicating the two variables (training time and training size) have a strong linear relationship.

The result of fixing the training instance size to 1000 and changing the time series length from 500 to 1000 is given in Figure 2.5. We start from the length of 500 as when the length is small, many of the subsequence lengths in wl are less than 10 and wl has many duplicate lengths according to the wl definition. These lengths are later removed from the wl, making the training time irregular.

In Figure 2.5 as in Figure 2.4, linear regression curve fitting is performed. The R^2 value is 0.99335 which is very close to 1, indicating the training time and training instance length have a strong linear relationship.

From Algorithm 1, at most 60 parameter combinations are used for testing. The testing time complexity is dominated by the BOP transformation process, which takes $O(m)$ time. So the total time complexity of Algorithm 1 is $O(nm)$.

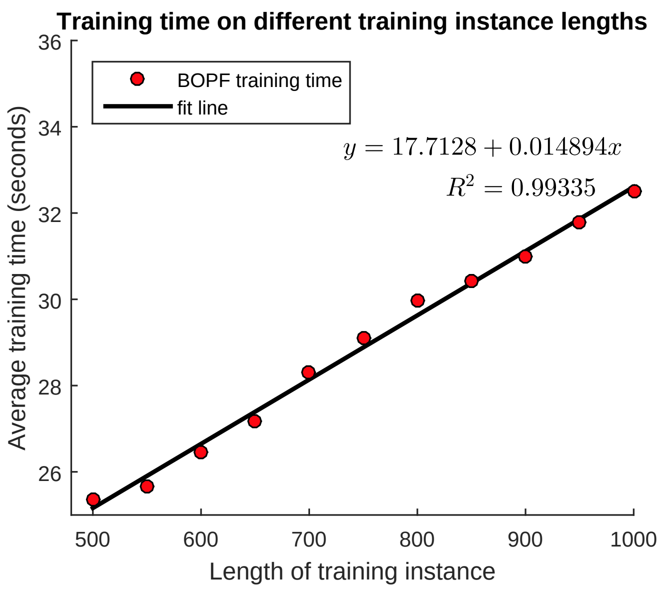


Figure 2.5: Training time of BOPF on different training instance lengths.

2.4 Experimental Evaluation

2.4.1 Experimental Setup

To evaluate the proposed approach, we test it on all 85 datasets in the widely used UCR time series classification archive [17]. The C++ source code of BOPF is available in the supplementary material[1]. The experiments are conducted on a batch-processing cluster [38]. A single core of AMD Opteron Processor 6276 (2299 MHz) and 16 GB memory are used. Note that the algorithms under comparison are all in the same language (C++) and are run under the same hardware condition and operating system. The running time of the algorithms is recorded by using the clock() function in C++.

2.4.2 Comparison between BOPF and Benchmarks

The 1-nearest neighbor classifier with Euclidean Distance (ED), 1-nearest neighbor with full size warping window Dynamic Time Warping (DTW) and 1-nearest neighbor DTW with

[1]`https://github.com/xiaoshengli/BOPF`

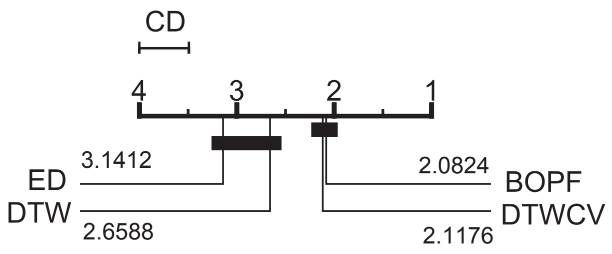


Figure 2.6: Critical difference diagram of the comparison with benchmarks.

the best warping window constraint selected from Cross Validation (DTWCV) are used as benchmarks. These methods are widely used benchmarks in time series classification and they are introduced in Section 2.1. The DTW methods are speeded up by the state-of-the-art UCR-Suite [19].

The four methods are run on 85 datasets and the respective classification error rates and running time are recorded. Table 2.1, 2.2 and 2.3 show the classification error rates of the four methods on the UCR archive.

Figure 2.6 shows the critical difference diagram [39] ($alpha = 0.05$) for the testing error rates comparison. The values in the figure are the respective rank means (lower is better) and the approaches connected by a bold bar have no significant difference to each other. From the figure one can see that the overall classification performance of BOPF in classification accuracy is significantly better than ED and DTW, and slightly better than DTWCV. Performing the Wilcoxon signed rank test on the error rates data, the p-values between BOPF and ED, DTW, DTWCV are 1.5×10^{-6}, 1.8×10^{-3}, 2.3×10^{-1} respectively. The conclusion drawn from these statistics coincides with that from the critical difference diagram.

The time complexity of DTWCV, DTW and BOPF are $O(n^2m^2)$, $O(nm^2)$ and $O(nm)$ respectively. Their actual total running time on the 85 datasets are 1.70×10^6, 9.67×10^4, and 2.73×10^3 seconds respectively. So BOPF is three orders of magnitude faster than DTWCV and one order of magnitude faster than DTW in actual running time.

Table 2.1: Testing error rates of BOPF and other methods on UCR Archive benchmarks

Dataset	ED	DTW	DTWCV	BOPF
50words	0.369	0.31	0.235	0.308
Adiac	0.389	0.396	0.391	0.381
ArrowHead	0.2	0.297	0.2	0.24
Beef	0.333	0.367	0.333	0.3
BeetleFly	0.25	0.3	0.3	0.2
BirdChicken	0.45	0.25	0.3	0.05
Car	0.267	0.267	0.233	0.267
CBF	0.148	0.003	0.006	0
ChlorineConcentration	0.35	0.352	0.35	0.444
CinC_ECG_torso	0.103	0.349	0.07	0.314
Coffee	0	0	0	0.036
Computers	0.424	0.3	0.38	0.308
Cricket_X	0.423	0.246	0.228	0.295
Cricket_Y	0.433	0.256	0.238	0.315
Cricket_Z	0.413	0.246	0.254	0.272
DiatomSizeReduction	0.065	0.033	0.065	0.049
DistalPhalanxOutlineAgeGroup	0.218	0.208	0.228	0.145
DistalPhalanxOutlineCorrect	0.248	0.232	0.232	0.212
DistalPhalanxTW	0.273	0.29	0.273	0.205
Earthquakes	0.326	0.258	0.258	0.183
ECG200	0.12	0.23	0.12	0.18
ECG5000	0.075	0.076	0.075	0.061
ECGFiveDays	0.203	0.232	0.203	0.017
ElectricDevices	0.45	0.399	0.375	0.468
FaceAll	0.286	0.192	0.192	0.204
FaceFour	0.216	0.17	0.114	0
FacesUCR	0.231	0.095	0.088	0.083
FISH	0.217	0.177	0.154	0.069
FordA	0.341	0.438	0.341	0.103
FordB	0.442	0.406	0.414	0.211
Gun_Point	0.087	0.093	0.087	0.02

Table 2.2: (Continue) Testing error rates of BOPF and other methods on UCR Archive benchmarks

Dataset	ED	DTW	DTWCV	BOPF
Ham	0.4	0.533	0.4	0.305
HandOutlines	0.199	0.202	0.197	0.139
Haptics	0.63	0.623	0.588	0.578
Herring	0.484	0.469	0.469	0.406
InlineSkate	0.658	0.616	0.613	0.676
InsectWingbeatSound	0.438	0.645	0.422	0.443
ItalyPowerDemand	0.045	0.05	0.045	0.05
LargeKitchenAppliances	0.507	0.205	0.205	0.243
Lighting2	0.246	0.131	0.131	0.295
Lighting7	0.425	0.274	0.288	0.342
MALLAT	0.086	0.066	0.086	0.057
Meat	0.067	0.067	0.067	0.167
MedicalImages	0.316	0.263	0.253	0.428
MiddlePhalanxOutlineAgeGroup	0.26	0.25	0.253	0.208
MiddlePhalanxOutlineCorrect	0.247	0.352	0.318	0.47
MiddlePhalanxTW	0.439	0.416	0.419	0.398
MoteStrain	0.121	0.165	0.134	0.082
NonInvasiveFatalECG_Thorax1	0.171	0.21	0.189	0.237
NonInvasiveFatalECG_Thorax2	0.12	0.135	0.12	0.176
OliveOil	0.133	0.167	0.133	0.133
OSULeaf	0.479	0.409	0.388	0.202
PhalangesOutlinesCorrect	0.239	0.272	0.239	0.35
Phoneme	0.891	0.772	0.773	0.903
Plane	0.038	0	0	0.01
ProximalPhalanxOutlineAgeGroup	0.215	0.195	0.215	0.156
ProximalPhalanxOutlineCorrect	0.192	0.216	0.21	0.22
ProximalPhalanxTW	0.293	0.263	0.263	0.203
RefrigerationDevices	0.605	0.536	0.56	0.483
ScreenType	0.64	0.603	0.589	0.6
ShapeletSim	0.461	0.35	0.3	0
ShapesAll	0.248	0.232	0.198	0.218

Table 2.3: (Continue) Testing error rates of BOPF and other methods on UCR Archive benchmarks

Dataset	ED	DTW	DTWCV	BOPF
SmallKitchenAppliances	0.659	0.357	0.328	0.195
SonyAIBORobotSurface	0.305	0.275	0.304	0.218
SonyAIBORobotSurfaceII	0.141	0.169	0.141	0.068
StarLightCurves	0.151	0.093	0.095	0.089
Strawberry	0.062	0.06	0.062	0.075
SwedishLeaf	0.211	0.208	0.154	0.163
Symbols	0.1	0.05	0.062	0.052
synthetic_control	0.12	0.007	0.017	0.01
ToeSegmentation1	0.32	0.228	0.25	0.057
ToeSegmentation2	0.192	0.162	0.092	0.085
Trace	0.24	0	0.01	0
TwoLeadECG	0.253	0.096	0.132	0
Two_Patterns	0.09	0	0.002	0.031
UWaveGestureLibraryAll	0.052	0.108	0.034	0.099
uWaveGestureLibrary_X	0.261	0.272	0.227	0.24
uWaveGestureLibrary_Y	0.338	0.366	0.301	0.34
uWaveGestureLibrary_Z	0.35	0.342	0.322	0.31
wafer	0.005	0.02	0.005	0.008
Wine	0.389	0.426	0.389	0.204
WordsSynonyms	0.382	0.351	0.252	0.382
Worms	0.635	0.536	0.586	0.481
WormsTwoClass	0.414	0.337	0.414	0.293
yoga	0.17	0.164	0.156	0.257
Rank mean	3.141	2.659	2.118	2.082

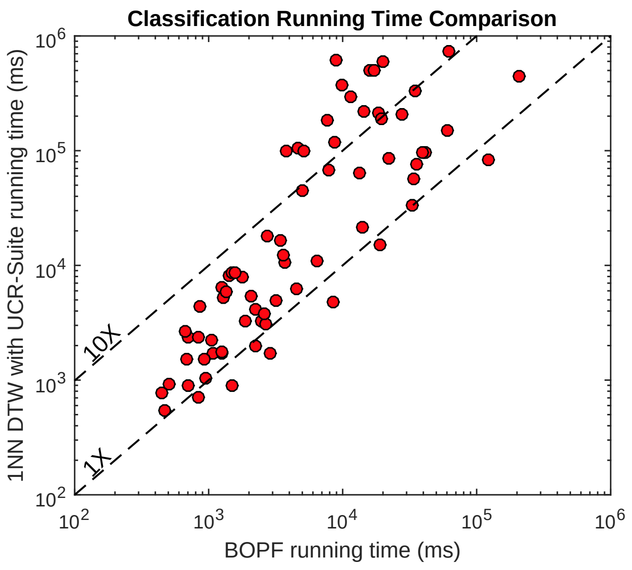


Figure 2.7: Pairwise comparison of running time between BOPF and DTW on 85 datasets.

Figure 2.7 (Figure 2.8) provides the pairwise running time comparison between BOPF and DTW (DTWCV) on the 85 datasets. Each dot represents a dataset and the x-axis value is the running time (in log-scale and milliseconds) of BOPF on the dataset. The y-axis value in Figure 2.7 (Figure 2.8) is the running time of DTW (DTWCV). The speed-up boundary lines are marked in the figures. From the figures and the raw data in [40], one observes that the larger the dataset is, the more speed-up one can receive from using BOPF.

The above experimental results indicate that BOPF can provide competitive classification results using quite a short time. In the situations where a linear time complexity is required, e.g. when dealing with large size datasets and when in real-time analytics, BOPF is considered as a good solution.

2.4.3 Effectiveness of Design Strategies

BOPF employs an incremental cross validation procedure to select a subset of features instead of using all of them. Also the top (30) parameter combinations are selected and used to predict the label with a majority vote. To verify the effectiveness of these strategies,

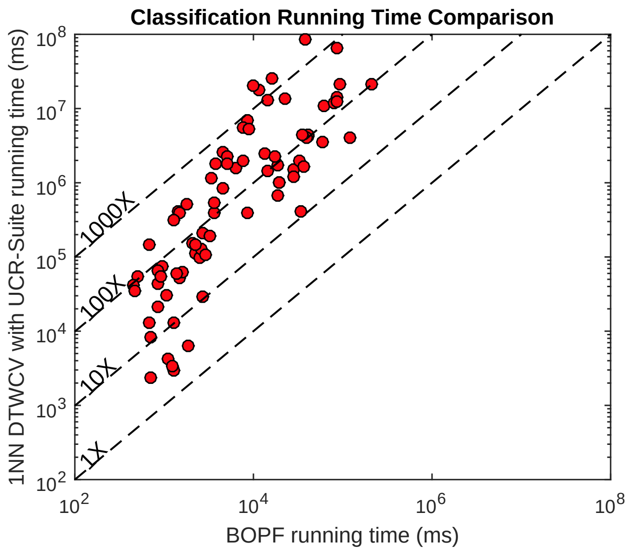


Figure 2.8: Pairwise comparison of running time between BOPF and DTWCV on 85 datasets.

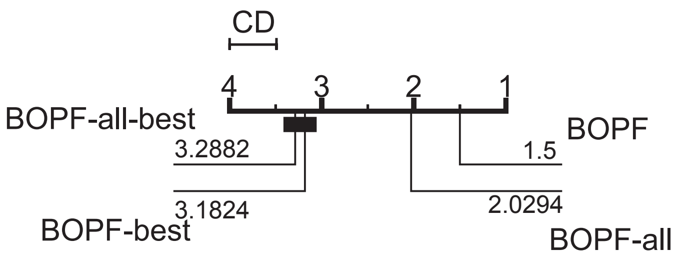


Figure 2.9: Critical difference diagram of comparison between BOPF and BOPF-all, BOPF-best, BOPF-all-best.

4 versions of BOPF are run on the 85 datasets and their classification performances are compared.

These versions are BOPF, BOPF using all the features (BOPF-all), BOPF using only the best parameter combination (BOPF-best) for prediction, BOPF with all the features and with only the best parameter combination for prediction (BOPF-all-best). Figure 2.9 presents the critical difference diagram for the comparison. The raw data is given in Table 2.4, 2.5 and 2.6.

Table 2.4: Testing error rates of BOPF and BOPF-all, BOPF-best, BOPF-all-best on UCR Archive benchmarks

Dataset	BOPF	BOPF-all	BOPF-best	BOPF-all-best
50words	0.308	0.325	0.376	0.396
Adiac	0.381	0.361	0.56	0.565
ArrowHead	0.24	0.286	0.343	0.583
Beef	0.3	0.3	0.633	0.633
BeetleFly	0.2	0.25	0.3	0.3
BirdChicken	0.05	0.05	0.4	0.35
Car	0.267	0.267	0.65	0.583
CBF	0	0.001	0.187	0.06
ChlorineConcentration	0.444	0.616	0.429	0.486
CinC_ECG_torso	0.314	0.263	0.499	0.391
Coffee	0.036	0.036	0.036	0.143
Computers	0.308	0.304	0.268	0.32
Cricket_X	0.295	0.31	0.718	0.718
Cricket_Y	0.315	0.318	0.844	0.844
Cricket_Z	0.272	0.272	0.674	0.664
DiatomSizeReduction	0.049	0.033	0.131	0.062
DistalPhalanxOutlineAgeGroup	0.145	0.153	0.19	0.218
DistalPhalanxOutlineCorrect	0.212	0.255	0.32	0.315
DistalPhalanxTW	0.205	0.265	0.338	0.345
Earthquakes	0.183	0.217	0.317	0.227
ECG200	0.18	0.19	0.32	0.25
ECG5000	0.061	0.087	0.082	0.121
ECGFiveDays	0.017	0.029	0.213	0.213
ElectricDevices	0.468	0.362	0.518	0.49
FaceAll	0.204	0.204	0.237	0.237
FaceFour	0	0	0.273	0.023
FacesUCR	0.083	0.08	0.141	0.141
FISH	0.069	0.091	0.183	0.177
FordA	0.103	0.178	0.197	0.227
FordB	0.211	0.355	0.241	0.267

Table 2.5: (Continue) Testing error rates of BOPF and BOPF-all, BOPF-best, BOPF-all-best on UCR Archive benchmarks

Dataset	BOPF	BOPF-all	BOPF-best	BOPF-all-best
Gun_Point	0.02	0.013	0.38	0.12
Ham	0.305	0.352	0.438	0.352
HandOutlines	0.139	0.146	0.138	0.151
Haptics	0.578	0.575	0.627	0.601
Herring	0.406	0.359	0.391	0.438
InlineSkate	0.676	0.669	0.869	0.691
InsectWingbeatSound	0.443	0.447	0.481	0.481
ItalyPowerDemand	0.05	0.051	0.127	0.114
LargeKitchenAppliances	0.243	0.472	0.245	0.365
Lighting2	0.295	0.295	0.508	0.393
Lighting7	0.342	0.425	0.836	0.63
MALLAT	0.057	0.035	0.481	0.503
Meat	0.167	0.2	0.25	0.3
MedicalImages	0.428	0.528	0.518	0.718
MiddlePhalanxOutlineAgeGroup	0.208	0.243	0.233	0.448
MiddlePhalanxOutlineCorrect	0.47	0.47	0.548	0.573
MiddlePhalanxTW	0.398	0.491	0.398	0.409
MoteStrain	0.082	0.088	0.292	0.301
NonInvasiveFatalECG_Thorax1	0.237	0.208	0.303	0.313
NonInvasiveFatalECG_Thorax2	0.176	0.149	0.217	0.226
OliveOil	0.133	0.1	0.133	0.2
OSULeaf	0.202	0.161	0.512	0.517
PhalangesOutlinesCorrect	0.35	0.366	0.387	0.367
Phoneme	0.903	0.913	0.918	0.913
Plane	0.01	0.01	0.029	0.01
ProximalPhalanxOutlineAgeGroup	0.156	0.22	0.176	0.22
ProximalPhalanxOutlineCorrect	0.22	0.203	0.265	0.258
ProximalPhalanxTW	0.203	0.23	0.228	0.25
RefrigerationDevices	0.483	0.501	0.557	0.685
ScreenType	0.6	0.587	0.555	0.576
ShapeletSim	0	0.156	0.05	0.133

Table 2.6: (Continue) Testing error rates of BOPF and BOPF-all, BOPF-best, BOPF-all-best on UCR Archive benchmarks

Dataset	BOPF	BOPF-all	BOPF-best	BOPF-all-best
ShapesAll	0.218	0.208	0.337	0.312
SmallKitchenAppliances	0.195	0.299	0.235	0.24
SonyAIBORobotSurface	0.218	0.211	0.16	0.16
SonyAIBORobotSurfaceII	0.068	0.071	0.248	0.171
StarLightCurves	0.089	0.234	0.148	0.124
Strawberry	0.075	0.086	0.064	0.093
SwedishLeaf	0.163	0.202	0.202	0.245
Symbols	0.052	0.058	0.333	0.152
synthetic_control	0.01	0.01	0.277	0.277
ToeSegmentation1	0.057	0.066	0.215	0.206
ToeSegmentation2	0.085	0.077	0.177	0.146
Trace	0	0	0.01	0
TwoLeadECG	0	0.001	0.193	0.091
Two_Patterns	0.031	0.032	0.419	0.419
UWaveGestureLibraryAll	0.099	0.099	0.143	0.143
uWaveGestureLibrary_X	0.24	0.243	0.361	0.356
uWaveGestureLibrary_Y	0.34	0.34	0.376	0.374
uWaveGestureLibrary_Z	0.31	0.312	0.427	0.41
wafer	0.008	0.009	0.005	0.003
Wine	0.204	0.185	0.574	0.296
WordsSynonyms	0.382	0.397	0.469	0.466
Worms	0.481	0.503	0.652	0.586
WormsTwoClass	0.293	0.254	0.315	0.381
yoga	0.257	0.269	0.471	0.496

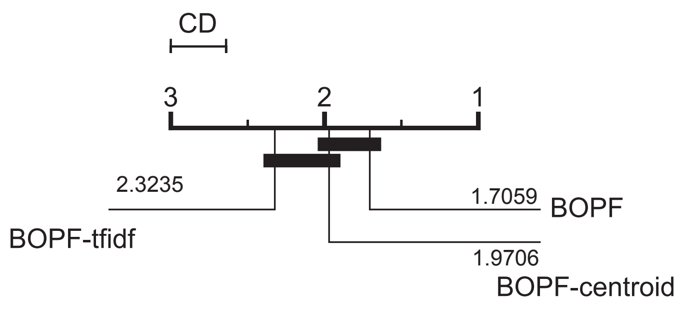


Figure 2.10: Critical difference diagram of comparison between BOPF and BOPF-centroid, BOPF-tfidf.

