Expert Systems With Applications 213 (2023) 118923

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Forest based on Interval Transformation (FIT): A time series classifier with adaptive features

Guiling Li [a,b], Shaolin Xu [a,*], Senzhang Wang [c], Philip S. Yu [d]

[a] School of Computer Science, China University of Geosciences, Wuhan, China
[b] Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan, China
[c] School of Computer Science and Engineering, Central South University, Changsha, China
[d] Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA

ARTICLE INFO

Keywords:
Time series classification
Feature selection
Random forest
Interval-based classifier

ABSTRACT

Time series classification (TSC) is an important task in time series data mining and has attracted a lot of research attention. Most TSC algorithms aim to achieve high classification accuracy while reducing the computational complexity. Currently, Time Series Combination of Heterogeneous and Integrated Embedding Forest (TS-CHIEF) is considered to be one of the state-of-the-art TSC algorithms. However, compared with fast algorithms such as Time Series Forest (TSF), TS-CHIEF still has high computation cost. On the premise that the TSF algorithm is fast, we propose a new TSC algorithm, Forest based on Interval Transformation (called FIT), which takes into account both accuracy and efficiency. FIT uses cross-validation to select appropriate transformation series and corresponding interval features, and adaptively converts the interval features of each series in the process of formal training. Subsequently, the transformed feature set is combined with the random forest training FIT model. We evaluate the performance of FIT on 85 UCR time series classification datasets. The experimental results demonstrate that FIT can achieve better accuracy while maintaining high efficiency compared with the state-of-the-art methods.

1. Introduction

In the past few decades, the research on time series classification (TSC) has gradually become an important research direction of machine learning. Numerous researchers have proposed a wide variety of algorithms, including time series similarity based methods and important features of time series extraction based methods. TSC can be applied in the area of electrocardiogram detection (Pourbabaee, Roshtkhari, & Khorasani, 2018; Wang, Liu, She, Nahavandi, & Kouzani, 2013), financial forecast (Tay and Cao, 2001) and motion recognition (Savadkoohi, Oladunni, and Thompson, 2021) and can reduce the workload of workers in the corresponding fields.

Among the existing TSC methods, a popular way is to use the integrated method to improve the classification performance. Through homogeneous integration or heterogeneous integration, the accuracy of the TSC algorithm can be greatly improved. Wherein, forest-based method is a classical integration algorithm, and Proximity Forest (PF) (Lucas et al., 2019) is a representative method with ensemble. PF uses a distance-based forest model through the random selection of distance measure at each node and the idea of dividing and conquering trees. The training complexity of the PF algorithm is quasi-linear with the number of training instances, and it has a quadratic relationship with the length of the time series. Moreover, PF can achieve classification accuracy close to that of the state-of-the-art TSC algorithm HIVE-COTE (Lines, Taylor, and Bagnall, 2018). Inspired by PF, Shifaz et al. propose a new algorithm TS-CHIEF (Shifaz, Pelletier, Petitjean, and Webb, 2020). In addition to distance-based representation in PF, TS-CHIEF uses interval-based and dictionary-based representations. It adopts the idea similar to PF, and uses heterogeneous integration to improve the classification accuracy. TS-CHIEF achieves high accuracy comparable to that of HIVE-COTE, and has the same training complexity as PF.

However, compared with fast algorithms such as BOSS (Schäfer, 2015) only based on symbolic representation and TSF (Deng, Runger, Tuv, and Vladimir, 2013) only based on interval representation, the computational complexity of PF and TS-CHIEF is still relatively high. Although TSF has low training time complexity, the accuracy of TSF is not competitive with other algorithms due to the randomness of interval selection in the training process. For interval-based classification method, if we can obtain the best interval to distinguish different classes of time series, the classification accuracy will be better

* Corresponding author.
E-mail addresses: guiling@cug.edu.cn (G. Li), cugxsl@163.com (S. Xu), szwang@csu.edu.cn (S. Wang), psyu@uic.edu (P.S. Yu).

https://doi.org/10.1016/j.eswa.2022.118923
Received 27 October 2021; Received in revised form 13 August 2022; Accepted 25 September 2022
Available online 30 September 2022

improved. On the basis of TSF, which is a fast algorithm based on intervals, we try to develop an algorithm that considers both accuracy and efficiency.

We propose a new more accurate TSC method based on interval features, named Forest based on Interval Transformation (FIT for short). Due to the feature limitation of the original time series in TSF and the randomness of the interval features, although the TSF algorithm is fast, the accuracy is not competitive. Therefore, our proposed algorithm FIT focuses on adding more transformation series and selecting discriminative feature representations. We design an overall scheme, that is, using cross-validation to select the appropriate transformation series and interval feature representation, and then combine the selected transformation series and interval feature representation with formal training. The scheme consists of two stages, cross-validation stage and formal training stage respectively. In the cross-validation stage, we use some transformation series and extracted features corresponding to random intervals to construct a classifier, and perform cross-validation to evaluate the performance of the transformation series on dataset, so as to select the appropriate transformation series and interval features. In the formal training stage, we use the selected transformation series and corresponding interval features to guide the training of the classifier. We use the corresponding interval features of the selected series to complete the calculation of interval features, thereby obtaining a new feature set as an input to construct a decision tree. FIT is a tree-based ensemble classification method, which adopts a random forest method to train similar trees as the final classifier. The corresponding interval features of these transformation series can more effectively identify the discriminative information on the time series of different classes, and thus FIT can more accurately classify the time series of unknown classes.

The main contributions of our work are summarized as follows.

- We propose a TSC algorithm based on interval feature transformation called FIT. FIT combines the two stages of cross-validation and formal training to achieve both efficiency and accuracy.
- In the cross-validation stage, we propose an adaptive selection method. This method can automatically select expressive transformation series and discriminative interval features for different datasets. In the formal training stage, we use the selected results of cross-validation to automatically assemble the interval features, and use the feature set combined with random forest to train the classifier.
- We conducted experiments on UCR time series classification archive (Dau et al., 2018) to evaluate the accuracy of classification. Experiments show that FIT is competitive with the state-of-the-art methods in terms of accuracy. Meanwhile, we also evaluated the efficiency of FIT on the UCR archive. Experiments show that FIT has higher efficiency compared with the candidates.

The rest of the paper is organized as follows. Section 2 is the related work of TSC algorithms. In Section 3, we present our new TSC method FIT in detail. Section 4 is the experimental evaluation. Finally, we conclude the paper and point out the future direction in Section 5.

2. Related work

In recent years, TSC has attracted much interest, and many algorithms have been proposed. We will make the reviews of the related work below.

2.1. Distance-based classifier

The distance-based feature classifier is a classifier that uses the real values of the entire time series as features. There are two main classification methods. One is to treat the entire time series as a vector, and then use traditional classification techniques. The second is to calculate the distance between the real values of time series based on the change of the distance measures. Wherein, the one nearest neighbor (1-NN) classifier combined with Euclidean Distance (ED) is the simplest classifier. However, the phase deviation between time series may occur, so the technology of elastic distance measure is used to solve this problem. The most common method is 1-NN combined with Dynamic Time Warping (1NN-DTW), which is the classic benchmark of TSC problems (Bagnall, Lines, Bostrom, Large, & Keogh, 2017; Sharabiani et al., 2017).

Later, Elastic Ensemble (EE) (Lines and Bagnall, 2015) integrates multiple 1-NN classifiers using eleven diverse distance measures, and proves that EE is more accurate than 1-NN classifiers using any single distance measure. However, EE takes more time to determine the parameters. Recently, Proximity Forest (PF) (Lucas et al., 2019) proposes to obtain the representative series of each class, and split the node according to the similarity to these series. Meanwhile, the similarity measure randomly selects one of the same eleven distance measure methods as EE. PF can get a higher accuracy than EE. In addition, due to the use of tree-based ensemble and random distance measure selection, PF has higher scalability.

2.2. Shapelet-based classifier

Shapelet is a continuous subsequence which can maximally represent a certain class in the time series. The shapelet-based classifier mainly includes two steps. First, we need discover shapelets. Second, we use the discovered shapelets to make classification.

