Data Mining and Knowledge Discovery (2024) 38:748–811
https://doi.org/10.1007/s10618-023-00978-w

Fast, accurate and explainable time series classification through randomization

Nestor Cabello[1] · Elham Naghizade[2] · Jianzhong Qi[1] · Lars Kulik[1]

Received: 30 November 2022 / Accepted: 22 August 2023 / Published online: 16 October 2023

Abstract

Time series classification (TSC) aims to predict the class label of a given time series, which is critical to a rich set of application areas such as economics and medicine. State-of-the-art TSC methods have mostly focused on classification accuracy, without considering classification speed. However, efficiency is important for big data analysis. Datasets with a large training size or long series challenge the use of the current highly accurate methods, because they are usually computationally expensive. Similarly, classification explainability, which is an important property required by modern big data applications such as *appliance modeling* and legislation such as the *European General Data Protection Regulation*, has received little attention. To address these gaps, we propose a novel TSC method – the *Randomized-Supervised Time Series Forest* (r-STSF). r-STSF is extremely fast and achieves state-of-the-art classification accuracy. It is an efficient interval-based approach that classifies time series according to aggregate values of the discriminatory sub-series (intervals). To achieve state-of-the-art accuracy, r-STSF builds an ensemble of randomized trees using the discriminatory sub-series. It uses four time series representations, nine aggregation functions and a supervised binary-inspired search combined with a feature ranking metric to identify highly discriminatory sub-series. The discriminatory sub-series enable explainable classifications. Experiments on extensive datasets show that r-STSF achieves state-of-the-art accuracy while being orders of magnitude faster than most existing TSC methods and enabling for explanations on the classifier decision.

Keywords Time series classification · Interval-based classifier · Feature selection · Randomized trees · Explainable classifier

Responsible editor: Johannes Fürnkranz.

Extended author information available on the last page of the article

1 Introduction

Time series classification (TSC) aims to predict the class label of a given time series (or its *feature-based representation*). A time series is an ordered time-stamped sequence of observations from a variable of interest. Various TSC methods have been proposed for a rich set of application areas such as economics (e.g., financial analysis (Pattarin et al. 2004)) and medicine (e.g., classification of electrocardiograms (Karpagachelvi et al. 2012)). Up to now, the key focus for TSC was on accuracy. However, while current techniques are able to generally achieve high levels of accuracy across a range of datasets, an important criterion has been largely unaddressed: the ability to classify large datasets. Most of current state-of-the-art TSC methods are impractical (i.e., they are computationally and/or memory expensive) when classifying large datasets. A recent approach, ROCKET (Dempster et al. 2020), has recognized the need for highly efficient TSC approaches and outperforms current state-of-the-art (SOTA) TSC approaches. The key contribution of our paper is to significantly increase the efficiency compared to ROCKET as the currently fastest approach. Our paper is the first that provides a comprehensive evaluation of TSC approaches for large datasets addressing an urgent research issue.

SOTA TSC methods such as the *hierarchical vote collective of transformation-based ensembles* (HIVE-COTE) (Lines et al. 2018) and *time series combination of heterogeneous and integrated embedding forest* (TS-CHIEF) (Shifaz et al. 2020) are highly accurate, but inefficient in terms of time and memory. TS-CHIEF's complexity is quadratic and HIVE-COTE has a bi-quadratic time complexity in the length of the time series. Recent variations of HIVE-COTE such as HIVE-COTE v1.0 (HC1) (Bagnall et al. 2020) and HIVE-COTE v2.0 (HC2) (Middlehurst et al. 2021) are no longer bi-quadratic, but are still among the slowest TSC methods. Another method, InceptionTime (Fawaz et al. 2020) recently reported SOTA classification accuracy with faster training times. However, while InceptionTime is faster than HIVE-COTE and its variations, this is largely due to the use of a GPU. When running on a CPU, InceptionTime is approximately two times slower than HC1 and HC2.

The *RandOm Convolutional KErnel Transform* (ROCKET) (Dempster et al. 2020) has recently claimed SOTA classification accuracy and faster training times. ROCKET is fast at training, approximately two orders of magnitude faster than HC1 and HC2. However, at testing, the difference between ROCKET and HC1 and HC2 is significantly reduced to one order of magnitude. ROCKET is currently the best suited SOTA TSC method for big data, however, its testing time could represent a drawback when classifying large datasets with very long series (Middlehurst et al. 2021).

Neither ROCKET or any other SOTA TSC method report a mechanism to explain their classifications. As these methods extract complex features based on latent features from many different time series representations/transformations, the integration of explainability is difficult. Whilst it might be possible to apply methods that provide post-hoc explanations for black box models, we will not discuss them as they are beyond the scope of this paper. Instead, we focus on techniques that integrate explanations by design. For time series it is key to identify a subset of features, in our case time intervals, that can explain which feature led to a certain classification.

In this paper, we propose a novel TSC method: the *Randomized-Supervised Time Series Forest* (r-STSF). r-STSF is the fastest TSC method at both training and testing; and to the best of our knowledge, r-STSF is the only TSC method that integrates explainability and achieves SOTA classification accuracy (cf. Fig. 1). As shown in Fig. 1, r-STSF is the fastest TSC method. The SOTA TSC methods such as ROCKET, HC1 and InceptionTime (ITime) are slower and do not integrate explanations. All other depicted methods do not achieve SOTA accuracy. The TSC methods HC2 and DrCIF were published while this paper was under review (hence they are in grey). HC2 is the most accurate SOTA TSC method and DrCIF is statistically as accurate as r-STSF. Both approaches are two orders of magnitude slower than r-STSF.

r-STSF is a tree-based ensemble classifier whose trees are grown using features derived from summary statistics over a number of randomly selected sub-series (sub-intervals). Classifiers based on this paradigm are known as interval-based TSC methods. Typical examples of such classifiers include *time series forest* (TSF) (Deng et al. 2013), *time series bag-of-features* (TSBF) (Baydogan et al. 2013), *learned*

