Accurate and Fast Time Series Classification based on Compressed Random Shapelet Forest

Jun Yang[1,2,3], Siyuan Jing[3]*, Guanying Huang[3]

(1) Chengdu Institute of Computer Application, Chinese Academy of Sciences, Chengdu 610041, China

(2) University of Chinese Academy of Sciences, Beijing 100049, China

(3) School of Artificial Intelligence, Leshan Normal University, Leshan 614000, China

yangjun192@mails.ucas.ac.cn, siyuan-jing@lsnu.edu.cn

Abstract: Achieving accurate, fast and interpretable time series classification (TSC) has attracted considerable attentions from data mining community over the past decade. In this paper, we propose an efficient algorithm, called compressed random shapelet forest (CRSF), to tackle this problem. Different from most of the shapelet-based TSC methods, CRSF obtains promising performance by greatly compressing the shapelet features space. In order to achieve the aim of compression, the time series dataset as well as the shapelets are represented by symbolic aggregate approximation (SAX) at first. Then, the shapelet-based decision trees are built upon a pool of high-quality shapelet candidates from which the useless shapelets and the self-similar shapelets have been pre-pruned. A new function for measuring distance between two SAX-represented time series is also introduced. Extensive experiments were conducted on 50 UCR time series datasets. The results show that, (1) CRSF can achieve the highest average accuracy on the datasets and it outperforms most of the existing shapelet-based TSC methods; (2) CRSF is slightly superior to gRSF in terms of accuracy and is significantly superior to gRSF in terms of the time cost. Specifically, it is averagely 41 times faster than gRSF according to the experimental results.

Keywords: Time series classification; shapelet; random forest; sax representation; compression

1. Introduction

Benefit from modern information technologies, such as 5G and Internet of things (IoT), data acquisition and transmission become more and more convenient. It is interesting that abundant of time series data appears in a wide range of real-life domains over the past two decades (Bagnall, Lines, Bostrom, Large, & Keogh, 2017; Katris, 2021). Generally, a time series refers to a sequence of ordered data points. Classical examples include transaction data of stock market, electrocardiogram, monitoring data of IoT, etc. Besides that, some data that seemingly do not have temporal relation can also be transformed into time series, e.g. food spectrographic, multimedia, motion trajectory, and so on (Gordon, Hendler, Kontorovich, & Rokach, 2015; Hong, Park, & Baek, 2020; Lahreche & Boucheham, 2021). Under this background, time series data mining (TSDM) attracts considerable attentions from academia and industry. Time series classification (TSC) is an important research branch of the TSDM.

Similar to other classification tasks, its aim is to correctly classify unlabeled time series instances into pre-defined classes. However, the intrinsic characteristic of time series presents a special challenge because the temporal relation within variables is often critical in finding the best discriminating features.

Up to now, the technologies of TSC can be grouped into three categories (Bagnall, et al., 2017; Ruiz, Flynn, Large, Middlehurst, & Bagnall, 2021). The first is the whole-series comparison based approach, in which two series are compared as a vector by a method that combines 1-Nearest Neighbor (1NN) and a distance measure, such as Euclidean Distance (ED) or Dynamic Time Warping distance (DTW). Most research interests in this direction are finding techniques that can compensate for small misalignments between times series using elastic distance measures, including Weighted DTW (Jeong, Jeong, & Omitaomu, 2011), Time Warp Edit (Marteau, 2009), Move-Split-Merge (Stefan, Athitsos, & Das, 2013), Complexity Invariant Distance (Batista, Keogh, Tataw, & de Souza, 2013), Derivative DTW (Górecki & Łuczak, 2012), and so on. The whole-series comparison based methods have the advantages of simplicity, robustness, and not requiring extensive parameter tuning. However, most of them suffer from the time and space overhead, and the obtained results lack interpretability. The second category is based on Deep Neural Network (DNN) (Ismail Fawaz, Forestier, Weber, Idoumghar, & Muller, 2019), which is the hottest topic in the field of machine learning recently. The representative works include Fully Convolutional Network (FCN) (Z. Wang, Yan, & Oates, 2017), InceptionTime (Ismail Fawaz, et al., 2020), TapNet (Zhang, Gao, Lin, & Lu, 2020), etc. The DNN-based methods have achieved very competitive results on most data sets. Unfortunately, their mining results are entirely a black-box, i.e. cannot tell us anything about why a particular instance is assigned to a particular class label. The final category is motif-based TSC which aims to identify motifs (or local patterns) that have discrimination power towards the class variable (Baydogan & Runger, 2015). Besides high discriminative power, motif also provides good interpretability. This characteristic is very important for some domains, including computer aided medical diagnoses, decision support system, etc. In this work, we focus on the motif-based algorithm of TSC.

Shapelet, which is a supervised subsequence of time series, is a case of time series motif that is maximally descriptive of the class variable (Ye & Keogh, 2009). Figure 1 gives an illustrating example. The data set is from UCR repository (Dau, et al., 2019), called *ECG200* in which class 1/2 represent a normal heartbeat and a myocardial infarction event respectively. From the figure, it is easy to understand that the shapelet-based TSC requires to find a set of high-quality shapelets which are used as discriminatory features to build a classifier. Classification via shapelet offers several advantages. First of all, shapelet can provide interpretive features to help researchers understand the difference among the target classes. Secondly, shapelet creates a very compact representation of the class concept, and this can significantly reduce the time and space overhead of classification. Due to this, shapelet-based TSC has attracted significant interests of the TSDM community over the past decade.

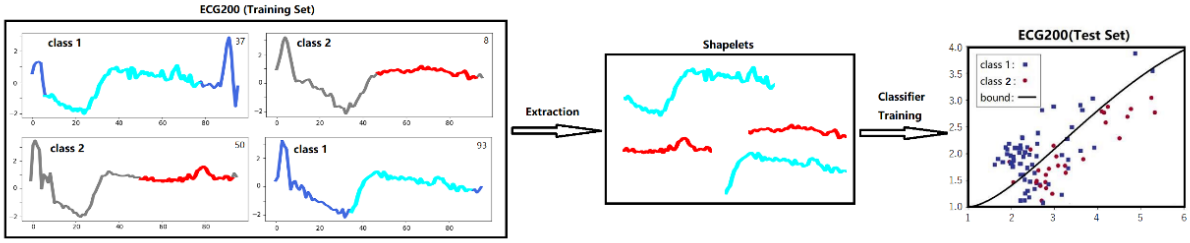

Figure 1. Time series classification via shapelet

In this paper, we propose an efficient algorithm, named Compressed Random Shapelet Forest (CRSF), for accurate and fast classification of time series. The proposed algorithm is designed based on gRSF (Karlsson, Papapetrou, & Boström, 2016) which is the first shapelet-based random forest classifier for TSC. There are two significant differences between CRSF and gRSF. On the one hand, CRSF employs SAX (Lin, Keogh, Wei, & Lonardi, 2007) to represent the time series dataset and shapelets. The advantage is that it not only eliminates the influence of noisy signal in time series, but also significantly reduces the feature space of shapelet. On the other hand, CRSF builds a set of decision trees upon a pool of high-quality shapelets which are diverse and non-self-similar. This is significantly different from gRSF in which the shapelets used for growing a tree node are sampled online. When generating a tree node, gRSF requires to randomly sample a number of shapelets from the dataset and select the shapelet that has the highest information gain to generate the node. However, the sampled shapelets maybe self-similar or without classification ability. To achieve satisfactory accuracy, gRSF has to increase the number of sampling shapelets which, in turn, greatly increases the training time of the algorithm. The tie is not easy to break in gRSF.

The contributions of this work can be summarized as follows:

(1) We propose a new shapelet-based TSC algorithm named CRSF, which uses SAX to represent time series and builds the decision trees upon a pool of “high quality” shapelet candidates. Moreover, some efficient strategies are employed in our algorithm, including a new function for measuring the distance between two SAX-represented time series, a roulette wheel strategy for shapelet selection, etc.

(2) We perform extensive experiments on 50 time series datasets from the UCR archive. Experimental results show that CRSF outperforms most of the existing shapelet-based methods in terms of accuracy, including FS (Rakthanmanon & Keogh, 2013), ELIS (Fang, Wang, & Wang, 2018), BSPCover (Guozhong Li, et al., 2021), etc. Moreover, it shows significant improvement in terms of efficiency compared with gRSF.

(3) We test different strategies of shapelet sampling and find that sampling shapelet under multiple SAX configurations is inferior to sampling shapelet under a single SAX configuration. In addition, we test the performance of CRSF under different parameter settings and provide insights into how to tune them for different situations.

The rest of the paper is organized as follows. In the next section, we review related work of TSC and focus on the shapelet-based approaches. Section 3 gives the concepts as well as the formal descriptions of the problem. In section 4, we introduce the details of CRSF and give the time complexity analysis. In section 5, the design of the experiment and analysis of experimental results are presented. Finally, we summarize the findings of this work in section 6.

2. Related work

In this section, we present the related work of time series representation and shapelet-based TSC algorithm.

Time series representation. It is well known that finding valuable knowledge from raw time series data is difficult because it maybe high dimensional or contains plenty of noise (X. Wang, et al., 2013). To tackle this problem, some researchers utilized domain transformation, such as Discrete Fourier Transformation (DTF) (Agrawal, Faloutsos, & Swami, 1993) and Discrete Wavelet Transformation (DWT) (Chan, Ada Wai-chee, & Clement, 2003), to convert the time series to the frequency domain. By this way, the time series is represented by few frequency coefficients, thus the dimension is greatly reduced. Another important method of dimension reduction is Piecewise Aggregate Approximation (PAA) (Keogh, Chakrabarti, Pazzani, & Mehrotra, 2001) which divides the time series data into segments with the same length, and then uses the mean value of the segment to represent the original data. However, both of the methods are non-adaptive. So far, the most successful technique for time series representation is SAX (Lin, et al., 2007), which transforms original real-value time series to symbolic representation, just like "ABCAD", "AACCB" and so on. The transformation is achieved by three steps. Firstly, it applies PAA to represent a time series as segmental mean value. Then, it divides the range into several regions based on Gaussian distribution, and each region corresponds to a character. Finally, a real-valued sequence obtained by PAA (i.e. segmental mean value) can be mapped to a string of characters. The SAX representation not only achieves much better performance than other methods, but also introduces abundant methods of bioinformatics and text mining into TSDM.

Shapelet-based TSC. Ye and Keogh (Ye & Keogh, 2009) firstly presented the concept of shapelet in KDD 2009. Shapelet refers to a subsequence of series that is maximally descriptive of the target variable. Shapelet-based TSC quickly becomes a hot issue in the data mining community because it not only creates a compact representation of the target variable but also provides an interpretable classification model. They employ information gain to evaluate shapelets and find the best shapelet via a brute-force algorithm. After that, the obtained shapelet is used to build a decision tree classifier. Although several speed-up techniques are adopted, its worst time complexity is $O\left(n^2 m^4\right)$, where n is the number of time series in the dataset and m is the length of the longest time series. It is not difficult to find that the exhaustive search is too time-consuming. Besides, the shapelet discovery is embedded into the training of a decision tree that affects the accuracy of the classification.

In order to reduce the time complexity, Mueen et al. proposed Logic Shapelet (LS) (Mueen, Keogh,

& Young, 2011) in which a technique of pre-computation of statistics is used to speed up the distance computation and an admissible pruning technique is used to skip the costly computation of entropy for the vast majority of candidate shapelets. It finds shapelets in less time than Ye's method by an order of magnitude. In addition, LS presents a novel way of shapelet representation, i.e. conjunctions or disjunctions of shapelets, which is more expressive than classical shapelet. Rakthanmanon and Keogh proposed Fast Shapelet (FS) (Rakthanmanon & Keogh, 2013) which employs SAX (Lin, et al., 2007) representation and a random projection technique to find potential shapelet candidates. The time complexity of FS algorithm is $O\left(nm^2\right)$. Grabocka et al. proposed a Scalable shapelet Discovery algorithm (SD) (Grabocka, Wistuba, & Schmidt-Thieme, 2015), in which the PAA is used to compress time series data and an online clustering/pruning technique is used to avoid measuring the prediction accuracy of similar candidates in Euclidean distance space. Since the SD algorithm prunes 99% of shapelet candidates, it can be 3–4 orders of magnitudes faster than FS.