One can see from the figure that BOPF is significantly better than BOPF-all, which verifies the effectiveness of using a subset of features instead of using all of them. Comparing BOPF and BOPF-best, BOPF is significantly better. This indicates that using the top combinations to make a majority vote is superior to using only the best combination. BOPF-all-best is the worst version in the experiment, which coincides with the above discussion.

BOPF adopts a combination of feature centroids and tf-idf weights. To verify this design, 3 versions of BOPF are compared: BOPF, BOPF with only centroids (BOPF-centriod), and BOPF with only tf-idf (BOPF-tfidf). Figure 2.10 gives the critical difference diagram of the comparison and the raw data is in Table 2.7, 2.8, and 2.9. The overall performance of BOPF is superior to those of BOPF-centroid and BOPF-tfidf. This result shows that using the combination of centroids and tf-idf weights can generate better performance than just using a single method.

2.5 Case Study

Xeno-canto [20] is a website of sharing bird sounds around the world. People across the world record the bird sounds in wild and upload the data to the website. We download 15 gigabytes bird sounds from the website and the birds are from two different orders, i.e. Charadriiformes and Piciformes respectively.

Table 2.7: Testing error rates of BOPF and BOPF-centroid, BOPF-tfidf on UCR Archive benchmarks

Dataset	BOPF	BOPF-centroid	BOPF-tfidf
50words	0.308	0.371	0.308
Adiac	0.381	0.34	0.627
ArrowHead	0.24	0.229	0.297
Beef	0.3	0.267	0.367
BeetleFly	0.2	0.2	0.3
BirdChicken	0.05	0.1	0
Car	0.267	0.25	0.3
CBF	0	0.003	0.004
ChlorineConcentration	0.444	0.452	0.45
CinC_ECG_torso	0.314	0.411	0.304
Coffee	0.036	0.036	0.036
Computers	0.308	0.288	0.296
Cricket_X	0.295	0.431	0.295
Cricket_Y	0.315	0.438	0.315
Cricket_Z	0.272	0.392	0.272
DiatomSizeReduction	0.049	0.049	0.046
DistalPhalanxOutlineAgeGroup	0.145	0.148	0.165
DistalPhalanxOutlineCorrect	0.212	0.227	0.25
DistalPhalanxTW	0.205	0.215	0.22
Earthquakes	0.183	0.18	0.193
ECG200	0.18	0.18	0.19
ECG5000	0.061	0.08	0.064
ECGFiveDays	0.017	0.009	0.091
ElectricDevices	0.468	0.479	0.481
FaceAll	0.204	0.221	0.203
FaceFour	0	0	0
FacesUCR	0.083	0.131	0.069
FISH	0.069	0.069	0.154
FordA	0.103	0.113	0.138
FordB	0.211	0.155	0.318

Table 2.8: (Continue) Testing error rates of BOPF and BOPF-centroid, BOPF-tfidf on UCR Archive benchmarks

Dataset	BOPF	BOPF-centroid	BOPF-tfidf
Gun_Point	0.02	0.007	0.1
Ham	0.305	0.295	0.343
HandOutlines	0.139	0.138	0.164
Haptics	0.578	0.552	0.62
Herring	0.406	0.453	0.359
InlineSkate	0.676	0.647	0.676
InsectWingbeatSound	0.443	0.428	0.443
ItalyPowerDemand	0.05	0.054	0.049
LargeKitchenAppliances	0.243	0.237	0.296
Lighting2	0.295	0.295	0.262
Lighting7	0.342	0.315	0.466
MALLAT	0.057	0.068	0.069
Meat	0.167	0.133	0.25
MedicalImages	0.428	0.42	0.58
MiddlePhalanxOutlineAgeGroup	0.208	0.208	0.203
MiddlePhalanxOutlineCorrect	0.47	0.47	0.477
MiddlePhalanxTW	0.398	0.386	0.431
MoteStrain	0.082	0.093	0.093
NonInvasiveFatalECG_Thorax1	0.237	0.188	0.452
NonInvasiveFatalECG_Thorax2	0.176	0.133	0.38
OliveOil	0.133	0.133	0.167
OSULeaf	0.202	0.145	0.401
PhalangesOutlinesCorrect	0.35	0.352	0.366
Phoneme	0.903	0.773	0.903
Plane	0.01	0	0.029
ProximalPhalanxOutlineAgeGroup	0.156	0.166	0.141
ProximalPhalanxOutlineCorrect	0.22	0.254	0.189
ProximalPhalanxTW	0.203	0.21	0.21
RefrigerationDevices	0.483	0.491	0.507
ScreenType	0.6	0.571	0.6
ShapeletSim	0	0	0.011

Table 2.9: Testing error rates of BOPF and BOPF-centroid, BOPF-tfidf on UCR Archive benchmarks

Dataset	BOPF	BOPF-centroid	BOPF-tfidf
ShapesAll	0.218	0.223	0.287
SmallKitchenAppliances	0.195	0.2	0.237
SonyAIBORobotSurface	0.218	0.183	0.298
SonyAIBORobotSurfaceII	0.068	0.097	0.061
StarLightCurves	0.089	0.104	0.115
Strawberry	0.075	0.073	0.072
SwedishLeaf	0.163	0.168	0.347
Symbols	0.052	0.074	0.043
synthetic_control	0.01	0.04	0.043
ToeSegmentation1	0.057	0.048	0.079
ToeSegmentation2	0.085	0.131	0.062
Trace	0	0	0
TwoLeadECG	0	0.001	0.015
Two_Patterns	0.031	0.082	0.015
UWaveGestureLibraryAll	0.099	0.126	0.108
uWaveGestureLibrary_X	0.24	0.284	0.263
uWaveGestureLibrary_Y	0.34	0.365	0.343
uWaveGestureLibrary_Z	0.31	0.344	0.339
wafer	0.008	0.011	0.002
Wine	0.204	0.222	0.185
WordsSynonyms	0.382	0.458	0.382
Worms	0.481	0.42	0.624
WormsTwoClass	0.293	0.298	0.309
yoga	0.257	0.278	0.27

The Mel-Frequency Cepstral Coefficients (MFCCs) [21] is adopted to transform each audio file into a 13-dimension time series. We only use the second dimension data in this study and only the first 1000 points of the transformed time series is used. This results in a time series dataset of 24471 instances and 1000 length. This dataset is much larger than any of the 85 datasets in the UCR time series classification archive and is one of the largest datasets for time series classification to date. The labels of the time series are the orders of the birds that produce the sounds. One third of the instances in the dataset are randomly selected to form a training set and the rest instances form a testing set. The dataset is provided in [40].

BOPF is run on this dataset to test its performance on large size data. Since the dataset is quite large, it is time-consuming to run super-linear time complexity algorithm on this dataset. So we just run the linear time complexity benchmark 1-nearest neighbor with Euclidean Distance (ED) for comparison. The same experimental setting described in Section 2.4 is used.

The ED does not have training phase and it takes 428 seconds for the testing phase on the bird sounds dataset. Its testing error rate is 0.474. On the other hand BOPF spends 525 seconds on the training phase and 171 seconds on the testing phase, receiving a testing error rate of 0.384. So compared with ED, BOPF can achieve a much better classification accuracy result using comparable total running time on this large dataset. If only considering the testing time, which is in the situations where the model has been trained before, BOPF is even much faster than ED. The above results and analysis show the utility of BOPF when dealing with large size data.

2.6 Conclusion of BOPF

The era of big data calls for low time complexity data mining methods to cope with the huge-volume and fast-changing data. This chapter proposes a linear time complexity time series classification method Bag-Of-Pattern-Features (BOPF). The experiments on all the 85 datasets in the UCR time series classification archive show that the method can provide competitive results in a quite short time. Further experiments demonstrate the effectiveness of the designs in BOPF. A case study on the bird sounds data shows the utility of the method on real-world large size data.

Chapter 3: Linear Time Motif Discovery

A time series is a sequence of ordered values, and time series motifs are defined as repeated patterns in the time series data. Figure 3.1 shows an example of a motif found in an Electro-Oculogram (EOG) dataset [41]. The motifs in time series provide important information about the systems that generate the data and offer valuable insights for the problem under investigation. Domain experts can use the motifs to confirm their understanding of the system. Analyzing motifs may result in new knowledge for the domain.

Motif discovery has been applied to various fields such as entomology [2–4], music [5,6], weather prediction [7], seismology [8] and so on. Finding motifs is an important task in time series data mining. There are several variants of definitions for time series motifs [42,43]. In this chapter, we adopt the classic definition: the most similar non-overlapping pair of subsequences of a given length in the time series data. The motif discovery problem can be formulated as, given a user-set subsequence length l, find the most similar non-overlapping pair of subsequences in a given time series of length n. In this chapter, we assume l is a fixed constant and $l \ll n$.

The state-of-the-art methods solve the exact motif discovery problem in quadratic time complexity [2,44,45]. In this chapter, we present an algorithm that can find the exact motif in expected $O(n)$ time complexity. The algorithm is further modified to find all pairs of subsequences whose distances are below a given threshold value in a linear expected time complexity.

Figure 3.2 shows a motif of zero-distance found by the proposed method in a DNA time series of length 230 million. The DNA sequence is the Chromosome 1 (Chr1) of the human genome (GRCh38 / hg38) from [46] and we use the method in [19] to transform the raw DNA sequence to time series. The start locations of the motif subsequences are given in the figure.

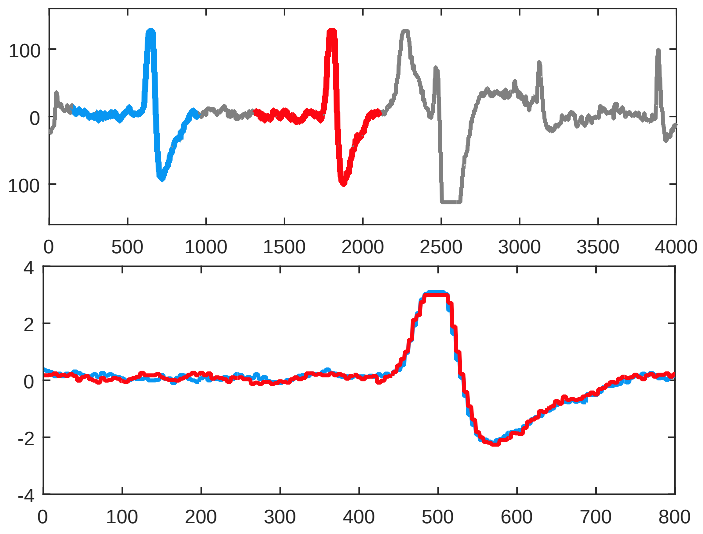


Figure 3.1: (top) Time series data from the EOG dataset with two motif subsequences (marked by blue and red respectively). (bottom) Normalizing and aligning the two motif subsequences from the top figure.

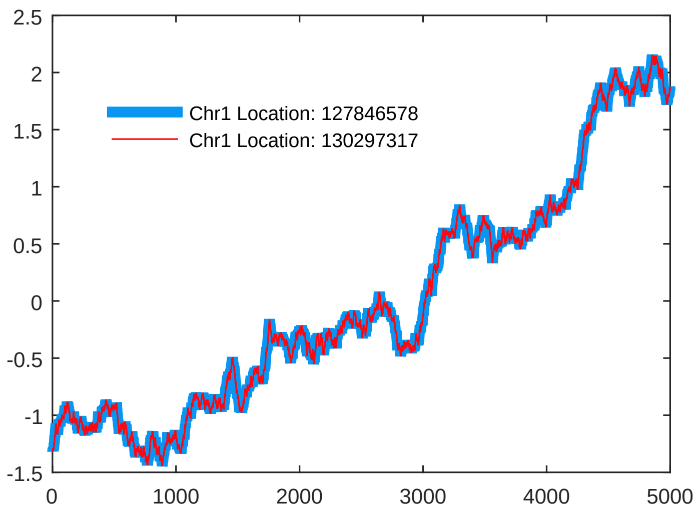


Figure 3.2: A motif found from a 230-million-length DNA time series.

If using the quadratic brute-force method to find the motif, i.e. comparing all pair-wise subsequences in the time series, it needs to calculate $(2.3 \times 10^8 - 5000 + 1) \times (2.3 \times 10^8 - 5000)/2 = 2.6449 \times 10^{16}$ pairs distances. Assume each distance can be computed in 10^{-6} seconds, then the total time the brute-force method takes is 2.6449×10^{10} seconds, or 838.7 years. The state-of-the-art STOMP (Matrix Profile) [45] takes 38.5 years (estimated) while our proposed algorithm takes only 7.9 hours to run on the same data.

The core idea of the proposed algorithm is to construct and maintain a data structure called the Grids. The subsequences of a given time series are inserted into the Grids in a random order, and the Grids holds an invariant that the origin offset in the Grids always equals the closest distance of the subsequences already inserted. After inserting all the subsequences, the origin offset of the Grids is the closest subsequence pair distance of the given time series.

We provide the proof of correctness and time complexity in the chapter. In practice, when true motifs exist (i.e. the most similar pairs have small distances) in the data or the threshold of the algorithm is set to a small value, the constant term in the time complexity becomes small and thus the algorithm runs very fast.

We carry out experiments to verify the theoretical analysis, and compare the actual running time of the proposed algorithm with that of the state-of-the-art. In addition, we conduct case studies on some real-world data to show the effectiveness of the proposed method. The work presented in this chapter has been published in [47].

3.1 Definitions and Related Work

Here we provides the definitions and notations that will be used in this chapter:

Definition 3.1.1. *The Minkowski distance Dist of two subsequences S and Q is:*

$$Dist(S, Q) = (\sum_{i=1}^{l} |\hat{s}_i - \hat{q}_i|^p)^{1/p} \tag{3.1}$$

where $\hat{S}$ and $\hat{Q}$ are the z-normalized subsequences of S and Q respectively. The subsequences are normalized to provide scale invariance.

When $p = 2$, Dist is the Euclidean distance, and when $p = \infty$, Dist is the Chebyshev distance:

$$Dist(S, Q) = \max_{i=1}^{l} |\hat{s}_i - \hat{q}_i| \tag{3.2}$$

Definition 3.1.2. *Given a time series T of length n and a subsequence length l, the* ***motif discovery*** *problem is to find the non-overlapping pair of subsequences of length l that have the minimum distance among all pairs of subsequences in T. Formally, the problem is to find (S_i, S_j) such that for $|k - q| \geq l$ and $|i - j| \geq l$, we have $Dist(S_i, S_j) \leq Dist(S_k, S_q)$ for $\forall k, q$.*

Definition 3.1.3. *The problem can be generalized to the* ***motif set discovery*** *problem: given a user-defined threshold value r, find all pairs of subsequences such that each pair has a distance smaller than r. Formally, the problem is to find $\{(S_i, S_j)\}$ such that $Dist(S_i, S_j) \leq r$, $|i - j| \geq l$ for $\forall i, j$.*

When the time series is long, the number of subsequence pairs in the time series becomes massive and thus very difficult to handle. So a popular research direction is to transform the time series into low-dimensional representations and find motif candidates in the new representations. Symbolic Aggregate approXimation [14] is one of the most used representations which transforms time series into strings and then other techniques such as random projection [48] and grammar induction [49, 50] can be adopted to find the motif candidates efficiently.

Mueen-Keogh (MK) [2] is the first practical method to find the exact motif in a long time series. This method randomly chooses some subsequences from the time series as references and calculates the distances between the references and all the subsequences in the time series. With these distances and the triangle inequality, large numbers of motif candidate pairs can be pruned off and thus the efficiency is enhanced.

Quick-Motif [44] groups the piece-wise aggregate approximation representations of subsequences into Minimum Bounding Rectangles (MBRs). The MBRs are grouped by a Hilbert R-tree and the distances between two MBRs are calculated with the incremental technique in [3].

STOMP (Matrix Profile) [45] calculates the pair-wise distances in a specific order such that the computation can be reused to improve the performance. To compute a pair-wise distance, the sum of element-wise products of the subsequences are used. As the subsequences in a time series have overlaps, the sum of element-wise product of (S_i, S_j) can be obtained from that of (S_{i-1}, S_{j-1}) in constant time.

The idea of the proposed algorithm in this chapter is inspired by [51], which finds the closest pair in a set of points. The algorithm uses a grid data structure and inserts points to the grid incrementally. During the process, the grid mesh size is equal to the closest pair distance among the points that have been inserted. In each insertion, the algorithm needs to check 3^D grid cells where D is the dimension of the points. We find that when $D > 6$, the algorithm's actual running time is greater than that of the brute-force algorithm ($n = 10000$). So it cannot be used for time series motif discovery as the motif dimension D (or subsequence length in our context) is usually much larger. The proposed method in this chapter devises a new data structure that only needs to check $D+1$ cells to ensure the exactness of the result in each insertion of a subsequence.

As mentioned before, the process of the proposed algorithm is to maintain and update a Grids data structure incrementally. So we first present the Grids data structure, and then detail the algorithms based on it.

3.2 The Grids Data Structure

For illustration purpose, in this section we set the subsequence length l to 2 so that we can draw the subsequences on the plane. The Grids data structure contains a serial of grids with different coordinate origins. Figure 3.3a shows an example of a grid. The grid divides

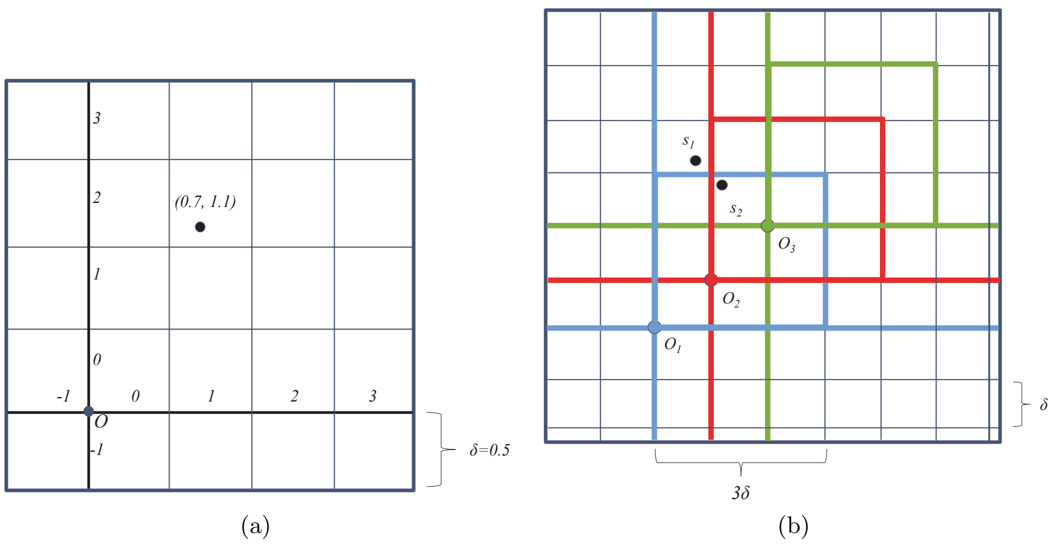


Figure 3.3: Illustrative examples of the grid and Grids data structure.

the space into cells with a cell size of $\delta = 0.5$. Each cell has an index and the cell index of a subsequence (x, y) can be computed by $(\lfloor x/\delta \rfloor, \lfloor y/\delta \rfloor)$. In Figure 3.3a, the cell index of the subsequence $(0.7, 1.1)$ is $(1, 2)$.