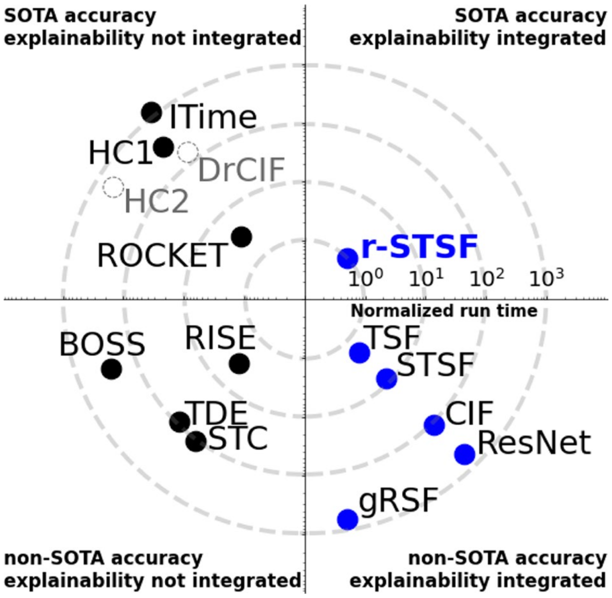


Fig. 1 Total run time (i.e., training plus testing time) of TSC methods when classifying 112 benchmark time series datasets normalized to r-STSF, i.e., r-STSF = 1. Each subsequent dashed circle represent an increase in the run time by a factor of 10. TSC methods that integrate explainability are in blue color; methods that do not integrate explainability are in black color. *r-STSF is the fastest approach and is the only TSC method that achieves both SOTA accuracy and integrates explainability*. All methods are implemented in Python and run under the same environment as detailed in Sect. 5 (Color figure online)

pattern similarity (LPS) (Baydogan and Runger 2016), the *supervised time series forest* (STSF) (Cabello et al. 2020), *canonical interval forest* (CIF) Middlehurst et al. (2020a), and *diverse representation canonical interval forest* (DrCIF) Middlehurst et al. (2021). Interval-based methods classify time series according to *discriminatory phase-dependent intervals* (discriminatory intervals for short hereafter), i.e., sub-series located at the same time regions over all time series that maximize class separability. As Fig. 2 shows, *interval 1* is an example of a discriminatory interval. This interval differentiates the blue time series from the red ones. In contrast, *interval 2* is non-discriminatory because it cannot separate the red and the blue time series.

Interval-based methods are generally fast and memory-efficient, while their tree-based structure in tandem with meaningful features (the *interval features*) integrate explainable classifications by design. The interval features are derived from simple transformations (i.e., statistics such as mean or standard deviation) over a number of sub-series and their meaning is easy to understand. The interactions between these features are modeled by binary trees, which are interpretable (Breiman 2001).

Our previous work (Cabello et al. 2020) proposed a highly accurate interval-based classifier, STSF, which showed that a "supervised" selection of intervals is significantly more efficient compared to one that is completely at random. Similar to TSF (Bagnall et al. 2017), it trains an ensemble of trees. The original TSF approach grows an ensemble of trees for classification, and a different set of intervals features is computed at node level (on each of the trees from the ensemble). STSF, however, extracts a set of candidate discriminatory interval features for the training of each tree classifier instead of each tree node. Similar to TSF, STSF uses the interval features that split the tree nodes (i.e., discriminatory interval features) to integrate explainability into the classification results. STSF proposes the regions of interests (ROIs) to highlight the location of discriminatory intervals, which are the intersected regions of such intervals.

To achieve high classification accuracy without relying on complex features, *r-STSF* uses a novel perturbation scheme to grow an ensemble of uncorrelated trees. Uncorrelated trees reduce the variance of the ensemble and increase the classification accuracy (Louppe et al. 2013). The perturbation scheme consists of (i) random partitions of intervals when searching for discriminatory features and (ii) an ensemble of randomized trees. To allow for each tree in the ensemble to use a different set

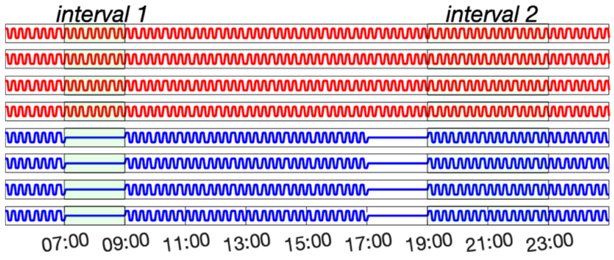


Fig. 2 Discriminatory *interval 1* versus non-discriminatory *interval 2*

of interval features for classification (i.e., ensemble of uncorrelated trees), r-STSF randomly partitions the sub-series when searching for discriminatory interval features. In comparison, our previous STSF method uses fixed partitions (following a binary search-based scheme) to extract interval features. Fixed partitions are not suitable to create uncorrelated trees, because they may allow for the repetition of some interval features across different trees in the ensemble. We show that random instead of fixed partitions makes r-STSF significantly more accurate. To further decrease the variance of ensembles, r-STSF adopts the extra-tree (ET) algorithm (Geurts et al. 2006), which uses randomized trees. In such randomized trees, the cut-point of each feature is randomly selected when looking for the feature that provides the best split. These trees are different from those used in random forest (RF), where for each feature only the best cut-point (i.e., split with the lowest entropy) is selected.

Although trees built by using RF are also considered randomized trees, they are only *weakly randomized* because the cut point selection still relies on the lowest entropy. Methods based on ET, however, are *strongly randomized* because the cut point selection is random. Thus, we will use the term *randomized trees* exclusively for trees built using the ET algorithm; all other type of trees will be considered non-randomized. We show that randomized trees instead of regular or non-randomized trees (as those based on RF) significantly increase the effectiveness of r-STSF.

Further, we introduce two new aggregation functions to form new features which are shown to improve the classification accuracy substantially: (i) the number of intersections of a sub-series with the mean axis of the entire sub-series, denoted as counts of mean-crossings in this paper, and (ii) the number of data points above the mean of a sub-series. These aggregation functions capture the shape of the time series and increase the classification accuracy more than generic statistics such as the mean or standard deviation.

To represent the temporal structure of time series data, we also include *autoregressive representations* of the series in our feature extraction process. This representation models a time series with an autoregressive process, where a value of a time series is represented as a linear combination of (some of) its previous values. r-STSF computes autoregressive representations of the raw time series (via the autoregression coefficients), and it captures lagged relationships of time series data points by extracting discriminatory sub-series from such representations. Our experiments show that the extraction of features from the autoregressive representation increases the classification accuracy considerably (i.e., more than 5%) for certain datasets.

r-STSF is very fast because it does not need to extract a set of candidate interval features for each tree classifier as STSF does. The training process of r-STSF is designed to require a small number (denoted by d) of sets of candidate discriminatory interval features. r-STSF extracts d sets of candidate interval features and merges all sets into a superset $\mathscr{F}$. Then, each of the r trees in the ensemble is built by using a set of randomly selected interval features from $\mathscr{F}$. Our experiments show that by setting $d = 0.1 \times r$, r-STSF achieves a similar classification accuracy (to computing a set of interval features for each tree) but being an order of magnitude faster.