Tight coupling of shapelet discovery and training of decision tree is another factor that hinders the accuracy improvement. Lines et al. proposed a transformation-based shapelet classification (TS)(Hills, Lines, Baranauskas, Mapp, & Bagnall, 2013) technique. The ties are resolved by constructing a new feature space based on the top *k* best shapelets and then transforming original time series data into the new feature space in which each data point is the similarity (i.e. distance) between the corresponding shapelet and the time series. Li et al. (Guiling Li, Yan, & Wu, 2019) proposed a shapelet discovery method PSKP that extracts shapelet candidates with key points found before.

Grabocka et al. presented a novel technique of Learning Time-series Shapelet (LTS) (Grabocka, Schilling, Wistuba, & Schmidt-Thieme, 2014) that, to our knowledge, firstly introduces machine learning into shapelet discovery. LTS models the shapelet learning task based on a logistic sigmoid function and proposes a method to learn the top-K true shapelets by optimizing the objective function. Hou et al. (Hou, Kwok, & Zurada, 2016) improve LTS via the generalized eigenvector method and fused lasso. Fang et al. (Fang, et al., 2018) proposed a shapelet-filtering strategy to improve LTS, i.e. the shapelet learning is not performed on all the shapelet extracted from entire raw time series data, but a set of screened PAA-represented shapelets.

In addition, Karlsson et al. proposed a generalized Random Shapelet Tree (gRSF) (Karlsson, et al., 2016) algorithm which consists of a set of shapelet-based decision trees, where both the choice of instances used for building a tree and the choice of shapelets are randomized. The experiments prove its effectiveness and efficiency, and moreover, the gRSF algorithm is easy to be implemented. Recently, Li et al. (Guozhong Li, et al., 2021)proposed BSPCover that focuses on the discovery of a set of high-quality shapelet candidates for model building. BSPCover employs Bloom filters and similarity matching to prunes identical and highly similar shapelet candidates, and then uses p-cover algorithm to determine discriminative shapelet candidates, and finally applies existing shapelet-learning technique (i.e. LTS) to build the classifier. Experimental results show that BSPCover achieves very completive performance compared to existing TSC methods.

3. Definitions and Symbols

In this paper, we study the classification of time series which can be formally described as a sequence of real numbers ordered by time and sampled at regular and fixed intervals. The purpose is to infer a classifier that is able to predict the class of unlabeled time series correctly. We start by introducing the necessary definitions and symbols.

Definition 1 (Time series): A time series $T = \langle t_1, t_2, \cdots, t_m \rangle$ is denoted as a sequence of m variables, where m is the length of the time series and $t_i \in \mathbb{R} \wedge i \in \{1, 2, \cdots, m\}$.

To simplify the presentation of the problem, we only consider the uni-variable time series here, but it is not difficult to extend the definition to multi-variable time series. A finite set of time series can be written as $\Gamma = \{T_1, T_2, \cdots, T_n\}$.

Definition 2 (Subsequence): A subsequence of a time series $T_{i,k,l}$ is a contiguous sequence of time series T_i, where k is the starting position and l is the length of the subsequence.

Definition 3 (Time series dataset): A time series dataset D is a set of pairs (a.k.a. instance) of a time series T_i and its label $c_i \in C$. Formally, $D = \{\langle T_1, c_1 \rangle, \langle T_2, c_2 \rangle, \cdots, \langle T_{|D|}, c_{|D|} \rangle\}$, where $|D|$ is the size of D.

In this paper, we consider employing SAX representation to reduce the feature space as well as the impact of noise. The definition of the SAX word is given below.

Definition 4 (SAX word): A SAX word is a sequence of characters, denoted as $\hat{T} = \langle \hat{t}_1, \hat{t}_2, \cdots, \hat{t}_\phi \rangle$, that mapped from $\overline{T} = \langle \bar{t}_1, \bar{t}_2, \cdots, \bar{t}_\phi \rangle$ with an alphabet $\Sigma = \{A_1, A_2, \cdots, A_{|\Sigma|}\}$ by the breakpoints of values following the Gaussian distribution. The symbol $\phi = \lceil m/\omega \rceil$ denotes the length of the SAX word where m is the length of the raw time series and ω is the window size that determines the length of the SAX word. The element in $\overline{T}$ can be calculated by formula (1)

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

Example 1: Assume a time series $T = \langle 0.12, 0.28, -0.06, -0.32, -0.94, -1.34, 1.32, 0.96, 0.43, 0.57 \rangle$, $\omega = 2$ and $\Sigma = \{A, B, C, D\}$. It is not difficult to obtain $\overline{T} = \langle 0.20, -0.19, -1.14, 1.14, 0.50 \rangle$. According to the Gaussian distribution based split point in (Lin, et al., 2007), the obtained $\overline{T}$ can be transformed into a SAX word $\hat{T} = \langle C, B, A, D, C \rangle$, or represented as $\hat{T} = \text{"}CBADC\text{"}$ for simplicity.

Furthermore, we use $\hat{T}_{i,k,l}$ to represent the sub-word (i.e. subsequence) of $\hat{T}_i$ whose starting position is k and length is l, and use $\widehat{D}$ to denote a SAX-represented time series dataset which is converted from D.

Definition 5 (Distance between two SAX words with equal length): Given two SAX words $\hat{T}_1 = \langle \hat{t}_{1,1}, \hat{t}_{1,2}, \cdots, \hat{t}_{1,m} \rangle$ and $\hat{T}_2 = \langle \hat{t}_{2,1}, \hat{t}_{2,2}, \cdots, \hat{t}_{2,m} \rangle$. The distance between the two SAX words is

denoted as:

$$Wdist\left(\hat{T}_1,\hat{T}_2\right)=\sqrt{\frac{1}{m}\sum_{i=1}^{m}dist\left(\hat{t}_{1,i},\hat{t}_{2,i}\right)^2} \tag{2}$$

The $dist()$ is a function that returns the distance of two characters. For example, Lin et al. proposed a distance function which is implemented as a lookup table (Lin, et al., 2007).

Definition 6 (Distance between two SAX words with different length): Given two SAX words $\hat{T}_1=\left\langle \hat{t}_{1,1},\hat{t}_{1,2},\cdots,\hat{t}_{1,m_1}\right\rangle$ and $\hat{T}_2=\left\langle \hat{t}_{2,1},\hat{t}_{2,2},\cdots,\hat{t}_{2,m_2}\right\rangle$. Without loss of generality, we assume $m_1<m_2$. Then, the distance between the two SAX words is denoted as:

$$Wdist\left(\hat{T}_1,\hat{T}_2\right)=\min_{j=0\ldots m_2-m_1}\sqrt{\frac{1}{m_1}\sum_{i=1}^{m_1}dist\left(\hat{t}_{1,i},\hat{t}_{2,j+i}\right)^2} \tag{3}$$

Example 2: Given two SAX words $\hat{T}_1=\langle C,B,A,D,C\rangle$, $\hat{T}_2=\langle A,D,C,C,D,A,C,D\rangle$, and $\Sigma=\{A,B,C,D\}$. We can easily find that the sub-word $\langle C,D,A,C,D\rangle$ in $\hat{T}_2$ is the closest segment to $\hat{T}_1$, thus their distance is $Wdist\left(\hat{T}_1,\hat{T}_2\right)=0.300$ (based on Lin's method).

Definition 7 (Shapelet): A shapelet S is a subsequence of the (raw or SAX-represented) time series which is representative for a class.

Early studies regard that a shapelet as a time series subsequence that is maximally representative for a class. Their aim is to find the unique shapelet which has the biggest discriminating power. However, most of the time, a unique shapelet cannot accurately describe a class (or known as a concept) and it usually requires a set of shapelets. Therefore, we relax the definition of shapelet, i.e. an arbitrary subsequence can be regarded as a shapelet.

We use information gain to evaluate the discriminating power of a shapelet S. Given a time series dataset D and a shapelet S, we first calculate the distances between S and each time series in D, and sort them. Then, the average between any two consecutive distances is regarded as a possible split point (denoted as τ). For each τ, it divides the dataset D into two disjoint subsets D_A and D_B. Next, we give the definition of information gain.

Definition 8 (Information Gain): Given a time series dataset D with n instances and a shapelet S, the information gain of S is denoted as:

$$IG_D(S)=\max_{\tau}\left\{E(D)-\left(\frac{|D_A|}{|D|}E(D_A)+\frac{|D_B|}{|D|}E(D_B)\right)\right\} \tag{4}$$

Where $E(D)$ is the entropy of D, denoted as:

$$E(D)=-\sum_{i=1}^{|C|}p_i\log p_i \tag{5}$$

The symbols introduced above are summarized in table 1.

Table 1. Summary of symbols

Symbol	Description
$\Gamma = \{T_1, T_2, \cdots, T_n\}$	A finite set of time series T, n is the number of time series
$T_{i,k,l}$	Subsequence of a time series T_i, k is the beginning position and l is the length
$\hat{T}$, $\hat{T}_{i,k,l}$	$\hat{T}$ and $\hat{T}_{i,k,l}$ are the SAX words converted by T and $T_{i,k,l}$ respectively
$C = \{c_1, c_2, \cdots, c_{\|C\|}\}$	A finite set of labels
D	Time series dataset having a finite set of instances $\langle T_i, c_i \rangle$, where $c_i \in C$
$\hat{D}$	Time series dataset having a finite set of SAX-transformed instances $\langle \hat{T}_i, c_i \rangle$
$Wdist(\hat{T}_1, \hat{T}_2)$	Distance between two SAX words
$\sum$, $\|\Sigma\|$	$\sum$ is the alphabet of SAX, and $\|\Sigma\|$ is the size of $\sum$
ω	Window size that determines the length of the SAX word
$\Omega = \{S_1, S_2, \cdots, S_{\|C\|}\}$	$S_i = \{S_{i,1}, S_{i,2}, \cdots\}$ is a pool of shapelets extracted from the time series of c_i
$IG_D(S)$	Information gain of shapelet S on D

4. Compressed random shapelet forest algorithm

4.1 Analysis of the gRSF algorithm

In this section, we first recall the gRSF algorithm. The pseudo code of gRSF is given in algorithm 1.1 and 1.2. The former is the framework of gRSF which aims to generate r shapelet-based decision trees. For simplicity, we use a decision tree in replace of the shapelet-based decision tree in the rest of the paper. The latter shows us how to build a decision tree. In algorithm 1.2, step 4-5 randomly sample sn shapelets and step 6 chooses the best one from these shapelets for growing the current tree node. Then, it divides the dataset D into two subsets, i.e. D_L and D_R, based on the selected shapelet. The obtained two datasets are used to recursively grow the left sub-tree and the right sub-tree respectively.

The weak points of gRSF are obvious. Firstly, the shapelets for growing each tree node are sampled online from the dataset without any constricts. In another word, it cannot guarantee the quality of the selected shapelets unless it increases the number of sampling times. However, gRSF requires to measure the quality of all sampled shapelets (i.e. calculate information gain of the shapelet on the current dataset) and then chooses the best one to generate the current tree node. This raises a dilemma. Furthermore, it is not difficult for us to calculate out that the worst time complexity of gRSF is $O(r \cdot sn \cdot n^2 \cdot m^2)$. To improve the performance of random shapelet forest, we consider compressing the feature space of the shapelets and build a pool of high-quality shapelets for growing decision trees.