In practice, we can implement the grid data structure with a hash table of key-value pairs. To insert a subsequence into the grid, one needs to compute the cell index it belongs to and use the index as the key. To save space, we can combine the hash values of the two index elements into one hash value and use it as the key for the subsequence. The value of a key-value pair in the hash table is a list of subsequences falling into the respective cell. Instead of storing the actual whole subsequence values in the hash table, we just store the start location of the subsequence in the time series. As the motif length is fixed, knowing the start location can retrieve all the values in the subsequence.

The Grids data structure contains g grids where g is a user-set parameter. Each grid has a grid size of $g\delta$ and their coordinate origins are offset by δ in each dimension from the previous grid origin. The first origin has a coordinate of $(0, 0)$, the second origin has a coordinate of (δ, δ), and the third has $(2\delta, 2\delta)$ and so on. Figure 3.3b shows an example of a Grids with $g = 3$. The origins of the three grids are O_1, O_2 and O_3 respectively (marked by blue, red and green in the figure, the respective coordinate axis are also in the same color). If $\delta = 1$ here then the coordinates of O_1, O_2 and O_3 are $(0, 0)$, $(1, 1)$, and $(2, 2)$ respectively. The $(0, 0)$ grid index cells of each grid are highlighted in the figure by their respective color (note that $g\delta = 3$ is the grid size). In the figure, the indices of $S_1(0.8, 3.2)$ are $(0, 1)$, $(-1, 0)$, $(-1, 0)$ and the indices of $S_2(1.1, 2.9)$ are $(0, 0)$, $(0, 0)$ $(-1, 0)$ for the three grids respectively.

For a subsequence (x, y), its indices for the three grids can be computed as $(\lfloor x/3\delta \rfloor, \lfloor y/3\delta \rfloor)$, $(\lfloor (x-\delta)/3\delta \rfloor, \lfloor (y-\delta)/3\delta \rfloor)$ and $(\lfloor (x-2\delta)/3\delta \rfloor, \lfloor (y-2\delta)/3\delta \rfloor)$ respectively. In general for a Grids of g grids, the indices of (x, y) are $(\lfloor x/g\delta \rfloor, \lfloor y/g\delta \rfloor)$, $(\lfloor (x-\delta)/g\delta \rfloor, \lfloor (y-\delta)/g\delta \rfloor)$, $\ldots$, $(\lfloor (x-(g-1)\delta)/g\delta \rfloor, \lfloor (y-(g-1)\delta)/g\delta \rfloor)$ respectively.

3.3 Algorithm for Motif Discovery

Algorithm 2 gives the pseudo-code of the algorithm for motif discovery. Given a time series T, a motif length l, and a grid number g, the algorithm outputs the motif subsequence pair M and the motif pair distance δ.

Line 1 gets the length of time series n. In Line 2, A is a random permutation of the start locations of all the subsequences in T. Line 3 calculates the mean μ and standard deviation σ for each subsequence. These statistics are used to normalize the subsequences during the process. With the running sum techniques [28], each mean and standard deviation can be computed in $O(1)$ time. Line 4 calculates the distance between the subsequences represented by A_1 and A_2 and assigns the distance to δ. In Line 5, M is used to record the subsequence pair.

Line 6 builds a Grids of g grids with grid size $g\delta$ and origin offset δ. The Grids contains the first 2 elements of A (A_1, A_2). More concretely, the buildGrids takes the subsequences values and δ, calculates the key values for the g grids, and inserts the key-value pairs to the Grids hash tables. Each subsequence is inserted to g tables corresponding to the g grids in

Algorithm 2

```
Input:
    $T$: time series
    $l$: motif length
    $g$: number of grids
Output:
    $\delta$: motif distance
    $M$: motif
1:  $n = T.length$
2:  $A = randomPermutation([1, \ldots, n - l + 1])$
3:  $\mu, \sigma = statistics(T, n, l)$
4:  $\delta = Dist(A_1, A_2, T, l, \mu, \sigma)$
5:  $M = (A_1, A_2)$
6:  $G = buildGrids(T, g, \delta, A, 2, l, \mu, \sigma)$
7:  for $i = 2$ to $n - l$ do
8:      $neighbors = retrieveGrids(G, A_{i+1}, \delta, \mu, \sigma)$
9:      $M, d = minDist(A_{i+1}, neighbors, \delta, M, \mu, \sigma)$
10:     if $d \geq \delta$ then
11:         $G = insertGrids(G, A_{i+1}, \delta, T, l, \mu, \sigma)$
12:     else
13:         $\delta = d$
14:         $G = buildGrids(T, g, \delta, A, i + 1, l, \mu, \sigma)$
15:     end if
16: end for
```

the Grids. When δ is zero, to avoid the divided-by-zero error, δ is set to a small precision constant, for example 10^{-5}.

In Line 7 to 16, each time a subsequence A_{i+1} is inserted to the Grids, the algorithm takes steps to ensure the invariant that δ always equals the closest pair distance holds. Line 8 retrieves the neighbors of A_{i+1}, which are the subsequences falling in the same cells as A_{i+1} in each of the g grids of the Grids.

Line 9 computes the minimum distance between A_{i+1} and its neighbors. If the distance is less than δ, M is updated to the new pair with the minimum distance. If d is greater than δ, A_{i+1} is directly inserted to the Grids with δ as the offset value in Line 11. If d is less than δ, δ is updated to d, the Grids is rebuilt to contain the first $i + 1$ elements in A with δ as the offset value. When two subsequences have overlaps, their distance is set to infinity to ensure they cannot become the motif.

In this chapter we assume l is a user-set fixed constant and $l \ll n$. Theorem 3.3.1 states the correctness of Algorithm 2.

Theorem 3.3.1. *Algorithm 2 guarantees to find the closest subsequence pair in T when $g = l + 1$.*

Proof. To prove Theorem 3.3.1, we need to prove the following invariant holds during the insertion of $n - l + 1$ subsequences to the Grids: δ always equals the smallest subsequence pair distance after inserting $i + 1$ subsequences for any i such that $1 \leq i \leq n - l$. The base case $i = 1$ is trivial. Assume δ is the closest pair distance after inserting i subsequences. When inserting A_{i+1}, according to Lemma 3.3.1 which will be introduced later, all the subsequences in Grids that have a distance to A_{i+1} less than δ will be retrieved to *neighbors* in Line 8.

If A_{i+1} forms a new pair with one of the previous i subsequences with a distance less than δ, d will be the new closest distance in Line 9. Then δ is updated to d and the Grids is rebuilt with the new δ. If A_{i+1} does not form a new pair with one of the previously inserted subsequences with a distance less than δ, δ is unchanged and is still the closest pair distance so far. So in either case δ is the closest pair distance among the $i + 1$ subsequences.

From induction, δ is the closest pair distance among the $n - l + 1$ subsequences after all the subsequences are inserted to the Grids. $\square$

Lemma 3.3.1. *Assume A_j is in the Grids and A_{i+1} is to be inserted, if $Dist(A_j, A_{i+1}) < \delta$, then they fall in the same cell in at least one of the grids.*

Proof. For a single dimension, the two subsequences fall in different cells only when the two coordinates fall on different sides of the cell border. As $Dist(A_j, A_{i+1}) < \delta$ and the origin offset is δ, the line connects the coordinates of A_j and A_{i+1} for a single dimension cannot cross more than one cell border. So for two subsequences of l dimensions, they can locate on the different sides of at most l cell borders. As the Grids have $l + 1$ cell borders, the two subsequences must locate on the same side of at least one cell border, i.e. fall in at least one same cell according to the pigeonhole principle. $\square$

Figure 3.3b gives an illustrative example of Lemma 3.3.1. S_1 and S_2 have a distance less than δ and fall in the same cell of the grid originated at O_3 (with the same cell index

$(-1, 0))$.

Theorem 3.3.2 states that the expected time complexity of Algorithm 2 is linear.

Theorem 3.3.2. *Algorithm 2 runs in $O(n)$ expected time.*

Proof. It is easy to see that Line 1 to 6 take $O(n)$ time and the main concern is the loop from Line 7 to 16. Let R_i, M_i, I_i, and B_i be the time of *retrieveGrids*, *minDist*, *insertGrids*, and *buildGrids* respectively at the i-th stage of the loop. Let Q_i be the set of subsequences including A_1, A_2, …to A_i, and let $\delta(Q_i)$ denote the closest pair distance among the subsequences in Q_i, define a random indicating variable $X(A_{i+1}, Q_{i+1})$:

$$X(A_{i+1}, Q_{i+1}) = \begin{cases} 1, & \text{if } \delta(Q_{i+1}) < \delta(Q_i) \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

Namely $X(A_{i+1}, Q_{i+1})$ takes the value of 1 when inserting A_{i+1} causes δ to be updated to a new value, otherwise it takes the value of 0. Let L_i be the time of the loop at stage i and we have:

$$L_i = O(R_i + M_i + I_i + X(A_{i+1}, Q_{i+1})B_i) \tag{3.4}$$

Then:

$$E[L_i] = O(E[R_i] + E[M_i] + E[I_i] + E[X(A_{i+1}, Q_{i+1})B_i]) \tag{3.5}$$

As the Grids is implemented by hash tables, we have $E[R_i] = O(1)$ and $E[I_i] = O(1)$. As the Grids offset size is δ and grid size is $g\delta$, as well as the fact that all the subsequences in Grids have distances no less than δ, the maximum number of subsequences that a cell can contain is constant. So the subsequences in *neighbors* in Line 8 is constant and thus $E[M_i] = O(1)$.

The random variable B_i depends on the hash algorithm and has nothing to do with the random indicating variable $X(A_{i+1}, Q_{i+1})$, so we have:

$$E[X(A_{i+1}, Q_{i+1})B_i] = E[X(A_{i+1}, Q_{i+1})]E[B_i] \tag{3.6}$$

From Lemma 3.3.2, $E[X(A_{i+1}, Q_{i+1})] \leq 2/(i+1)$. B_i is the time to insert i subsequences to the Grids so $E[B_i] = i$. So we have:

$$E[L_i] = O(O(1) + O(1) + O(1) + 2/(i+1) \times O(i)) = O(1) \tag{3.7}$$

The loop is executed at most $O(n)$ times so $E[L] = O(n)$. $\square$

Lemma 3.3.2. $E[X(A_{i+1}, Q_{i+1})] \leq 2/(i+1)$

Proof.

$$E[X(A_{i+1}, Q_{i+1})] = Pr(\delta(Q_{i+1}) < \delta(Q_i)) \times 1 \tag{3.8}$$

where $Pr(\delta(Q_{i+1}) < \delta(Q_i))$ is the probability that inserting A_{i+1} causes δ to be updated to a new value.

When $\delta(Q_{i+1}) < \delta(Q_i)$, let C be the set of closest pairs formed by A_{i+1}.

If $|C| = 2$, A_{i+1} is one of the two subsequences forming the closest pair and as $A_1, A_2, \ldots, A_{i+1}$ is randomly permuted, we have:

$$Pr(\delta(Q_{i+1}) < \delta(Q_i)) = 2/(i+1) \tag{3.9}$$

If $|C| > 2$, A_{i+1} is the only subsequence forming multiple pairs that have distances less than $\delta(Q_i)$. If not so, there must be two subsequences in Q_i that have the distance $\delta(Q_{i+1})$, which contradicts the fact that $\delta(Q_i)$ is the closest pair distance in Q_i. So one has:

$$Pr(\delta(Q_{i+1}) < \delta(Q_i)) = 1/(i+1) \tag{3.10}$$

Combining the two cases, we have:

$$Pr(\delta(Q_{i+1}) < \delta(Q_i)) \leq 2/(i+1) \tag{3.11}$$

From (3.8) we have $E[X(A_{i+1}, Q_{i+1})] \leq 2/(i+1)$.

□

Algorithm 2 maintains the mean, standard deviation and start location for each subsequence, which takes $O(n)$ space complexity. Also $n - l + 1$ subsequences locations are inserted to the Grids which takes $O(n)$ as well so the overall space complexity is $O(n)$.

In Algorithm 2, when δ is small, the subsequences become sparse in the Grids. The number of subsequences in *neighbors* in Line 8 becomes small. Therefore the constant term in the time complexity is also small. So the algorithm is especially suitable for datasets that have similar motif subsequences.

From the proof of Lemma 3.3.1, we can see that Algorithm 2 fails to find the closest subsequence pair only if the two subsequences of the closest pair fall in different cells in all the g grids, when one subsequence is already in the Grids and the other one is to be inserted. Assume whether the two coordinates of the closest pair locate on different sides of the cell border is independent for each dimension, then the probability that the pair has coordinates locating on different sides of all the g cell borders decreases fast super-linearly as g increases.

So in practice we can fix the g value at a sufficient large value to ensure exactness instead of setting it to $l + 1$. In our experiments in Section 3.5, when g takes 14, we find that the algorithm never fails to find the exact motifs for the datasets used (motif length up to 1000). The parameter g provides a tradeoff choice to the users. When g is set to a small value, the probability that the result is suboptimal is high, but the algorithm is fast and the space used is small. When g is set to a large value, the algorithm is less fast and uses more space, but the probability that the result is suboptimal is low.

As the subsequences are considered in a random order, it is possible that the closest pair with a small distance is inserted to the Grids at the end which makes the algorithm less fast. To alleviate this and make the performance stable, we can cascade Algorithm 2 with different g settings. First run the algorithm with a small g value like 2 to decrease the δ value fast. In the next run g is set to a large value like 8 to ensure good result. The closest pair found in the first round is the first pair to be inserted to the Grids in the second round and then other subsequences are considered in a random order. We can use $g = (2, 8)$ to denote this cascading setting.

3.4 Algorithm for Motif Set Discovery

Algorithm 3 gives the pseudo-code of the algorithm for *motif set* discovery. Given a time series T, a motif length l, a grid size g, and a threshold value r, the algorithm outputs all subsequence pairs M such that the two subsequences in the pair have a distance less than r.

Algorithm 3

```
Input:
    $T$: time series
    $l$: motif length
    $g$: number of grids
    $r$: threshold
Output:
    $M$: motif set
 1: $n = T.length$
 2: $A = randomPermutation([1, \ldots, n-l+1])$
 3: $\mu, \sigma = statistics(T, n, l)$
 4: $\delta = r$
 5: $M = \{\}$
 6: $G = buildGrids(T, g, \delta, A, 1, l, \mu, \sigma)$
 7: for $i = 1$ to $n - l$ do
 8:     $neighbors = retrieveGrids(G, A_{i+1}, \delta, \mu, \sigma)$
 9:     if $|neighbors| > 0$ then
10:         $M = recordMotifs(neigbors, M, A_{i+1}, \delta, T, l, \mu, \sigma)$
11:     end if
12:     $G = insertGrids(G, A_{i+1}, \delta, T, l, \mu, \sigma)$
13: end for
```

The main difference between Algorithm 3 and Algorithm 2 is that in Algorithm 3, δ is fixed at r and is not updated during the insertion process. Line 4 assigns the threshold value r to δ and M is initialized as an empty set in Line 5. The first subsequence is inserted to the Grids with δ in Line 6. In Line 9 to 11, when the neighbor set of A_{i+1} is not empty, the distances between A_{i+1} and its neighbors are calculated. The pairs that have distances less than δ are added to M. A_{i+1} is inserted to the Grids with δ in Line 12.

Algorithm 3 finds all the pairs that have a distance less than the given threshold r if g is set to $l + 1$. When the number of pairs that have distances less than r is constant, Algorithm 3 runs in $O(n)$ expected time. The proof of correctness and time complexity of Algorithm 3 are similar to the proof of Algorithm 2 and thus is omitted here.

As analyzed in Section 3.3, when the threshold value r is set to a small value, δ is small and Algorithm 3 is fast. So the algorithm is especially suitable for finding similar motifs. This is meaningful as people usually care more about similar repeated patterns and less about the not so similar patterns.

The user can set the r value to a small value to check if there are similar patterns in the time series fast. If no such patterns exists, the return set is empty. Then the user can gradually increase the threshold to interact with the data. Setting r to a large value is not very efficient. Therefore if the dataset is not likely to contain similar patterns, the users can consider other methods such as STOMP [45], whose performance is independent of the data.

The correctness and time complexity of the proposed algorithms hold for Minkowski distance (Eq. (3.1)) of any $p \geq 1$. In our experiments we adopt the case $p = \infty$, i.e. Chebyshev distance (Eq. (3.2)) as its value is on a single dimension. So the distances of subsequences of different lengths can be compared and it is convenient to set a uniform distance threshold for the motifs of different lengths.

Also the Chebyshev distance value does not accumulate along with the subsequence length, this leads to a small δ value, which makes the algorithms fast as discussed in the previous sections.

In the returned motif set of Algorithm 3, the subsequences in different pairs may have overlaps. In this case we can post-process the results to pick the one with minimum distance from the pairs that have overlaps and remove others.

3.5 Experimental Evaluation

Experiments are conducted to verify the previous theoretical analysis and compare the proposed algorithm with the state-of-the-art. The C++ source code of the proposed algorithms is available in the supplementary material[1]. The experiments are conducted in a batch-process cluster, a single core of AMD Opteron Processor 6276 (2299 MHz) and 500 GB memory are used. The running time is recorded with the clock() function in C++. Algorithm 2 is denoted as LL (Li-Lin) and Algorithm 3 is called LL2.

To verify the correctness of LL, for each time series length n and motif length l combination, it is run on 1000 random walk time series (generated by accumulating standard Gaussian distribution random values, each time series use a different random seed). g is set to a cascading $(2, 8, 14)$. A motif is planted in each time series as the ground truth and the motif is generated by randomly picking a subsequence from the time series, copying it with added Gaussian noise, adding the copy to a randomly selected location that does not overlap with the original subsequence. The noise is set to follow the standard Gaussian distribution and is multiplied by a factor of 10^{-4} to make sure the planted motif is the closest pair in the time series.

The detailed results are provided in Table 3.1. From the results one observes that LL succeeds to discover the exact motifs in all the tested situations in all runs, this shows the effectiveness of the algorithm.

To verify the time complexity, LL is run on the random walk time series of different lengths which are used in the previous experiment and the average running time of 30 runs on each input size is recorded. g takes the same values as the correctness experiment. Figure 3.4a and Figure 3.4b show the running time results of LL with motif length 100 and 1000 respectively.

[1]`https://github.com/xiaoshengli/LL`

Table 3.1: Number of times LL succeeds to find the motifs out of 1000 random walk time series

Motif length l	Time series length $n = 1 \times 10^6$	Time series length $n = 1 \times 10^7$
100	1000/1000	1000/1000
1000	1000/1000	1000/1000

Table 3.2: Number of times LL2 succeeds to find all the 10 planted motifs out of 10000 random walk time series

Motif length l	Time series length $n = 1 \times 10^6$	Time series length $n = 1 \times 10^7$
100	10000/10000	10000/10000
1000	10000/10000	10000/10000

In the figures, the x-coordinate of a red dot is its input time series length n, the y-coordinate is the respective average running time. Linear regression curve fitting is performed on the running time results and the black lines in the figures are the fit lines.

From the figures one observes that the R^2 values of the linear fitting, which are the coefficients of determination, are 0.99623 and 0.99971 for the two figures respectively. These values are very close to 1, which indicates the average running time and the input time series length have a strong linear relationship. The results coincide with the previous theoretical analysis.

To verify the correctness of LL2, it is run on random walk time series of different lengths n with different motif lengths l. For each combination of n and l, LL2 is run on 10000 random walk time series with planted motifs. The motifs are generated in the same way as in the experiment for LL but here for each time series, 10 different motifs (i.e. 10 different pairs of patterns) are planted . The detailed experimental results are provided in Table 3.2.

From the results one can see LL2 succeeds to discover all the 10 planted motifs in all the tested combinations in all runs, which demonstrates the effectiveness of LL2.