An extensive experimental study on 112 time series datasets (Bagnall et al. 2019) shows that r-STSF is not significantly different in accuracy than most of the current SOTA classifiers, except for HC2, which was developed while this paper was under review. Whilst HC2 is more accurate than r-STSF, its long running time renders it impractical for classifying large time series datasets. Another approach, DrCIF which was proposed jointly with HC2, is slightly more accurate but not competitive in terms of runtime as it is two orders of magnitude slower. HC2 and DrCIF were published while r-STSF was under review. When reporting our results (e.g., ranking) comparing r-STSF against evaluated TSC methods we use the notation $b(a)$, where b denotes r-STSF's position *before* HC2 and DrCIF, whereas a includes them. r-STSF is ranked as the fourth (sixth) best classifier (according to the average ranks). Among all TSC methods that integrate explainability, only r-STSF achieves SOTA classification accuracy. Since r-STSF is at least two orders of magnitude faster than most of the SOTA TSC methods, it is ideal for classifying large data sets with long series. Its explainability supports applications requiring insights on the classifier decision.

In our previous work STSF (Cabello et al. 2020), we show that computing interval features in a supervised manner from different time series representations by using robust statistics is a highly efficient and accurate interval-based classification strategy. In this article, we design r-STSF with with the following new contributions:

- We propose a novel TSC method, r-STSF, which is currently the fastest current TSC method (Sect. 5.3). r-STSF is highly competitive to SOTA TSC methods in terms of average ranks and its interval-based design offers explainable classification results (Sect. 7). For large time series datasets, r-STSF is also the most suitable (i.e., accuracy-efficiency trade-off) TSC method (Sect. 5.3.1).
- We propose a novel perturbation scheme to create an ensemble of uncorrelated trees which improves the classification accuracy of interval-based techniques. Our scheme employs (i) random partitions when assessing the discriminatory quality of sub-series (Sect. 4.2) and (ii) randomized trees to build the ensemble of trees for classification (Sect. 4.3). Our selection of sets of sub-series can decrease the correlation of trees in the ensemble without affecting the strength of each tree.
- r-STSF's redesigned training process (Sect. 4.4) shows that by (i) computing and merging a small number of sets of candidate discriminatory interval features into a superset $\mathscr{F}$, and (ii) randomly selecting interval features from $\mathscr{F}$ to build each tree classifier, we achieve a similar classification accuracy (to computing a set of interval features for each tree) but being an order of magnitude faster.
- We show that time-specific aggregation functions (e.g., slope) have a larger positive impact in the classification accuracy than generic statistics (e.g., mean). To capture details of the shape of the sub-series, we propose two time-specific aggregation functions: (i) the counts of mean-crossings, and (ii) the counts of data points above the mean (Sect. 4.2).
- We propose to capture lagged relationships of the time series data points by using autoregressive representations (i.e., autoregressive coefficients) of the time series

(Sect. 4.1). Our experiments show that extracting interval features from autoregressive representations of the series can increase the classification accuracy.
– We significantly enhance the discussion of explainability for TSC (Sect. 7), presenting different case studies and showing the utility of our approach in explaining classification results on different time series representations.
– To validate the contribution of each component of r-STSF to the classification accuracy, we conduct an ablation study on the method (Sect. 6).

We provide the experimental results and source code at https://github.com/stevcabello/r-STSF.

2 Related work

Time series classification (TSC) methods can be categorised into *distance-based*, *shapelet-based*, *dictionary-based*, *interval-based*, *kernel-based*, *hybrid*, and *deep learning-based* methods. Distance-based methods, such as *elastic ensemble* (EE) (Lines and Bagnall 2015) and *proximity forest* (PF) (Lucas et al. 2019) focus on classify a time series according to the similarity of its ordered data points to those of time series with known class labels, where localized distortions of the data points are allowed. EE uses an ensemble of 11 elastic nearest neighbour classifiers, whereas PF builds an ensemble of proximity trees for classification. Proximity trees use elastic distance measures as the splitting criteria. Both EE and PF have a quadratic time complexity over the length of the time series, which makes them computationally inefficient for long series.

Shapelet-based methods (Rakthanmanon and Keogh 2011; Hills et al. 2014; Grabocka et al. 2014) use shapelets, i.e, sub-series that are representative of class membership, and time series are classified according to their similarity to the (discriminatory) shapelets. By conducting a post-transform clustering of the shapelets, these methods may provide some level of explainability. *Shapelet Transform* (ST) (Hills et al. 2014) achieves a competitive classification accuracy Bagnall et al. (2017) with a bi-quadratic time complexity over the length of the series (due to full enumeration of all possible shapelets), which makes it impractical for long series. To improve the efficiency and hence the applicability of shapelet-based classifications, methods such as *random shapelet forest* (RSF) (Karlsson et al. 2015) and its more recent and general version, the *generalized random shapelet forest* (gRSF) (Karlsson et al. 2016) construct an ensemble of randomized shapelet trees, where each tree is built using a random selection of time series instances and a random selection (rather than a full enumeration) of shapelets. RSF and gRSF integrate explainability by identifying relevant shapelet lengths and time-points for the classifier decision. Recently, Bagnall et al. (2020) found that a full enumeration of all possible shapelets is not only unnecessary, but often results in over-fitting. Bagnall et al. (2020) proposed the *shapelet transform classifier* (STC), which randomly selects shapelets (i.e., no longer bi-quadratic time complexity as ST) and uses rotation forest (Rodriguez et al. 2006) for classification.