Algorithm 1.1: $gRSF(D, r, l, u, N)$

Input: D : a training data set, r : number of trees, l : lower bound of shapelet length, u : upper bound of shapelet length, sn : number of shapelets

Output: $\Psi = \{ST_1, ST_2, \cdots, ST_r\}$: an ensemble of random shapelet trees

$\Psi \leftarrow \varnothing$;
for j = 1 to r **do**
  $D_j \leftarrow sample(D)$;
  $ST_j \leftarrow randomShapeletTree(D_j, l, u, sn)$;
  $\Psi \leftarrow \Psi \bigcup ST_j$;
return Ψ;

Algorithm 1.2: $randomShapeletTree(D, l, u, sn)$

Input: D : a training data set, l : lower bound of shapelet length, u : upper bound of shapelet length, sn : number of shapelets

Output: ST : a randomized shapelet tree

if $isLeaf(D)$ **then**
  return $makeLeaf(D)$;
$\Omega \leftarrow \varnothing$;
for i = 1 to sn **do**
  $\Omega \leftarrow \Omega \bigcup sampleShapelet(D, rand(l, u))$;
$(\tau, S) \leftarrow bestSplit(D, \Omega)$;
$(D_L, D_R) \leftarrow distribute(D, S, \tau)$;
$ST_L \leftarrow randomShapeletTree(D_L, l, u, sn)$;
$ST_R \leftarrow randomShapeletTree(D_R, l, u, sn)$;
return $ST \leftarrow \{ST_L, ST_R, S, \tau\}$;

4.2 The CRSF algorithm

4.2.1 Framework of the CRSF

In this section, we begin to introduce the CRSF algorithm. The pseudo code of CRSF's framework is given in algorithm 2.1. Step 1 is in charge of the transformation of the raw time series dataset D into a SAX-represented dataset $\widehat{D}$ according to parameters ω and Σ, which denote the window size of SAX and the alphabet list respectively. Step 3 invokes the $generateShapeletPool$ function to generate sn shapelets based on $\widehat{D}$. From step 4 to 7, the algorithm invokes the function $createRandomShapeletTree$ circularly to create r decision trees that consist of the random shapelet forest Ψ. The variable SP is used to remember the shapelets which have been used in a decision tree.

Compared with gRSF, it is easy to find that there are several innovations (or differences). Firstly, the dataset is converted into a SAX-represented dataset whose aim is to compress the data and eliminate

noise. By this way, the space of the shapelet can be significantly reduced. Secondly, a pool of high-quality shapelets is generated before training the decision trees, and all the decision trees are built upon the shapelet pool. Thirdly, we propose a new distance function between two SAX words to replace the method proposed by Lin et al. (Lin, et al., 2007). The distance function will be presented in 4.2.2. Next, we explain the details of the CRSF algorithm.

Algorithm 2.1: $CRSF\left(D,r,l,u,sn,\omega,\Sigma,C\right)$

Input: D : a training data set, r : number of trees, l : lower bound of shapelet length, u : upper bound of shapelet length, sn : number of shapelets, ω : SAX window size, Σ : alphabet of SAX representation, C : label set

Output: $\Psi=\{ST_1,ST_2,\cdots,ST_r\}$: an ensemble of random shapelet trees

1 $\widehat{D}\leftarrow convert\left(D,\omega,\Sigma\right)$;
2 $\Psi\leftarrow\varnothing$;
3 $\Omega\leftarrow generateShapeletPool\left(\widehat{D},l,u,sn,C\right)$;
4 **for** j = 1 to r **do**
5 $\quad SP\leftarrow\varnothing$;
6 $\quad ST_j\leftarrow createRandomShapeletTree\left(\widehat{D}_j,\Omega,SP\right)$;
7 $\quad \Psi\leftarrow\Psi\bigcup ST_j$;
8 **return** Ψ;

4.2.2 SAX transformation and a new distance function

The transformation of a time series to a SAX word in our work is similar to (Lin, et al., 2007). At first, a time series is cut into $\lceil m/\omega\rceil$ segments where m represents the length of the time series. Then, we calculate the average value of each segment by formula (1) (see section 3). Finally, the obtained compressed data sequence (also known as PAA sequence) are mapped into a SAX word based on an alphabet list Σ. A toy example has been given in section 3. To determine the parameters ω and Σ, we perform a grid-search over the two parameters as well as a 5-fold cross-validation experiment. The details will be shown in section 5.

An important issue in TSC is to measure the similarity (i.e. distance) of two time series. For SAX-represented time series data, Lin et al. have proposed a distance function for SAX words with the same length which represented as a lookup table. The method is shown in formula (6).

$$Wdist_{Lin}\left(\widehat{T}_1,\widehat{T}_2\right)=\sqrt{\frac{1}{m}\sum_{i=1}^{m}cell\left(\hat{t}_{1,i},\hat{t}_{2,i}\right)^2}$$
$$cell\left(\hat{t}_1,\hat{t}_2\right)=\begin{cases}0 & \left\|\hat{t}_1-\hat{t}_2\right\|\le 1\\ \beta_{\max\left(\hat{t}_1,\hat{t}_2\right)-1}-\beta_{\min\left(\hat{t}_1,\hat{t}_2\right)} & \left\|\hat{t}_1-\hat{t}_2\right\|>1\end{cases} \quad (6)$$

Where symbol $\|\ \|$ indicates the position difference of two characters in the alphabet, and β

refers to the split point used in SAX transformation. More details can be found in (Lin, et al., 2007). We can find that the distance between two characters which are adjacent in the alphabet is set to zero in the function. This is for robustness, however it often leads to bad predictive results. For example, assume two SAX words $\hat{T}_1 = "BBBBB"$, $\hat{T}_2 = "ACBAC"$ which are obviously different. According to Lin's method, we can get $Wdist_{Lin}\left(\hat{T}_1, \hat{T}_2\right) = 0$. It is obviously unreasonable. To solve this problem, we propose a new distance function, which is shown in formula (7).

$$Wdist\left(\hat{T}_1, \hat{T}_2\right) = \begin{cases} Wdist_{Lin}\left(\hat{T}_1, \hat{T}_2\right) & \max_i\left(\left\|\hat{t}_{1,i} - \hat{t}_{2,i}\right\|\right) > 1 \\ \sqrt{\frac{\sum_{i=1}^{m}\left\|\hat{t}_{1,i} - \hat{t}_{2,i}\right\|}{m^2} \cdot cell\left(A,C\right)^2} & \text{otherwise} \end{cases} \tag{7}$$

Where $cell\left(A,C\right)$ is the distance of two characters whose position difference is 2, just as 'A' and 'C'. $\sum_{i=1}^{m}\left\|\hat{t}_{1,i} - \hat{t}_{2,i}\right\|$ refers to the number of characters that their position difference is 1 in two SAX words. Continue the above example, we can get $Wdist\left(\hat{T}_1, \hat{T}_2\right) = \sqrt{0.67^2 \times \left(4/25\right)} = 0.268$. Similarly, let $\hat{T}_3 = "ACBBC"$, we can get $Wdist(\hat{T}_1, \hat{T}_3) = \sqrt{0.67^2 \times \left(3/25\right)} = 0.232$. It demonstrates that, compared to $\hat{T}_2$, $\hat{T}_3$ is closer to $\hat{T}_1$. Furthermore, consider $\hat{T}_4 = "DBBBB"$ and the distance between $\hat{T}_1$ and $\hat{T}_4$ is $Wdist\left(\hat{T}_1, \hat{T}_4\right) = \sqrt{0.67^2/5} = 0.300$, it de demonstrates that $\hat{T}_2$ is closer to $\hat{T}_1$ than $\hat{T}_4$. We think the new distance function is more reasonable than Lin's method.

4.2.3 Generation of a pool of shapelet candidates

This section introduces the function $generateShapeletPool$ whose aim is to generate a pool of high-quality shapelets. Before that, we give an illustrating example to help understand the characteristics of a high-quality shapelet. In figure 1, there are four time series from the *ECG200* dataset (the indices of them are 8, 37, 50, and 93 respectively). Let $\omega = 8$, $\Sigma = \{A, B, C, D, E, F\}$, the raw time series can be transformed into four SAX words, $\hat{T}_8 = "FCBAACEEEEDD"$, $\hat{T}_{37} = "EAAAEEEDECCE"$, $\hat{T}_{50} = "FBAAEEDDDEED"$ and $\hat{T}_{93} = "FCAAAEEEDDCC"$. The SAX words $\hat{T}_8, \hat{T}_{50}$ belong to class 1, and the other two belong to class 2. Assume that we extract a shapelet $S_1 = "EED"$, it is easy to find that it appears in all the time series. It means that S_1 has no discriminating power and can not provide any useful knowledge for classification. Thus, such a shapelet should be removed from the pool. Assume another shapelet $S_2 = "BAA"$ extracted from $\hat{T}_8$, we can find that it does not appear in the other class. It means that the shapelet may be helpful for classification. Therefore, such a shapelet should be kept. If two identical shapelets are extracted from the same class, one of them should be discarded to avoid redundancy.

Through the analysis, we can conclude that a high-quality shapelet should be exclusive and non-self-similar. The former means the shapelet should only appear in its own class. The latter means a shapelet should not be same as other shapelets extracted from the same class. Li et al. (Guozhong Li, et al., 2021) employ the Bloom filter, which is a very efficient technique for information retrieve, to query whether a shapelet exists in other classes or not. However, they do not consider the self-similar shapelets. In this work, we propose a two-step strategy to obtain high-quality shapelets. First, we also

employ a Bloom filter $bloomFilter(C,S)$ to judge whether the shapelet S exists in other classes C or not. Then, we exploit a simple rule to filter out the self-similar shapelets. We say two shapelets are self-similar if (1) they are entirely identical and from the same class, (2) they are overlapping and from the same time series. In this work, the overlap is determined by formula (9), where $startPos$ and $endPos$ denote the starting position and the ending position of the shapelets in the time series, λ is a threshold to control the overlapping ratio and it is set to 0.9 in this work.

$$\frac{\min(endPos_1, endPos_2) - \max(startPos_1, startPos_2)}{\max(endPos_1, endPos_2) - \min(startPos_1, startPos_2)} \geq \lambda \tag{9}$$

The pseudo code of the $generateShapeletPool$ function is shown in algorithm 2.2. The variable $SP[]$ is used to store the shapelets of different classes. Step 4 initializes a set of Bloom filters B_C. Step 5-15 aims to generate $sn \times |C|$ shapelets circularly, where $|C|$ is the number of classes. The first three steps of the loop randomly sample a shapelet S, and then get its class label c as well as the corresponding complementary set C'. In step 9-15, the algorithm firstly checks whether S appears in other classes C' by the Bloom filters. If so, it removes S from $SP[C']$; otherwise, it continues to check whether S is self-similar with other shapelets in class c. If not, it adds S to $SP[c]$ and updates the Bloom filter. Step 16-17 collect all the shapelets to build the pool Ω.

In fact, we also considered sampling shapelets under multiple SAX configurations. For example, we try to randomly select a pair of SAX parameters $(\omega, |\Sigma|)$ from predefined SAX configurations $\{\{\omega = 2, |\Sigma| = 4\}, \{\omega = 4, |\Sigma| = 6\}, \cdots\}$, and then sample a shapelet S from the corresponding SAX dataset. However, experimental results show that sampling shapelets under multiple SAX configurations is not superior to sampling under a single SAX configuration.