There are two ways to discover shapelets. One is to select shapelets in the candidate set. Ye and Keogh (2009) first proposed the discriminative feature of shapelet, and the shapelet with the highest information gain was selected in the candidate set for classification. However, the complexity of the searching process is too high. Subsequent work on shapelet-based classification dedicated to speed up the searching process of shapelet (Ji et al., 2017; Li, Yan, & Wu, 2019). Fast shapelets (FS) (Rakthanmanon and Keogh, 2013) uses Symbol Aggregation Approximation (SAX) (Lin, Keogh, Wei, and Lonardi, 2007) to transform time series, thus accelerating the shapelet discovery. The other is to generate shapelets by optimizing the objective function. Learned shapelet (LS) (Grabocka, Schilling, Wistuba, and Schmidt-Thieme, 2014) initializes a shapelet, and optimizes the loss function to obtain a shapelet close to the optimal shapelet. The accuracy of LS is higher than that of FS, but it has higher time consumption.

After the shapelets are obtained, there are two ways to use the shapelets for classification. One is to embed the shapelets into the decision tree as the split feature of the node such as the work in Ye and Keogh (2009). The other is to use shapelets to transform the original time series into a new feature space to construct a classifier such as the Shapelet Transform (ST) in Hills, Lines, Baranauskas, Mapp, and Bagnall (2014). ST can obtain high accuracy, but its computational complexity is high.

2.3. Dictionary-based classifier

Many dictionary-based classification algorithms have been proposed, including the Bag of Patterns (BOP) (Lin, Khade, and Li, 2012), Symbolic Aggregate approXimation-Vector Space Model (SAX-VSM) (Senin and Malinchik, 2013), Bag-of-SFA-Symbols (BOSS) (Schäfer, 2015), Bag-of-SFA-Symbols in Vector Space (BOSS-VS) (Schäfer, 2016), Word Extraction for Time Series Classification (WEASEL) (Schäfer and Leser, 2017), contractable BOSS (cBOSS) (Middlehurst, Vickers, and Bagnall, 2019).

Among the dictionary-based methods, BOSS is the most commonly used benchmark. BOSS first uses Symbolic Fourier Approximation (SFA) to convert real-valued time series into SFA words. Then it counts the frequency of these SFA words to get a histogram, and uses the

frequency histogram to train the BOSS model. BOSS-VS is a variant of BOSS in vector space, which can greatly reduce the training time of BOSS, but it will reduce the accuracy of BOSS. Schäfer and Leser (2017) proposed a new variant WEASEL, it can obtain higher accuracy than BOSS, but it increases the training time and costs more memory. The recently proposed cBOSS can reduce the training time of the BOSS, meanwhile, it can obtain the comparable accuracy with BOSS. Therefore, cBOSS is selected as a new component of HIVE-COTE 1.0 (Bagnall, Flynn, Large, Lines, and Middlehurst, 2020).

2.4. Interval-based classifier

Classifiers based on intervals usually randomly extract a set of intervals from time series. These intervals are then transformed into new features, which are used as the input of traditional TSC methods, such as decision trees. Time Series Forest (TSF) (Deng et al., 2013) proposes to randomly select a set of intervals in a time series and calculate three features of intervals, including mean, standard deviation and slope. These new features are used to train the random forest. Besides, a new measure entrance gain is adopted as the splitting criterion at the node of decision tree. TSF uses a random sampling strategy to greatly reduce time complexity. The inspiration for this paper is also from TSF.

TSBF (Baydogan, Runger, and Tuv, 2013) proposes a bag-of-features framework to classify time series. TSBF first randomly extracts intervals from time series to capture the local information. Second, the learner use these intervals to generate class probability estimates. Then, histograms of these class probability estimates are generated, and the interval information is obtained by concatenating them. Finally, global features are added as input and random forest is combined to train final classifier. Different from TSF and TSBF, Random Interval Spectral Ensemble (RISE) (Lines et al., 2018) uses the spectral features of intervals. RISE first obtains the spectral features of the entire time series and trains a random tree on these features. Then, the remaining random trees are trained on four transformed features of an interval to obtain the corresponding classifier. Finally, all the random trees are combined for classification.

2.5. Ensemble-based classifier

There are some work on TSC by ensemble methods such as He et al. (2020) and Lines et al. (2018) . Ensemble methods include homogeneous ensemble and heterogeneous ensemble. Homogeneous ensemble means that, by changing the training data or training scheme, each of the same base classifiers is trained to obtain a homogeneous set. The ensemble of trees is a typical homogeneous ensemble, such as PF and TSF.

Heterogeneous ensemble refers to the set formed by training each base classifier using different classification algorithms. Two commonly used heterogeneous ensembles are the Flat Collective of Transformation-based Ensembles (FLAT-COTE) (Bagnall, Lines, Hills, and Bostrom, 2015) and HIVE-COTE (Lines et al., 2018). FLAT-COTE divides 35 different classifiers into four types, but the final result is a meta-ensemble of 35 classifiers. HIVE-COTE adds two new classifiers, BOSS and RISE respectively. Moreover, HIVE-COTE divides different classifiers into five modules, and obtains individual predictions on each module, so that the weight of the influence of each module is more balancing. HIVE-COTE is considered to be the state-of-the-art algorithm. However, due to two components EE and ST, the training complexity of HIVE-COTE is relatively high. Recently, TS-CHIEF (Shifaz et al., 2020) proposes to use three different types of features to construct each tree, including similarity-based, dictionary-based or interval-based representations. TS-CHIEF can obtain higher accuracy comparable to that of HIVE-COTE.

2.6. Classifier combined with deep learning

Deep learning technology has achieved great success in image classification (Krizhevsky, Sutskever, and Hinton, 2012) and speech recognition (Hinton et al., 2012). At present, some TSC algorithms by deep learning have been proposed(Fawaz, Forestier, Weber, Idoumghar, and Muller, 2019), including Fully Convolutional Network (FCN) (Wang, Yan, and Oates, 2017), Residual Network (ResNet) (Wang et al., 2017), Random Convolution Kernel Transformation (ROCKET) (Dempster, Petitjean, and Webb, 2020) and InceptionTime (Fawaz et al., 2020). Compared with HIVE-COTE, FCN and ResNet are still unable to obtain the competitive accuracy. While ROCKET and InceptionTime are able to compete with HIVE-COTE in terms of accuracy and also have scalability. These algorithms combined with deep learning can achieve high training efficiency. Combining deep learning with TSC research is a direction worth exploring.

3. The proposed FIT algorithm

We propose a new TSC algorithm, a forest classifier based on interval transformation called FIT. FIT can adaptively select the transformation series and the interval feature representation. TSF proposes to select a set of intervals randomly in time series and transforms them into three features, mean, standard deviation and slope respectively. These interval features are then combined with random forest to construct classifier. Inspired by TSF, our proposed FIT algorithm can obtain discriminative features which can better distinguish different classes of time series. Fig. 1 shows the overall framework of the FIT model. First, we extract the interval of the six kinds of series, and at the same time perform a variety of feature representation calculations on these intervals. Second, we make cross-validation to obtain the corresponding accuracy and number of features selected for each series. According to the cross-validation accuracy, we figure out the suitable series type for the current dataset. Then, we select the appropriate feature representation for each series, and perform interval extraction and adaptive feature representation calculation. Finally, we use the obtained adaptive feature set combined with random forest for training to obtain our FIT classification model.

In this section, first, we introduce six features of the interval. Second, we present the proposed adaptive series and feature acquisition methods and the mechanism to use these features to train the FIT model respectively. Third, after training the FIT model, we describe how to use FIT to classify time series. Finally, we make complexity analysis of the FIT algorithm.

3.1. Features of the interval

When we obtain the interval from the time series, there are many types of features that can be calculated. For example, the HCTSA toolbox (Fulcher and Jones, 2017) is a comprehensive library containing more than 7500 features, including basic statistics of time series values, entropy, linear relationships, stability, and many other features. However, simple features are usually more intuitive and effective for interval performance, such as mean, standard deviation.

In this paper, we use six simple yet effective features to describe the time series interval, i.e., mean, standard deviation, slope, interquartile range, maximum and minimum respectively. There are two methods to calculate the slope, linear interpolation and linear regression respectively. Linear interpolation uses a straight line to connect the two endpoints of the series and obtains the slope of the line. Linear regression uses the slope of the regression line of all data points on the series, in which the independent variable is the time index and the dependent variable is the series value corresponding to the time index. We use the linear regression method because it can better represent the characteristics of all data points on the series.