Dictionary-based methods (Lin et al. 2012; Schäfer 2015) use the relative frequency of discriminatory sub-series for classification. These methods transform each sub-series into a symbolic representation such as the *Symbolic Fourier Approximation* (SFA) Schäfer and Högqvist (2012) or the *Symbolic Aggregate Approximation* (SAX) (Lin et al. 2003) and employ a bag-of-patterns (BOP) model (Lin et al. 2012) to transform the symbols into a histogram representation. The *bag of SFA symbols* (BOSS) (Schäfer 2015) is the best dictionary-based method according to Bagnall et al. (2017). Although more recent dictionary-based classifiers such as the *Word ExtrAction for time SEries cLassification* (WEASEL) Schäfer and Leser (2017), *Spatial BOSS* (S-BOSS) (Large et al. 2019), and *Contractable BOSS* (cBOSS) (Middlehurst et al. 2019) are more accurate than BOSS, their difference in classification accuracy is not significant (Middlehurst et al. 2020b), while WEASEL is even more memory-intensive than BOSS (Le Nguyen et al. 2019). This makes BOSS a more practical TSC method. BOSS computes the number of times that a discriminatory sub-series (represented as a symbol) appears in the time series. This provides more accurate classifications if the discriminatory sub-series appears in different classes, i.e., represents more than one class. In BOSS, the size of the alphabet grows exponentially to the word length, which makes this method memory intensive for long series. A recent TSC method, *temporal dictionary ensemble* (TDE) (Middlehurst et al. 2020b) combines features from four dictionary-based TSC methods BOSS, WEASEL, S-BOSS, and cBOSS. TDE is significantly more accurate than all other dictionary-based TSC methods. However, as mentioned in its original proposal, TDE is slow at testing and less scalable, which makes it impractical to classify large time series datasets. To the best of our knowledge, none of the previously discussed dictionary-based TSC methods has focused on providing explainable classifications. However, a recent dictionary-based method, mtSAX-SEQL+LR (Le Nguyen et al. 2019) enables classification explainability. It transforms the time series into a symbolic representation, but it uses a *Sequential Learner* (SEQL) (Ifrim and Wiuf 2011) for feature selection and a *logistic regression* (LR) classifier for classification. The LR classifier learns a linear classification model which is essentially a set of symbols and their coefficients. The symbols can be mapped back to their original location in the time series. Since the coefficients can be interpreted as the discriminatory power of the symbols, mtSAX-SEQL+LR uses such coefficients to enable explainability into their classification results. mtSAX-SEQL+LR has quasi-quadratic time complexity over the length of the series. It is more efficient than BOSS and WEASEL (quadratic time over the length of the series). However, it is not competitive to SOTA TSC methods in terms of accuracy. Instead of using SAX or SFA symbols, Baydogan and Runger (2015) propose SMTS. It employs a tree-based ensemble to compute a symbolic representation of the time series, identifying discriminatory sub-series by their relative frequency at the terminal nodes. Building upon this, Rand-TS (Görgülü and Baydogan 2021) includes a post-processing step with a supervised learner. This further refines the symbolic representation by filtering out irrelevant features. Classification is done using *k*-nearest neighbour with Manhattan distance. We compared the reported classification accuracy values of Rand-TS and SMTS against those of SOTA methods and found that neither is significantly more accurate than BOSS nor competitive with SOTA methods. This aligns with existing literature (Bailly et al. 2016).

Interval-based methods (Deng et al. 2013; Baydogan et al. 2013; Baydogan and Runger 2016) are well known to be highly efficient TSC methods (Bagnall et al. 2017). These methods, similar to the *random patches* (RP) approach (Louppe and Geurts 2012), rely on random subsets of features. However, unlike most TSC methods including the interval-based ones which use all training instances for model training, RP uses randomly sampled training instances. Interval-based methods explore sets of sub-series to extract discriminatory intervals. The idea is that time series from the same class tend to have intervals with similar characteristics. *Time series forest* (TSF) (Deng et al. 2013) relies on random searches to reduce the high-dimensional interval space (i.e., quadratic to the length of the time series) and to explore intervals of different lengths. TSF uses three statistical measurements (mean, standard deviation, and slope) and tree-based ensembles to capture discriminatory intervals. TSF is fast and memory efficient (Bagnall et al. 2017), and the discriminatory intervals can be identified by using the *temporal importance curve* (Deng et al. 2013), which enables explainable classifications. The *random interval spectral ensemble* (RISE) (Lines et al. 2018) was introduced to capture frequency-domain features. RISE works similar to TSF, but it extracts spectral features over each random interval instead of statistical measurements as in TSF. Both TSF and RISE are highly efficient but less accurate than other feature-based TSC methods. More recent interval-based TSC methods such as our previous *supervised time series forest* (STSF) (Cabello et al. 2020) and the *canonical interval forest* (CIF) (Middlehurst et al. 2020a) have improved considerably the classification accuracy, and are competitive to highly accurate shapelet and dictionary-based classifiers. STSF selects its intervals in a supervised manner rather than completely at random as TSF does, and it uses a set of seven summary statistics to compute the interval features. Besides, STSF computes interval features not only from the original (i.e. raw) time series but also from additional time series representations such as the periodogram and first-order difference representations. CIF is similar to TSF, while it randomly selects eight out of the twenty-two descriptive statistics proposed by Lubba et al. (2019) when building each tree. Both STSF and CIF can provide explainable classifications but are less accurate than SOTA TSC methods. A recent interval-based method proposed in a parallel work, *diverse representation of canonical interval forest* (DrCIF) (Middlehurst et al. 2021) draws on ideas presented in STSF and CIF, and achieves SOTA accuracy. However, as shown in Sect. 5.3, it requires of a long running time and does not scale to large time series datasets.

Kernel-based methods such as the *RandOm Convolutional KErnel Transform* (ROCKET) (Dempster et al. 2020) transforms the series into a feature-based representation by using a large number of random convolutional kernels, and it uses such features to train a linear ridge regression classifier. ROCKET achieves SOTA classification accuracy, but it has not reported classification explainability. ROCKET's time complexity is quadratic to the number of time series or to the number of extracted features (depending on which is smaller). Large datasets with very long series may affect ROCKET's scalability. An ensemble of smaller ROCKET classifiers, *Arsenal*, is proposed by Middlehurst et al. (2021) to integrate ROCKET into HC2's ensemble. Arsenal by itself is statistically as accurate as ROCKET. However, HC2 with Arsenal is significantly better than HC2 with ROCKET.

Hybrid methods are heterogeneous ensembles. They classify time series datasets by combining information from different types of TSC methods. These methods are highly accurate but usually time and memory expensive. Two representatives are the *hierarchical vote collective of transformation-based ensembles* (HIVE-COTE) (Lines et al. 2018) and *time series combination of heterogeneous and integrated embedding forest* (TS-CHIEF) (Shifaz et al. 2020). HIVE-COTE uses EE, ST, BOSS, TSF, and RISE for classification and employs a modular hierarchical structure to allow a single probabilistic prediction from each classifier. HIVE-COTE achieves SOTA classification accuracy but is impractical for long series (bi-quadratic time complexity over the length of the time series). TS-CHIEF builds on PF and incorporates BOSS and RISE features as splitting criteria. TS-CHIEF is statistically similar to HIVE-COTE in classification accuracy, but more scalable (similar to PF). TS-CHIEF time complexity is quadratic to the series length, which makes it costly for long series.