Figure 3.4c and Figure 3.4d give the running time experiment results of LL2 on random

Table 3.3: Running time comparison with the state-of-the-art on the hg38 DNA data

Method	$n = 1 \times 10^6$	$n = 1 \times 10^7$	$n = 1 \times 10^8$	$n = 2.3 \times 10^8$
STOMP (estimated)	4.2 hours	17.8 days	6.7 years	38.5 years
LL2	221 seconds	23.9 minutes	6.3 hours	7.9 hours

walk time series that are generated in the same way as the previous experiment. The parameters of LL2 take the same values as in the correctness experiment. Each running time in the figures is the average time of 30 runs. In the figures, we can see the R^2 values of the linear curve fitting are 0.99742 and 0.99979 for the two cases respectively. These values are very close to 1, indicating a strong linear relationship between the time series length n and the average running time of LL2.

Figure 3.4e shows the situation if we change the Chebyshev distance in LL2 to Euclidean distance and the parameters are kept the same. From the figure we can see the previous conclusion of linear relationship still holds.

LL2 is run on different lengths of the hg38 DNA data introduced in the beginning of this chapter and for comparison the state-of-the-art STOMP [45] is also run on the same data. The motif length l is set to 5000 and Table 3.3 provides the experiment results. The threshold of LL2 is set to 10^{-15} and g is set to 6. The STOMP code used here is from its authors and is also in C++. The actual running time of STOMP is quite long so we adopt the estimated values. The estimation of STOMP's running time is performed with the method introduced in its original paper [45]. From the table one can see LL2 is orders of magnitude faster than STOMP.

3.6 Case Studies

3.6.1 Bird Sound in the Wild

In this case study the proposed method is used to analyze bird sound data. Xeno-canto [20] is a website where people around the world can upload and share the bird sound they record in the wild. We downloaded 1072 Antthrushes sound records from the website and used the Mel-Frequency Cepstral Coefficients (MFCCs) [21] to transform the audio files into time series. Each record is transformed to a 13-dimension time series and we only take the second dimension data. The time series are concatenated and the result is a long time series of 11 million length. LL2 is run on this time series with $l = 1000$, $g = 6$ and $r = 0.5$ and it takes 17.7 minutes to run. The parameters can be set by trial-and-error to capture the meaningful patterns. Figure 3.5a shows one of the motifs found by the algorithm.

According to the annotations in the webstite, the two motif subsequences are from two different records (XC117911 and XC117926) but are under the same species (Chamaeza Turdina). This may suggest the pattern found is a characteristic of this kind of bird and can be used as features for other data mining tasks like classification.

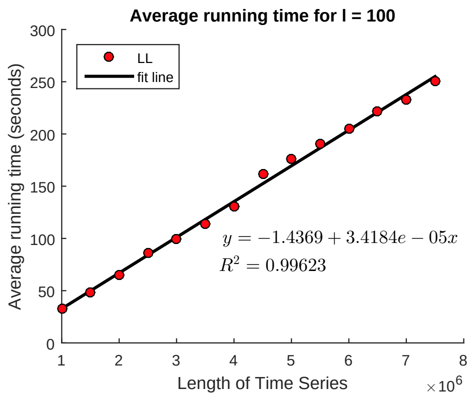
3.6.2 Electrical Load Measurements

The proposed algorithm is applied to analyze the electrical load measurements data [52] in this case study. This dataset contains the electrical consumption in Watts for 20 households of different appliances. Here the data of household 1 is used and each electrical device has a consumption time series of 7 million length. LL2 is run on these time series with $l = 2000$, $g = 6$ and $r = 0.1$ and the parameters can be selected by trial-and-error to find the meaningful patterns. Figure 3.5b shows an example pattern detected from the Fridge consumption data.

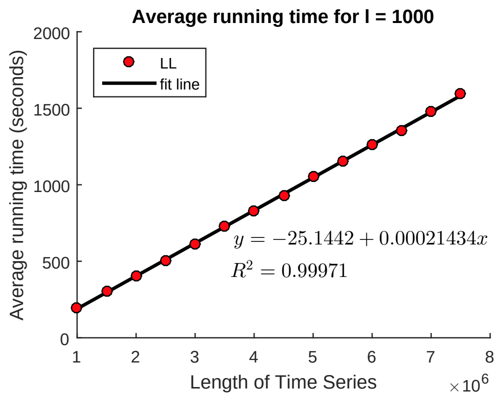
Domain experts can check the details of patterns to give diagnosis of the electrical appliances. Also tracking the patterns in the long time series of each device can reveal the behavior of their users. The above results show the effectiveness of the proposed method on detecting similar patterns in the real-world data.

3.7 Conclusion of LL

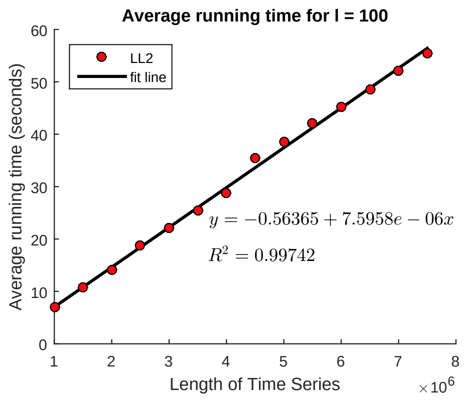
Discovering motifs in time series data provides valuable information for the domain experts and has many applications. The state-of-the-art exact motif discovery methods have quadratic time complexities over the length of the time series. This chapter proposes an algorithm that can find the exact motifs in a linear expected time complexity. The algorithm is further modified to find a motif set under a given threshold. In practice the algorithms are very fast if true motifs exist in the data or the threshold is set to a small value. The detailed proof of correctness and time complexity are presented. Experiment results of the proposed algorithms coincide with the theoretical analysis. The proposed algorithm is orders of magnitude faster than the state-of-the-art on the tested dataset and the case studies on the bird sound and electrical consumption data demonstrate its effectiveness in real-world situations.



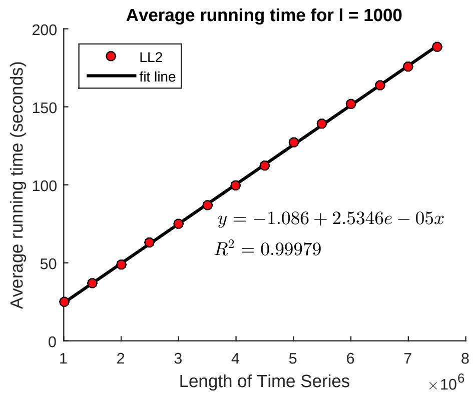
(a)



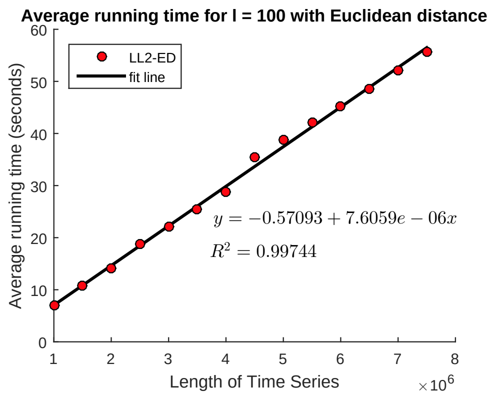
(b)



(c)

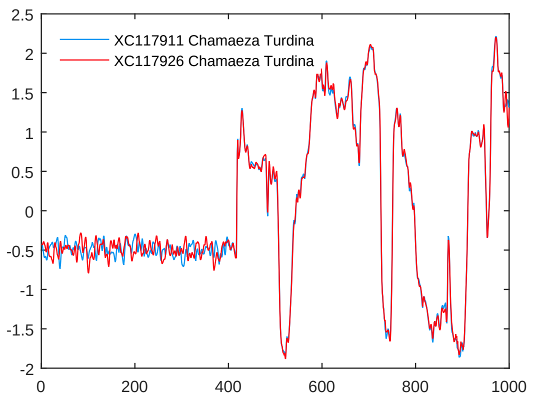


(d)

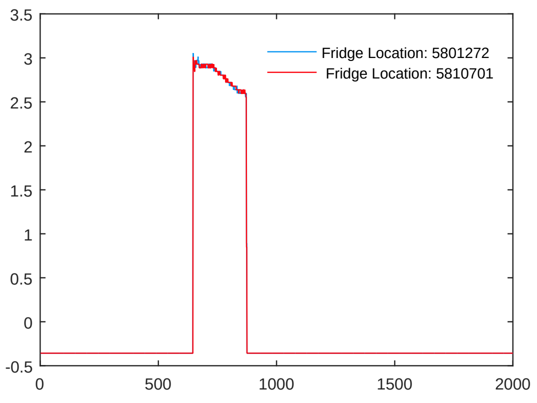


(e)

Figure 3.4: Running time of LL and LL2 on different lengths of input time series with different motif lengths



(a)



(b)

Figure 3.5: Example motifs found by the proposed method in the bird sound data and fridge power consumption data.

Chapter 4: Time Series Clustering with Symbolic Pattern Forest

Time series clustering problem can be formulated as, given a set of unlabeled time series instances, to place them into homogeneous and separated groups. In this chapter, we consider the partitional clustering problem of whole time series, i.e. we regard a time series instance as an object and cluster the time series objects into pairwise-disjoint groups.

With the advance of technologies, for example the sensors are becoming lighter, smaller and cheaper, hence widely embedded in various devices and machines, the amount of time series data becomes huge and the content is changing rapidly. This requires the data mining algorithms to have low time complexity. Although there have been a wealth of work on time series clustering, little work is on providing a linear time solution with reasonable performance. Existing super-linear time complexity methods may not be applicable when the dataset is large, or when real-time analytics are required.

In this chapter we propose a novel time series clustering algorithm, called Symbolic Pattern Forest (SPF), which has linear time complexity. The approach checks if some randomly selected symbolic patterns exist in the time series to partition the data instances. This partition process is executed multiple times, and the partitions are combined by ensemble to generate the final partition. We demonstrate that group structures in the data can emerge from the random partition process. Further analysis shows that the ensemble size needed to achieve good results does not directly depend on the input data size, and thus we can set the ensemble size to a proper fixed value for a specific data pattern.

The application of symbolic patterns in SPF has several benefits. Real-world time series often contain noise, amplitude change, phase-shift and irrelevant portion of data. The normalization step in symbolization can provide scale-invariance against the amplitude change. In the average and symbolization step, some noise can be smoothed out. The symbolic patterns do not preserve the pattern location information in the time series, thus they are not affected by the phase-shift. If a time series contains a symbolic pattern in some portion, changing the values in other portion does not affect the appearance of the pattern, making SPF robust against irrelevant data.

Further, the utilization of symbolic patterns makes the pattern space finite, and we can use the symbolic patterns to partition the data without using a distance measure. Checking the boolean indicating array to assign clusters in SPF is efficient as boolean operations are very fast. Boolean values are space-efficient which can take more advantage of the CPU cache to speed up the program.

We evaluate the SPF algorithm on all 85 datasets in the well-known UCR time series archive [17], and compare with other state-of-the-art approaches with statistical analysis. The results show that SPF is better in accuracy compared with other rival methods. The work presented in this chapter has been published in [53].

4.1 Definition and Related Work

This section provides the definitions and notations to precisely describe the problem under investigation and to present the proposed method.

Definition 4.1.1. *Given a set of time series $\{T_i\}_{i=1}^{n}$, where n is the number of time series instances, time series partitional clustering assigns a group relationship c_i for each T_i, with $c_i = r_j$, $j \in \{1, 2, \ldots, k\}$. r_j is a group value and k is the number of clusters. Usually we have $k \ll m$ and $k \ll n$. For presentation simplicity, we assume all the time series in the dataset have the same length m. The proposed algorithm can also work on datasets with varying-length time series.*

Some research on time series clustering is based on the k-means algorithm [54]. In the k-means algorithm, the clustering group relationship is generated by an iterative refinement procedure. In initialization, k centroids are randomly selected. In each iteration, the distances from the instances to the centroids are computed and the instances are assigned to their nearest centroids. Centroids are then updated according to the new assignment.

In the standard k-means algorithm, Euclidean Distance (ED) [55] is used as the distance metric and arithmetic mean is adopted to calculate the centroids. It is common for real-world time series data to contain phase-shift, warping, distortion and amplitude change. The simple ED may not be able to cope with these situations. Therefore, many time series distances measures are proposed [56], and one of the most popular ones is the Dynamic Time Warping (DTW) [18] which can align the data points from the two time series under comparison to find the optimal matching.

Methods of generating centroids under new time series distance measures have been proposed. Examples of these methods are NonLinear Alignment and Averaging Filters (NLAAF) [57], Prioritized Shape Averaging (PSA) [58] and Dynamic Time Warping Barycenter Averaging (DBA) [59].

K-shape [60] is one of the state-of-the-art time series algorithms based on k-means. It proposes a distance measure called Shape Based Distance (SBD), which is based on the cross-correlation of the time series. The centroids are generated by optimizing the within-cluster squared normalized cross-correlation between the centroids and the time series instances.

Another category of algorithms on time series clustering transform the time series into flat features and then apply classic clustering algorithms on the features to generate the cluster assignment. In [61], the authors transform a time series into a bitmap, which is composed of the counts of all the symbolic patterns in time series. The bitmap representation provides a new distance measure for the classic clustering algorithms to run on time series data.

In the work by Zakaria et al. [62], the authors propose to enumerate all the subsequences in the time series dataset to select a subset of subsequences called U-shapelets that can best separate the data. The distances between the time series and these subsequences are computed and regarded as new feature values. Finally k-means is applied on the new feature values to get the clustering result. Although the shapelet-based method and the proposed technique both use local shapes in time series for clustering, they are quite different. The shapelet-based method adopts an iterative procedure to refine the clusters, while the proposed algorithm directly partitions the data and combines the partitions to get the clusters.

In a recent work [63], instead of enumerating all the subsequences in the time series dataset, the shapelets are learned by optimizing an objective function.

There are some research work on time series clustering that are based on certain models. These methods assume the clusters are generated by some models and fit the models with the time series data. Examples of the models are Gaussian mixed models [64], Markov Chain [65] and Hidden Markov Models [66] and Self-Organization Map [67]. The model-based approaches assume the time series are generated by certain distributions which may not be true and have the drawback of being slow in processing the data [68].

4.2 Symbolic Pattern Forest

For clarity, we present a small concrete example to illustrate the idea of the proposed method. Then we provide the formal description and analysis of the algorithm.

4.2.1 A Concrete Illustrative Example

Figure 4.1 (Left) shows a small dataset of 4 time series instances belonging to 2 different classes (in blue and red respectively). These time series are taken from the FaceFour dataset from the UCR time series archive [17] and the time series in the dataset reflect the face outlines of different individuals under different conditions [69].

Given a set of SAX parameters, we can enumerate all possible symbolic patterns. The proposed method randomly picks a symbolic pattern from all available patterns, and checks if the pattern exists in the time series instances. More specifically, we use a sliding window of pre-defined length to go through the time series and extract all subsequences of the length. Each subsequence is transformed to a symbolic pattern and compared with the randomly

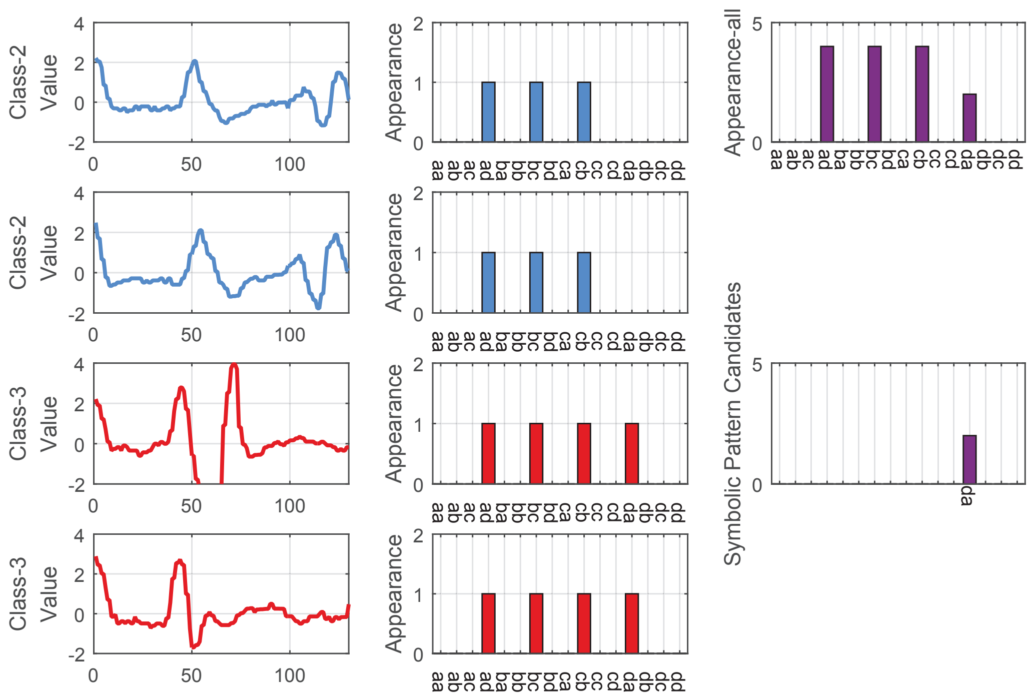


Figure 4.1: An illustrative example of the SPF clustering process. (Left) Time series from 2 classes. (Center) Boolean indicating arrays of the time series on the left. (Right) Total appearance count of each pattern and final symbolic pattern candidates.

chosen pattern.

This process is repeated multiple times so here we can optimize the process by scanning the time series in the beginning and storing the appearance of each pattern in a boolean indicating array. Boolean false (0) means the pattern does not appear in the time series, and boolean true (1) indicates the pattern appears. When checking a specific random pattern, we can directly look at the boolean array without needing to scan the time series again. Figure 4.1 (Center) shows the boolean indicating arrays for the four time series on the left respectively.

According to whether the time series contain a certain randomly selected pattern, the time series are partitioned into two groups. Those containing the pattern go to one group and the others are assigned to the other group. These two groups form two clusters. If the number of clusters k is more than 2, we perform the division with a new random symbolic pattern on the larger group. Previously chosen symbolic patterns are excluded from the pool of available pattern candidates. This process continues until we obtain k clusters, and the tree constructed from the procedure is called Symbolic Pattern Tree (SPT).

The above procedure are repeated multiple times, and multiple trees are constructed as a result, hence the name Symbolic Pattern Forest (SPF). Each tree is a partition of the data, and we combine the partitions in the forest by ensemble to get the final output partition.

The main idea of the symbolic pattern tree is that it uses a symbolic pattern to separate different time series groups. If a pattern appears in all the instances, it cannot separate the instances and thus we can exclude it from the symbolic pattern candidate pool. The same applies to the patterns that do not appear in any instances.

In SPF, the number of occurrences for each pattern in the dataset is counted, and the patterns with a count greater than an upper bound or less than a lower bound are excluded from the symbolic pattern candidate pool. The settings of these two bounds will be introduced later. Figure 4.1 (Right) shows the total occurrence count of each pattern in the dataset, as well as the final symbolic pattern candidate "da". SPF will select "da", which can separate the two classes into two clusters correctly.

4.2.2 SPF Algorithm

Cluster Ensemble

In SPF, each SPT generates a cluster assignment for all the time series instances, and these clusters are combined by ensemble to generate the final cluster assignment. Hybrid Bipartite Graph Formulation (HBGF) [70] is adopted in SPF to perform the cluster ensemble. HBGF has a linear time complexity. The idea of HBGF is to build a graph model where the instances and clusters of the ensemble are the vertices. Partitioning the graph generates the ensemble consensus clusters. In our implementation, we use Metis [71] to partition the graph.

Efficient SAX Computation

In SPF, we need to compute the SAX symbolic pattern of each subsequence in the time series. The cumulative sum technique in [22] is applied to calculate the symbolic pattern of an arbitrary subsequence in $O(1)$ time (given the cumulative sums). The cumulative sum series U and cumulative squared sum series V of a time series T can be computed as: $u_j = \sum_{i=1}^{j} t_i$, $v_j = \sum_{i=1}^{j} t_i^2$, where $j = 1, \ldots, m$ and u_0, v_0 are set to 0.

For a subsequence $x = [t_i, \ldots, t_{i+l-1}]$, the mean value μ_x and standard deviation σ_x can be calculated as: $\mu_x = (u_{i+l} - u_{i-1})/l$, $\sigma_x = \sqrt{(v_{i+l} - v_{i-1})/l - \mu_x^2}$. x is divided in ω segments and each of the segment is mapped to a symbol. The normalized mean value of a segment $y = [t_i, \ldots, t_j]$ is $\mu_y = ((u_j - u_{i-1})/(j - i + 1) - \mu_x)/\sigma_x$. Then μ_y is mapped to a fixed break points region and transformed to a respective symbol [14].