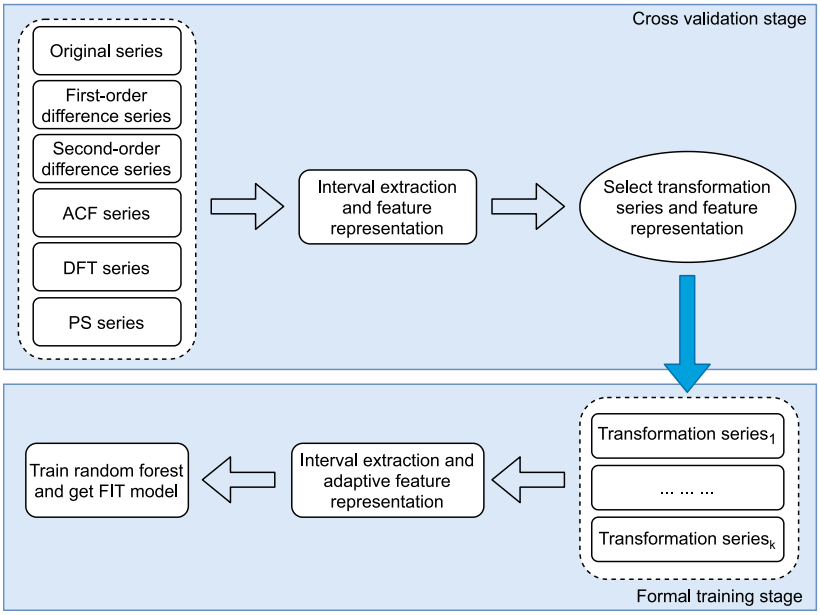

Fig. 1. The overall framework of the FIT model.

In order to better introduce our work, some symbolic representations in this paper are given. The time series T is a series of m real values that change with time t, where $T = x_1, \ldots, x_m$. The generation of a time series is actually a stochastic process, and the real values of a time series are a set of random variables that depend on the time index. Thus, n instances of time series generated by the random process form the training set D. For simplicity, we use $F_f(D, I)$ to represent the fth feature of the interval I on the dataset D, where the interval I includes the starting point t_a and the ending point t_b of the interval, where $1 \leq t_a \leq t_b \leq m$.

3.2. Cross-validation stage

We have described six types of interval features above. Next, we will introduce how to train our FIT model. In this section, we first introduce how to obtain the appropriate series transformation method and the corresponding interval feature representation.

3.2.1. Adaptive selection of series and interval features

In order to extract more effective interval features, we have added several series transformation methods to enrich the expressiveness of the original series. The adding methods include the First-order Difference and Second-order Difference series transformation of the trend feature, the Power Spectrum (PS) and Discrete Fourier Transform (DFT) series transformation of the frequency domain feature, and the Auto-Correlation Function (ACF) series transformation of the time domain feature. We extract the interval features for all the transformation series, and evaluate the performance of the transformation series and the interval features by cross-validation. Therefore, we can obtain a highly expressive series transformation and feature representation method for the current cross-validated dataset. And we call this procedure as self-adaptation selection.

Algorithm 1 describes the specific realization of self-adaptation selection for the series transformation and the corresponding interval feature. The input parameters include the training set D, selection proportion of accuracy $ratio$, number of features n_F, number of trees for cross-validation k_{CvTree}, and number of intervals extracted $n_{Interval}$. Here we perform 10-fold cross-validation five times, and set the value of $ratio$ to 0.9. In Algorithm 1, we first transform the original training set to obtain six different types of transformation series (line 1). Next, we calculate the average accuracy by cross-validation for each transformation series, and the number of corresponding feature nodes selected as tree nodes when each classifier is constructed (lines 3–13).

Algorithm 1 AdaptiveFeatureSelection

Input: Training set, D; Selection proportion of accuracy, $ratio$; Number of features, n_F; Number of trees for cross validation, k_{CvTree}; Number of intervals extracted, $n_{Interval}$;
Output: Transformation series and corresponding interval feature, $adaptiveFeatureSet$;

```
1: series ← TransDataset(D)
2: featureNodes ← ∅
3: for i=1 to Size(series) do
4:     for j=1 to 5 do
5:         for k=1 to 10 do
6:             train_k, valid_k ← series_i
7:             oneFoldAcc += GetCVAcc(train_k, valid_k, featureNodes_i,
       k_CvTree, n_Interval)
8:         end for
9:         oneFoldAcc /= 10
10:        cvAcc_i += oneFoldAcc
11:    end for
12:    cvAcc_i /= 5
13: end for
14: for i=1 to Size(series) do
15:    if cvAcc_i ≥ ratio*Max(cvAcc) then
16:        for j=1 to n_F do
17:            if featureNodes_ij ≥ Mean(featureNodes_i) then
18:                adaptiveFeatureSet.Add(i * n_F + j)
19:            end if
20:        end for
21:    end if
22: end for
23: return adaptiveFeatureSet
```

First, we divide the original training set into a training set and a validation set (line 6). The training set is used to train the model, and the validation set is used to calculate one fold of the cross-validation accuracy (line 7). After five times 10-fold cross-validation, we obtain an average cross-validation accuracy for each series transformation method (line 12). Then, we can use the results of cross-validation

to select the series transformation methods and interval features that are beneficial to the classification of the current dataset (lines 14–22). Here, we select the transformation method with the accuracy greater than $ratio$*Max($cvAcc$), where Max($cvAcc$) is the maximum cross-validation accuracy (line 15). Subsequently, we need to select effective interval features for each transformation method. The criterion we choose is to keep the feature whose number of feature nodes is greater than the average of the total number of six feature nodes (line 17). Due to the uncertainty of node selection, the selection method of interval features is different from that of series transformation. This uncertainty is mainly due to the fact that when the training tree classifier splits at each node, the attribute selected to calculate the information gain is part of randomness, so a certain degree of certainty is sacrificed in order to improve the speed of cross-validation. By the selection method of interval features, poor feature representations can be automatically filtered. For example, if the number of nodes of six features is [10, 10, 10, 10, 10, 40], then only one feature with a number of 40 will be retained. If the number of nodes of the six features is [10, 20, 30, 40, 50, 60] in this relatively scattered situation, we can guarantee that about half of the better features are retained. If the number of nodes for the six features is [10, 40, 40, 40, 40, 40], then we will undoubtedly filter out the feature with the worst node number of 10. Finally, the selected features will be stored in our adaptive feature set (line 18).

Algorithm 2 GetCVAcc

Input: Training set, $train$; Validation set, $valid$; Number of feature nodes for cross-validation, $featureNodes$; Number of trees for cross validation, k_{CvTree}; Number of intervals extracted, $n_{Interval}$;
Output: The accuracy of validation set, $oneFoldAcc$;

1: **for** i=1 to k_{CvTree} **do**
2: $intervals \leftarrow$ BuildInterval($n_{Interval}$, Length($train$))
3: **for** j=1 to $n_{Interval}$ **do**
4: **for** k=1 to Size($train$) **do**
5: **for** f=1 to n_F **do**
6: $transTrain_k$.Add($F_f(train_k, intervals_j)$)
7: **end for**
8: **end for**
9: **end for**
10: $cvTrees_i \leftarrow$ BuildRandomTree($transTrain$, $featureNodes$)
11: **end for**
12: **for** k=1 to Size($valid$) **do**
13: **for** i=1 to k_{CvTree} **do**
14: **for** j=1 to $n_{Interval}$ **do**
15: **for** f=1 to n_F **do**
16: $transValid_k$.Add($F_f(valid_k, intervals_j)$)
17: **end for**
18: **end for**
19: $classes_i \leftarrow$ ClassifyWithRandomTree($cvTrees_i$, $transValid$)
20: **end for**
21: $preClass \leftarrow$ MaxOccurClass($classes$)
22: **if** $preClass = trueClass$ **then**
23: $preTrue$++
24: **end if**
25: **end for**
26: $oneFoldAcc = preTrue$/Size($valid$)
27: **return** $oneFoldAcc$

3.2.2. Training process of cross-validation

Algorithm 2 describes the training process of the classifier in cross-validation, including training the classifier through training set and calculating cross-validation accuracy by using validation set. The input parameters include training set $train$, validation set $valid$, number of feature nodes for cross-validation $featureNodes$, number of trees for cross-validation k_{CvTree}, and number of intervals extracted $n_{Interval}$. Here we set the number of trees in cross-validation k_{CvTree} to 10 and the number of intervals extracted $n_{Interval}$ to $\sqrt{m}$ (we take an integer close to the square root of m). First, for each tree, we need to generate a set of random intervals for the transformation of multiple features (line 2). After obtaining a set of random intervals (presented by Algorithm 3 later), we can make feature transform for the training set. For each interval of each series, the corresponding feature transformation is performed (line 6). Subsequently, we take the feature set of training set transformation and the number of nodes of the feature as the input to build a random tree (line 10). By this way, a tree is successfully built. After building the tree 10 times, a classifier of cross-validation is obtained (lines 1–11). Then we use the classifier to evaluate the accuracy of the validation set. Similarly, we perform feature transformation on the validation set (line 16), and obtain a prediction class for each tree (line 19). We set the class with the most occurrences as the class of the instance (line 21), and obtain the total number of instances with correct predictions (lines 22–24). Finally, we obtain cross-validation accuracy by dividing the number of instances with the correct class prediction by the total number of instances in the validation set (line 26).