To make HIVE-COTE faster and more scalable, Bagnall et al. (2020) proposed HIVE-COTE v1.0 (HC1). HC1 makes modifications such as dropping the elastic ensemble (EE), no longer fully enumerating the shapelet space (i.e., STC), and setting running time limits. In terms of accuracy, HC1 is statistically similar to HIVE-COTE, but significantly faster. The recent HIVE-COTE v2.0 (HC2) (Middlehurst et al. 2021) method, which is proposed in a parallel work, replaces three of the four classifiers from HC1. HC2 still uses STC, but it replaces BOSS with TDE and TSF with DrCIF. HC2 not longer uses RISE, and it adds Arsenal to the ensemble. To the best of our knowledge, HC2 is the new SOTA TSC method in terms of accuracy. However, as shown in Sect. 5.3, HC2 is among the slowest and most memory expensive classifiers. It does not scale to large datasets. No hybrid methods have reported classification explainability.

Deep learning-based methods such as the *fully convolutional networks* (FCN) and *residual networks* (ResNet) obtain competitive accuracy (Wang et al. 2017) and integrate explanations into the model decisions using the *class activation map* (CAM) (Zhou et al. 2016) to highlight relevant sub-series. However, they are also computationally expensive for long series and require a GPU. InceptionTime (Fawaz et al. 2020) is the best deep learning-based classifier, and it is competitive to SOTA TSC methods. It improves ResNet's accuracy by using an ensemble of five different Inception networks which are randomly initialized. InceptionTime is faster than HIVE-COTE as it uses GPU parallelization; however, it is very slow when running without GPU (Bagnall et al. 2020) and does not report classification explainability.

Table 1 summarizes the representative TSC methods. Most current TSC methods have time complexities that are quadratic or bi-quadratic to the length of the time series, which makes them less practical on long time series. Besides, to the best of our knowledge, with the exception of TSF, RSF, gRSF, ST, FCN, ResNet, mtSAX-SEQL+LR, STSF and CIF no existing TSC methods provide insights about the classifier decision.

Table 1 Summary of representative TSC methods: The time and space complexities show that these methods (except for r-STSF, STSF and TSF) are not suitable for large datasets with long time series

Approach	Training	Training memory cost	SOTA accuracy	Integrates explain-ability
r-STSF	$\mathcal{O}(r \cdot n \cdot \log n \cdot \log m)$	$\mathcal{O}(n \cdot m + r \cdot m)$	**Yes**	**Yes**
TSF	$\mathcal{O}(r \cdot n \cdot \log n \cdot m)$	$\mathcal{O}(rm)$	No	**Yes**
RSF	$\mathcal{O}(n^2 \log n)$	*	No	**Yes**
gRSF	$\mathcal{O}(n^2m^2 \log nm^2)$	*	No	**Yes**
ST	$\mathcal{O}(n^2m^4)$	$\mathcal{O}(\kappa n)$	No	**Yes**
FCN	*	*	No	**Yes**
ResNet	*	*	No	**Yes**
mtSAX-SEQL+LR	$\mathcal{O}(n \cdot m^{3/2} \log m)$	*	No	**Yes**
STSF	$\mathcal{O}(r \cdot n \cdot \log n \cdot \log m)$	$\mathcal{O}(rm)$	No	**Yes**
CIF	*	*	No	**Yes**
BOSS	$\mathcal{O}(n^2m^2)$	$\mathcal{O}(n\alpha^{\iota})$	No	No
EE	$\mathcal{O}(n^2m^2)$	$\mathcal{O}(m^2)$	No	No
PF	$\mathcal{O}(r \cdot n \cdot \log n \cdot C_e \cdot c \cdot m^2)$		No	No
TDE	*	*	No	No
STC	*	*	No	No
HIVE-COTE	$\mathcal{O}(n^2m^4)$	$\mathcal{O}(\kappa nm^2)$	**Yes**	No
TS-CHIEF	$\mathcal{O}(r \cdot n \cdot \log n \cdot C_e \cdot c \cdot m^2)$	$\mathcal{O}(n \cdot m + r \cdot n \cdot c + D_t \cdot n \cdot m)$	**Yes**	No
InceptionTime	*	*	**Yes**	No
ROCKET	$\mathcal{O}(Knm + n^2v)$ **or** $\mathcal{O}(Knm + nv^2)$		**Yes**	No
HC1	*	*	**Yes**	No
HC2	*	*	**Yes**	No
DrCIF	*	*	**Yes**	No
Arsenal	*	*	**Yes**	No

r-STSF is the only TSC method that integrates explainability and achieves SOTA classification accuracy. Other SOTA TSC methods do not provide explainable classifications. Other TSC methods that integrate explainability do not achieve SOTA classification accuracy

n number of time series, m length of a series, κ number of shapelets, α alphabet size, ι word length, r number of trees, c number of classes, C_e number of candidate splits, D_t number of dictionary-based transformations, K number of kernels, v number of extracted features. For space, in TS-CHIEF's training time, we have only included the complexity of the similarity-based splitting

*Indicates that the information is not explicitly stated in the associated paper

Bold "Yes" is used to highlight that r-STSF is the only TSC method that achieves SOTA accuracy and integrates explainability

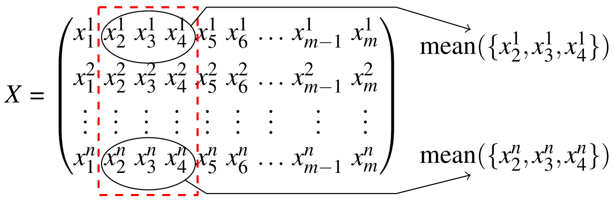
3 Preliminaries

We take an interval-based approach to classify time series and identify discriminatory features. The basic idea is to extract sub-series for which aggregates (e.g., mean and standard deviation) are computed and used as features. To allow for explainable classifications, we focus on *phase-dependent intervals*, i.e., discriminatory features located at the same time regions over all time series in a given dataset. Although r-STSF's explainability is oriented towards phase-dependent intervals, this does not imply that r-STSF cannot provide explanations in case of phase-independent discriminatory intervals (or in the absence of phase-dependent discriminatory intervals). As discussed in Sect. 4.1, r-STSF uses different time series representations as a proxy to *indirectly* detect phase-independent discriminatory intervals, i.e., phase-independent intervals in the time domain may become phase-dependent when being moved into the frequency-domain.