Algorithm 2.2: $generateShapeletPool(\widehat{D}, l, u, sn, C)$

Input: $\widehat{D}$: a SAX-represented training data set, l : lower bound of shapelet length, u : upper bound of shapelet length, sn : number of shapelets, C : label set

Output: Ω : a pool of shapelets

1 $\Omega \leftarrow \varnothing$;
2 $SP[] \leftarrow \varnothing$;
3 $k \leftarrow 0$;
4 $InitializeBloomFilter(\mathrm{B}_C)$;
5 **while** $k < sn \times |C|$ **do**
6 $\quad S \leftarrow sampleShapelet(\widehat{D}, rand(l,u))$;
7 $\quad c \leftarrow typeOf(S)$;
8 $\quad C' \leftarrow C \setminus c$;
9 $\quad$ **if** $bloomFilter(\mathrm{B}_C, C', S)$ **do**
10 $\quad\quad cnt \leftarrow removeShapelet(SP[C'], S)$;
11 $\quad\quad k \leftarrow k - cnt$;

12 | **else if** $!isSelfSemilar\left(SP[c], S\right)$ **do**

13 | | $SP[c] \leftarrow SP[c] \cup S$;

14 | | $updateBloomFilter\left(\mathrm{B}_C, S, c\right)$;

15 | | $k \leftarrow k+1$;

16 **for** i = 1 to $|C|$ **do**

17 | $\Omega \leftarrow \Omega \cup SP[i]$;

18 **return** Ω ;

4.2.4 Training a shapelet-based decision tree

This section introduces the function $createRandomShapeletTree$ whose aim is to generate a shapelet-based decision tree based on a pre-learned shapelet pool. The pseudo code is given in algorithm 2.3. The logic in the algorithm is simple. In step 3, the algorithm randomly selects a specific shapelet S from $\Omega \setminus SP$ firstly. That means a shapelet that has been used in the decision tree can not be selected again. Here, we propose a specific strategy of roulette wheel selection in which the probability of a shapelet being selected is calculated by formula (10). The variable $\#unselected_S$ is in charge of counting the number of times that the shapelet S is not selected during the training of the random forest. The variable would be incremented by 1 if the corresponding shapelet is not selected when growing a tree node. The rationale behind the consideration is that the fewer times a shapelet is selected, the greater the probability that it will be selected next time.

$$prob(S) = \frac{\#unselected_S}{\sum_{S' \in \Omega} \#unselected_{S'}} \tag{10}$$

We can find that gRSF, essentially, employs a tournament selection strategy, i.e. it firstly samples a number of shapelet candidates, and then chooses the best one for the current tree node. Compared with gRSF, our algorithm only selects one shapelet for each tree node, therefore it can reduce considerable time costs. In steps 5-6, the algorithm calculates the distances between S and $\widehat{T}_i \in \widehat{D}$. After that, it finds the best split point τ and divides the original data set $\widehat{D}$ into two disjoint subsets $\widehat{D}_L, \widehat{D}_R$. The principle of this step can be found in (Ye & Keogh, 2009). Finally, the shapelet S is added into SP, and the algorithm recursively creates the left sub-tree and the right sub-tree respectively.

Algorithm 2.3: $createRandomShapeletTree\left(\widehat{D}, \Omega, SP\right)$

Input: $\widehat{D}$: a SAX-represented training data set, Ω : a pool of shapelet candidates, SP : a set storing the selected shapelets in the shapelet tree

Output: ST : a randomized shapelet tree

1 **if** $isLeaf\left(\widehat{D}\right)$ **then**

2 | **return** $makeLeaf\left(\widehat{D}\right)$;

3 $S \leftarrow randomSelect\left(\Omega, SP\right)$;

4 $ds[] \leftarrow 0$;

5 **for** i = 1 to $\left|\widehat{D}\right|$ **do**

6 $\quad ds[i] = Wdist\left(\widehat{T}_i, S\right)$;

7 $\left(\tau, \widehat{D}_L, \widehat{D}_R\right) \leftarrow splitDataset\left(\widehat{D}, ds\right)$;

8 $SP \leftarrow SP \bigcup S$;

9 $ST_L \leftarrow createRandomShapeletTree\left(\widehat{D}_L, \Omega, SP\right)$;

10 $ST_R \leftarrow createRandomShapeletTree\left(\widehat{D}_R, \Omega, SP\right)$;

11 **Return** $ST \leftarrow \left\{ST_L, ST_R, S, \tau\right\}$;

4.3 Time complexity

The time cost of CRSF is composed of two parts, i.e. generation of a pool of shapelets and training a set of decision trees. The time complexity of the former part is uncertain, but it ideally far less than the second part. Therefore, we only focus on the analysis of the time complexity of the latter part. For each tree node, the algorithm requires to compute the distances between S and $\widehat{T}_i \in \widehat{D}$, and then it sorts the distances and finds the best split point among all possible splitting points. According to (Karlsson, et al., 2016), the time complexity of generating a tree node is $O\left(n \times m^2 / \omega^2\right)$, where n denotes the number of time series in the current node, m is the length of the raw time series and ω is the SAX window size. Based on the analysis, it is not difficult to obtain that the worst time complexity of CRSF is $O\left(rn^2m^2/\omega^2\right)$. It is far less than the worst time complexity of gRSF which is $O\left(r \cdot sn \cdot n^2 \cdot m^2\right)$.

5. Experiments

In this section, we perform extensive experiments for evaluating the predictive performance of the CRSF algorithm. Firstly, the datasets and experimental design will be introduced. Secondly, we explore the parameter setting strategies, including the number of decision trees, the size of shapelet pool, etc. Thirdly, we prove the effectiveness of techniques in the proposed algorithm. After that, we evaluate the predictive performance of CRSF by comparing it with six shapelet-based TSC algorithms as well as 1NN+DTW which is claimed to be difficult to defeat. Moreover, we evaluate the efficiency of our algorithm. Finally, we go through the experimental results again and give an in-depth analysis.

5.1 Environment and datasets

The proposed algorithm is coded in Java. The part that generates decision trees is implemented using multi-threading, and the rest part is single-threaded. All the experiments were performed on a server equipped with Intel Xeon Bronze 3106 (1.7GHz) * 2 and 128GB RAM, running on Windows Server 2016 (64 bit).

A well-known benchmark in the field of TSC, called the UCR archive (Dau, et al., 2019), was employed in the experiments. It provides 128 time series datasets from different fields, such as sensor, ECG, image, motion trajectory, etc. Due to page limitation, we selected 50 datasets and the descriptions are given in Table 2, where Train, Test, Class, and Length columns are the numbers of time series in the

training set and the testing set, the number of classes, and the length of time series, respectively. The datasets are carefully selected because (1) they cover most of the domains in the archive, (2) they have different characteristics, for example, the minimal value in the column of “Train” is 20 but the maximal value is 8926; the minimal value in the column of “Length” is 15 but the maximal value is 2709.

For fairness, all the experiments were performed 20 times on each data set because the proposed algorithm as well as some algorithms involved in comparison are stochastic. We calculated the average value and reported them in the paper. Moreover, in order to limit the experimental time, we set a threshold of running time to 2 hours, i.e. the algorithms would be terminated if their running time is over 2 hours on a dataset. The code of CRSF and the experimental results can be found in http://tsdm.lsnu.edu.cn/crsf.

Table 2. Description of the datasets

Domain	Name	Abbreviation	Train	Test	Class	Length
Image	ArrowHead	ArrowHead	36	175	3	251
Image	BeetleFly	BeetleFly	20	20	2	512
Image	BirdChicken	BirdChicken	20	20	2	512
Simulated	CBF	CBF	30	900	3	128
Sensor	ChlorineConcentration	ChlConcen	467	3840	3	166
Spectro	Coffee	Coffee	28	28	2	286
Image	DiatomSizeReduction	DiatomSizRed	16	306	4	345
Image	DistalPhalanxOutlineAgeGroup	DistPhaOutAgGrp	400	139	3	80
Image	DistalPhalanxOutlineCorrect	DistalPhaOutCor	600	276	2	80
Sensor	Earthquakes	Earthquakes	322	139	2	512
ECG	ECG200	ECG200	100	100	2	96
ECG	ECGFiveDays	ECGFiveDays	23	861	2	136
Device	ElectricDevices	ElectricDev	8926	7711	7	96
Image	FaceAll	FaceAll	560	1690	14	131
Image	FaceFour	FaceFour	24	88	4	350
Image	FacesUCR	FacesUCR	200	2050	14	131
Sensor	FordA	FordA	3601	1320	2	500
Spectro	Ham	Ham	109	105	2	431
Image	HandOutlines	HandOutlines	1000	370	2	2709
Motion	Haptics	Haptics	155	308	5	1092
Image	Herring	Herring	64	64	2	512
Motion	InlineSkate	InlineSkate	100	550	7	1882
Sensor	InsectWingbeatSound	InsectWingSnd	220	1980	11	256
Device	LargeKitchenAppliances	LargeKitchenApp	375	375	3	720
Sensor	Lightning2	Lightning2	60	61	2	637
Sensor	Lightning7	Lightning7	70	73	7	319
Simulated	Mallat	Mallat	55	2345	8	1024
Spectro	Meat	Meat	60	60	3	448
Image	MiddlePhalanxOutlineAgeGroup	MidPhaOutAgeGrp	400	154	3	80

Image	MiddlePhalanxTW	MiddlePhalanxTW	399	154	6	80
Sensor	Phoneme	Phoneme	214	1896	39	1024
Image	ProximalPhalanxOutlineAgeGroup	ProPhaOutAgeGrp	400	205	3	80
Device	RefrigerationDevices	RefrigDev	375	375	3	720
Simulated	ShapeletSim	ShapeletSim	20	180	2	500
Image	ShapesAll	ShapesAll	600	600	60	512
Sensor	SonyAIBORobotSurface2	SonyAIBORoSur2	27	953	2	65
Spectro	Strawberry	Strawberry	613	370	2	235
Image	Symbols	Symbols	25	995	6	398
Simulated	SyntheticControl	SyntheticContrl	300	300	6	60
Motion	ToeSegmentation1	ToeSegmentat1	40	228	2	277
Sensor	Trace	Trace	100	100	4	275
ECG	TwoLeadECG	TwoLeadECG	23	1139	2	82
Simulated	TwoPatterns	TwoPatterns	1000	4000	4	128
Motion	UWaveGestureLibraryY	UWaveGestLibY	896	3582	8	315
Sensor	Wafer	Wafer	1000	6164	2	152
Spectro	Wine	Wine	57	54	2	234
Motion	WormsTwoClass	WormsTwoClass	181	77	2	900
Image	Yoga	Yoga	300	3000	2	426
HRM	Fungi	Fungi	18	186	18	201
Simulated	SmoothSubspace	SmoothSubspace	150	150	3	15

5.2 Parameters setting

In this section, we conduct experiments to explore how to set the parameters of CRSF, including the parameters of SAX, i.e. ω and $|\Sigma|$, the size of the shapelet pool sn and the number of decision trees r. To shorten the time, the experiments were conducted on 15 datasets which were carefully selected from the 50 datasets. The selected datasets are ArrowHead, BirdChicken, CBF, Earthquakes, ECG200, ElectricDev, Haptics, InlineSkate, LargeKitchenApp, Lightning7, Mallat, ShapesAll, Symbols, Wine and Fungi. For each parameter, we preset the scope and searched for the optimal parameter setting by 5-fold cross-validation experiment.

5.2.1 SAX configuration

Firstly, we test different strategies of SAX parameters setting. Besides, we wish to figure out that whether sampling shapelet on multiple SAX configurations is helpful to improve the performance of the algorithm. We designed three strategies, which are sampling on a single, or two, or three SAX configuration(s) respectively. According to the findings in (Grabocka, et al., 2015; Lin, et al., 2007), the scopes of parameter ω and $|\Sigma|$ were set to (1, 2, 4, 8, 16) and (4, 6, 8, 10). The optimal SAX configuration (or composite SAX configurations) was achieved by grid search. Both of the size of shapelet pool and the number of decision trees were set to 100. The experimental results are shown in table 3.

In table 3, the first column are the used datasets. The column “SAX” shows the optimal SAX

configuration of the CRSF algorithm under the corresponding strategy. “SAX 1”, “SAX 2” and “SAX 3” mean sampling on a single SAX configuration, two SAX configurations and three SAX configurations respectively. The column “Accu.” shows the average accuracy as well as the ranking obtained by CRSF on the corresponding SAX configuration. For example, the “SAX 2” column of the first row means that CRSF achieved the highest accuracy when it samples on the SAX configurations (2, 10) and (4, 8). The obtained average accuracy is 0.723 and its ranking is 2 among three strategies. The last row in the table shows the average ranking obtained by CRSF under different strategies of SAX configuration. The highest accuracy as well as its ranking in each row is marked in bold.

We can see that sampling on a single SAX configuration is slightly better than sampling on multiple SAX configurations. Nevertheless, the former only needs to search for $4\times 5=20$ different SAX configurations but the other two strategies require to search for C_{20}^{2} and C_{20}^{3} different SAX configurations. Even though we restrict the searching scope of the latter two strategies, e.g. do not allow for same ω value in different SAX configurations, the running time of the latter two strategies is still far more than that of the former. This is the reason why the proposed algorithm only samples on a single SAX configuration.