Parameters of SAX

In SAX the number of segments ω, alphabet size γ, and subsequence length l are user-set parameters. In SPF, γ is set to 4 as previous research [14] suggests this value is suitable for most datasets. A grid search on the combinations of ω and l is performed. Each combination will generate a cluster assignment and all these assignments are combined by ensemble to give the final algorithm output. ω takes the values from $wd = \{3, 4, 5, 6, 7\}$ and l takes the values from $wl = \{0.025, 0.05, 0.075, \ldots, 1\}m$, i.e. an arithmetic sequence from $0.025m$ to m with a common difference of $0.025m$, where m is the length of the time series. Duplicate values and values less than 10 are removed. In total at most 200 l and ω combinations are tested.

Algorithm 4 gives the pseudo-code of SPF. Given a time series dataset D, an ensemble size q and the number of clusters k, the algorithm returns a cluster assignment C as output.

In Line 1, the dataset D is loaded and stored in T. Line 2 computes the cumulative sum U and cumulative squared sum V from T, which will be used to calculate the symbolic patterns. Line 3-4 perform the grid search on ω and l discussed before. Line 5 calculates

Algorithm 4 Symbolic Pattern Forest (SPF)

```
Input:
    $D$: time series dataset
    $q$: ensemble size
    $k$: number of clusters
Output:
    $C$: cluster assignment of $D$
1:  $T = loadData(D)$
2:  $[U, V] = CumulativeSum(T)$
3:  for each $\omega \in wd$ do
4:      for each $l \in wl$ do
5:          $SP = SymbolicPattern(T, \omega, l, U, V)$
6:          $PC = PatternCount(SP)$
7:          $SPC = FindCandidates(PC)$
8:          for $i = 1$ to $q$ do
9:              $CA = SPT(SPC, SP, k)$
10:             $ES.add(CA)$
11:         end for
12:         $CA2 = Ensemble(ES)$
13:         $ES2.add(CA2)$
14:     end for
15: end for
16: $C = Ensemble(ES2)$
```

the symbolic pattern appearance indicating boolean array SP. Line 6 counts the number of occurrence of each symbolic pattern and stores the result in PC. Line 7 takes the pattern count PC to select the symbolic pattern candidate SPC that will be used in the SPT random selection process. The patterns that have a count greater than an upper bound or less than a lower bound are removed from the candidate pool. In our method, we set the lower bound to $0.25 \times n/k$ where n is the number of instances and k is the number of clusters. The upper bound is set to $n - 0.25 \times n/k$. These bounds are set so to remove non-distinguishing patterns according to our experiments.

In Line 8-11, SPT uses the symbolic pattern candidates to generate a cluster assignment CA, and CA is added to an ensemble set ES. This process is repeated q times. In Line 12, we combine the cluster assignments in ES by ensemble and get the consensus cluster assignment $CA2$. $CA2$ is added to the ensemble set $ES2$ in Line 13. Finally in Line 16, the cluster assignments in $ES2$ is combined by ensemble to receive the final cluster assignment C as the output of the algorithm.

4.2.3 Analysis of SPF

Time Complexity

Recall n denotes the number of time series instances, and m denotes the length of time series. k is considered as a constant much smaller than m and n. In Algorithm 4 the computation of cumulative sums takes $O(nm)$ time. The number of grid search combinations is at most 200. It takes $O(nm)$ time to obtain the symbolic pattern boolean indicating arrays and the symbolic pattern candidates. SPT takes $O(n)$ time and the ensemble process also takes $O(n)$ time. So the total time complexity of SPF is $O(nm)$, which is linear to the input data size.

Effectiveness of SPF

In this subsection we show that if there exists a group structure in the dataset related to one pattern or some patterns, then SPF will output such group relationship. The intuition is that the unrelated patterns distribute uniformly among the instances so their effects cancel each other out in the ensemble, and only the structure on the related patterns is maintained and revealed. More formally, We have the following theorem:

Theorem 4.2.1. *Consider two instances T_1 and T_2 in the same class, if they agree with each other on some related patterns, then SPF will put them in the same cluster.*

Proof. Assume β is the percentage of related patterns in the symbolic candidate patterns. In the random selection process, if a related pattern is selected, then $P(C(T_1) = C(T_2)) = 1$, where $P(\cdot)$ is the probability function and $C(\cdot)$ is the cluster assignment function. If an unrelated pattern is selected, $P(C(T_1) = C(T_2)) = P(C(T_1) \neq C(T_2)) = 1/2$. So overall $P(C(T_1) = C(T_2)) = \beta \times 1 + (1 - \beta) \times 1/2$ and $P(C(T_1) \neq C(T_2)) = (1 - \beta) \times 1/2$. We have:

$$P(C(T_1) = C(T_2)) > P(C(T_1) \neq C(T_2)) \tag{4.1}$$

Each tree is independent, according to the law of the large numbers, when we have sufficiently large ensemble size:

$$Count(C(T_1) = C(T_2)) > Count(C(T_1) \neq C(T_2)) \tag{4.2}$$

where $Count(C(T_1) = C(T_2))$ is the count of cases that T_1 and T_2 are in the same cluster. So in the ensemble result, T_1 and T_2 are assigned in the same cluster. □

In the above analysis, we assume the ensemble size is sufficiently large, the following theorem quantifies how large the size should be.

Theorem 4.2.2. *Assume the ensemble size is q, the lower bound of q to receive a good clustering result is $-2\ln\alpha/\beta^2$, where 1-α is the confidence level, β is the related pattern percentage in the symbolic pattern candidate pool.*

Proof. Let X denote the random variable where there are X cases with $C(T_1) = C(T_2)$. Then X follows the binomial distribution:

$$P(X = z) = \binom{q}{z} p^z (1-p)^{q-z} \tag{4.3}$$

where $p = P(C(T_1) = C(T_2))$. Equation (4.2) should hold with high probability, so our goal can be formulated as:

$$P(X \leq z) = \sum_{i=0}^{z} \binom{q}{i} p^i (1-p)^{q-i} \leq \alpha \tag{4.4}$$

where $z = q/2$, $1 - \alpha$ is the confidence level, e.g. 95%. Here considering Hoeffding's inequality [72]:

$$P(E[\bar{X}] - \bar{X} \geq t) \leq e^{-2qt^2} \tag{4.5}$$

where $t \geq 0$. Considering $E[\bar{X}] = p$, we have:

$$P(E[\bar{X}] - \bar{X} \geq t) = P(qE[\bar{X}] - q\bar{X} \geq qt) \tag{4.6}$$

$$= P(X \leq qp - qt) \leq e^{-2qt^2} \tag{4.7}$$

Let $z = qp - qt$, we have $t = (qp - z)/q$:

$$P(X \leq z) \leq e^{-2(qp-z)^2/q} \leq \alpha \tag{4.8}$$

Recall $z = q/2$, $p = \beta \times 1 + (1 - \beta) \times 1/2$, we can solve the above inequality and get:

$$q \geq \frac{-2 \ln \alpha}{\beta^2} \tag{4.9}$$

$\square$

Here is a concrete example of this boundary value: let the confidence level be 99% and assume 50% of the patterns in the pattern candidate pool are related patterns. So $\alpha = 0.01$ and $\beta = 0.5$, we get $q \geq 36.84$.

One observation from equation (4.9) is that the ensemble size lower bound does not directly depend on the number of instances n and time series length m. In the experiment part we will see the accuracy results do not change with fixed ensemble size and varying instances numbers and time series lengths, given the same data input type .

4.3 Experimental Evaluation

4.3.1 Experimental Setup

To evaluate the proposed algorithm, we run it on all 85 datasets from the UCR time series archive [17]. This public archive contains different types of labeled time series from various fields. Each dataset in the archive contains a training set and a testing set. We fuse both sets and use all the data in the experiment. The results of SPF are compared with other rival methods. The widely used k-means algorithm [54] is selected as the baseline. Standard k-means adopts ED as the distance metric and uses arithmetic mean to calculate centroids. K-shape [60] introduced before is selected for comparison as in [60] the authors show k-shape is superior to other state-of-the-art time series clustering algorithms (including those based on DTW).

The source code of k-shape and k-means are obtained from the author of [60] and the code is in Matlab. The number of iterations of k-shape and k-means are set to 100 which is the same as in [60]. The ensemble size of SPF is set to 100 in all the experiments. k is set to equal the number of classes of the dataset in use. Following [60], we use the Rand Index to measure the accuracy of the clustering results. Rand Index is defined as: $Rand\ Index = (TP+TN)/(TP+TN+FP+FN)$, where TP is the number of instances belonging to the same class and assigned in the same cluster, TN is the number of instances belonging to different classes and assigned in different clusters, FP is the number of instances belonging to different classes but assigned in the same cluster, and FN is the number of instances belonging to the same class but assigned in different clusters.

To verify the time complexity of SPF, we run it on the widely used CBF dataset [73] of different sizes and record the average running time for each size. This dataset is a synthetic dataset, so with its underlying data generation rules in [73], we can conveniently generate the datasets with different number of instances and time series lengths.

The C++ source code of SPF is available in the supplementary material[1]. The experiments are conducted in a batch-processing cluster. A single core of AMD Opteron Processor 6276 (2299 MHz) and 16 GB memory are used.

4.3.2 Experimental Results

k-means, k-shape and SPF are run on the 85 datasets 10 times; the average running time and average Rand Index on each dataset are recorded. Table 4.1, 4.2 and 4.3 show the

[1] `https://github.com/xiaoshengli/SPF`

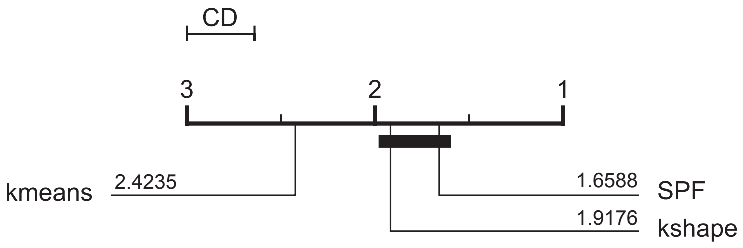


Figure 4.2: Critical difference diagram of the comparison on Rand Index.

clustering Rand Index of the three methods on the UCR archive. Here we present the summarization of the comparison. Figure 4.2 shows the critical difference diagram [39] (at 95% confidence level) for the Rand Index comparison. The value beside each method in the figure is the rank mean (lower is better) for the respective method. The methods that are connected by a bold bar have no significant difference.

From the figure one can see the accuracy of SPF is significantly better than that of k-means. SPF is also better than k-shape in rank mean, but the difference is not significant. So the overall accuracy of SPF is quite competitive. Performing the Wilcoxon signed rank test on the Rand Index result, the p-values between SPF and k-means, k-shape are 2.15×10^{-4} and 5.69×10^{-2} respectively. The conclusion from the Wilcoxon signed rank test coincides with that from the critical diagram.

The time complexities of k-means, k-shape and SPF are $O(nm)$, $O(\max(nm^2, m^3))$, $O(nm)$ respectively. Compared with k-means, SPF has the same time complexity but is significantly more accurate. Compared with k-shape, SPF has lower time complexity and slightly better accuracy. The total actual running time of k-means, k-shape and SPF on the 85 datasets are 1278, 19322, 1241 seconds respectively. Note that these methods are implemented in different languages so these time values just show how fast we can cluster the data using the available source code.

Figure 4.3 shows the average running time of SPF on the CBF datasets. The number of instances is changed from 1000 to 10000 and the length of time series is fixed at 1000. In the figure, the x-axis value is the number of instances and the y-axis value is the average

Table 4.1: Clustering Rand Index of SPF and other methods on UCR Archive benchmarks

Dataset	kmeans	kshape	SPF
50words	0.95009	0.95122	0.954649
Adiac	0.94146	0.94842	0.960146
ArrowHead	0.60894	0.62328	0.643958
Beef	0.65938	0.66429	0.717797
BeetleFly	0.51923	0.52282	0.494103
BirdChicken	0.52295	0.55423	0.598462
Car	0.65835	0.65354	0.817773
CBF	0.70209	0.87521	0.984026
ChlorineConcentration	0.52696	0.52696	0.53569
CinC_ECG_torso	0.67822	0.61987	0.659128
Coffee	0.77578	0.79136	0.834416
Computers	0.49985	0.52851	0.539517
Cricket_X	0.85395	0.87355	0.864838
Cricket_Y	0.85406	0.87373	0.874013
Cricket_Z	0.85743	0.87187	0.866868
DiatomSizeReduction	0.92262	0.953	0.916244
DistalPhalanxOutlineAgeGroup	0.72001	0.7033	0.627936
DistalPhalanxOutlineCorrect	0.49944	0.49945	0.500212
DistalPhalanxTW	0.84023	0.84313	0.740148
Earthquakes	0.49971	0.51915	0.499589
ECG200	0.61412	0.61232	0.603015
ECG5000	0.75818	0.78094	0.642027
ECGFiveDays	0.50003	0.80945	0.582481
ElectricDevices	0.71946	0.7164	0.77173
FaceAll	0.87555	0.91292	0.934548
FaceFour	0.73325	0.77431	1
FacesUCR	0.88029	0.90997	0.940633
FISH	0.77797	0.78389	0.862852
FordA	0.49996	0.56705	0.500163
FordB	0.50018	0.51482	0.500872
Gun_Point	0.49749	0.49749	0.497487

Table 4.2: (Continue) Clustering Rand Index of SPF and other methods on UCR Archive benchmarks

Dataset	kmeans	kshape	SPF
Ham	0.52808	0.52601	0.506871
HandOutlines	0.67337	0.68065	0.585073
Haptics	0.67249	0.68536	0.714803
Herring	0.50277	0.50242	0.504232
InlineSkate	0.72091	0.72711	0.757905
InsectWingbeatSound	0.88696	0.81027	0.879232
ItalyPowerDemand	0.49997	0.55844	0.637753
LargeKitchenAppliances	0.55503	0.59058	0.57165
Lighting2	0.51336	0.52959	0.537218
Lighting7	0.80036	0.80656	0.81959
MALLAT	0.93436	0.91658	0.963025
Meat	0.7819	0.82353	0.834832
MedicalImages	0.66443	0.67938	0.676133
MiddlePhalanxOutlineAgeGroup	0.73311	0.72959	0.705882
MiddlePhalanxOutlineCorrect	0.49983	0.49977	0.500369
MiddlePhalanxTW	0.80248	0.81394	0.77377
MoteStrain	0.70449	0.80419	0.699834
NonInvasiveFatalECG_Thorax1	0.94945	0.95443	0.967836
NonInvasiveFatalECG_Thorax2	0.96385	0.96697	0.971028
OliveOil	0.82915	0.85203	0.869153
OSULeaf	0.74686	0.78646	0.799241
PhalangesOutlinesCorrect	0.50528	0.50535	0.505203
Phoneme	0.91376	0.9289	0.930075
Plane	0.88267	0.90802	0.993962
ProximalPhalanxOutlineAgeGroup	0.77883	0.77072	0.679702
ProximalPhalanxOutlineCorrect	0.53367	0.5342	0.528511
ProximalPhalanxTW	0.80498	0.8036	0.740752
RefrigerationDevices	0.55601	0.55658	0.557137
ScreenType	0.56193	0.5592	0.564222
ShapeletSim	0.49937	0.70932	0.518955
ShapesAll	0.95512	0.97829	0.981383

Table 4.3: (Continue) Clustering Rand Index of SPF and other methods on UCR Archive benchmarks

Dataset	kmeans	kshape	SPF
SmallKitchenAppliances	0.55468	0.45509	0.634841
SonyAIBORobotSurface	0.7545	0.70861	0.708936
SonyAIBORobotSurfaceII	0.66152	0.59137	0.701271
StarLightCurves	0.76968	0.77214	0.71625
Strawberry	0.50417	0.50417	0.529104
SwedishLeaf	0.8785	0.90159	0.921184
Symbols	0.84391	0.8857	0.970886
synthetic_control	0.86796	0.89033	0.981982
ToeSegmentation1	0.49852	0.50365	0.606328
ToeSegmentation2	0.49844	0.59831	0.584549
Trace	0.74948	0.72469	0.977302
TwoLeadECG	0.5021	0.54089	0.688457
Two_Patterns	0.63024	0.67439	0.690577
UWaveGestureLibraryAll	0.89914	0.90029	0.847431
uWaveGestureLibrary_X	0.85611	0.85663	0.859987
uWaveGestureLibrary_Y	0.84573	0.83129	0.832854
uWaveGestureLibrary_Z	0.84447	0.84512	0.843882
wafer	0.53533	0.53966	0.508936
Wine	0.49648	0.49648	0.500541
WordsSynonyms	0.89533	0.89437	0.901371
Worms	0.64792	0.64882	0.677528
WormsTwoClass	0.49939	0.50189	0.517872
yoga	0.5	0.4999	0.499982

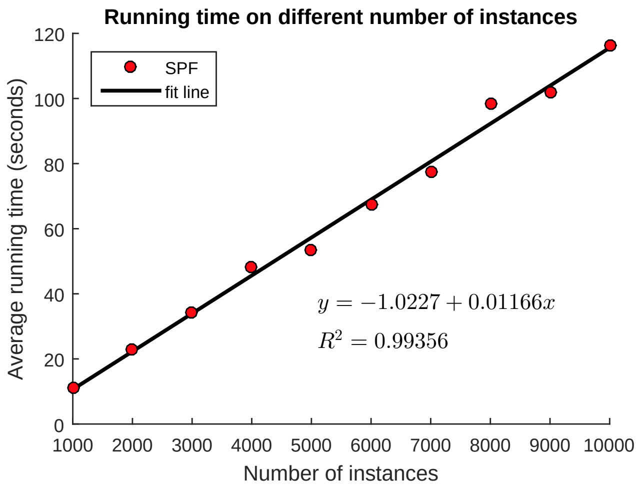


Figure 4.3: Running time of SPF on different number of instances.

running time of 30 runs. Linear regression curve fitting is performed on the data and the black line in the figure is the fit line. From the figure one can see the R^2 value, which is the coefficient of determination of the fitting, is 0.99356. This value is very close to 1, indicating the average running time of SPF and the number of instances have a strong linear relationship.

Figure 4.4 gives the average running time of SPF on the CBF datasets of different lengths. The number of instances is fixed at 1000 and the length is changing from 1000 to 10000. Each time value in the figure is the average time of 30 runs. The coefficient of determination R^2 value is 0.99923. This value is very close to 1, indicating the average running time of SPF and the length of time series have a strong linear relationship.

The above results coincide with the time complexity analysis in Section 4.2.3. Also in the experiment we record the average Rand Index of SPF on different input combinations. The Rand Index values remain the same at 1 under all the cases. The ensemble size is fixed in the experiment, and this result is in accord with the analysis in Section 4.2.3, that given a certain data type, the ensemble size to receive good accuracy results does not directly depend on the number of instances or time series lengths.

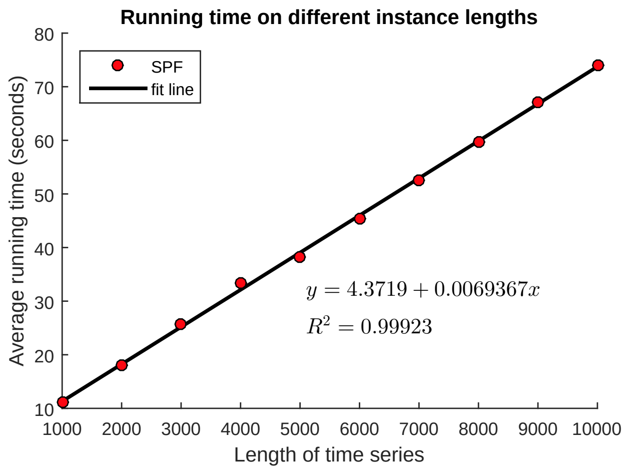


Figure 4.4: Running time of SPF on different time series lengths.