Interval feature: *Given a set of time series* $X = \{\boldsymbol{x^1}, \boldsymbol{x^2}, \boldsymbol{x^3}, \dots, \boldsymbol{x^n}\}$, *where* $\boldsymbol{x^i} = \{x_1^i, x_2^i, \dots, x_m^i\}$, *an aggregation function* $a(\cdot)$, *and an interval* (s,e), *an interval feature* $\boldsymbol{f} = a(X, s, e)$ *is a vector of length* n, *defined as follows:*

$$\boldsymbol{f} = \{a(\boldsymbol{x}^1, s, e), a(\boldsymbol{x}^2, s, e), a(\boldsymbol{x}^3, s, e), \dots, a(\boldsymbol{x}^n, s, e)\}$$

where $a(\boldsymbol{x}^i, s, e) = a(\{x_s^i, x_{s+1}^i, \dots, x_{e-1}^i, x_e^i\})_{1 \leqslant s \leqslant e \leqslant m}$. With s=2, e=4, a=mean, an interval feature $\boldsymbol{f} =$ mean $(X, 2, 4)$ is represented with the dashed rectangle as:


$X = \begin{pmatrix} x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 & x_6^1 & \dots & x_{m-1}^1 & x_m^1 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 & x_6^2 & \dots & x_{m-1}^2 & x_m^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_1^n & x_2^n & x_3^n & x_4^n & x_5^n & x_6^n & \dots & x_{m-1}^n & x_m^n \end{pmatrix}$
mean$(\{x_2^1, x_3^1, x_4^1\})$
mean$(\{x_2^n, x_3^n, x_4^n\})$

Problem statement: Consider a set of n univariate time series $X = \{\boldsymbol{x^1}, \boldsymbol{x^2}, \dots, \boldsymbol{x^n}\}$, where each time series $\boldsymbol{x^i} = \{x_1^i, x_2^i, \dots, x_m^i\}$ has m ordered real-valued observations, sampled at equally-spaced time intervals. Each time series $\boldsymbol{x^i}$ is also associated with a class label y^i. We aim to find the set of interval features that yield the highest time series class prediction accuracy. Finding such a set of interval features is NP-hard. For a time series of length m, there are $\mathcal{O}(m^2)$ different intervals, and hence $\mathcal{O}(2^{m^2})$ subsets of intervals. For a large m, it is prohibitively expensive to explore all subsets. We present an efficient heuristic to avoid the exhaustive search while retaining a high classification accuracy

Table 2 summarizes the symbols frequently used in this paper.

Table 2 Frequently used symbols and their meanings

Symbol	Meaning
UCR_{112}	The 112 benchmark time series datasets from the UCR repository
n	Number of time series instances
m	Time series length
X	Time series set of size $n \times m$
y	Class labels vector of size $n \times 1$
c	Number of class labels
$\boldsymbol{x}^i$	Time series instance
y^i	Class label
$\boldsymbol{f}$	Interval feature of size $n \times 1$
$a(\cdot)$	Aggregation function
T	Tree classifier
r	Number of tree classifier in the ensemble
X_O, X_P, X_D, X_G	Original (raw), periodogram, derivative and autoregressive representations of X, respectively
F_O, F_P, F_D, F_G	Candidate discriminatory interval features extracted from X_O, X_P, X_D and X_G, respectively
d	Number of sets of candidate discriminatory interval features
$\mathscr{F}$	Set of candidate discriminatory interval features (union of F_O, F_P, F_D, F_G)
$\mathscr{F}^*$	Set of discriminatory interval features
i, j, k	Indices to iterate across time series instances, time series data points, class labels, etc

4 Our approach

Our proposed TSC algorithm, *r-STSF*, takes a stochastic optimization approach to select a set of interval features with a high discriminating power (i.e., *candidate discriminatory interval features*) from the high dimensional interval feature space (i.e., all $\mathcal{O}(m^2)$ possible interval features). For a time series of length m, we reduce the interval feature space size to $\mathcal{O}(\log m)$. We search for the best interval features subset $\mathscr{F}^*$ (i.e., *discriminatory interval features*) through an ensemble of binary trees, which has a time complexity of $\mathcal{O}(r \cdot n \cdot \log n \cdot \log m)$, where r is the total number of trees in the ensemble and n is the number of time series instances. The trees in the ensemble are built in a randomized manner following the extra-trees algorithm (see Sect. 4.3) to reduce the variance of the ensemble and improve the classification accuracy. Figure 3 shows an overview of r-STSF when training the ensemble of randomized binary trees.

For a given time series training set X, r-STSF does not only uses its original (raw) representation, i.e., X_O, but derives its periodogram, i.e., X_P, derivative, i.e., X_D, and autoregressive representation, i.e., X_G. For each representation, r-STSF selects a group of candidate discriminatory interval features F_O, F_P, F_D, and F_G, respectively. Next, a union set $\mathscr{F}$ is formed by merging all candidate discriminatory interval features. It is worth noting that the process of extracting candidate interval features is repeated for d times, where d is a system constant parameter. In other words, d sets of candidate

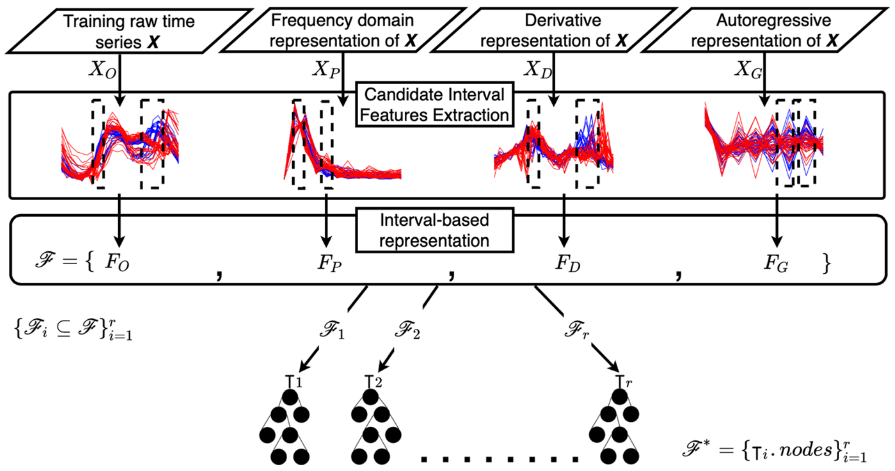

Training raw time series X
Frequency domain representation of X
Derivative representation of X
Autoregressive representation of X
X_O
X_P
X_D
X_G
Candidate Interval Features Extraction
Interval-based representation
$\mathscr{F} = \{ F_O, F_P, F_D, F_G \}$
$\{\mathscr{F}_i \subseteq \mathscr{F}\}_{i=1}^r$
$\mathscr{F}_1$
$\mathscr{F}_2$
$\mathscr{F}_r$
T_1
T_2
T_r
$\mathscr{F}^* = \{\mathsf{T}_i.nodes\}_{i=1}^r$