Algorithm 3 BuildInterval

Input: Number of intervals extracted, $n_{Interval}$; The length of series, m;
Output: A set of random intervals, $intervals$;

1: $intervals \leftarrow \emptyset$
2: **for** i=1 to $n_{Interval}$ **do**
3: **if** $i = 1$ **then**
4: $startPos \leftarrow 0$
5: $endPos \leftarrow m$
6: **else**
7: $startPos \leftarrow$ RandGenStart()
8: $length \leftarrow$ RandGenLength()
9: $endPos \leftarrow startPos + length$
10: **end if**
11: **if** $endPos > m$ **then**
12: $endPos \leftarrow m$
13: $intervals$.Add(Interval($startPos$, $endPos$))
14: **end if**
15: **end for**
16: **return** $intervals$

Algorithm 3 describe how to extract random intervals. The input parameters include number of intervals extracted $n_{Interval}$, and the length of time series m. In Algorithm 3, we extract two kinds of the interval. One is the global interval, that is, an interval of length m that contains all data points (lines 3–5). And the other is the random interval (lines 6–10). It is worth pointing out that we have set certain rules for the extraction of random interval. The starting point of the interval is randomly obtained in the range of [0, m-3] (line 7). The reason is to ensure that the starting point of the interval does not appear in the last three data points, and also to ensure that the interval length selected later is greater than 3. Then, the length of the interval is randomly selected in the range of [3, m/2] (line 8). The ending point of the interval is the sum of the starting point of the interval and the random interval length (line 9). But if the ending point of the interval exceeds the length of the time series, it is required to replace the ending point of the interval with the last data point of the time series (lines 11–14). Once the starting point and ending point of the interval are obtained, the corresponding random interval is also generated (line 13).

3.3. Formal training stage

After obtaining the discriminative series transformation method and interval features representation, we propose how to construct the FIT classification model in Algorithm 4. The input parameters include training set D, selection proportion of accuracy $ratio$, number of features n_F, number of trees k_{Tree}, number of trees for cross validation k_{CvTree}, and

number of intervals extracted $n_{Interval}$. First, we can acquire the transformation method and the corresponding interval feature suitable for the current dataset by Algorithm 1 (line 1), and make the corresponding transformation on the training set to get transformation series (line 2). Second, in each tree, the interval is extracted for each transformation series (line 5). Then, the interval extracted from each transformation series is calculated according to the selected feature representation, so as to obtain the feature set we use for the random tree (line 8). Finally, the feature set is used as the input to construct the random tree, and the information gain is used as the splitting criterion of the random tree. Thereby, a tree based on interval transformation is successfully constructed (line 12). After constructing the tree 200 times, we obtain the forest based on interval transformation, that is, the FIT classification model training is completed (line 14).

Algorithm 4 TrainingFIT

Input: Training set, D; Selection proportion of accuracy, *ratio*; Number of features, n_F; Number of trees, k_{Tree}; Number of trees for cross validation, k_{CvTree}; Number of intervals extracted, $n_{Interval}$;
Output: FIT model, *trees*;
1: $adaptiveFeatureSet \leftarrow$ AdaptiveFeatureSelection($D, ratio, n_F, k_{CvTree}, n_{Interval}$)
2: $series \leftarrow$ TransDataset(D)
3: **for** i=1 to k_{Tree} **do**
4: **for** s=1 to Size($adaptiveFeatureSet$) **do**
5: $intervals \leftarrow$ BuildInterval($n_{Interval}$, Length($selectedSeries$))
6: **for** j=1 to $n_{Interval}$ **do**
7: **for** k=1 to Size(D) **do**
8: $transTrain_k$.Add($F_f(selectedSeries_k, intervals_j)$)
9: **end for**
10: **end for**
11: **end for**
12: $trees_i \leftarrow$ BuildRandomTree($transTrain$)
13: **end for**
14: **return** *trees*

3.4. Use FIT for classification

When a time series with unknown class need to be classified, we describe how FIT classifies the time series in Algorithm 5. The input parameters include test instance Q, number of trees k_{Tree}, transformation series and corresponding interval feature $adaptiveFeatureSet$, number of intervals extracted $n_{Interval}$, and FIT model *trees*. First, according to a set of intervals of the current tree, the test instance of the unknown class is represented by the corresponding feature of each transformation series, and a transformed time series is obtained (line 4). Then, we use the transformed test instance as the input of FIT model, and use the classification method of decision tree to classify until reaching the leaf node to yield the classification result, which is the test result of the tree for the time series (line 7). Finally, a majority vote is performed on the 200 test results obtained by 200 trees to obtain the final prediction class (line 9), thus we obtain the classification result of the test instance by FIT.

3.5. Complexity analysis

The training of the FIT model contains two stages, the cross-validation stage and the formal training stage respectively.

In the cross-validation stage, we select the appropriate series transformation method and the corresponding interval feature. For each tree, we set the number of split candidate features on each node to $\sqrt{m}$, and the maximum number of instances in each layer to n, so the complexity of each layer is $O(n * \sqrt{m})$. The average depth of the tree can be regarded as $log(n)$, so the complexity of each tree is $O(n * log(n) * \sqrt{m})$. In five times 10-fold cross-validation, k_{CvTree} trees

Algorithm 5 Classification

Input: Test instance, Q; Number of trees, k_{Tree}; Transformation series and corresponding interval feature, $adaptiveFeatureSet$; Number of intervals extracted, $n_{Interval}$; FIT model, *trees*;
Output: Predicted class, *preClass*;
1: **for** i=1 to k_{Tree} **do**
2: **for** s=1 to Size($adaptiveFeatureSet$) **do**
3: **for** j=1 to $n_{Interval}$ **do**
4: $transInst$.Add($F_f(Q, intervals_j)$)
5: **end for**
6: **end for**
7: $classes \leftarrow$ ClassifyWithRandomTree($trees_i, transInst$)
8: **end for**
9: $preClass \leftarrow$ MaxOccurClass($classes$)
10: **return** *preClass*

are trained for s time series(here s denotes the number of transformed series), assuming a total of k_{cv} trees are trained. Therefore, the time complexity of the cross-validation phase is $O(k_{cv} * n * log(n) * \sqrt{m})$, where k_{cv} is the total number of trees in the cross-validation phase, n is the number of instances of the data set, and m is the length of the time series.

In the formal training phase, we use the features extracted in the cross-validation phase for formal training of the FIT model. Similar to the first stage, the complexity of each tree is $O(f * n * log(n) * \sqrt{m})$, where the number of features f can be regarded as a constant. Therefore, the complexity of each tree is $O(n * log(n) * \sqrt{m})$. The complexity of training k_{Tree} is $O(k_{Tree} * n * log(n) * \sqrt{m})$, wherein, k_{Tree} is the number of trees for constructing the classifier in the formal training phase.

Therefore, the total time complexity of the two stages is $O((k_{cv} + k_{Tree}) * n * log(n) * \sqrt{m})$. In simple terms, the time complexity of the FIT model is $O(k * n * log(n) * \sqrt{m})$, where k is the total number of trees in the cross-validation phase and the formal training phase.

4. Experiments

We conducted comprehensive experiments to evaluate the performance of FIT. We use the dataset of the UCR Time Series Classification Archive (Dau et al., 2018), which is the standard library in the TSC research field. We compared the accuracy and efficiency with the state-of-the-art TSC algorithms. Besides, we make a case study of the series transformation method.

All the experimental results are obtained on the computer with AMD Ryzen 5 4600U with Radeon Graphics (2.10 GHz), 16 GB. The algorithms are implemented by Java.