Table 3. Grid search of SAX configurations

Dataset	**Accu. 1**	**SAX 1**	**Accu. 2**	**SAX 2**	**Accu. 3**	**SAX 3**
ArrowHead	0.703(3)	(2,10)	0.723(2)	(2,10),(4,8)	**0.726(1)**	(1,10),(2,6),(4,8)
BirdChicken	**0.925(1)**	(4,8)	0.915(2)	(4,8),(8,6)	0.910(3)	(2,6),(4,4),(8,8)
CBF	**0.990(2)**	(2,8)	**0.990(2)**	(1,8),(2,4)	**0.990(2)**	(1,10),(2,6),(4,6)
Earthquakes	**0.748(2)**	(16,6)	**0.748(2)**	(2,6),(4,10)	**0.748(2)**	(2,10),(4,8),(8,8)
ECG200	**0.847(1)**	(8,6)	0.808(3)	(2,10),(4,10)	0.816(2)	(1,8),(2,10),(4,10)
ElectricDev	**0.724(1)**	(1,6)	0.720(2)	(1,6),(4,6)	0.713(3)	(1,6),(2,6),(4,8)
Haptics	0.464(2)	(8,10)	**0.466(1)**	(4,6),(8,10)	0.458(3)	(2,8),(4,10),(8,10)
InlineSkate	0.383(3)	(2,10)	**0.391(1)**	(2,6),(8,10)	0.385(2)	(2,10),(4,6),(8,6)
LargeKitchenApp	0.818(2)	(2,10)	**0.822(1)**	(2,10),(4,10)	0.818(2)	(2,10),(4,10),(8,4)
Lightning7	**0.752(1)**	(4,10)	0.729(3)	(2,10),(4,10)	0.736(2)	(1,8),(2,4),(4,10)
Mallat	**0.971(1.5)**	(16,8)	0.970(3)	(2,8),(4,8)	**0.971(1.5)**	(2,10),(4,8),(8,6)
ShapesAll	0.835(2)	(2,10)	**0.840(1)**	(2,10),(8,10)	0.824(3)	(2,10),(4,8),(8,6)
Symbols	**0.943(1)**	(8,10)	0.931(3)	(2,4),(4,8)	0.939(2)	(1,4),(2,8),(4,8)
Wine	0.752(3)	(1,10)	**0.782(1)**	(1,10),(2,8)	0.774(2)	(1,10),(2,8),(4,8)
Fungi	0.968(2)	(4,10)	**0.971(1)**	(1,10),(4,8)	0.963(3)	(1,10),(2,10),(4,6)
Ave. Ranking	1.83	-	1.87	-	2.23	-

5.2.2 Size of shapelet pool

From algorithm 2.2, we can find that the size of shapelet pool is $sn\times|C|$, i.e. a coefficient sn multiplies the number of class $|C|$. We fixed the number of the decision tree to 100. The parameters of SAX were set based on the results in table 3. The scope of the parameter sn was set to $\{10,20,30,40,50,60,70,80,100,200\}$. The results are shown in table 4. In each row, we also marked the highest accuracy value in bold. For comparison, we give the total running time of the proposed

algorithm on 15 datasets in last row.

From the table, we can find that the best average ranking is 3.37 obtained when the size of shapelet pool is set to 100. The ranking which is closest to the value is 4.03 which is obtained when the size of shapelet pool is set to 60. Furthermore, their total running time are 44.42 seconds and 47.14 seconds respectively. It demonstrates that the difference between their running time is small. Therefore, we conclude that setting the size of shapelet pool to 100 is a safe choice.

Table 4. Prediction performance based on different shapelets sizes sn

Dataset	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**100**	**200**
ArrowHead	0.676(10)	0.691(9)	0.703(5)	0.707(3)	0.702(7)	0.702(7)	**0.709(1.5)**	0.706(4)	**0.709(1.5)**	0.702(7)
BirdChicken	0.825(10)	0.890(9)	0.910(6)	0.925(4)	0.905(7.5)	0.930(3)	0.915(5)	0.905(7.5)	0.940(2)	**0.950(1)**
CBF	0.980(10)	0.989(7)	0.986(9)	0.991(3)	0.990(4.5)	0.989(7)	0.990(4.5)	0.989(7)	**0.992(1.5)**	**0.992(1.5)**
Earthquakes	0.745(8.5)	0.746(7)	0.737(10)	0.745(8.5)	**0.750(1)**	0.747(6)	0.749(2)	0.748(4)	0.748(4)	0.748(4)
ECG200	0.792(10)	0.807(8)	0.809(7)	0.816(2)	0.810(6)	0.814(3)	0.802(9)	0.812(4)	**0.817(1)**	0.811(5)
ElectricDev	0.719(10)	0.721(8.5)	0.721(8.5)	0.724(5)	0.724(5)	0.724(5)	**0.726(1.5)**	**0.726(1.5)**	0.724(5)	0.724(5)
Haptics	0.428(10)	0.452(6.5)	0.452(6.5)	0.451(8)	0.456(3)	0.455(4.5)	0.450(9)	0.455(4.5)	0.457(2)	**0.469(1)**
InlineSkate	0.366(10)	0.375(9)	0.379(8)	0.381(6.5)	**0.391(1)**	0.386(4)	0.381(6.5)	0.387(3)	0.384(5)	0.388(2)
LargeKitchenApp	0.783(10)	0.797(9)	0.815(6)	0.810(8)	0.814(7)	0.822(2.5)	0.822(2.5)	**0.823(1)**	0.821(4.5)	0.821(4.5)
Lightning7	0.719(10)	0.743(8)	0.744(6)	0.749(4)	**0.756(1)**	0.753(2.5)	0.744(6)	0.738(9)	0.744(6)	0.753(2.5)
Mallat	0.964(10)	0.968(7)	0.968(7)	0.968(7)	0.969(4.5)	**0.970(2)**	0.966(9)	**0.970(2)**	0.969(4.5)	**0.970(2)**
ShapesAll	0.829(10)	0.836(3)	0.831(8)	0.832(7)	0.837(2)	0.834(4.5)	0.834(4.5)	0.833(6)	**0.838(1)**	0.830(9)
Symbols	0.917(10)	0.936(6.5)	0.930(9)	0.935(8)	0.938(3)	**0.940(1)**	0.938(3)	0.937(5)	0.938(3)	0.936(6.5)
Wine	0.694(10)	0.720(9)	0.722(8)	0.756(4)	0.741(7)	0.750(5)	0.746(6)	**0.772(1.5)**	0.767(3)	**0.772(1.5)**
Fungi	**0.975(1)**	0.968(5)	0.966(6.5)	0.963(9)	0.970(3.5)	0.970(3.5)	0.957(10)	0.971(2)	0.966(6.5)	0.965(8)
Ave. ranking	9.30	7.43	7.37	5.80	4.20	4.03	5.33	4.13	**3.37**	4.04
Total time(s)	28.14	32.41	36.65	41.18	43.16	44.42	46.04	45.75	47.14	48.44

5.2.2 Number of decision tree

In this section, we explore how to set the number of decision tree r in the CRSF algorithm. We set the size of shapelet pool to 100 according to the result in previous section. The scope of the parameter r was set to $\{50,100,150,200,250,300,400,500\}$. The experimental results are shown in table 5. It is easy to find that CRSF achieves the highest average ranking when the parameter r is set to 300. The second and third highest average ranking is 3.17 ($r=500$) and 3.80 ($r=400$) respectively. The latter is significantly inferior to the gold medal winner ($r=300$). Furthermore, when the parameter r is set to 300, the total running time of the proposed algorithm on 15 datasets is 65.55 seconds which is less than 86.97 seconds which is the total running time of the algorithm when the parameter r is set to 500. Therefore, we fix the number of the decision trees to 300 in CRSF.

Table 5. Prediction performance based on different size of decision tree

Dataset	**50**	**100**	**150**	**200**	**250**	**300**	**400**	**500**
ArrowHead	0.701(8)	0.705(7)	0.725(2)	0.714(4.5)	0.723(3)	0.711(6)	**0.726(1)**	0.714(4.5)
BirdChicken	0.910(8)	0.930(6)	0.955(4)	0.945(5)	0.915(7)	**0.965(2)**	**0.965(2)**	**0.965(2)**
CBF	0.987(8)	0.992(4)	0.992(4)	0.992(4)	0.991(7)	**0.993(1)**	0.992(4)	0.992(4)

Earthquakes	0.748(4.5)	**0.749(1)**	0.748(4.5)	0.748(4.5)	0.748(4.5)	0.747(8)	0.748(4.5)	0.748(4.5)
ECG200	0.828(2)	0.809(8)	0.811(7)	0.820(3)	**0.832(1)**	0.816(5)	0.812(6)	0.818(4)
ElectricDev	0.719(8)	0.722(7)	0.726(5.5)	0.727(4)	0.726(5.5)	0.728(2.5)	0.728(2.5)	**0.729(1)**
Haptics	0.438(8)	0.458(7)	0.463(6)	**0.473(2)**	**0.473(2)**	**0.473(2)**	0.468(5)	0.471(4)
InlineSkate	0.383(7)	0.389(4)	0.386(5)	0.384(6)	0.382(8)	0.391(3)	**0.392(1.5)**	**0.392(1.5)**
LargeKitchenApp	0.809(8)	0.820(7)	0.822(5)	**0.828(1)**	0.825(2)	0.823(3.5)	0.821(6)	0.823(3.5)
Lightning7	0.752(3)	0.738(8)	0.758(2)	0.740(6.5)	0.740(6.5)	0.745(5)	0.748(4)	**0.762(1)**
Mallat	0.968(8)	0.969(7)	0.970(5.5)	**0.972(2)**	0.971(4)	**0.972(2)**	0.970(5.5)	**0.972(2)**
ShapesAll	0.803(8)	0.831(7)	0.840(6)	0.844(5)	0.849(4)	**0.855(1.5)**	0.852(3)	**0.855(1.5)**
Symbols	0.924(8)	0.943(3)	0.943(3)	0.942(5)	**0.944(1)**	0.943(3)	0.940(7)	0.941(6)
Wine	0.724(8)	0.750(7)	0.757(5)	0.759(4)	0.770(3)	**0.778(1)**	0.774(2)	0.754(6)
Fungi	0.941(8)	0.962(6)	0.959(7)	0.973(5)	0.974(4)	**0.977(1)**	0.975(3)	0.976(2)
Ave. ranking	6.97	5.93	4.77	4.10	4.17	**3.10**	3.80	3.17
Total time(s)	31.74	46.02	45.87	56.58	60.48	65.55	72.21	86.97

5.3 Effectiveness of our methods

In this section, we conduct experiments to prove the effectiveness of the techniques used in CRSF, including the strategy of roulette wheel selection which is used to choose a shapelet when growing a tree node, and the new function for measuring distance between two SAX words.

5.3.1 Roulette wheel selection vs. tournament selection

In this section, we prove the effectiveness of the roulette wheel selection strategy which is employed to choose a shapelet when growing a tree node. The method was compared with the tournament selection strategy that firstly samples a number of shapelets as the candidates and then chooses the best one among them to grow the tree node. In the experiments, the number of shapelet candidates was set to 1, 3, 5 and 7 respectively. It is easy to find that the tournament selection degenerates to the roulette wheel selection when the parameter is set to 1. Other parameters were set according to the conclusions obtained before.

The experimental results are shown in table 6. The average accuracy and the average running time of CRSF on each dataset are calculated. Similarly, the results obtained under different strategies on each dataset are ranked. The average ranking as well as the total running time obtained by CRSF under four strategies are given in the last row. From the table, we can find that the average rankings of the four strategies are 2.6, 2.77, 2.43, and 2.20 respectively. The difference of the obtained average ranking among the four strategies is not significant. Therefore, we employ paired t-test to detect whether there exist significant difference among these strategies or not. The formula is given below.

$$|t| = \left| \frac{\overline{d}}{s_D / \sqrt{n}} \right| \geq t_{a/2}(n-1) \tag{11}$$

Where $\overline{d}$ and s_D represent the mean value and the standard deviation respectively. Symbol n is the number of datasets, α is the significance level and it is set to 0.01. Since the CRSF algorithm

uses the first strategy, i.e. selecting one shapelet to grow a tree node, we only calculate the t-test value between the first strategy and other three strategies. The results of the t-test value are 0.7320, 0.0358 and 0.4641 respectively. All the t-test value are lower than $t_{0.005}(14) = 2.9768$. It demonstrates that there is no significant difference between the roulette wheel selection and the tournament selection strategy according to their accuracy on 15 datasets. However, from the last row in table 5, we can find that the total running time when using the roulette wheel selection strategy is far less than that using the tournament selection strategy. In another word, the roulette wheel selection reduces the time cost of the proposed algorithm without deteriorating the accuracy. This proves the effectiveness of the strategy.