Fig. 3 Overview of r-STSF when training an ensemble of r randomized trees. Sets of candidate discriminatory interval features F_O, F_P, F_D, F_G are selected from the time series representations X_O, X_P, X_D, X_G, respectively. All sets of candidate discriminatory interval features are merged into a single superset $\mathscr{F}$. Each randomized tree is built by using a number of randomly selected interval features from $\mathscr{F}$. It is worth noting that each tree node selects its own group of interval features (from $\mathscr{F}$) for the split. The set of discriminatory intervals $\mathscr{F}^*$ from the nodes of the randomized trees offers explainability to the classification outcome

discriminatory interval features are extracted. In our empirical study, we find that a small value, e.g., $d = 0.1 \times r$, is sufficient to yield a high TSC accuracy. This means that, for an ensemble of 500 trees, we just extract 50 sets of candidate discriminatory intervals. Lastly, each randomized tree is built by using a number of randomly selected interval features from $\mathscr{F}$. We set this number to the square root of the size of $\mathscr{F}$ following other tree-based ensemble approaches such as *random forest* (RF) (Breiman 2001). A tree classifier, due to its intrinsic feature selection capability, enables the selection of a set of discriminatory intervals $\mathscr{F}^*$ with which TSC is performed. Features from $\mathscr{F}^*$ enable the explainability of the TSC task, which will be discussed in Sect. 7.

r-STSF and STSF share a similar logic, i.e., extract features in a supervised manner and use them to build an ensemble of trees for classification. However, every component of r-STSF is designed with the goal of improving classification accuracy and decreasing training/testing time, while retaining explainability. A summary of the main differences between r-STSF and STSF is discussed in Sect. 4.4.

4.1 Time series representation

We first discuss time series representations to extract intervals features. We use intervals from original (i.e., the time domain as described in Sect. 3), periodogram (i.e., the frequency domain), derivative, and autoregressive representations. We focus on the latter three representations.

Periodogram representation: Several TSC methods (Bagnall et al. 2012; Lines et al. 2018) use the *periodogram representation* when examining time series similarity in the frequency domain. We adopt this and exploit the periodogram representation of each time series derived from the *discrete Fourier transform* (DFT). The DFT decomposes a real-valued time series $\boldsymbol{x^i}$ into a linear combination of sinusoidal functions with amplitudes o and q, and phase ω:

$$\boldsymbol{x_t} = \sum_{j=1}^{m} (o_j \cos(2\pi\omega_j t) + q_j \sin(2\pi\omega_j t)) \tag{1}$$

The periodogram $\boldsymbol{x^i_P} = \{p^i_1, p^i_2, \dots, p^i_m\}$ of series $\boldsymbol{x^i}$ is represented using the set of amplitudes $\{(o_1, q_1), (o_2, q_2), \dots, (o_m, q_m)\}$ from Equation (1), i.e., $p^i_j = \sqrt{o^2_j + q^2_j}$. A property of the DFT of a real-valued series is that it is symmetric, i.e., $(o_j, q_j) = (o_{m-j-1}, q_{m-j-1})$. Thus, we can shrink the size of the periodogram by half, resulting in $\boldsymbol{x^i_P} = \{p^i_1, p^i_2, \dots, p^i_{m/2}\}$. For long series, this reduces the computation cost substantially in assessing the discriminatory power of the interval features. A side benefit of this representation is that it helps to *indirectly* detect *phase-independent discriminatory intervals,* i.e., discriminatory features located at different time regions of the original series. Take Fig. 4 as an example. r-STSF assesses the similarity of a group of series based on the discriminatory power of the extracted phase-dependent interval features. Hence, time series $\boldsymbol{x^i}$ and $\boldsymbol{x^k}$, with a same class label (i.e., $y^i = y^k$) but time-shifted discriminatory interval, are more likely to be identified as similar by using the periodogram representation of the series (i.e., $\boldsymbol{x^i_P}$ and $\boldsymbol{x^k_P}$) rather than their original (time domain) representation.

Derivative representation: Using a (first-order) derivative representation of a time series rather than the original time series improves the classification accuracy (Keogh and Pazzani 2001; Górecki and Łuczak 2013) as it provides trend information. Given a time series $\boldsymbol{x^i}$, its derivative representation is $\boldsymbol{x^i_D} = \{x^i_2 - x^i_1, x^i_3 - x^i_2, \dots, x^i_m - x^i_{m-1}\}$.

Autoregressive representation: An autoregressive model, as shown in Eq. (2), assumes that the current observation of a time series, x_t, can be explained as a linear

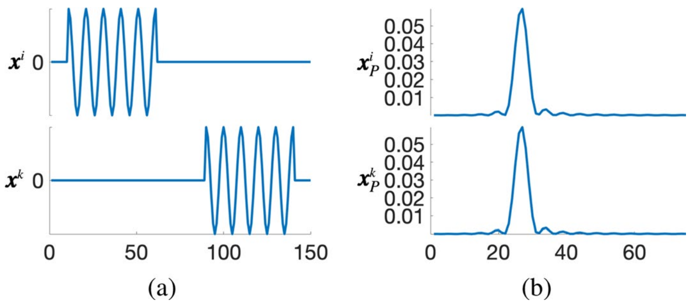


Fig. 4 **a** Time series $\boldsymbol{x^i}$ and $\boldsymbol{x^k}$, with a same class label (i.e., $y^i = y^k$) present a similar sub-series but located at different time regions (i.e., phase-independent). **b** The periodogram representation $\boldsymbol{x^i_P}$ and $\boldsymbol{x^k_P}$ of series $\boldsymbol{x^i}$ and $\boldsymbol{x^k}$, respectively. Our algorithm searches for discriminatory phase-dependent interval features, which in **a** are challenging to identify. The periodogram representation provides more flexibility as it considers the frequency of the discriminatory sub-series (ignoring its location in time) and thus helps identify discriminatory sub-series even when they appear at non-identical location in time across different time series

combination of its past l observations, model coefficients $\{\beta_i\}_{i=1}^{l}$, a constant b, and the error term ϵ_t.