4.1. Accuracy comparison

First we evaluate the classification accuracy of FIT on 85 UCR time series datasets. In order to eliminate the randomness of the accuracy generated by random operations in the algorithm, the reported accuracy is the average of results by running 30 times. Here, FIT uses 200 trees in the forest. We use 11 baseline methods, i.e., DTW, BOSS, ST, PF, TS-CHIEF, TSF, TSBF, RISE, STSF (Cabello, Naghizade, Qi, and Kulik, 2020), FCN and TS2Vec (Yue et al., 2021) respectively. Most of these methods are among the state-of-the-art algorithms indicated by the website.[1] Wherein, the accuracy results of TS-CHIEF, STSF, PF and TS2Vec are from the papers of each algorithm, the accuracy of RISE comes from the published results in STSF, the accuracy of FCN comes from the published results in TSC review of deep learning (Fawaz et al., 2019), and the accuracy of other algorithms are obtained from the published results in TS-CHIEF (Shifaz et al., 2020).

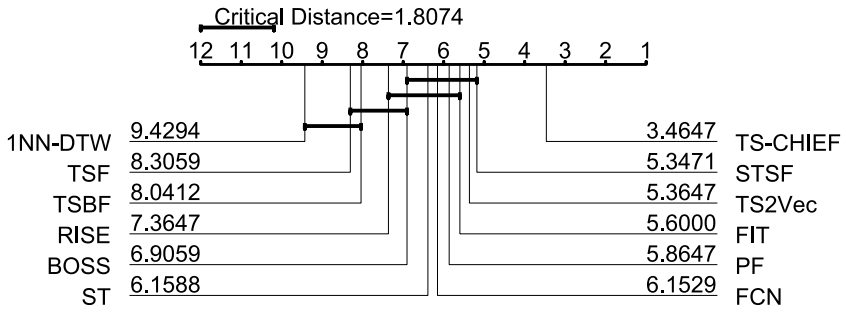

Fig. 2. The critical difference diagram over the accuracy average ranks of FIT and other 11 TSC methods on the 85 datasets of the UCR archive.

Table 1
The accuracy comparison among FIT and other 11 TSC classifiers.

Dataset	DTW	BOSS	TSF	PF	RISE	STSF	CHIEF	TSBF	ST	FIT	FCN	TS2Vec
Adiac	0.609	0.765	0.732	0.734	0.780	0.828	0.798	0.770	0.783	0.779	**0.844**	0.762
ArrHead	0.800	0.834	0.726	**0.875**	0.794	0.675	0.833	0.754	0.737	0.752	0.843	0.857
Beef	0.667	0.800	0.767	0.720	0.833	0.840	0.706	0.567	**0.900**	0.717	0.697	0.767
BeetFly	0.650	0.900	0.750	0.875	0.750	**0.940**	0.914	0.800	0.900	0.725	0.860	0.900
BirdChi	0.700	0.950	0.800	0.865	0.950	0.900	0.909	0.900	0.800	0.950	**0.955**	0.800
Car	0.767	0.833	0.767	0.847	0.800	0.815	0.855	0.783	**0.917**	0.808	0.905	0.833
CBF	0.994	0.998	0.994	0.993	0.951	0.979	0.998	0.988	0.974	0.988	0.994	**1.000**
ChloCon	0.650	0.661	0.720	0.634	0.769	0.780	0.717	0.692	0.700	0.752	0.814	**0.832**
CinCECG	0.930	0.887	0.983	0.934	**0.986**	0.985	0.983	0.712	0.954	0.983	0.824	0.827
Coffee	**1.000**	**1.000**	0.964	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.964	**1.000**	**1.000**	**1.000**
Comput	0.624	0.756	0.720	0.644	0.764	0.756	0.705	0.756	0.736	0.752	**0.822**	0.660
CricketX	0.780	0.736	0.664	0.802	0.697	0.683	**0.814**	0.705	0.772	0.627	0.792	0.782
CricketY	0.756	0.754	0.672	0.794	0.718	0.748	**0.802**	0.736	0.780	0.696	0.787	0.749
CricketZ	0.736	0.746	0.672	0.801	0.705	0.722	**0.834**	0.715	0.787	0.696	0.811	0.792
DiaSizRed	0.935	0.931	0.931	0.966	0.928	0.966	0.973	0.899	0.925	0.945	0.313	**0.984**
DisPhAG	0.626	0.748	0.748	0.731	0.763	0.728	0.746	0.712	**0.770**	0.723	0.710	0.761
DisPhOC	0.725	0.728	0.772	**0.793**	0.775	0.788	0.782	0.783	0.775	0.787	0.760	0.727
DisPhTW	0.633	0.676	0.669	0.660	0.676	0.683	0.670	0.676	0.662	0.663	0.690	**0.698**
Earthqua	0.727	0.748	0.748	0.754	0.748	**0.769**	0.748	0.748	0.741	0.752	0.727	0.748
ECG200	0.880	0.870	0.870	0.909	0.880	0.880	0.862	0.840	0.830	0.889	0.889	**0.920**
ECG5000	0.925	0.941	0.939	0.937	0.937	0.942	**0.945**	0.940	0.944	0.943	0.940	0.935
ECG5Days	0.797	**1.000**	0.956	0.849	0.999	0.978	**1.000**	0.877	0.984	**1.000**	0.987	**1.000**
ElecDev	0.631	**0.799**	0.693	0.706	0.664	0.741	0.755	0.703	0.747	0.738	0.702	0.721
FaceAll	0.808	0.782	0.752	0.894	0.762	0.789	0.841	0.744	0.779	0.731	**0.945**	0.771
FaceFour	0.898	**1.000**	0.932	0.974	0.898	0.977	**1.000**	**1.000**	0.852	0.992	0.928	0.932
FacesUCR	0.908	0.957	0.883	0.946	0.875	0.886	**0.966**	0.867	0.906	0.873	0.946	0.924
50Words	0.765	0.706	0.741	0.831	0.692	0.771	**0.845**	0.758	0.706	0.749	0.627	0.771
Fish	0.834	0.989	0.794	0.935	0.846	0.903	**0.994**	0.834	0.989	0.909	0.958	0.926
FordA	0.665	0.930	0.815	0.855	0.941	0.963	0.941	0.850	0.971	**0.981**	0.904	0.936
FordB	0.599	0.711	0.688	0.715	0.811	0.794	0.830	0.599	0.807	0.815	**0.878**	0.794
GunPoint	0.913	**1.000**	0.973	0.997	0.980	0.920	**1.000**	0.987	**1.000**	0.947	**1.000**	0.980
Ham	0.600	0.667	0.743	0.660	0.686	0.738	0.715	**0.762**	0.686	0.739	0.718	0.714
HandOut	0.878	0.903	0.919	0.921	0.884	0.920	**0.932**	0.854	**0.932**	0.923	0.806	0.922
Haptics	0.416	0.461	0.445	0.445	0.458	0.508	0.517	0.490	0.523	0.495	0.480	**0.526**
Herring	0.531	0.547	0.609	0.580	0.641	0.630	0.588	0.641	**0.672**	0.627	0.608	0.641
InlSkate	0.387	0.516	0.376	0.542	0.349	0.555	0.527	0.386	0.373	**0.636**	0.339	0.415
InsWinSou	0.574	0.523	0.633	0.619	0.655	**0.666**	0.643	0.625	0.627	0.655	0.393	0.630
ItPowDem	0.955	0.909	0.960	0.967	0.953	**0.971**	**0.971**	0.883	0.948	0.969	0.961	0.925
LaKitAp	0.795	0.765	0.571	0.782	0.637	0.794	0.807	0.528	0.859	0.782	**0.902**	0.845
Light2	**0.869**	0.836	0.803	0.866	0.705	0.725	0.748	0.738	0.738	0.767	0.739	**0.869**
Light7	0.712	0.685	0.753	0.822	0.699	0.770	0.763	0.726	0.726	0.760	0.827	**0.863**
Mallat	0.914	0.938	0.919	0.958	0.922	0.969	**0.975**	0.960	0.964	0.961	0.967	0.914
Meat	0.933	0.900	0.933	0.933	0.933	0.932	0.888	0.933	0.850	0.933	0.853	**0.950**
MedImg	0.747	0.718	0.755	0.758	0.662	0.786	**0.796**	0.705	0.670	0.794	0.779	0.789
MidPhAG	0.520	0.546	0.578	0.562	0.597	0.568	0.583	0.578	0.643	0.592	0.553	**0.838**
MidPhOC	0.766	0.780	0.828	0.836	0.821	0.823	**0.854**	0.814	0.794	0.831	0.801	0.636
MidPhTW	0.507	0.546	0.565	0.529	0.591	0.589	0.550	**0.597**	0.520	0.592	0.512	0.584
MotStr	0.866	0.879	0.869	0.902	0.872	0.924	**0.948**	0.903	0.897	0.891	0.937	0.861

(*continued on next page*)

Table 1 shows the accuracy comparison between FIT and the other 11 baseline methods. From Table 1, we can observe that FIT ranks 4th among 12 methods in terms of average accuracy on 85 datasets, reaching 82.4%. The accuracy obtained by FIT is closer to that of TS2Vec and STSF. However, compared with TS2Vec, FIT can achieve more than 15% improvement in accuracy on some datasets, such as CinCECG, InlSkate, and UWaAll. TS-CHIEF can obtain the best average accuracy 84.6%, because TS-CHIEF uses a more diverse feature space. We also can observe FIT can achieve better accuracy than ST, PF, BOSS, RISE, TSF, TSBF, and FCN.