Table 6. Prediction performance of different shapelet selection strategies

Dataset	**1**		**3**		**5**		**7**	
	Accu.	**Time(s)**	**Accu.**	**Time(s)**	**Accu.**	**Time(s)**	**Accu.**	**Time(s)**
ArrowHead	**0.723(1)**	0.46	0.688(4)	0.45	0.697(3)	0.46	0.704(2)	0.47
BirdChicken	**0.965(2)**	0.26	0.950(4)	0.27	**0.965(2)**	0.27	**0.965(2)**	0.32
CBF	0.992(4)	0.36	**0.996(1.5)**	0.38	**0.996(1.5)**	0.40	0.995(3)	0.38
Earthquakes	0.748(3)	0.91	0.748(3)	1.20	**0.750(1)**	1.49	0.748(3)	1.77
ECG200	0.847(4)	0.36	0.849(3)	0.38	**0.861(1)**	0.38	0.852(2)	0.42
ElectricDev	0.726(4)	36.35	0.728(2)	109.73	0.727(3)	185.89	**0.729(1)**	266.03
Haptics	**0.471(1)**	1.27	0.470(2)	1.34	0.465(4)	1.40	0.466(3)	1.40
InlineSkate	0.386(4)	12.42	0.407(3)	13.52	0.411(2)	13.24	**0.414(1)**	13.07
LargeKitchenApp	0.822(4)	4.37	0.834(3)	4.32	**0.848(1)**	4.99	0.844(2)	5.59
Lightning7	**0.764(1)**	0.77	0.749(4)	0.81	0.756(2)	0.87	0.751(3)	0.92
Mallat	0.959(4)	0.62	0.963(3)	0.63	0.966(2)	0.62	**0.967(1)**	0.65
ShapesAll	**0.851(1)**	11.18	0.847(2)	12.91	0.839(4)	15.07	0.840(3)	17.62
Symbols	**0.945(1)**	0.57	0.938(3)	0.61	0.935(4)	0.58	0.939(2)	0.59
Wine	**0.759(1)**	0.50	0.750(2)	0.50	0.741(3)	0.51	0.704(4)	0.50
Fungi	0.930(4)	1.10	0.933(2)	1.20	0.931(3)	1.09	**0.937(1)**	1.14
Ave. ranking	2.60	-	2.77	-	2.43	-	2.20	-
Running time(s)	-	71.5	-	148.25	-	227.26	-	310.87

5.3.2 Our distance function vs. Lin's distance function

In this section, we evaluate the effectiveness of the new distance function by comparing it with Lin's distance function (Lin, et al., 2007). We collect the accuracy obtained by the two methods on 15 datasets. The results are shown in a scatter plot (see figure 2). The horizontal ordinate and the vertical ordinate denote the accuracy obtained by Lin's method and our method respectively. From the scatter plot, we can find that our method is slightly better than Lin's method on 7 datasets, and it is far better than Lin's method on 2 datasets. For the rest 6 datasets, the accuracy obtained by the two methods is very close. The results are quite expectable because our method, in fact, is an extension of Lin's method. In most cases, the calculating results of two methods are identical except that two SAX words are close, i.e. $\max_i \left(\left\| \hat{t}_{1,i} - \hat{t}_{2,i} \right\| \right) \le 1$. The results show that our method, in deed, performs much better on two

datasets. This proves its effectiveness.

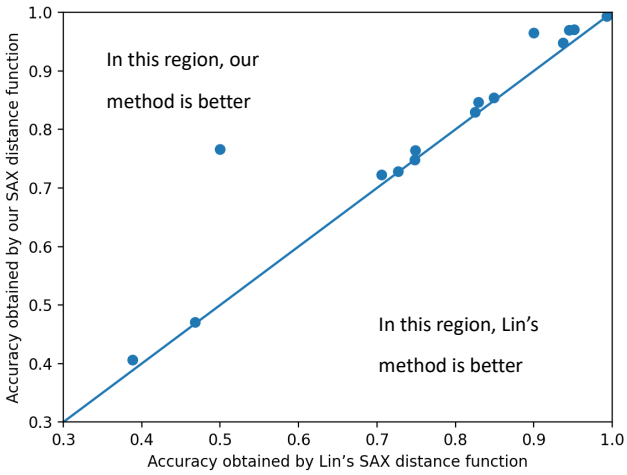

Figure 2. Accuracy comparison between our distance function and Lin's distance function

5.4 Comparison with state-of-the-art algorithms

5.4.1 Baselines

We compared CRSF with seven different classifiers, including 1NN+DTW, and six shapelet-based algorithms. Due to page limitation, we only provide brief details of each method.

(1) gRSF (Karlsson, et al., 2016): This algorithm firstly proposes the random shapelet forest for TSC. The principle of the gRSF algorithm is introduced in section 4.1. The executable jar file was downloaded from https://github.com/isaksamsten/rsf-cmdline.

(2) 1NN+DTW: This algorithm is usually employed as a baseline in most TSC research. Wang et al. (X. Wang, et al., 2013) has proven that it is hard to beat. We used a toolkit *tsml* implemented by Bagnall et al. (Bagnall, et al., 2017) and it can be downloaded from https://github.com/uea-machine-learning/tsml.

(3) Fast Shapelet (FS) (Rakthanmanon & Keogh, 2013): FS is the first shapelet-based TSC algorithm employed SAX representation. The source code was downloaded from the website http://alumni.cs.ucr.edu/~rakthant/FastShapelet/.

(4) Learning Time-series Shapelets (LTS) (Grabocka, et al., 2014): LTS is a classical shapelet-based TSC algorithm using machine learning technique. The LTS algorithm was implemented in *tsml*.

(5) Scalable shapelet Discovery (SD) (Grabocka, et al., 2015): SD is known as one of the fastest shapelet-based TSC algorithms which uses random sampling and online pruning technique to obtain shapelet candidates. The executable jar file was downloaded from http://fs.ismll.de/publicspace/ScalableShapelets.

(6) ELIS (Fang, et al., 2018): ELIS is a state-of-the-art algorithm for TSC which is based on shapelet learning. It first employs PAA and TF-IDF to discover shapelet candidates. Then, the logistic regression

is applied to adjust the shapelets. The source code was downloaded from https://github.com/House1993/ELIS.

(7) BSPCover(Guozhong Li, et al., 2021): BSPCover is a state-of-the-art TSC algorithm. It employs an efficient method to obtain a set of shapelet candidates, and then applies logistic regression classifier to adjust the shapelets. The source code was downloaded from https://goo.gl/sN9Kz7.

The parameters of these algorithms were set according to the corresponding references. If datasets do not appear in the references, we use the default parameters. The detailed parameter settings can be found in http://tsdm.lsnu.edu.cn/crsf. The experiments were conducted on 50 datasets.

5.4.2 Accuracy

The results are shown in table 7. It should be emphasized that most of the algorithms are stochastic, so these algorithms were run 20 times on each dataset. The reported results in table 7 are the average value of the accuracy. Moreover, we calculate the ranking of eight algorithms on each dataset. The best is marked in bold. Besides, it should be explained that short dashes in the table mean that the corresponding algorithm did not return the result within 2 hours on the dataset. In order to describe the experimental results more intuitively, we make some comparisons in the last five rows of the table. The first row shows the average ranking of the algorithm on 50 datasets. From the row, we can find the average ranking of CRSF is 2.75, which is obviously better than the second one, i.e. gRSF whose average ranking is 3.18. The “total best” row shows the number of datasets on which the corresponding algorithm won the gold medal. The last three rows show the number of datasets that CRSF wins, draws, and loses across all datasets compared with the corresponding algorithms, respectively. For example, compared with gRSF, CRSF won 28 times, drew 2 times, lost 20 times on 50 datasets. From the results, it is not difficult to conclude that CRSF is the best classifier among the eight methods.

Next, we apply the Nemenyi test to detect whether there exist significant differences among the eight algorithms. A critical difference diagram is shown in figure 3. Classifiers that are not significantly different at p = 0.05 are connected. From the figure, it clearly shows that CRSF is significantly superior to other five algorithms, which are 1NN+DTW, SD, FS, ELIS and BSPCover. Since the difference among CRSF, gRSF and LTS is not significant, the empirical findings indicate that when aiming for highest accuracy, either CRSF or gRSF or LTS can be safely recommended. However, it is known to us that LTS is based on machine learning, therefore it is more time consuming than other two algorithms based on random sampling. For example, LTS did not finish the classifier training within 2 hours on two Haptics and ShapesAll. Due to this, we prefer to recommend CRSF and gRSF.

Table 7. Accuracy comparisons among 8 classifiers on 50 data sets

Dataset	**gRSF**	**1NN+DTW**	**FS**	**LTS**	**SD**	**ELIS**	**BSPCover**	**CRSF**
ArrowHead	0.714(6)	0.783(4)	0.600(8)	**0.823(1)**	0.699(7)	0.800(3)	0.806(2)	0.723(5)
BeetleFly	0.900(2.5)	0.800(5.5)	0.650(8)	0.850(4)	0.683(7)	0.800(5.5)	0.900(2.5)	**0.960(1)**
BirdChicken	0.800(5.5)	0.900(2.5)	0.750(7)	0.800(5.5)	0.633(8)	0.900(2.5)	0.850(4)	**0.965(1)**
CBF	0.992(3)	0.991(4.5)	0.958(7)	0.991(4.5)	0.981(6)	0.927(8)	**0.997(1)**	0.994(2)