4.4 Conclusion of SPF

This chapter presents a Symbolic Pattern Forest (SPF) algorithm for time series clustering. The method partitions the time series instances by checking some randomly selected symbolic patterns and the partitions of multiple runs are combined to give a final cluster assignment. Analysis is conducted on the time complexity and effectiveness of the algorithm. We evaluate the algorithm extensively on all 85 datasets from the UCR time series archive and the results show that SPF is very competitive compared with other rival methods.

Chapter 5: Evolving Separating References for Time Series Classification

The idea behind the state-of-the-art time series classification techniques is to identify and extract patterns or characteristics from the labeled time series and use these patterns to classify new unlabeled data. For example, the shapelet-based techniques recently attract a lot of interests [27–31,34]. A shapelet is a subsequence in a time series that can best split the data into two subsets [27]. Shapelet-based approaches check the candidate subsequences in the training data and extract short patterns that are the most discriminative for separating different classes. These short patterns are then embedded in a decision tree as split test criteria [27, 28, 34] or are employed to transform the time series data to a new space with their distances to the patterns as new features [29–31].

Figure 5.1 (top and middle) shows the training time series instances in the ECGFiveDays dataset from the UCR time series classification repository [17]. The data originates from PhysioNet.Org [74] and records the ECGs (Electrocardiogram) of a patient in two different days. There are 23 training instances and 861 testing instances falling into two classes. For simplicity we denote one class as positive class and the other one as negative class.

Figure 5.1 (bottom) presents two separating references for this dataset. These two sequences have very different patterns and characteristics from the time series in the dataset. At first glance one may regard them as unrelated series, and if calculating the distance (to be defined later) between these two sequences and the time series data, one will obtain large distance values, indicating high degree of dissimilarity. However, if we use the distances to the two references as features and map the time series instances into a new space, meaningful results appear as shown in Figure 5.2.

In the new space, one can see that the training instances from the two classes are

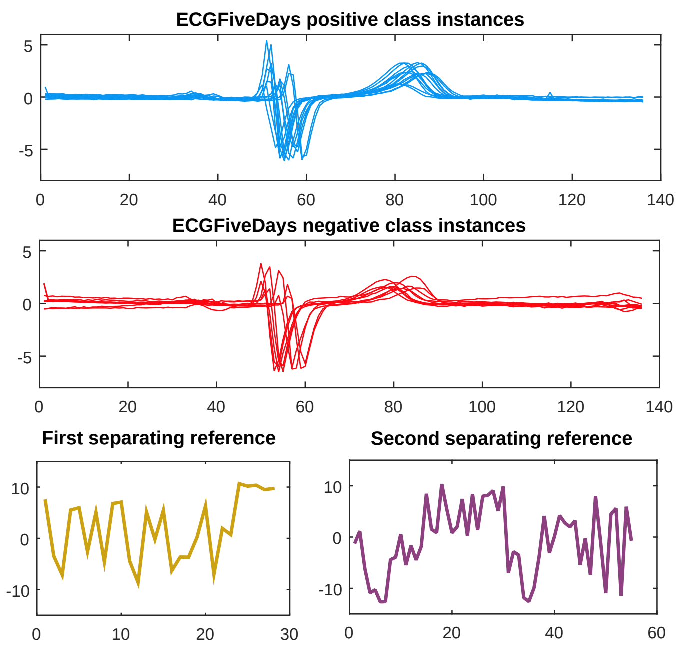


Figure 5.1: (top) Instances from the positive class of the ECGFiveDays dataset. (middle) Instances from the negative class of the ECGFiveDays dataset. (bottom) Two separating references for the ECGFiveDays dataset.

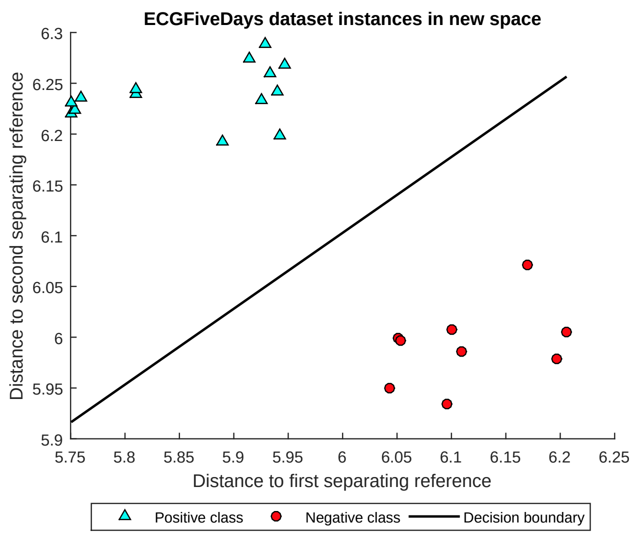


Figure 5.2: Mapping the time series in ECGFiveDays dataset to a new space using the two separating references in Figure 5.1

perfectly separated, with a large margin (distance between instances from different classes). Applying these two separating references and the linear decision boundary in Figure 5.2 can achieve a 100% classification accuracy on the testing dataset.

An evolution process is proposed in this chapter to obtain the separating references and the respective classification decision boundary for a given dataset to separate instances from different classes with large margins. The large margins between different classes provided by the proposed method facilitate good generalization performance from the training data to the testing data and help to prevent over-fitting. It is well-known that when the training data size is small, a classifier tends to over-fit the training data and thus performs poorly on the testing data. So the proposed approach is especially suitable for the situations where not much labeled data is available. Even though data is becoming cheaper and ubiquitous, these situations are not uncommon as labeled data is expensive to obtain [15].

The Separating References (SRs) differ from the previous concepts in time series classification such as shapelets [27–31, 34] or representative patterns [75] in that SRs are not actual patterns from the data (or similar to the actual patterns). They are generated from an evolution process to map the time series to a new space where the instances from different classes are separated with large margins. The work presented in this chapter has been published in [76].

5.1 Definitions and Related Work

The following are some definitions and notations that will be used in this chapter:

Definition 5.1.1. *A time series dataset D is a set of time series T_i with their respective class labels y_i, where $i = 1, \ldots, n$, n is the number of time series instances in the dataset.*

Definition 5.1.2. *Assume S and Q are time series of the same length l, then the (non-normalized) distance $D(S,Q)$ between them is given by Equation (5.1):*

$$D(S,Q) = \sqrt{\frac{1}{l}\sum_{i=1}^{l}(s_i - q_i)^2} \tag{5.1}$$

Definition 5.1.3. *Assume S and Q are series of the same length l, before calculating the distances, the two series are z-normalized to two new series $\hat{S}$ and $\hat{Q}$ with 0 means and 1 standard deviation. Then the normalized distance $D2(S,Q)$ is given by Equation (5.2):*

$$D2(S,Q) = \sqrt{\frac{1}{l}\sum_{i=1}^{l}(\hat{s}_i - \hat{q}_i)^2} \tag{5.2}$$

The same distance can be calculated using Equation (5.3) [28]:

$$D2(S,Q) = \sqrt{2(1 - \frac{\sum_{i=1}^{l} s_i q_i - l\mu_S\mu_Q}{l\sigma_S\sigma_Q})} \tag{5.3}$$

where $\mu_S = \frac{1}{l}\sum_{i=1}^{l} s_i$, $\sigma_S^2 = \frac{1}{l}\sum_{i=1}^{l} s_i^2 - \mu_S^2$, $\mu_Q = \frac{1}{l}\sum_{i=1}^{l} q_i$ and $\sigma_Q^2 = \frac{1}{l}\sum_{i=1}^{l} q_i^2 - \mu_Q^2$. The caching speedup techniques in [28] is adopted in our experiments to accelerate the distance computation.

Definition 5.1.4. *The distance $\hat{D}(S,T)$ between a short sequence S of length l and a long sequence T of length m ($l < m$) is the minimum distance between S and any subsequences of T of the same length l:*

$$\hat{D}(S,T) = \min_j D(S,Q_j) \tag{5.4}$$

where $Q_j = [t_j, t_{j+1}, \dots, t_{j+l-1}]$ and $1 \le j \le m - l + 1$.

Definition 5.1.5. *The **Separating References** SR for a given dataset is a set of sequences of values that can be used to map time series to a new space where the instances from different classes are separated with large margins.*

A shapelet is a subsequence that can best split the data into two subsets [27] such that the information gain is maximized. In [27], candidate subsequences of different lengths are enumerated to find the one that can best split the training set into two subsets based on their distances to the subsequence. The best subsequence is regarded as the shapelet and embedded in a decision tree classifier as the split test to classify unseen data.

As the total number of candidate subsequences can be very large, the shapelet-discovery process is time-consuming. Fast Shapelet (FS) [34] method addresses this problem by transforming the original real-value time series into a lower dimensional symbolic representation using Symbolic Aggregate approXimation (SAX) [14]. A random projection and a hashing process are used to identify the promising candidates. The quality of these candidates are then verified in the original space.

Instead of enumerating candidate subsequences, Learning Shapelet (LS) method [30] learns the shapelet via optimizing a classification objective function. The method initializes the shapelets as the cluster centroids of the subsequences in the dataset and applies a greedy gradient descent method to update the shapelets. The shapelets learned by LS, albeit not actual subsequences from the training set, resemble the shapes of their respective best matching subsequences. Thus the learned shapelets are quite different from the separating references.

In [75], the authors proposes the concept of representative patterns – a set of class-specific subsequences that best distinguish one class of time series from another. SAX is used to transform the time series into sequences of symbols and a grammar induction (GI) procedure is adopted to discover frequent subsequences of variable lengths as pattern candidates. The candidates are further refined to ensure the greatest discriminative power on different classes.

Instead of identifying some short patterns from the data for classification, another type of techniques apply a sliding window of certain length to extract all subsequences from the time series data, and transform them into symbolic patterns. A histogram to count the frequency of each symbolic pattern is constructed for a time series instance [12], or a time series class [35]. These histograms are used as new features for classification. Bag-of-Patterns (BOP) [12], SAX-Vector Space Model (SAX-VSM) [35] and Bag-of-SFA-Symbols (BOSS) [32] fall into this group.

5.2 Overview

In this section, we propose the Evolving Separating References (ESR) method which, given a time series dataset, randomly samples subsequences of different lengths in the dataset and uses them to form an initial population of individuals. Then several evolutionary computation operations are carried out on the population to gradually evolve it generation after generation with a specially designed evaluation function and some constraints. After a certain generation of evolution, the best individual in the population will provide the separating references and the respective decision boundary. The framework of the evolution process is based on the Minimum Penalty (MP) algorithm [77], an evolutionary algorithm that solves constrained optimization problems. Each step of the process will be explained in the following subsections.

Figure 5.3 presents the evolution process of the proposed approach on the ECGFiveDays dataset with two separating references of length 28 and 55 ($K = 1, L = (0.2, 0.4)$, with non-normalized distance. We will introduce the parameters in a later subsection). Each subfigure shows the time series instances and the decision boundary in the new space provided by the best individual in the population at a specific generation.

At generation 1, which is just after initialization, one observes that the distances of all the instances to the separating references are small, as the current separating references are actual subsequences in the time series. The instances from the two classes are mixed and cannot be linearly separated. As the evolution process goes on, instances within the

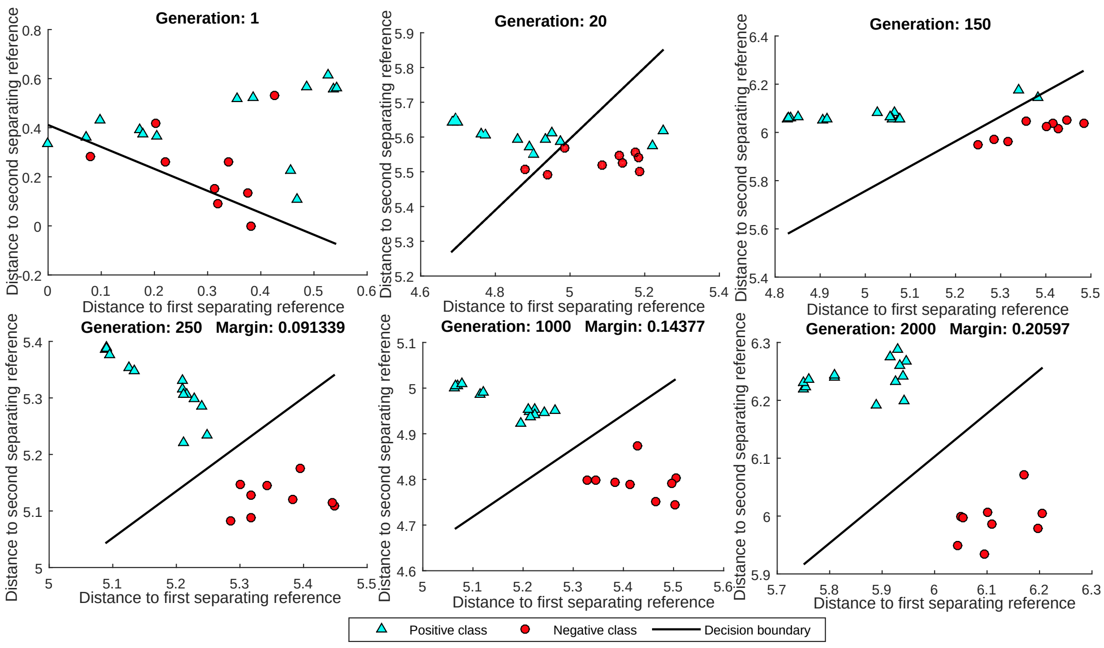


Figure 5.3: Evolution process of the proposed method on ECGFiveDays dataset with two separating references of length 28 and 55.

same class gradually gather together. At generation 250 shown in the left bottom sub-figure of Figure 5.3, all the instances are separated by the decision boundary with a margin of 0.091339. As the evolution further proceeds, the margin between the instances from the two different classes gradually enlarges. At generation 1000, the margin is 0.14377 and becomes 0.20597 at generation 2000. The two separating references in Figure 5.1 are taken from generation 2000.

5.3 Initialization

The population is initialized to have NP individuals (vectors). Each individual contains a number of Separating References (SRs) of variable lengths. The lengths are determined by a length pattern vector $L = (l_1, \ldots, l_i, \ldots, l_r)$ where l_i is a number between zero and one, denoting a percentage of the length of the time series. The absolute length of an SR is $l_i \times m$ (ceiled to integer) where m is the length of the whole time series. For each length, we generate K SRs, so the total number of SRs is $K \times Q$ where $Q = r$ (i.e. $|L|$). For example, if $L = (0.2, 0.4), K = 2, m = 100$, we will have 2 SRs of length 20 and 2 SRs of length 40 in an individual, for a total of length 120.

Let $P = (p_1, \ldots, p_{KQ}) = (l_{1,1}, \ldots, l_{1,K}, \ldots, l_{Q,1}, \ldots, l_{Q,K})$, which is the total length pattern in an individual. For the above example, $P = (0.2, 0.2, 0.4, 0.4)$. The individual also includes the Decision Boundary (DB). DB has the weights and bias of the linear boundary with respective to the SRs in the same individual. With the same example, an individual is a vector of length 125, where the lengths of the SRs are $(20, 20, 40, 40)$ and DB is length of 5 (4 weights for each SR and 1 bias).

More formally, we have:

$$\vec{x}_{i,G} = (x_{i,1,G}, \ldots, x_{i,d,G}), \quad i = 1, \ldots, NP \tag{5.5}$$

$$\vec{x}_{i,G} = (SR_{i,G}, DB_{i,G}) \tag{5.6}$$

$$SR_{i,G} = (SR_{i,1,G}, \ldots, SR_{i,KQ,G}) \tag{5.7}$$

$$SR_{i,j,G} = (x_{i,q+1,G}, \ldots, x_{i,q+s,G}) \tag{5.8}$$

$$DB_{i,G} = (\vec{w}_{i,G}, b_{i,G}) \tag{5.9}$$

$$\vec{w}_{i,G} = (x_{i,v+1,G}, \ldots, x_{i,v+KQ,G}) \tag{5.10}$$

where d is the dimension of the individual vector $\vec{x}_{i,G}$ and G denotes the generation number starting from 1. $q = m\sum_{e=1}^{j-1} p_e$, $s = mp_j$, $v = m\sum_{e=1}^{KQ} p_e$ and $b_{i,G} = x_{i,d,G}$.

For initialization, each SR is assigned a randomly selected subsequence of the same length from the training dataset. Each element in DB is initialized by a standard normally distributed random number. NP is fixed at 40 in all the experiments presented in this chapter.

5.4 Evolutionary Operations

Three operations (i.e., mutation, crossover, and selection) are conducted in every generation, each of them is explained as follows.

5.4.1 Mutation Operation

The mutation strategy in [78] is employed here. Concretely, for each individual (target vector) $\vec{x}_{i,G}$ in the population, three mutant vectors $\vec{v}$ are attained using the "rand/1" (5.11), "current-to-rand/1" (5.12) and "rand/2" (5.13) strategies respectively [78]:

$$\vec{v}_{i,G,1} = \vec{x}_{r1,G} + F \times (\vec{x}_{r2,G} - \vec{x}_{r3,G}) \tag{5.11}$$

$$\vec{v}_{i,G,2} = \vec{x}_{i,G} + rand \times (\vec{x}_{r1,G} - \vec{x}_{i,G}) + F \times (\vec{x}_{r2,G} - \vec{x}_{r3,G}) \tag{5.12}$$

$$\vec{v}_{i,G,3} = \vec{x}_{r1,G} + rand \times (\vec{x}_{r2,G} - \vec{x}_{r3,G}) + F \times (\vec{x}_{r4,G} - \vec{x}_{r5,G}) \tag{5.13}$$

where indexes $r1$, $r2$, $r3$, $r4$ and $r5$ are distinct random numbers chosen from $\{1, \ldots, NP\} \setminus \{i\}$, F is the scale factor to amplify the difference and $rand$ is a uniformly distributed random number between 0 and 1.

To prevent the absolute values in SRs becoming too large, lower bounds $LB = (lb_1, \ldots, lb_d)$ and upper bounds $UB = (ub_1, \ldots, ub_d)$ are set for the components in the individuals with respective to SRs. The upper bound values equal to 2 times of the maximum value in the training set and the lower bound values equal to 2 times of the minimum value in the training set (as the data is z-normalized, the minimum value is always negative). There is no boundary constraints for DBs.

More formally, one has:

$$lb_k = \begin{cases} 2\min_{i,j} t_{i,j}, & \text{if } \ 1 \le k \le v \\ -\infty, & \text{if } \ v+1 \le k \le d \end{cases} \tag{5.14}$$

$$ub_k = \begin{cases} 2\max_{i,j} t_{i,j}, & \text{if } 1 \le k \le v \\ +\infty, & \text{if } v+1 \le k \le d \end{cases} \tag{5.15}$$

where $v = m\sum_{e=1}^{KQ} p_e$, $1 \le i \le n$ and $1 \le j \le m$. Some components of the newly produced mutant vector may violate the boundary constraints and are treated as follows:

$$v_{i,k,G} = \begin{cases} \min(2lb_k - v_{i,k,G}, ub_k), & \text{if } v_{i,k,G} < lb_k \\ \max(2ub_k - v_{i,k,G}, lb_k), & \text{if } v_{i,k,G} > ub_k \end{cases} \tag{5.16}$$

where $1 \le i \le NP$ and $1 \le k \le d$.

5.4.2 Crossover Operation

After mutation, crossover is performed on each mutant to generate a trial vector $\vec{u}_{i,G}$:

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, & \text{if } rand_j(0,1) < CR \text{ or } j = j_{rand} \\ x_{i,j,G}, & \text{otherwise} \end{cases} \tag{5.17}$$

where $1 \le i \le NP$ and $1 \le j \le d$, $rand_j(0,1)$ is a uniformly distributed random number between 0 and 1 generated anew for each j. j_{rand} denotes an integer randomly selected from $\{1, \ldots, d\}$ and CR is the crossover rate to control the information to inherit from the mutant vector.

The crossover operation is not applied to the mutant vector produced by the "current-to-rand/1" strategy, so $\vec{u}_{i,G,2} = \vec{v}_{i,G,2}$. The scale factor F and crossover rate CR for each individual in each generation are randomly selected from three combinations: $(F = 1.0, CR = 0.1)$, $(F = 1.0, CR = 0.9)$ and $(F = 0.8, CR = 0.2)$.