Fig. 2 shows the critical difference diagram for Nemenyi test (Demšar, 2006) over the accuracy average ranks of 12 classifiers. We can observe from Fig. 2 that there is no significant difference

[1] https://www.timeseriesclassification.com/.

Table 1 (*continued*).

Dataset	DTW	BOSS	TSF	PF	RISE	STSF	CHIEF	TSBF	ST	FIT	FCN	TS2Vec
NoECGT1	0.829	0.838	0.876	0.907	0.901	0.933	0.911	0.842	0.950	0.915	**0.956**	0.930
NoECGT2	0.870	0.901	0.910	0.940	0.916	0.941	0.945	0.862	0.951	0.930	**0.953**	0.938
OliveOil	0.867	0.867	0.867	0.867	0.900	**0.933**	0.888	0.833	0.900	0.887	0.723	0.900
OSULeaf	0.599	0.955	0.583	0.827	0.649	0.798	**0.991**	0.760	0.967	0.848	0.977	0.851
PhalOC	0.761	0.772	0.803	0.824	0.814	0.832	**0.845**	0.830	0.763	0.839	0.820	0.809
Phoneme	0.227	0.265	0.212	0.320	0.356	0.325	**0.369**	0.276	0.321	0.326	0.325	0.312
Plane	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.999	**1.000**	**1.000**
ProPhAG	0.785	0.834	0.849	0.846	0.854	0.844	0.850	0.849	0.844	0.842	0.831	**0.887**
ProPhOC	0.790	0.849	0.828	0.873	0.876	**0.905**	0.888	0.873	0.883	0.888	0.903	0.834
ProPhTW	0.761	0.800	0.815	0.779	0.810	0.765	0.819	0.810	0.805	0.774	0.767	**0.824**
RefrigDev	0.440	0.499	**0.589**	0.532	0.544	0.580	0.558	0.472	0.581	0.552	0.508	**0.589**
ScrType	0.411	0.464	0.456	0.455	0.528	0.533	0.508	0.509	0.520	0.536	**0.625**	0.411
ShapSim	0.694	**1.000**	0.478	0.776	0.783	0.983	**1.000**	0.961	0.956	0.940	0.724	**1.000**
ShapAll	0.802	0.908	0.792	0.886	0.833	0.852	**0.930**	0.185	0.842	0.831	0.895	0.902
SmKitAp	0.672	0.725	0.811	0.744	0.811	**0.834**	0.822	0.672	0.792	0.806	0.783	0.731
SonyAIR1	0.696	0.632	0.787	0.846	0.822	0.907	0.826	0.795	0.844	0.915	**0.960**	0.903
SonyAIR2	0.859	0.859	0.810	0.896	0.911	0.833	0.925	0.778	0.934	0.940	**0.979**	0.871
StarCur	0.898	0.978	0.969	0.981	0.975	0.978	**0.982**	0.977	0.979	0.981	0.961	0.969
Strawber	0.946	**0.976**	0.965	0.968	0.965	0.964	0.966	0.954	0.962	0.966	0.972	0.962
SwedLeaf	0.846	0.922	0.914	0.947	0.936	0.943	0.966	0.915	0.928	0.960	**0.969**	0.941
Symbols	0.938	0.967	0.915	0.962	0.933	0.884	**0.977**	0.946	0.882	0.918	0.955	0.976
SynCon	0.983	0.967	0.987	0.995	0.667	0.990	**0.998**	0.993	0.983	0.987	0.985	0.997
ToeSeg1	0.750	0.939	0.741	0.925	0.908	0.844	**0.965**	0.781	**0.965**	0.856	0.961	0.917
ToeSeg2	0.908	**0.962**	0.815	0.862	0.900	0.885	0.954	0.800	0.908	0.851	0.880	0.892
Trace	0.990	**1.000**	0.990	**1.000**	0.960	0.990	**1.000**	0.980	**1.000**	0.996	**1.000**	**1.000**
2LeECG	0.868	0.981	0.759	0.989	0.888	0.987	0.995	0.866	0.997	0.924	**1.000**	0.986
2Patterns	0.999	0.993	0.991	**1.000**	0.435	0.998	**1.000**	0.976	0.955	0.994	0.871	**1.000**
UWaAll	0.962	0.939	0.957	**0.972**	0.921	0.955	0.969	0.926	0.942	0.959	0.817	0.795
UWaX	0.774	0.762	0.804	0.829	0.619	0.811	**0.841**	0.831	0.803	0.823	0.754	0.719
UWaY	0.702	0.685	0.727	0.762	0.667	0.742	**0.772**	0.736	0.730	0.742	0.639	0.770
UWaZ	0.675	0.695	0.743	0.764	0.650	0.759	0.784	0.773	0.749	0.768	0.726	**0.930**
Wafer	0.996	0.995	0.996	0.996	0.996	**1.000**	0.999	0.995	**1.000**	0.999	0.997	0.998
Wine	0.611	0.741	0.630	0.569	0.648	0.669	**0.891**	0.611	0.796	0.757	0.587	0.870
WordSyn	0.749	0.638	0.647	0.779	0.589	0.636	**0.787**	0.688	0.571	0.697	0.564	0.676
Worms	0.533	0.558	0.610	0.718	0.662	0.768	**0.802**	0.688	0.740	0.762	0.765	0.701
Worms2	0.584	**0.831**	0.623	0.784	**0.831**	0.791	0.816	0.753	**0.831**	0.829	0.726	0.805
Yoga	0.843	**0.918**	0.859	0.879	0.818	0.828	0.835	0.819	0.818	0.879	0.839	0.887
Avg.Acc	0.759	0.810	0.779	0.819	0.788	0.826	**0.846**	0.778	0.822	0.824	0.809	0.829
Avg.Rank	9.429	6.906	8.306	5.865	7.365	5.347	**3.465**	8.041	6.159	5.600	6.153	5.365

between FIT and STSF, TS2Vec, PF, FCN. Fig. 2 also shows that there is a significant difference between FIT and TS-CHIEF, indicating the competitiveness of TS-CHIEF. In addition, FIT significantly outperforms 1NN-DTW, TSF, and TSBF.

In order to observe the accuracy comparison between the two classifiers more intuitively, we show the scatter plot of the accuracy of the pairwise comparison between FIT and other algorithms in Fig. 3. From the scatter plot between FIT and TSF, we can see that FIT performs better on 68 datasets. A large number of data points located above the diagonal of the scatter plot, which indicates that FIT has better performance over TSF in terms of accuracy. The scatter plot of the comparison between FIT and the deep learning algorithm FCN shows that, the data points located in the upper left are farther away from the diagonal position. This indicates that FIT has better classification advantages over FCN on some datasets. Compared FIT with TS-CHIEF, our FIT wins 22 datasets. However, it can be seen that a large number of data points are basically located near the diagonal, indicating that the accuracy of FIT on these datasets is not very different from that of TS-CHIEF. Because TS-CHIEF uses multiple feature representations for integration, it is difficult for us to achieve the accuracy by using FIT only with interval features. The scatter plots by comparising FIT with DTW, BOSS, PF, RISE, TSBF, ST show that FIT can win more datasets, which can also prove that FIT has better classification performance.