ChlConcen	**0.662(1)**	0.647(2)	0.583(6)	0.585(5)	0.538(7)	0.274(8)	0.612(4)	0.625(3)
Coffee	0.929(6)	**1.000(2.5)**	0.893(7)	**1.000(2.5)**	0.845(8)	0.964(5)	**1.000(2.5)**	**1.000(2.5)**
DiatomSizRed	0.827(8)	**0.980(1.5)**	0.886(6)	**0.980(1.5)**	0.914(3.5)	0.899(5)	0.873(7)	0.914(3.5)
DistPhaOutAgGrp	0.734(2)	0.727(4)	0.669(6)	0.719(5)	0.667(7)	0.734(3)	0.468(8)	**0.736(1)**
DistalPhaOutCor	0.772(5)	0.768(6)	0.779(4)	0.793(2)	0.731(7)	0.721(8)	**0.832(1)**	0.783(3)
Earthquakes	0.748(3)	0.683(6)	0.684(5)	0.741(4)	0.648(7)	**0.776(1)**	-(8)	0.748(2)
ECG200	0.840(4)	0.840(4)	0.770(7)	**0.860(1)**	0.840(4)	0.810(6)	0.640(8)	0.847(2)
ECGFiveDays	0.999(3)	0.700(8)	0.998(4)	**1.000(1.5)**	0.724(7)	0.955(6)	**1.000(1.5)**	0.978(5)
ElectricDev	0.748(3)	0.754(2)	0.549(6)	**0.929(1)**	0.593(5)	0.087(8)	0.242(7)	0.728(4)
FaceAll	0.750(4)	**0.946(1)**	0.624(8)	0.749(5)	0.713(7)	0.756(3)	0.763(2)	0.737(6)
FaceFour	0.875(6)	0.830(7)	0.921(5)	0.966(2.5)	0.805(8)	0.955(4)	0.966(2.5)	**0.975(1)**
FacesUCR	0.905(2)	0.886(3)	0.674(7)	**0.939(1)**	0.849(5)	0.636(8)	0.783(6)	0.872(4)
FordA	0.932(3)	0.586(8)	0.837(6)	0.940(2)	0.848(5)	0.676(7)	**0.963(1)**	0.889(4)
Ham	**0.791(1)**	0.695(5)	0.724(4)	0.638(6.5)	0.575(8)	0.638(6.5)	0.762(2)	0.745(3)
HandOutlines	0.905(3)	0.838(6)	0.876(4)	**0.954(1)**	0.862(5)	-(7.5)	-(7.5)	0.920(2)
Haptics	**0.471(1.5)**	0.416(4.5)	0.351(7)	-(8)	0.353(6)	0.416(4.5)	0.451(3)	**0.471(1.5)**
Herring	0.547(5)	0.469(7)	0.609(3)	0.594(4)	0.506(6)	-(8)	**0.656(1)**	0.654(2)
InlineSkate	0.378(5)	0.384(4)	0.295(7)	**0.438(1)**	0.398(3)	0.355(6)	-(8)	0.406(2)
InsectWingSnd	**0.637(1)**	0.358(8)	0.490(7)	0.606(2)	0.495(6)	0.596(3)	0.545(5)	0.583(4)
LargeKitchenApp	0.808(3)	0.789(4)	0.333(7.5)	0.701(5)	0.652(6)	0.333(7.5)	**0.851(1)**	0.830(2)
Lightning2	0.754(6)	0.803(3.5)	0.639(7)	0.803(3.5)	0.774(5)	-(8)	**0.820(1)**	0.813(2)
Lightning7	0.685(2.5)	0.685(2.5)	0.534(6)	-(7.5)	0.625(4)	-(7.5)	0.534(5)	**0.764(1)**
Mallat	0.946(3)	0.935(4)	0.958(2)	0.753(8)	0.931(5)	0.816(6)	0.768(7)	**0.971(1)**
Meat	0.917(3)	**1.000(1)**	0.783(6)	0.817(5)	0.880(4)	0.550(8)	0.750(7)	0.948(2)
MidPhaOutAgeGrp	0.604(2)	0.584(3.5)	0.533(5)	0.584(3.5)	0.518(6)	-(8)	0.188(7)	**0.635(1)**
MiddlePhalanxTW	0.552(2)	0.487(4)	0.500(3)	0.481(5)	0.461(6)	-(8)	0.273(7)	**0.602(1)**
Phoneme	**0.315(1)**	0.228(3)	0.174(5)	0.218(4)	0.146(6)	-(7.5)	-(7.5)	0.236(2)
ProPhaOutAgeGrp	0.844(2)	0.805(6)	0.824(4)	0.815(5)	0.767(7)	0.834(3)	0.488(8)	**0.854(1)**
RefrigDev	**0.589(1)**	0.464(5)	0.333(7)	0.515(3)	0.510(4)	0.400(6)	-(8)	0.564(2)
ShapeletSim	0.989(3)	0.650(8)	**1.000(1.5)**	0.972(4)	0.697(7)	**1.000(1.5)**	0.844(5)	0.820(6)
ShapesAll	0.835(2)	0.768(4)	0.610(5)	-(7)	0.786(3)	-(7)	-(7)	**0.854(1)**
SonyAIBORoSur2	0.880(5)	0.831(6)	0.789(7)	0.893(4)	0.770(8)	0.914(3)	**0.935(1)**	0.927(2)
Strawberry	**0.949(1)**	0.941(3)	0.914(5)	0.905(6)	0.898(7)	0.839(8)	0.943(2)	0.938(4)
Symbols	0.776(8)	**0.950(1)**	0.925(5)	0.938(3)	0.856(6)	0.808(7)	0.934(4)	0.948(2)
SyntheticContrl	**0.997(2)**	0.993(4.5)	0.923(8)	**0.997(2)**	0.976(7)	0.993(4.5)	**0.997(2)**	0.991(6)
ToeSegmentat1	0.956(3)	0.772(8)	0.921(6)	0.939(4)	0.857(7)	**0.978(1)**	0.965(2)	0.925(5)
Trace	**1.000(3.5)**	**1.000(3.5)**	**1.000(3.5)**	**1.000(3.5)**	0.958(7)	**1.000(3.5)**	0.810(8)	**1.000(3.5)**
TwoLeadECG	0.990(4)	0.904(7)	0.925(6)	0.997(2.5)	0.869(8)	**0.998(1)**	0.997(2.5)	0.951(5)
TwoPatterns	0.995(4)	**1.000(1)**	0.259(8)	0.993(5)	0.981(7)	0.998(2.5)	0.998(2.5)	0.988(6)
UWaveGestLibY	**0.722(1)**	0.634(6)	0.611(7)	0.703(2.5)	0.671(5)	0.693(4)	-(8)	0.703(2.5)
Wafer	**1.000(1)**	0.980(8)	0.998(2.5)	0.996(4)	0.993(7)	0.994(6)	0.998(2.5)	0.995(5)
Wine	0.759(2.5)	0.574(4)	0.759(2.5)	0.500(7)	0.548(5)	0.500(7)	0.500(7)	**0.766(1)**
WormsTwoClass	**0.792(1)**	0.623(8)	0.727(4.5)	0.727(4.5)	0.649(7)	0.718(6)	0.746(3)	0.777(2)
Yoga	**0.843(1)**	0.836(3)	0.700(7)	0.828(4)	0.801(6)	0.839(2)	-(8)	0.827(5)

Fungi	0.613(5)	0.796(2)	0.726(4)	0.748(3)	0.358(7)	0.505(6)	0.113(8)	**0.970(1)**
SmoothSubspace	0.920(3)	0.833(5)	0.767(7)	0.807(6)	0.859(4)	0.927(2)	0.333(8)	**0.951(1)**
Ave. ranking	3.18	4.49	5.62	3.86	6.07	5.36	4.67	**2.75**
Total Best	13	7	2	11	0	5	10	**16**
CRSF 1v1 Wins	28	36	46	28	49	39	31	-
CRSF 1v1 Draws	2	2	1	3	1	1	1	-
CRSF 1v1 Losses	20	12	3	19	0	10	18	-

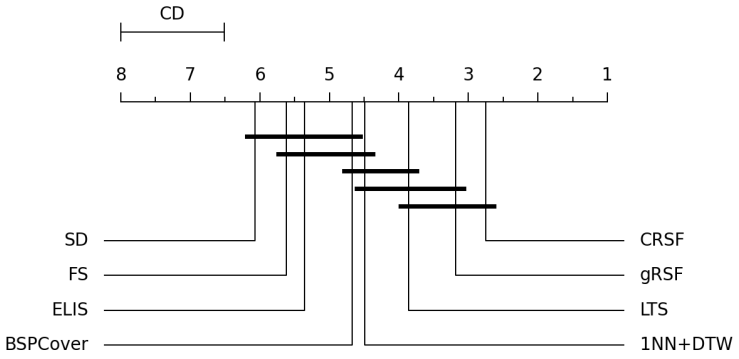

Figure 3. Nemenyi tests for 8 classifiers (p = 0.05)

5.4.3 Time cost

In this section, we discuss the time cost of the proposed algorithm. We just compare it with gRSF. There are three reasons, (1) CRSF is, essentially, an improved version of gRSF which aims to achieve higher efficiency, therefore it is necessary to compare the efficiency of the two algorithms; (2) It can be seen from table 6 that the algorithms which are based on machine learning, including LTS, ELIS and BSPCover, are slower than other algorithms; (3) FS and SD may have higher efficiency than CRSF, but both of the two algorithms are obviously inferior to CRSF in terms of accuracy.

We calculate the speedup ratio between CRSF and gRSF, which is the running time of gRSF divided by the running time of CRSF. The higher the speedup ratio value is, the more efficient the proposed algorithm is. Results are shown in figure 4 in which the horizontal ordinate denotes the index of datasets and the vertical ordinate denotes the speedup ratio between two algorithms. It can be seen that the speedup ratio obtained on FaceAll is the minimal which is 4.3, and the speedup ratio obtained on HandOutlines is the maximal which is 323.5. For all datasets, the average speedup ratio obtained by CRSF is 41. This result proves the efficiency of the proposed algorithm. Furthermore, we find that the datasets whose class number is large, such as FaceAll, Phoneme, ShapesAll, Fungi etc., consume more time cost than those whose class number is small. The finding is reasonable because the size of the shapelet pool which is the most time-consuming part in CRSF depends on the number of classes.

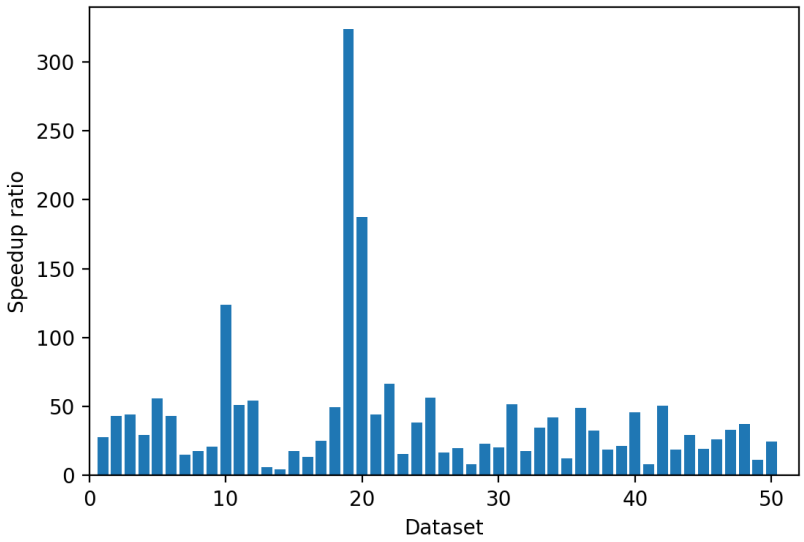

Figure 4. Speedup ratio between CRSF and gRSF

5.5 Domain analysis

In this section, we analyze the impact of the domain on the accuracy of the algorithms. According to the dataset description in table 2, the datasets can be categorized into 8 groups in terms of their domains, which are image, sensor, simulated, motion, spectro, device, ECG and HRM. We calculate the average ranking of the eight algorithms on each domain. The results are shown in table 8. The best ranking in each row is marked in bold. From the table, we find that CRSF performs best in three domains, which are image, spectro and HRM. Besides, it wins silver medal in other four domains, which are sensor, simulated, motion and device. Moreover, the difference of the average ranking between the gold medal winner (i.e. gRSF) and CRSF is very small. Thus, we can conclude that CRSF performs well in the seven domains. The only domain that CRSF performs not well is ECG. In this domain, LTS performs much better than other algorithms. CRSF only wins a bronze medal and its average ranking is 4.00, which is just on the median line.

Table 8. Average ranking of 8 classifiers on different domains

Domain	Dataset num.	gRSF	1NN+DTW	FS	LTS	SD	ELIS	BSPCover	CRSF
Image	17	3.88	4.06	5.59	3.53	6.15	5.44	4.97	**2.38**
Sensor	10	**2.70**	5.05	5.50	3.95	6.20	5.45	4.30	2.85
Simulated	6	**3.00**	4.50	5.58	4.92	6.00	4.08	4.25	3.67
Motion	5	**2.30**	6.10	6.30	4.00	5.60	4.30	4.80	2.60
Spectro	5	2.70	3.10	4.90	5.40	6.40	6.90	4.10	**2.50**
Device	3	**2.33**	3.67	6.83	3.00	5.00	7.17	5.33	2.67
ECG	3	3.67	6.33	5.67	**1.67**	6.33	4.33	4.00	4.00
HRM	1	5.00	2.00	4.00	3.00	7.00	6.00	8.00	**1.00**

5.6 Case study

In this section, we deeply analyze two datasets, i.e. FaceAll and ElectricDevice, on which the

accuracy obtained by CRSF is about 20 percent below the highest accuracy.

(1) FaceAll

The time series instances of FaceAll belong to 14 categories. From table 7, it is easy to find that 1NN+DTW achieving the highest accuracy 0.946 and the accuracies achieved by other classifiers are all below 0.770. Figure 5b is a picture that shows one exemplar per class in FaceAll (Dau, et al., 2019). From the picture, we find that these time series are very similar. In another word, it is difficult to find segments (i.e. shapelets) with great discriminating power. We think this is the reason why all shapelet-based classifiers perform worse than the 1NN+DTW which is based on whole-series comparison.