5.4.3 Selection Operation

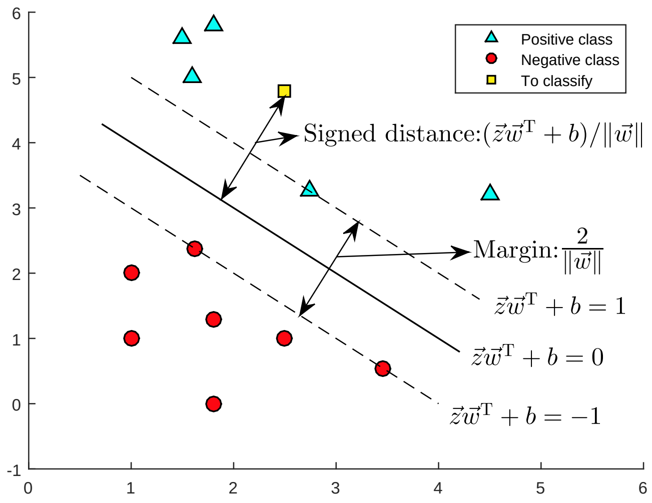
For each individual (target vector $\vec{x}_{i,G}$), after the mutation and crossover operations, there will be three trial vectors ($\vec{u}_{i,G,1}$, $\vec{u}_{i,G,2}$ and $\vec{u}_{i,G,3}$). An evaluation function is applied on the four vectors and the one with the minimum evaluation value will become the individual selected for the next generation ($\vec{x}_{i,G+1}$). The evaluation function $F(\vec{x})$ is presented in the next subsection.

$$\vec{x}_{i,G+1} = \begin{cases} \vec{x}_{i,G}, & \text{if } F(\vec{x}_{i,G}) < \\ & \ \min(F(\vec{u}_{i,G,1}), F(\vec{u}_{i,G,2}), F(\vec{u}_{i,G,3})) \\ \vec{u}_{i,G,1}, & \text{else if } F(\vec{u}_{i,G,1}) < \\ & \ \min(F(\vec{u}_{i,G,2}), F(\vec{u}_{i,G,3})) \\ \vec{u}_{i,G,2}, & \text{else if } F(\vec{u}_{i,G,2}) < F(\vec{u}_{i,G,3}) \\ \vec{u}_{i,G,3}, & \text{otherwise} \end{cases} \tag{5.18}$$

5.5 Evaluation Function

For simplicity, we discuss the design of the evaluation function on the binary classification problem (the class label is either $+1$ or -1). In next section we will present the generalization to multi-class classification. Figure 5.4 shows an illustrative example with some time series instances in the SR-transformed space. Each of the instances can be denoted as $(\vec{z}_i, y_i)$ where $\vec{z}_i$ is the distance vector to the respective SRs and y_i is the class label ($i = 1, \ldots, n$, n is the number of instances). For positive class $y_i = 1$, otherwise $y_i = -1$. In the figure, $\vec{z}\vec{w}^{\mathrm{T}} + b = 0$ is the decision boundary.

Our goal is to maximize the margin between the two classes to ensure good generalization performance. As shown in the figure, the margin value is $2/\left\|\vec{w}\right\|$ given $y_i(\vec{z}_i\vec{w}^{\mathrm{T}} + b) \geq 1$ for



Positive class
Negative class
To classify
Signed distance:$(\vec{z}\vec{w}^{\mathrm{T}}+b)/\|\vec{w}\|$
Margin:$\frac{2}{\|\vec{w}\|}$
$\vec{z}\vec{w}^{\mathrm{T}}+b=1$
$\vec{z}\vec{w}^{\mathrm{T}}+b=0$
$\vec{z}\vec{w}^{\mathrm{T}}+b=-1$

Figure 5.4: Illustration of the design of the evaluation function

$i=1,\ldots,n$ are satisfied [79]. Hence we can formulate the problem as (5.19):

$$\begin{aligned} &\min_{SR,\vec{w},b} && \|\vec{w}\| \\ &s.t. && y_i(\vec{z}_i\vec{w}^{\mathrm{T}}+b)\geq 1, \qquad i=1,\ldots,n \end{aligned} \tag{5.19}$$

In the Support Vector Machine (SVM) [79], the goal is also to maximize the margin via a constrained optimization formulation. However, the problem here is different. In SVM, the location of instances are fixed, and only the decision boundary ($\vec{w}$, b) is considered to maximize the margin. In our case, the locations of both the decision boundary and the instances are changed (via changing SR) to find a maximum margin. Unlike the situation in the SVM formulation, the problem here is non-convex and non-differentiable.

As in SVM, a slack variable ξ is added to the constraints to soften the margin to enhance the performance. So the problem formulation becomes (5.20):

$$\begin{aligned} &\min_{SR,\vec{w},b} && \|\vec{w}\|+C\xi \\ &s.t. && y_i(\vec{z}_i\vec{w}^{\mathrm{T}}+b)\geq 1-\xi, \qquad i=1,\ldots,n \end{aligned} \tag{5.20}$$

where C is the penalty factor and is fixed at 10^4 in our experiments. ξ is determined by evolution (by adding ξ to each individual $\vec{x}$ and setting its range between 0 and 0.1).

The Minimum Penalty (MP) method [77] is adopted here to handle the constraints and form the final evaluation function. The problem in (5.20) can be further formulated as:

$$\begin{array}{lll} \min_{\vec{x}} & f(\vec{x}) & \\ s.t. & g_i(\vec{x}) \leq 0, & i = 1, \ldots, n \end{array} \tag{5.21}$$

where $\vec{x}$ is the individual containing SR, $\vec{w}$, b, ξ, $f(\vec{x}) = \|\vec{w}\| + C\xi$ and $g_i(\vec{x}) = 1 - y_i(\vec{z}_i\vec{w}^{\mathrm{T}} + b) - \xi$ for $i = 1, \ldots, n$.

Let $G_i(\vec{x}) = \max(g_i(\vec{x}), 0)$, then $G(\vec{x}) = \sum_{i=1}^{n} G_i(\vec{x})$ reflects the constraint violation of individual $\vec{x}$. The constraint violation is normalized as:

$$G(\vec{x}_i) = (G(\vec{x}_i) - G_{\min})/(G_{\max} - G_{\min}),\ i = 1, \ldots, 4NP \tag{5.22}$$

where G_{max} is the maximum constraint violation in the population, G_{min} is the minimum constraint violation in the population and $4NP$ is the number of individuals in the population (including the NP target vectors and $3NP$ trial vectors). $f(\vec{x})$ is also normalized to have the same order of magnitude as $G(\vec{x})$:

$$f(\vec{x}_i) = (f(\vec{x}_i) - f_{\min})/(f_{\max} - f_{\min}),\ i = 1, \ldots, 4NP \tag{5.23}$$

where f_{max} and f_{max} are the maximum and minimum $f(\vec{x})$ values in the population respectively.

An individual that satisfies all the constraints is regarded as feasible and otherwise infeasible. When there is only infeasible individuals in the population, the final evaluation function $F(\vec{x})$ is defined as:

$$F(\vec{x}) = G(\vec{x}) \tag{5.24}$$

This design aims at steering the population to have some feasible individuals. After the presence of feasible individuals, the final evaluation function $F(\vec{x})$ is:

$$F(\vec{x}) = f(\vec{x}) + r \times G(\vec{x}) \tag{5.25}$$

According to the Minimum Penalty Method [77], $r = 1.0001 r_{\min}$ and $r_{\min}$ is provided by (5.26):

$$r_{\min} = \begin{cases} \max_k(\dfrac{f_{S1\min} - f(\vec{x}_k)}{G(\vec{x}_k)}), & \text{if} \ \ f_{S2\min} < f_{S1\min} \\ 0, & \text{otherwise} \end{cases} \tag{5.26}$$

where $f_{S1\min}$ is the minimum $f(\vec{x})$ value in the set of feasible individuals in the population and $f_{S2\min}$ denotes the minimum $f(\vec{x})$ value in the set of infeasible ones. $k \in S3$ and $S3$ is the subscript set containing the infeasible individuals with $f(\vec{x})$ values smaller than $f_{S1\min}$. With this evaluation function, individuals that can separate the time series instances with large margins will obtain small evaluation function values and thus survive the evolution process.

5.6 Generalization to Multi-class Classification

For binary classification problem, the evolution process described above finally produces a best individual (the one with the minimum evaluation function value in the population) containing the SRs and the respective DB ($\vec{w}$, b). Given a new unlabeled time series T, the method maps the series to an instance $\vec{z}$ in the new space by calculating its distances to the SRs. The class of this series (positive or negative) is predicted by the sign of $\vec{z}\vec{w}^{\mathrm{T}} + b$.

To generalize the method to multiple-class situation, the one-versus-all strategy is adopted. For each class in a dataset D with C classes, one classifier is trained using the class as positive class and the remaining classes as negative class. For an unlabeled time series T, there are C predictions of $\vec{z}_j\vec{w}_j^{\mathrm{T}} + b_j$, where $j = 1, \ldots, C$.

Figure 5.4 (yellow square instance) indicates the signed distance of an instance $\vec{z}$ to the decision boundary equals $(\vec{z}\vec{w}^{\mathrm{T}} + b)/\left\|\vec{w}\right\|$. This distance value reflects the confidence of the prediction. The larger the distance is, the farther the instance is away from the boundary, and thus the more confident the prediction is. When comparing the distances from different classes, the absolute distance may have different scales in different spaces. To compensate this, the distance is normalized by dividing the half margin value $1/\left\|\vec{w}\right\|$ in the respective space, resulting in $\vec{z}\vec{w}^{\mathrm{T}} + b$ as the relative distance.

The proposed approach then picks the class which produces the largest relative distance to the decision boundary as the final prediction h:

$$h = \arg\max_{j}(\vec{z}_j\vec{w}_j^{\mathrm{T}} + b_j), \quad j = 1, \ldots, C \tag{5.27}$$

Algorithm 5 provides the pseudo-code of the proposed ESR algorithm. In line 1 the label and the time series are separated. Also the number of total classes is counted and the time series are z-normalized if they have not been normalized. Then for each class the multi-value class labels are transformed to binary labels (+1 or -1) in Line 2-3. Afterwards a population is initialized using the time series and then the evolutionary operations detailed in the previous sections are conducted for a specified number of generations (Line 4-9). Line 10 extracts the final SRs and decision boundary from the best individual (the one with the minimum evaluation function value) in the population. Line 11-12 calculate the relative distance of the tested time series to the decision boundary. Finally in Line 14, the class producing the maximum relative distance is outputted as the predicted class value.

The time complexity of the proposed ESR is $O(n \times m^2 \times maxGen)$ where n is the number of instances in the training set, m is the time series length, $maxGen$ is the conducted generation number.

Algorithm 5 Evolving Separating References (ESR)

```
Input:
    $D$: training time series dataset
    $Ts$: to classify time series
    $K$: number of SRs
    $L$: length pattern of SRs
    $Dist$: distance formula
    $maxGen$: number of generation to perform
Output:
    $h$: predicted label of $Ts$
 1: $[T, Label, C] = loadData(D)$
 2: for $i = 1$ to $C$ do
 3:     $Y = binaryLabel(i, Label)$
 4:     $Population = initialization(T, K, L)$
 5:     for $G = 1$ to $maxGen$ do
 6:         $Population = mutation(Population)$
 7:         $Population = crossover(Population)$
 8:         $Population = selection(Population, T, Y, Dist)$
 9:     end for
10:     $[SRs, \vec{w}, b] = extractBest(Population)$
11:     $\vec{z}_i = transform(Ts, SRs, Dist)$
12:     $RD_i = \vec{z}_i \vec{w}^{\mathrm{T}} + b$
13: end for
14: $h = \arg\max_i(RD_i), i = 1, \ldots, C$
```

5.7 Experimental Evaluation

5.7.1 Experimental Setup

To evaluate the proposed approach, we perform experiments on the widely used UCR time series classification benchmark datasets [17], and compare the results with those of the state-of-the-art. The source code of the ESR is available in the supplementary material[1], the random seed in the source code is fixed to ensure reproducibility. The experiments were conducted on a batch processing cluster [38].

ESR has a binary parameter "Setting" which can take two value: $S1$ or $S2$. $S1$ corresponds to: applying the non-normalized distance formula (5.1) when calculating distances, $L = (0.05, 0.1, 0.2, 0.4, 0.6)$, $K = 5$. $S2$ is: adopting the normalized distance formula (5.3), $K = 5$, $L = (0.2, 0.4, 0.6)$. The Setting parameter is selected using 3-fold cross-validation on the training set. The parameter that yields lower average error rate in the cross-validation

[1] `https://github.com/xiaoshengli/ESR`

is chosen and ties are broken by taking $S1$. The evolution generation number is fixed at 5000 (extends to 10000 when no feasible individual exits in the 5000th generation) in all the experiments.

For comparison, 1-Nearest Neighbor (1NN) with Euclidean Distance (ED), 1NN with Dynamic Time Warping (DTW), 1NN with Dynamic Time Warping with Best warping window (DTWB) are chosen as standard baselines. We also compare with Fast Shapelet (FS) [34], SAX-VSM [35], Learning Shapelet (LS) [30], Representative Pattern Mining (RPM) [75], Bag-of-SFA-Symbols (BOSS) [32], and Collective of Transformation-based Ensembles (COTE) [16]. COTE is an ensemble of 35 classifiers on different data transformations and is the current best-performing method on accuracy [15].

5.7.2 Experimental Results

40 common datasets used by the above rivals from the UCR archive are taken as our test candidates and for fair comparison, the results of the rivals are directly taken from literature [16,32,75].

The testing error rates of the methods under comparison on the tested datasets are given in Table 5.1, 5.2, 5.3, and 5.4. The critical difference diagrams [39] of the results are presented to summarize the comparison.

Figure 5.5 shows the critical difference diagram ($alpha = 0.05$) for the comparison on the 40 tested datasets. The values in the figure are the respective rank means (lower is better) and the approaches connected by a bold bar have no significant difference to each other at the 95% confidence level. From the figure one can see the overall performance of ESR is better than DTWB, which is considered as a hard-to-beat standard baseline [23], but the difference is not significant. COTE, BOSS, LS have lower rank means than ESR, but the difference is not significant.

As analyzed previously, the proposed ESR method is especially fit for dealing with small size training data. Therefore we compare the results on the 20 smallest size datasets (measured by $n \times m$, where n is the number of instances in the training set and m is the

Table 5.1: Testing error rates of ESR and other methods on UCR Archive benchmarks

DataSet	ED	DTW	DTWB	SAX-VSM	ESR
50words	0.369	0.31	0.242	0.374	0.255
Adiac	0.389	0.396	0.391	0.417	0.299
Beef	0.333	0.367	0.333	0.233	0.167
CBF	0.148	0.003	0.004	0.01	0.001
ChlorineConcentration	0.352	0.352	0.35	0.341	0.39
CinC_ECG_torso	0.103	0.349	0.07	0.344	0.18
Coffee	0	0	0	0	0
Cricket_X	0.423	0.246	0.252	0.308	0.274
Cricket_Y	0.433	0.256	0.228	0.318	0.244
Cricket_Z	0.413	0.246	0.238	0.297	0.267
DiatomSizeReduction	0.065	0.033	0.065	0.121	0.023
ECGFiveDays	0.203	0.232	0.203	0.001	0
FaceAll	0.286	0.192	0.192	0.245	0.078
FaceFour	0.216	0.17	0.114	0.114	0
FacesUCR	0.231	0.095	0.088	0.109	0.068
Fish	0.217	0.177	0.154	0.017	0.029
Gun_Point	0.087	0.093	0.087	0.013	0.013
Haptics	0.63	0.623	0.588	0.584	0.523
ItalyPowerDemand	0.045	0.05	0.045	0.089	0.042
Lighting2	0.246	0.131	0.131	0.213	0.295

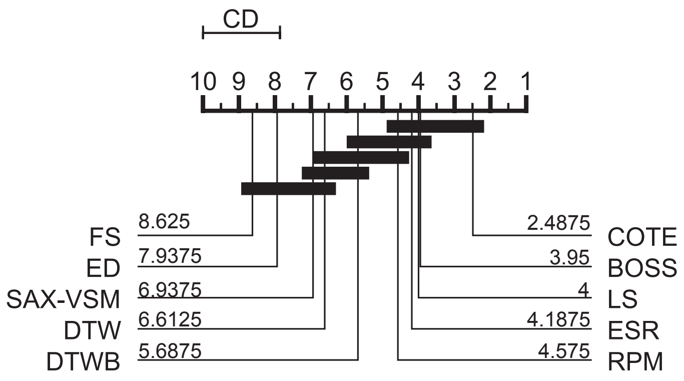


Figure 5.5: Critical difference diagram for the comparison on the 40 datasets

Table 5.2: (Continue) Testing error rates of ESR and other methods on UCR Archive benchmarks

DataSet	ED	DTW	DTWB	SAX-VSM	ESR
Lighting7	0.425	0.274	0.288	0.397	0.274
MALLAT	0.086	0.066	0.086	0.199	0.066
MedicalImages	0.316	0.263	0.253	0.516	0.242
MoteStrain	0.121	0.165	0.134	0.125	0.133
OliveOil	0.133	0.167	0.133	0.133	0.067
OSULeaf	0.479	0.409	0.388	0.165	0.174
SonyAIBORobotSurface	0.304	0.275	0.305	0.306	0.062
SonyAIBORobotSurfaceII	0.141	0.169	0.141	0.126	0.072
SwedishLeaf	0.211	0.208	0.157	0.278	0.077
Symbols	0.101	0.05	0.062	0.109	0.048
synthetic_control	0.12	0.007	0.017	0.017	0.003
Trace	0.24	0	0.01	0	0
Two_Patterns	0.093	0	0.002	0.004	0.014
TwoLeadECG	0.253	0.096	0.132	0.014	0.001
uWaveGestureLibrary_X	0.261	0.273	0.227	0.323	0.207
uWaveGestureLibrary_Y	0.338	0.366	0.301	0.364	0.313
uWaveGestureLibrary_Z	0.35	0.342	0.322	0.356	0.273
Wafer	0.005	0.02	0.005	0.001	0.002
WordsSynonyms	0.382	0.351	0.252	0.44	0.346
Yoga	0.17	0.164	0.155	0.151	0.22
Rank Mean	7.938	6.613	5.688	6.938	4.188

Table 5.3: (Continue) Testing error rates of ESR and other methods on UCR Archive benchmarks

DataSet	FS	LS	RPM	BOSS	COTE	ESR
50words	0.483	0.232	0.242	0.301	0.191	0.255
Adiac	0.425	0.437	0.355	0.22	0.233	0.299
Beef	0.467	0.24	0.233	0.2	0.133	0.167
CBF	0.092	0.006	0.001	0	0.001	0.001
ChlorineConcentration	0.667	0.349	0.355	0.34	0.314	0.39
CinC_ECG_torso	0.225	0.167	0.125	0.125	0.064	0.18
Coffee	0.071	0	0	0	0	0
Cricket_X	0.549	0.209	0.236	0.259	0.154	0.274
Cricket_Y	0.495	0.249	0.379	0.208	0.167	0.244
Cricket_Z	0.533	0.201	0.19	0.246	0.128	0.267
DiatomSizeReduction	0.078	0.033	0.069	0.046	0.082	0.023
ECGFiveDays	0	0	0	0	0	0
FaceAll	0.408	0.218	0.231	0.21	0.105	0.078
FaceFour	0.091	0.048	0.045	0	0.091	0
FacesUCR	0.296	0.059	0.034	0.042	0.057	0.068
Fish	0.189	0.066	0.065	0.011	0.029	0.029
Gun_Point	0.04	0	0.027	0	0.007	0.013
Haptics	0.61	0.532	0.562	0.536	0.481	0.523
ItalyPowerDemand	0.063	0.031	0.044	0.053	0.036	0.042
Lighting2	0.361	0.177	0.262	0.148	0.164	0.295
Lighting7	0.397	0.197	0.219	0.342	0.247	0.274

Table 5.4: (Continue) Testing error rates of ESR and other methods on UCR Archive benchmarks

DataSet	FS	LS	RPM	BOSS	COTE	ESR
MALLAT	0.054	0.046	0.02	0.058	0.036	0.066
MedicalImages	0.443	0.271	0.262	0.288	0.258	0.242
MoteStrain	0.202	0.087	0.113	0.073	0.085	0.133
OliveOil	0.3	0.56	0.1	0.1	0.1	0.067
OSULeaf	0.421	0.182	0.211	0.012	0.145	0.174
SonyAIBORobotSurface	0.315	0.103	0.058	0.321	0.146	0.062
SonyAIBORobotSurfaceII	0.211	0.082	0.114	0.098	0.076	0.072
SwedishLeaf	0.229	0.087	0.07	0.072	0.046	0.077
Symbols	0.091	0.036	0.042	0.032	0.046	0.048
synthetic_control	0.107	0.007	0.007	0.03	0	0.003
Trace	0	0	0	0	0	0
Two_Patterns	0.741	0.003	0.005	0.004	0	0.014
TwoLeadECG	0.075	0.003	0.048	0.016	0.015	0.001
uWaveGestureLibrary_X	0.316	0.2	0.226	0.241	0.196	0.207
uWaveGestureLibrary_Y	0.412	0.287	0.303	0.313	0.267	0.313
uWaveGestureLibrary_Z	0.353	0.269	0.279	0.312	0.265	0.273
Wafer	0.003	0.004	0.013	0.001	0.001	0.002
WordsSynonyms	0.542	0.34	0.353	0.345	0.266	0.346
Yoga	0.335	0.15	0.165	0.081	0.113	0.22
Rank Mean	8.625	4	4.575	3.95	2.488	4.188

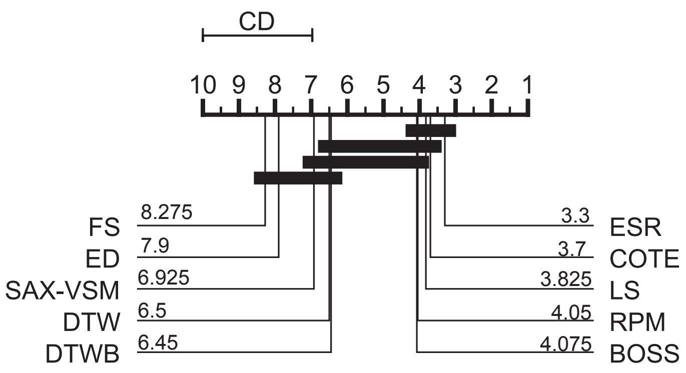


Figure 5.6: Critical difference diagram for the comparison on the 20 smallest datasets

length of time series). The comparison result is presented in Figure 5.6 and the raw data is in Table 5.5 and 5.6. From the figure one observes that ESR achieves the best rank means among all the methods under comparison. Further ESR is the only method that significantly outperforms DTWB on the tested datasets. Conducting the Wilcoxon signed rank test on the results, the p-values between ESR and DTWB is 0.0019. The small p-value (< 0.05) indicates the results are significantly different between ESR and DTWB.