4.2. Time comparison

We compare the training time and testing time between FIT and other three algorithms, TSF, PF and BOSS respectively. The running time of these four algorithms is obtained by using the same programming language on the same machine. Due to the limited experimental equipment, the time results of FIT and TSF are obtained by running 10 times, and the time results of PF and BOSS are obtained by running once. This has no influence on BOSS, because the result of BOSS is fixed. But it may have influence on PF, because the PF algorithm has a random process.

Table 2
The total time required to train the classifier on 79 UCR datasets and the average time for each dataset. The time unit is minute.

	TSF	FIT	BOSS	PF
Mean time	0.1472	1.6082	14.1066	17.3717
Total time	11.6270	127.0509	1114.4185	1372.3620

The training time of the four classifiers on 79 UCR datasets is shown in Table 2. We have not made time comparison on the remaining six datasets because using BOSS training model takes up too much memory, which leads to memory overflow. It can be seen from Table 2 that TSF has the shortest training time, followed by FIT. Here the training time of FIT is about 10 times that of TSF. The training time of BOSS and PF is about 10 times that of FIT.

The testing time of the four classifiers on 79 UCR datasets is shown in Table 3. We can observe that the testing time of FIT is greater than that of TSF, which is close to the testing time of BOSS. The testing time of PF is still very long, close to its training time. The reason is that the PF algorithm uses similarity measure, so that it costs more time to calculate distance.

There are two main reasons for the increase in time consumption compared to TSF. On the one hand, series transformation costs extra time. On the other hand, we make cross-validation to obtain the appropriate series transformation method and interval features

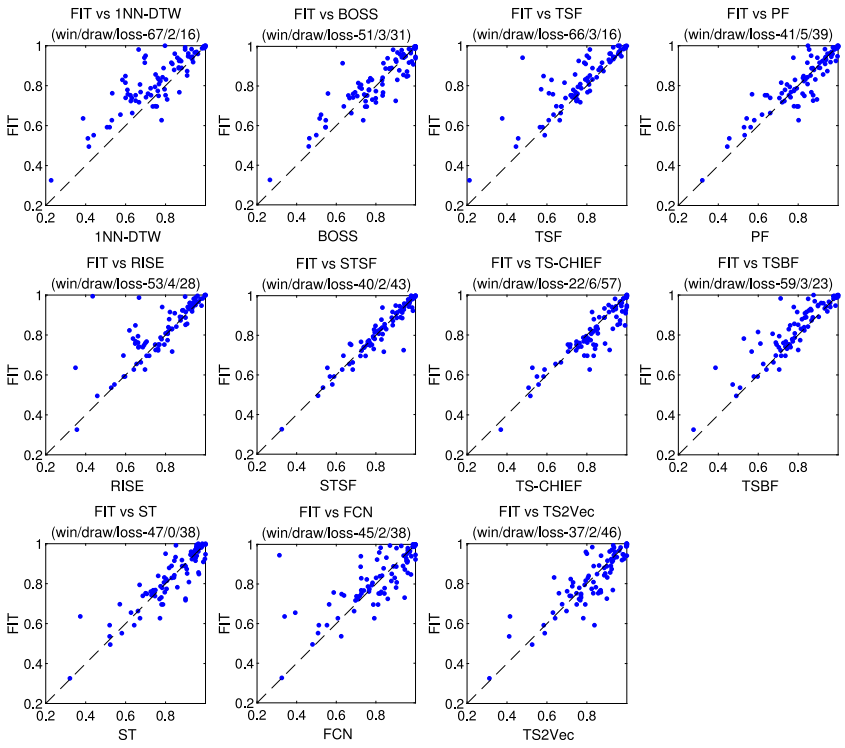

Fig. 3. The scatter plot of accuracy comparison between FIT and other 11 competitive methods. Each data point represents a dataset. The data point located above the diagonal indicates that the accuracy of FIT on this dataset is better than the corresponding competitive method.

suitable for the current dataset, and it costs time to build trees in the cross-validation stage.

4.3. Case study

In this section, we conduct a case study of the series transformation method. We use the ShapeletSim dataset to investigate the effect of the PS series transformation method. In Fig. 4, we can clearly observe that there is an obvious difference between class 1 and class 2, that is, the segment marked with a red box in series of class 2. This segment will be a discrimination feature for classifying the two classes. However, as shown in Fig. 5(c), given two series belonging to class 2, the position of the segment appearing in their interval is very different, and using the interval feature of the original series in Fig. 5(c) cannot solve this phase deviation problem in the same class.

Fig. 5(d) shows the PS series from the interval of two original series in class 2 shown in Fig. 5(c). Since the original series have large fluctuations, there still exists fluctuations in Fig. 5(d), but we can observe that the PS series of the two intervals have crests at the third and fourth time points, which are significantly different from other positions. Using the maximum feature of this interval can solve the phase deviation problem, so PS series transformation can effectively use the interval feature that cannot be used in the original series to effectively make classification.

In addition, comparing the interval between the two original series of class 1 in Fig. 5(a) and the corresponding PS series in Fig. 5(b), it can be clearly seen that Fig. 5(d) can clearly present the peaks while Fig. 5(b) cannot present them. It clarifies that using this feature after PS series transformation can effectively achieve effective classification. Therefore, PS series can distinguish the inter-class differences between Fig. 5(b) and (d). At the same time, it can also eliminate the intra-class difference, as the phase deviation in Fig. 5(c) is shown in Fig. 5(d).

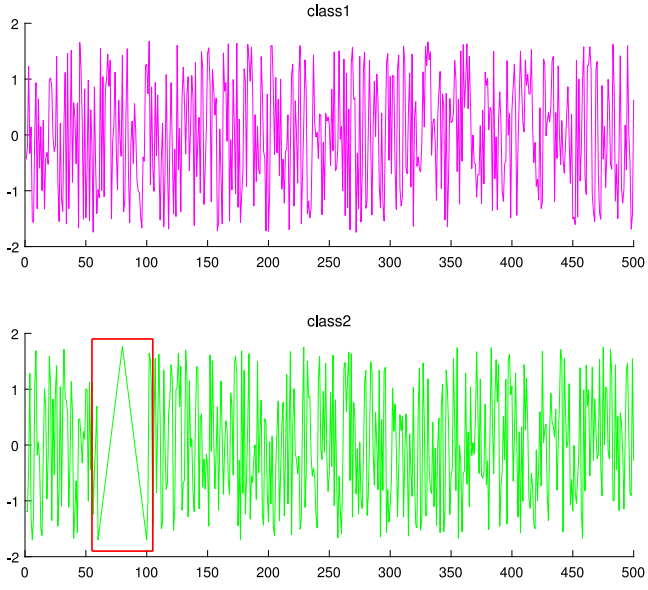

Fig. 4. The original series of class 1 and class 2 of the ShapeletSim dataset.

Table 3
The total time required to test the classifier on 79 UCR datasets and the average time for each dataset. The time unit is minute.

	TSF	FIT	BOSS	PF
Mean time	0.0470	1.3775	1.4679	16.3285
Total time	3.7133	108.8249	115.9656	1289.9550

5. Conclusion

In this paper, we propose a new TSC algorithm called FIT by selecting time series interval features. FIT first extracts the appropriate series transformation method for the current dataset through cross-validation,

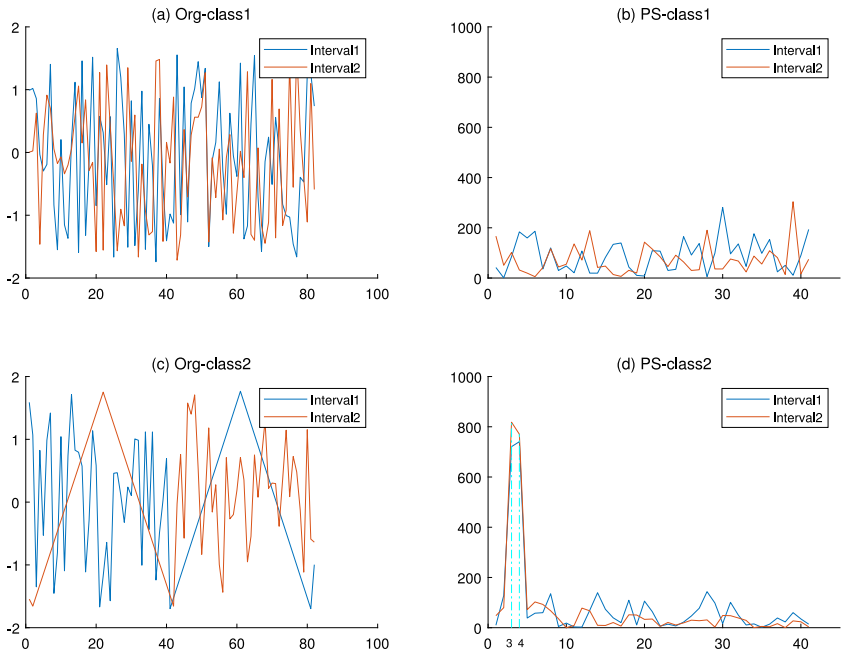

Fig. 5. Case study on ShapeletSim dataset. (a) The interval of two original series in class 1. (b) The PS series from Fig. 5(a). (c) The interval of two original series in class 2. (d) The PS series from Fig. 5(c).

and then selects the appropriate interval feature through the number of feature nodes of the classifier constructed in the cross-validation, and uses the obtained interval feature set to train the FIT model. Experiments show that FIT can obtain high classification accuracy on the 85 datasets of UCR archives. In addition, FIT is also an efficient algorithm, which can have lower time complexity after obtaining the transformation series.