(2) ElectricDevices

ElectricDevices is a time series dataset that collects the electric consuming data of 251 families in U.K. within a month. From table 7, we can find the highest accuracy 0.929 is achieved by LTS. Three classifiers, i.e. gRSF, CRSF and 1NN+DTW, belong to the second echelon whose accuracies on the dataset are 0.748, 0.754 and 0.728 respectively. We analyze the predicting results of CRSF in depth and find that the instances in class 1 are most likely to be misclassified and its predicting error rate is over 0.9. Interestingly, most of them are misclassified into class 5. The reason behind the phenomenon, we think, is that the shapelets extracted from class 1 which have discriminating power are abandoned by Bloom filter. Figure 5b is a picture that shows one exemplar per class in the dataset (Dau, et al., 2019). Let us focus on the sharp parts in class 1, 3 and 6. We can see that the shapes of these segments are very similar. In other words, the SAX word converted from these segments may be identical, thus they all would be discarded by Bloom filters. These shortcomings may be overcome by improving the sampling strategy.

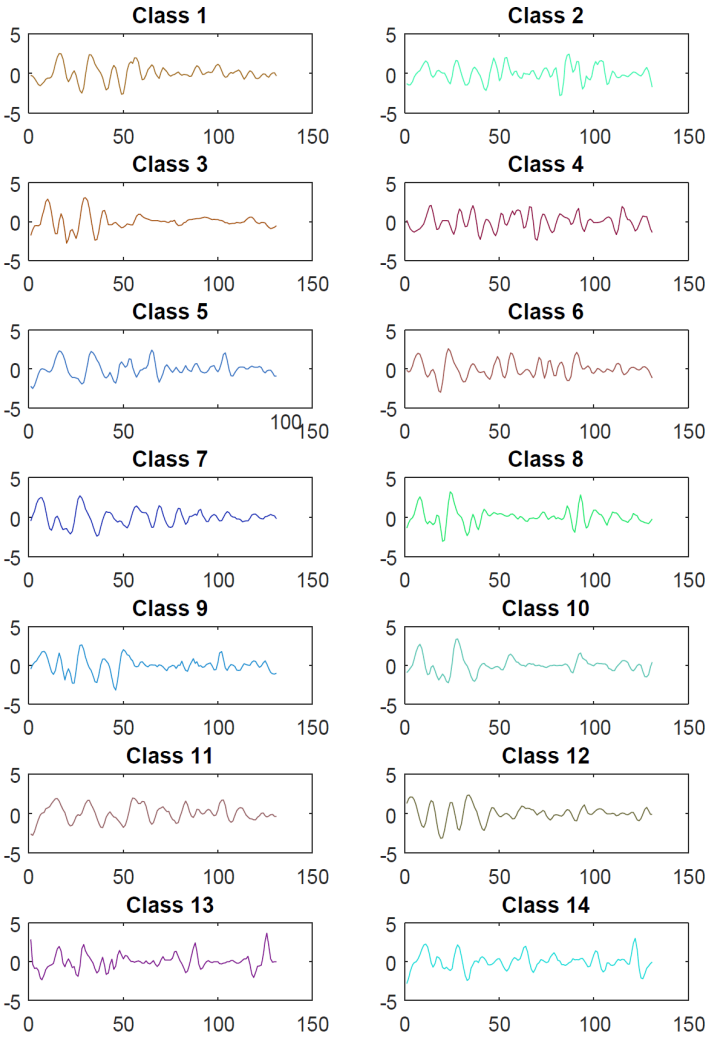

Figure 5a. One exemplar per class of FaceAll

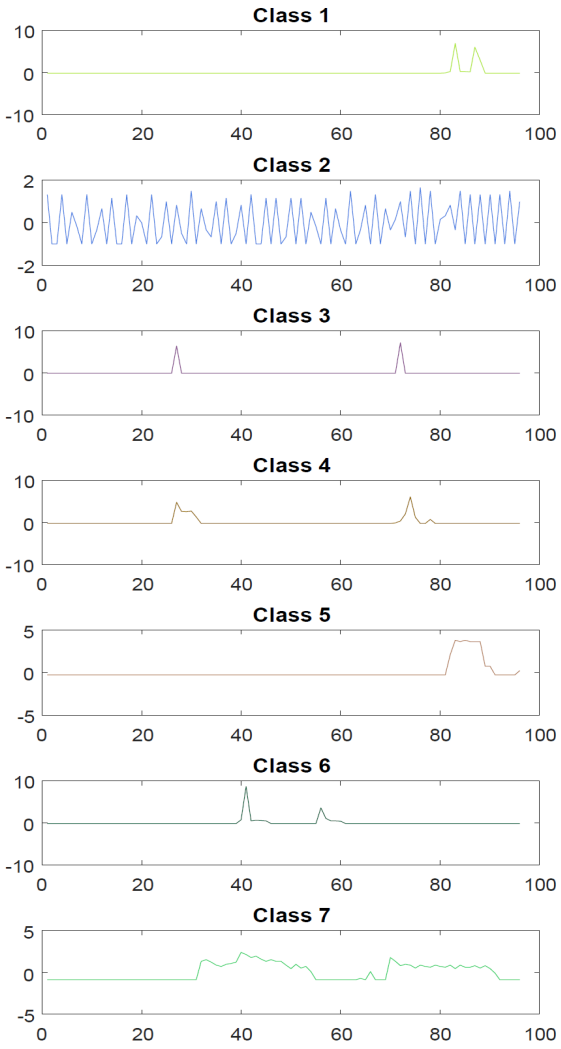

Figure 5b. One exemplar per class of ElectricDevies

6. Conclusions

In this paper, we propose a compressed random shapelet forest algorithm, named CRSF, for TSC task. Compared with another shapelet-based random forest classifier gRSF, CRSF improves the accuracy and efficiency by compressing the shapelet feature space. The compression is achieved by SAX representation as well as constructing a high-quality shapelet pool in which a decision tree randomly select a shapelet when growing a tree node. Besides, we also propose a roulette wheel selection strategy for shapelet selection and a new distance function to measure two SAX words. We perform extensive experiments on 50 popular time series datasets to evaluate the effectiveness and efficiency of the CRSF algorithm. Some conclusions can be drawn.

(1) Compared with 6 well-known shapelet-based classifiers and the 1NN+DTW classifier, CRSF can achieve the highest average accuracy on 50 datasets. Although there is no significant difference in terms of the accuracy among CRSF, gRSF and LTS statistically, CRSF is much more efficient than gRSF and LTS. Specifically, the average running time of CRSF is 41 times faster than that of gRSF.

(2) We conducted abundant experiments to evaluate the proposed strategies and explored the strategies of parameter setting. The experimental results show that the strategies, i.e. sampling shapelet on multiple SAX configurations and employing tournament selection when growing a tree node, do not show superiority, but deteriorate the efficiency of the algorithm. Therefore, the CRSF algorithm only samples shapelets on a single SAX configuration and employs roulette wheel selection strategy. Besides, setting parameters sn and r to 100 and 300 respectively is a safe choice.

(3) By analyzing the predicting results on different domains, it shows that CRSF performs well in seven of the eight domains. In the last domain (i.e. ECG), the average ranking achieved by CRSF is 4.0 which is on the median line (not bad). The results demonstrate that the domain of the problem has little impact on the accuracy of CRSF.

Acknowledgements

The authors thank the reviewers for their work and the contributors of the UCR archive.

References

Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. In *International Conference on Foundations of Data Organization and Algorithms* (pp. 69-84).

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery, 31*, 606-660.

Batista, G. E. A. P. A., Keogh, E. J., Tataw, O. M., & de Souza, V. M. A. (2013). CID: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery, 28*, 634-669.

Baydogan, M. G., & Runger, G. (2015). Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery, 30*, 476-509.

Chan, F. K., Ada Wai-chee, F., & Clement, Y. (2003). Haar wavelets for efficient similarity search of time-series: with and without time warping. *IEEE Transactions on Knowledge and Data Engineering, 15*, 686-705.

Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., & Keogh, E. (2019). The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica, 6*, 1293-1305.

Fang, Z., Wang, P., & Wang, W. (2018). Efficient Learning Interpretable Shapelets for Accurate Time Series Classification. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (pp. 497-508).

Gordon, D., Hendler, D., Kontorovich, A., & Rokach, L. (2015). Local-shapelets for fast classification of spectrographic measurements. *Expert Systems with Applications, 42*, 3150-3158.

Górecki, T., & Łuczak, M. (2012). Using derivatives in time series classification. *Data Mining and Knowledge Discovery, 26*, 310-331.

Grabocka, J., Schilling, N., Wistuba, M., & Schmidt-Thieme, L. (2014). Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and*

data mining (pp. 392-401).

Grabocka, J., Wistuba, M., & Schmidt-Thieme, L. (2015). Fast classification of univariate and multivariate time series through shapelet discovery. *Knowledge and Information Systems, 49*, 429-454.

Hills, J., Lines, J., Baranauskas, E., Mapp, J., & Bagnall, A. (2013). Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery, 28*, 851-881.

Hong, J. Y., Park, S. H., & Baek, J.-G. (2020). SSDTW: Shape segment dynamic time warping. *Expert Systems with Applications, 150*.

Hou, L., Kwok, J. T., & Zurada, J. M. (2016). Efficient Learning of Timeseries Shapelets. In *Thirtieth AAAI Conference on Artificial Intelligence* (pp. 1209-1215).

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery, 33*, 917-963.

Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., Webb, G. I., Idoumghar, L., Muller, P.-A., & Petitjean, F. (2020). InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery, 34*, 1936-1962.

Jeong, Y.-S., Jeong, M. K., & Omitaomu, O. A. (2011). Weighted dynamic time warping for time series classification. *Pattern Recognition, 44*, 2231-2240.

Karlsson, I., Papapetrou, P., & Boström, H. (2016). Generalized random shapelet forests. *Data Mining and Knowledge Discovery, 30*, 1053-1085.

Katris, C. (2021). A time series-based statistical approach for outbreak spread forecasting: Application of COVID-19 in Greece. *Expert Syst Appl, 166*, 114077.

Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems, 3*, 263-286.

Lahreche, A., & Boucheham, B. (2021). A fast and accurate similarity measure for long time series classification based on local extrema and dynamic time warping. *Expert Systems with Applications, 168*.

Li, G., Choi, B., Xu, J., Bhowmick, S. S., Chun, K.-P., & Wong, G. L. H. (2021). Efficient Shapelet Discovery for Time Series Classification (Extended Abstract). In *2021 IEEE 37th International Conference on Data Engineering (ICDE)* (pp. 2336-2337).

Li, G., Yan, W., & Wu, Z. (2019). Discovering shapelets with key points in time series classification. *Expert Systems with Applications, 132*, 76-86.

Lin, J., Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery, 15*, 107-144.

Marteau, P. F. (2009). Time warp edit distance with stiffness adjustment for time series matching. *IEEE Trans Pattern Anal Mach Intell, 31*, 306-318.

Mueen, A., Keogh, E., & Young, N. (2011). Logical-Shapelets: An Expressive Primitive for Time Series Classification. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*.

Rakthanmanon, T., & Keogh, E. (2013). Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets. In *Proceedings of the 2013 SIAM International Conference on Data Mining* (pp. 668-676).

Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., & Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min Knowl Discov, 35*, 401-449.

Stefan, A., Athitsos, V., & Das, G. (2013). The Move-Split-Merge Metric for Time Series. *IEEE Transactions on Knowledge and Data Engineering, 25*, 1425-1438.

Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., & Keogh, E. (2013). Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery, 26*, 275-309.

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 1578-1585).

Ye, L., & Keogh, E. (2009). Time series shapelets. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*.

Zhang, X., Gao, Y., Lin, J., & Lu, C.-T. (2020). TapNet: Multivariate Time Series Classification with Attentional Prototypical Network. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, pp. 6845-6852).