The above results demonstrate the effectiveness of ESR for time series classification. ESR is highly competitive in the situations where only small size training data is available. This coincides with the previous theoretical analysis.

5.8 Conclusion of ESR

This chapter presents a novel perspective that sequences of values that are very different from the patterns in the time series can be used as references to classify the data effectively. We propose an evolution process to obtain these sequences, i.e. the separating references, from a given time series dataset. Instead of mining and extracting patterns from the time series as performed by the state-of-the-arts, the proposed method focuses on constructing some references to separate instances from different classes with large margins. The experiments on the UCR time series classification benchmarks demonstrate the effectiveness of the proposed approach.

Table 5.5: Testing error rates of ESR and other methods on the 20 smallest datasets

Dataset	ED	DTW	DTWB	SAX-VSM	ESR
Beef	0.333	0.367	0.333	0.233	0.167
CBF	0.148	0.003	0.004	0.01	0.001
Coffee	0	0	0	0	0
DiatomSizeReduction	0.065	0.033	0.065	0.121	0.023
ECGFiveDays	0.203	0.232	0.203	0.001	0
FaceFour	0.216	0.17	0.114	0.114	0
FacesUCR	0.231	0.095	0.088	0.109	0.068
Gun_Point	0.087	0.093	0.087	0.013	0.013
ItalyPowerDemand	0.045	0.05	0.045	0.089	0.042
Lighting2	0.246	0.131	0.131	0.213	0.295
Lighting7	0.425	0.274	0.288	0.397	0.274
MedicalImages	0.316	0.263	0.253	0.516	0.242
MoteStrain	0.121	0.165	0.134	0.125	0.133
OliveOil	0.133	0.167	0.133	0.133	0.067
SonyAIBORobotSurface	0.304	0.275	0.305	0.306	0.062
SonyAIBORobotSurfaceII	0.141	0.169	0.141	0.126	0.072
Symbols	0.101	0.05	0.062	0.109	0.048
synthetic_control	0.12	0.007	0.017	0.017	0.003
Trace	0.24	0	0.01	0	0
TwoLeadECG	0.253	0.096	0.132	0.014	0.001
Rank Mean	7.9	6.5	6.45	6.925	3.3

Table 5.6: (Continue) Testing error rates of ESR and other methods on the 20 smallest datasets

Dataset	FS	LS	RPM	BOSS	COTE	ESR
Beef	0.467	0.24	0.233	0.2	0.133	0.167
CBF	0.092	0.006	0.001	0	0.001	0.001
Coffee	0.071	0	0	0	0	0
DiatomSizeReduction	0.078	0.033	0.069	0.046	0.082	0.023
ECGFiveDays	0	0	0	0	0	0
FaceFour	0.091	0.048	0.045	0	0.091	0
FacesUCR	0.296	0.059	0.034	0.042	0.057	0.068
Gun_Point	0.04	0	0.027	0	0.007	0.013
ItalyPowerDemand	0.063	0.031	0.044	0.053	0.036	0.042
Lighting2	0.361	0.177	0.262	0.148	0.164	0.295
Lighting7	0.397	0.197	0.219	0.342	0.247	0.274
MedicalImages	0.443	0.271	0.262	0.288	0.258	0.242
MoteStrain	0.202	0.087	0.113	0.073	0.085	0.133
OliveOil	0.3	0.56	0.1	0.1	0.1	0.067
SonyAIBORobotSurface	0.315	0.103	0.058	0.321	0.146	0.062
SonyAIBORobotSurfaceII	0.211	0.082	0.114	0.098	0.076	0.072
Symbols	0.091	0.036	0.042	0.032	0.046	0.048
synthetic_control	0.107	0.007	0.007	0.03	0	0.003
Trace	0	0	0	0	0.01	0
TwoLeadECG	0.075	0.003	0.048	0.016	0.015	0.001
Rank Mean	8.275	3.825	4.05	4.075	3.7	3.3

Chapter 6: Conclusion and Future Work

In this dissertation, we present three linear time complexity methods for the classification, motif discovery, and clustering of time series data respectively. For linear time complexity time series classification, we propose the Bag-Of-Pattern-Features (BOPF) method. The classification results on the 85 datasets in the UCR time series classification archive demonstrate the effectiveness of the method. We design further experiments to verify the design strategies in BOPF. A case study on the bird sounds data shows the utility of the approach on real-world large size data.

For time series motif discovery, this dissertation proposes an algorithm that can find the exact motifs in a linear expected time complexity. The algorithm is further modified to find a motif set under a given threshold. The dissertation gives the detailed proof of correctness and time complexity. Experimental results of the proposed algorithms show that the algorithms can always find the motifs. The proposed algorithm is orders of magnitude faster than the state-of-the-art on the tested dataset and the case studies on the bird sound and electrical consumption data show its effectiveness in real-world situations.

For time series clustering, we introduce a Symbolic Pattern Forest (SPF) algorithm. The method partitions the time series instances by checking some randomly selected symbolic patterns and the partitions of multiple runs are combined to give a final cluster assignment. The dissertation provides analysis on the time complexity and effectiveness of the algorithm. We evaluate the algorithm extensively on all 85 datasets from the UCR time series archive and the results show that SPF is very competitive compared with other rival methods.

To demonstrate the linear time complexity, we run these methods on inputs of different sizes multiple times and record the average running time on each size. We perform linear curve fitting on the experiment results and the analysis shows that the average running time of these methods have strong linear relationships with the input sizes.

The research in the dissertation shows that we can have linear time complexity methods for the classification, motif discovery, and clustering of time series with effectiveness comparable or superior to the state-of-the-arts. With these methods, we can handle large size data and process the time series fast, which will benefit a variety of fields where the three time series data mining tasks have applications.

In addition to investigating on linear time solutions, this dissertation also presents a perspective that sequences of values that are very different from the patterns in the time series can be used as references to classify the data effectively. We propose an evolution process to receive these sequences, i.e. the separating references, from a given time series dataset. Instead of mining and extracting patterns from the time series as performed by the state-of-the-arts, the proposed approach focuses on constructing some references to separate instances from different classes with large margins. The experiments on the UCR time series classification benchmarks show the effectiveness of the proposed method.

All the proposed methods in this dissertation are designed for univariate time series data. A future research direction is to modify and extend the proposed approaches to make them work on multivariate time series data. A device can have multiple sensors for different measurements in the same time period, which will lead to multivariate time series data. The time series data mining methods should be able to figure out which dimensions of the multivariate time series are helpful for the tasks at hand and ignore those dimensions that are irrelative. The methods should be able to combine the features from different helpful dimensions correctly for the data mining tasks, meanwhile still maintain the desirable linear time complexity property.

Another future direction is to apply the deep learning techniques in time series clustering. The deep learning methods have been successfully applied in many applications and show great potential in various fields. Applying the deep learning techniques may bring benefits for time series clustering.

Bibliography

Bibliography

[1] T.-c. Fu, "A review on time series data mining," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[2] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover, "Exact discovery of time series motifs," in *Proceedings of the 2009 SIAM international conference on data mining.* SIAM, 2009, pp. 473–484.

[3] A. Mueen, "Enumeration of time series motifs of all lengths," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on.* IEEE, 2013, pp. 547–556.

[4] M. Shokoohi-Yekta, Y. Chen, B. Campana, B. Hu, J. Zakaria, and E. Keogh, "Discovery of meaningful rules in time series," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining.* ACM, 2015, pp. 1085–1094.

[5] Y. Hao, M. Shokoohi-Yekta, G. Papageorgiou, and E. Keogh, "Parameter-free audio motif discovery in large data archives," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on.* IEEE, 2013, pp. 261–270.

[6] A. Muscariello, G. Gravier, and F. Bimbot, "An efficient method for the unsupervised discovery of signalling motifs in large audio streams," in *Content-Based Multimedia Indexing (CBMI), 2011 9th International Workshop on.* IEEE, 2011, pp. 145–150.

[7] A. McGovern, D. H. Rosendahl, R. A. Brown, and K. K. Droegemeier, "Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction," *Data Mining and Knowledge Discovery*, vol. 22, no. 1-2, pp. 232–258, 2011.

[8] C. Cassisi, M. Aliotta, A. Cannata, P. Montalto, D. Patanè, A. Pulvirenti, and L. Spampinato, "Motif discovery on seismic amplitude time series: The case study of mt etna 2011 eruptive activity," *Pure and Applied Geophysics*, vol. 170, no. 4, pp. 529–545, 2013.

[9] U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. Alcock, "Finding anomalous periodic time series," *Machine learning*, vol. 74, no. 3, pp. 281–313, 2009.

[10] M. Kumar, N. R. Patel, and J. Woo, "Clustering seasonality patterns in the presence of errors," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2002, pp. 557–563.

[11] N. Subhani, L. Rueda, A. Ngom, and C. J. Burden, "Multiple gene expression profile alignment for microarray time-series data clustering," *Bioinformatics*, vol. 26, no. 18, pp. 2281–2288, 2010.

[12] J. Lin, R. Khade, and Y. Li, “Rotation-invariant similarity in time series using bag-of-patterns representation,” *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012.

[13] G. Salton, A. Wong, and C.-S. Yang, “A vector space model for automatic indexing,” *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[14] J. Lin, E. Keogh, L. Wei, and S. Lonardi, “Experiencing sax: a novel symbolic representation of time series,” *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[15] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, “The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances,” *Data Mining and Knowledge Discovery*, pp. 1–55, 2016.

[16] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, “Time-series classification with cote: the collective of transformation-based ensembles,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2522–2535, 2015.

[17] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, “The ucr time series classification archive,” July 2015, www.cs.ucr.edu/~eamonn/time_series_data/.

[18] D. J. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series.” in *KDD workshop*, vol. 10, no. 16. Seattle, WA, 1994, pp. 359–370.

[19] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, “Searching and mining trillions of time series subsequences under dynamic time warping,” in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2012, pp. 262–270.

[20] Xeno-canto, “The xeno-canto website,” Dec 2020, http://www.xeno-canto.org.

[21] P. Mermelstein, “Distance measures for speech recognition, psychological and instrumental,” *Pattern recognition and artificial intelligence*, vol. 116, pp. 374–388, 1976.

[22] X. Li and J. Lin, “Linear time complexity time series classification with bag-of-pattern-features,” in *2017 IEEE International Conference on Data Mining (ICDM).* IEEE, 2017, pp. 277–286.

[23] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, “Querying and mining of time series data: experimental comparison of representations and distance measures,” *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[24] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, “Weighted dynamic time warping for time series classification,” *Pattern Recognition*, vol. 44, no. 9, pp. 2231–2240, 2011.

[25] P.-F. Marteau, “Time warp edit distance with stiffness adjustment for time series matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 306–318, 2009.

[26] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM (JACM)*, vol. 24, no. 4, pp. 664–675, 1977.

[27] L. Ye and E. 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.* ACM, 2009, pp. 947–956.

[28] A. Mueen, E. Keogh, and N. 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.* ACM, 2011, pp. 1154–1162.

[29] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, "Classification of time series by shapelet transformation," *Data Mining and Knowledge Discovery*, vol. 28, no. 4, pp. 851–881, 2014.

[30] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2014, pp. 392–401.

[31] L. Hou, J. T. Kwok, and J. M. Zurada, "Efficient learning of timeseries shapelets," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[32] P. Schäfer, "The boss is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.

[33] P. Schäfer and U. Leser, "Fast and accurate time series classification with weasel," *arXiv preprint arXiv:1701.07681*, 2017.

[34] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *Proceedings of the thirteenth SIAM conference on data mining (SDM).* SIAM, 2013, pp. 668–676.

[35] P. Senin and S. Malinchik, "Sax-vsm: Interpretable time series classification using sax and vector space model," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on.* IEEE, 2013, pp. 1175–1180.

[36] P. Schäfer, "Scalable time series classification," *Data Mining and Knowledge Discovery*, vol. 30, no. 5, pp. 1273–1298, 2016.

[37] D. C. Montgomery and G. C. Runger, *Applied statistics and probability for engineers.* John Wiley & Sons, 2010.

[38] "The argo cluster," 2020, http://wiki.orc.gmu.edu/index.php/About_ARGO.

[39] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[40] "Supporting web page for bopf," 2020, http://mason.gmu.edu/~xli22/BOPF.

[41] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[42] A. Mueen, "Time series motif discovery: dimensions and applications," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 2, pp. 152–159, 2014.

[43] S. Torkamani and V. Lohweg, "Survey on time series motif discovery," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 2, 2017.

[44] Y. Li, M. L. Yiu, Z. Gong *et al.*, "Quick-motif: An efficient and scalable framework for exact motif discovery," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on.* IEEE, 2015, pp. 579–590.

[45] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on.* IEEE, 2016, pp. 739–748.

[46] P. A. M. for Aging People, "hg38, grch38 genome reference consortium human reference 38," 2018, http://hgdownload.soe.ucsc.edu/goldenPath/hg38/chromosomes/.

[47] X. Li and J. Lin, "Linear time motif discovery in time series," in *Proceedings of the 2019 SIAM International Conference on Data Mining.* SIAM, 2019, pp. 136–144.

[48] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2003, pp. 493–498.

[49] Y. Li, J. Lin, and T. Oates, "Visualizing variable-length time series motifs," in *Proceedings of the 2012 SIAM international conference on data mining.* SIAM, 2012, pp. 895–906.

[50] Y. Gao, J. Lin, and H. Rangwala, "Iterative grammar-based framework for discovering variable-length time series motifs," in *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on.* IEEE, 2016, pp. 7–12.

[51] M. J. Golin, R. Raman, C. Schwarz, and M. Smid, "Simple randomized algorithms for closest pair problems," *Nordic Journal of Computing*, vol. 2, pp. 3–27, 1995.

[52] D. Murray, J. Liao, L. Stankovic, V. Stankovic, R. Hauxwell-Baldwin, C. Wilson, M. Coleman, T. Kane, and S. Firth, "A data management platform for personalised real-time energy feedback," in *8th International Conference on Energy Efficiency in Domestic Appliances and Lighting*, 2015.

[53] X. Li, J. Lin, and L. Zhao, "Linear time complexity time series clustering with symbolic pattern forest." in *IJCAI*, 2019, pp. 2930–2936.

[54] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[55] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, *Fast subsequence matching in time-series databases.* ACM, 1994, vol. 23, no. 2.

[56] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining and Knowledge Discovery*, vol. 26, no. 2, pp. 275–309, 2013.

[57] L. Gupta, D. L. Molfese, R. Tammana, and P. G. Simos, "Nonlinear alignment and averaging for estimating the evoked potential," *IEEE Transactions on Biomedical Engineering*, vol. 43, no. 4, pp. 348–356, 1996.

[58] V. Niennattrakul and C. A. Ratanamahatana, "Shape averaging under time warping," in *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, vol. 2. IEEE, 2009, pp. 626–629.

[59] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern Recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[60] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1855–1870.

[61] N. Kumar, V. N. Lolla, E. Keogh, S. Lonardi, C. A. Ratanamahatana, and L. Wei, "Time-series bitmaps: a practical visualization tool for working with large time series databases," in *Proceedings of the 2005 SIAM international conference on data mining*. SIAM, 2005, pp. 531–535.

[62] J. Zakaria, A. Mueen, and E. Keogh, "Clustering time series using unsupervised-shapelets," in *2012 IEEE 12th International Conference on Data Mining*. IEEE, 2012, pp. 785–794.

[63] Q. Zhang, J. Wu, H. Yang, Y. Tian, and C. Zhang, "Unsupervised feature learning from time series." in *IJCAI*, 2016, pp. 2322–2328.

[64] C. Biernacki, G. Celeux, and G. Govaert, "Assessing a mixture model for clustering with the integrated completed likelihood," *IEEE transactions on pattern analysis and machine intelligence*, vol. 22, no. 7, pp. 719–725, 2000.

[65] M. Ramoni, P. Sebastiani, and P. Cohen, "Multivariate clustering by dynamics," in *AAAI/IAAI*, 2000, pp. 633–638.

[66] M. Bicego, V. Murino, and M. A. Figueiredo, "Similarity-based clustering of sequences using hidden markov models," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2003, pp. 86–95.

[67] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[68] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

[69] C. A. Ratanamahatana and E. Keogh, "Everything you know about dynamic time warping is wrong." Citeseer, 2004.

[70] X. Z. Fern and C. E. Brodley, "Solving cluster ensemble problems by bipartite graph partitioning," in *Proceedings of the twenty-first international conference on Machine learning.* ACM, 2004, p. 36.

[71] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[72] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding.* Springer, 1994, pp. 409–426.

[73] N. Saito and R. R. Coifman, "Local feature extraction and its applications using a library of bases," Ph.D. dissertation, Yale University, 1994.

[74] P. A. M. for Aging People, 2017, http://www.pamap.org/.

[75] X. Wang, J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein, "Rpm: Representative pattern mining for efficient time series classification," in *EDBT*, 2016, pp. 185–196.

[76] X. Li and J. Lin, "Evolving separating references for time series classification," in *Proceedings of the 2018 SIAM international conference on data mining.* SIAM, 2018, pp. 243–251.

[77] X. Li and G. Zhang, "Minimum penalty for constrained evolutionary optimization," *Computational Optimization and Applications*, vol. 60, no. 2, pp. 513–544, 2015.

[78] Y. Wang, Z. Cai, and Q. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 55–66, 2011.

[79] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning.* Springer series in statistics Springer, Berlin, 2001, vol. 1.

Curriculum Vitae

Xiaosheng Li received his Bachelor of Engineering from Tianjin University in 2011. He received his Master of Engineering from Tianjin University in 2014. He was employed as a teaching assistant for six years in George Mason University and received his PhD in Computer Science from George Mason University in 2020. His research focuses on data mining and machine learning, especially on time series data. He is interested in the classification, clustering, motif discovery, deep learning of time series.