In future work, we will study more effective interval extraction methods, so that the classification accuracy will be greatly improved. At the same time, we think it is worth trying to extend FIT to multivariate time series classification tasks.

CRediT authorship contribution statement

Guiling Li: Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing, Funding acquisition. **Shaolin Xu:** Methodology, Software, Visualization, Formal analysis, Writing – original draft, Writing – review & editing. **Senzhang Wang:** Methodology, Writing – original draft, Writing – review & editing. **Philip S. Yu:** Formal analysis, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The authors would like to thank Prof. Eamonn Keogh and all the people who have contributed to the UCR time series classification archive for their selfless work.

The work is supported by the National Natural Science Foundation of China (No. 61702468), Open Research Project of The Hubei Key Laboratory of Intelligent Geo-Information Processing, China(No. KLIGIP-2018B03).

References

Bagnall, A., Flynn, M., Large, J., Lines, J., & Middlehurst, M. (2020). On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (hive-cote v1. 0). In *International workshop on advanced analytics and learning on temporal data* (pp. 3–18). Springer.

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*(3), 606–660. http://dx.doi.org/10.1007/s10618-016-0483-9.

Bagnall, A., Lines, J., Hills, J., & Bostrom, A. (2015). Time-series classification with COTE: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, *27*(9), 2522–2535. http://dx.doi.org/10.1109/TKDE.2015.2416723.

Baydogan, M. G., Runger, G., & Tuv, E. (2013). A bag-of-features framework to classify time series. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(11), 2796–2802. http://dx.doi.org/10.1109/TPAMI.2013.72.

Cabello, N., Naghizade, E., Qi, J., & Kulik, L. (2020). Fast and accurate time series classification through supervised interval search. In *2020 IEEE International Conference on Data Mining (ICDM)* (pp. 948–953). IEEE.

Dau, H. A., Keogh, E., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., et al. (2018). The UCR time series classification archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

Dempster, A., Petitjean, F., & Webb, G. I. (2020). ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 1–42. http://dx.doi.org/10.1007/s10618-020-00701-z.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*(Jan), 1–30.

Deng, H., Runger, G., Tuv, E., & Vladimir, M. (2013). A time series forest for classification and feature extraction. *Information Sciences*, *239*, 142–153. http://dx.doi.org/10.1016/j.ins.2013.02.030.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, *33*(4), 917–963. http://dx.doi.org/10.1007/s10618-019-00619-1.

Fawaz, H. I., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., et al. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, *34*(6), 1936–1962. http://dx.doi.org/10.1007/s10618-020-00710-y.

Fulcher, B. D., & Jones, N. S. (2017). Hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell Systems*, *5*(5), 527–531. http://dx.doi.org/10.1016/j.cels.2017.10.001.

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).

He, G., Xin, X., Peng, R., Han, M., Wang, J., & Wu, X. (2020). Online rule-based classifier learning on dynamic unlabeled multivariate time series data. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, http://dx.doi.org/10.1109/TSMC.2020.3012677.

Hills, J., Lines, J., Baranauskas, E., Mapp, J., & Bagnall, A. (2014). Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, *28*(4), 851–881. http://dx.doi.org/10.1007/s10618-013-0322-1.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, *29*(6), 82–97. http://dx.doi.org/10.1109/MSP.2012.2205597.

Ji, C., Zhao, C., Pan, L., Liu, S., Yang, C., & Wu, L. (2017). A fast shapelet discovery algorithm based on important data points. *International Journal of Web Services Research (IJWSR)*, *14*(2), 67–80. http://dx.doi.org/10.4018/IJWSR.2017040104.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*.

Li, G., Yan, W., & Wu, Z. (2019). Discovering shapelets with key points in time series classification. *Expert Systems with Applications*, *132*, 76–86. http://dx.doi.org/10.1016/j.eswa.2019.04.062.

Lin, J., Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, *15*(2), 107–144. http://dx.doi.org/10.1007/s10618-007-0064-z.

Lin, J., Khade, R., & Li, Y. (2012). Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, *39*(2), 287–315. http://dx.doi.org/10.1007/s10844-012-0196-5.

Lines, J., & Bagnall, A. (2015). Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, *29*(3), 565–592. http://dx.doi.org/10.1007/s10618-014-0361-2.

Lines, J., Taylor, S., & Bagnall, A. (2018). Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, *12*(5), http://dx.doi.org/10.1145/3182382.

Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., et al. (2019). Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, *33*(3), 607–635. http://dx.doi.org/10.1007/s10618-019-00617-3.

Middlehurst, M., Vickers, W., & Bagnall, A. (2019). Scalable dictionary classifiers for time series classification. In *International conference on intelligent data engineering and automated learning* (pp. 11–19). Springer.

Pourbabaee, B., Roshtkhari, M. J., & Khorasani, K. (2018). Deep convolutional neural networks and learning ECG features for screening paroxysmal atrial fibrillation patients. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *48*(12), 2095–2104. http://dx.doi.org/10.1109/TSMC.2017.2705582.

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). SIAM.

Savadkoohi, M., Oladunni, T., & Thompson, L. A. (2021). Deep neural networks for human's fall-risk prediction using force-plate time series signal. *Expert Systems with Applications*, *182*, Article 115220. http://dx.doi.org/10.1016/j.eswa.2021.115220.

Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, *29*(6), 1505–1530. http://dx.doi.org/10.1007/s10618-014-0377-7.

Schäfer, P. (2016). Scalable time series classification. *Data Mining and Knowledge Discovery*, *30*(5), 1273–1298. http://dx.doi.org/10.1007/s10618-015-0441-y.

Schäfer, P., & Leser, U. (2017). Fast and accurate time series classification with weasel. In *Proceedings of the 2017 ACM on conference on information and knowledge management* (pp. 637–646).

Senin, P., & Malinchik, S. (2013). Sax-vsm: Interpretable time series classification using sax and vector space model. In *2013 IEEE 13th international conference on data mining* (pp. 1175–1180). IEEE.

Sharabiani, A., Darabi, H., Rezaei, A., Harford, S., Johnson, H., & Karim, F. (2017). Efficient classification of long time series by 3-d dynamic time warping. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(10), 2688–2703. http://dx.doi.org/10.1109/TSMC.2017.2699333.

Shifaz, A., Pelletier, C., Petitjean, F., & Webb, G. I. (2020). TS-CHIEF: a scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, *34*(3), 742–775. http://dx.doi.org/10.1007/s10618-020-00679-8.

Tay, F. E., & Cao, L. (2001). Application of support vector machines in financial time series forecasting. *Omega*, *29*(4), 309–317. http://dx.doi.org/10.1016/S0305-0483(01)00026-3.

Wang, J., Liu, P., She, M. F., Nahavandi, S., & Kouzani, A. (2013). Bag-of-words representation for biomedical time series classification. *Biomedical Signal Processing and Control*, *8*(6), 634–644. http://dx.doi.org/10.1016/j.bspc.2013.06.004.

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 international joint conference on neural networks (IJCNN)* (pp. 1578–1585). IEEE.

Ye, L., & Keogh, E. (2009). Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 947–956).

Yue, Z., Wang, Y., Duan, J., Yang, T., Huang, C., Tong, Y., et al. (2021). TS2Vec: Towards universal representation of time series. CoRR, abs/2106.10466.