$$x_t = b + \sum_{k=1}^{l} \beta_k x_{t-k} + \epsilon_t \tag{2}$$

If a time series dataset satisfies this assumption, *the time series can be modeled as an autoregressive process which can be used for the classification task, since series from different classes may have different autoregressive models*. The model (autoregressive) coefficients $\{\beta_k\}_{k=1}^{l}$ form an autoregressive representation. Given a time series $\boldsymbol{x^i}$, its autoregressive representation is $\boldsymbol{x^i_G} = \{\beta^i_1, \beta^i_2, \dots, \beta^i_{l-1}, \beta^i_l\}$. Three well-known methods to estimate such coefficients are the *ordinary least-squares* (OLS), *Yule-Walker*, and *Burg's method*. We use Burg's method as it usually yields better model fitting than the Yule-Walker approach, and it is computationally efficient (Proakis and Manolakis 2014; Brockwell et al. 2002), while OLS has a time complexity quadratic to the length of the time series. Moreover, we set the lag order $l = 12(m/100)^{1/4}$, where m is the series length, as suggested by Schwert (1989).

The idea of using autoregressive coefficients for classification has been previously adopted in TSC methods (Lines et al. 2018). Nonetheless, unlike these approaches, we do not use the autoregressive coefficients directly as features. Instead, we build an autoregressive representation (i.e., series of autoregressive coefficients), from which interval features are derived by using a set of aggregation functions. Our insight is that whilst the individual coefficients may not be discriminatory, the summary representation of a group of them is likely to retain more discriminatory information.

4.2 Extraction of candidate discriminatory interval features

Our process to extract candidate discriminatory interval features is summarized in Algorithms 1 and 2. This process is similar to that in STSF. The key difference is that, unlike STSF, r-STSF does not partition each interval into halves to assess the *quality* of sub-intervals. Instead, r-STSF creates partitions with random cut points (Lines 4 and 5 in Algorithm 2). This strategy plays a significant role in further boosting the TSC accuracy (see Sect. 6.6), since it allows to explore a larger number of sub-intervals and hence generates less correlated trees.

As detailed in Algorithm 1, given a time series set X of size $n \times m$, a set of aggregation functions A, and a feature ranking metric f_r, r-STSF computes a set of candidate discriminatory interval features, $\mathscr{F}$, using all time series representations (i.e., original, periodogram, derivative and autoregressive) (Line 2). For each representation, r-STSF finds discriminatory features from each aggregation function $a \in A$ (Line 4). A cut point u is randomly selected from $\{1, 2, \dots, m\}$ (Line 5), which partitions X into two sub-sets: (i) X_L of size $n \times u$, and (ii) X_R of size $n \times (m - u)$ (Line 6). The initial random partition (according to u) enables r-STSF to explore more diverse sub-series (i.e., at different locations and of different lengths). Next, in Lines 7 and 8, X_L and X_R are used to extract a set of interval features F_L and F_R in a supervised manner (i.e., Algorithm 2). Finally, F_L and F_R are added to $\mathscr{F}$ (Line 9).

Algorithm 1: GetIntervalFeatures

```
   Input: $X$: set of $n$ time series of length $m$; $y$: class label vector; $A$: set of aggregation functions;
          $f_r$: feature ranking metric.
 1 $\mathscr{F} \leftarrow \emptyset$;
 2 for each time series representation do
 3     $X$ changes according to the corresponding representation;
 4     for each $a \in A$ do
 5         $u \leftarrow$ GetRandomCutPoint$(m)$;
 6         $X_L \leftarrow X(1:u)$; $X_R \leftarrow X(u+1:m)$;
 7         $F_L \leftarrow$ SupervisedIntervalSearch$(X_L, y, a, f_r, \emptyset)$;
 8         $F_R \leftarrow$ SupervisedIntervalSearch$(X_R, y, a, f_r, \emptyset)$;
 9         $\mathscr{F} \leftarrow \mathscr{F} \cup \{F_L + F_R\}$;
10     end
11 end
12 return $\mathscr{F}$;
```

Our supervised search algorithm for the candidate discriminatory intervals is summarized in Algorithm 2. The algorithm recursively breaks a given interval into two and computes interval features for the two resultant intervals (Line 5). The size of each interval is not fixed, but set based on a random cut point u (Line 4). The *Fisher score* Duda et al. (2012) (detailed below) is further computed for each interval feature (Line 6). The interval feature with a higher score is added to the set of candidate discriminatory interval features (the starting and ending time indices for each interval as well as the aggregation function used to represent the interval feature), and the search continues within this interval (Lines 7 to 13). The algorithm stops when the interval cannot be partitioned further (with less than two points, Lines 1 to 3).

Algorithm 2: SupervisedIntervalSearch

```
   Input: $X'$: set of $n$ time series of length $m'$; $y$: class label vector; $a$: aggregation function; $f_r$:
          feature ranking metric; $F$: subset of candidate discriminatory intervals.
 1 if $m' < 2$ then
 2     return $F$;
 3 else
 4     $u \leftarrow$ GetRandomCutPoint$(m')$;
 5     $f_L \leftarrow a(X', 1, u)$; $f_R \leftarrow a(X', u, m)$;
 6     $score_L \leftarrow f_r(f_L, y)$; $score_R \leftarrow f_r(f_R, y)$;
 7     if $score_L >= score_R$ then
 8         $F \leftarrow F \cup \{f_L\}$;
 9         SupervisedIntervalSearch$(X'(1:u), y, a, f_r, F)$;
10     else
11         $F \leftarrow F \cup \{f_R\}$;
12         SupervisedIntervalSearch$(X'(u:m), y, a, f_r, F)$;
13     end
14 end
15 return $F$;
```

Feature ranking metric: We assess the quality of sub-intervals using Fisher Score (Duda et al. 2012) as the feature ranking metric. The Fisher score of an interval feature indicates how well the feature separates a class of time series from the other classes. A higher Fisher score suggests a more discriminatory feature. For a given time series subset of size $n \times m'$, an interval feature $\boldsymbol{f}$ of size $n \times 1$ is extracted (by applying $a(\cdot)$ on each time series row-wise). Interval feature $\boldsymbol{f}$ and a vector of class labels $\boldsymbol{y} \in \{1, 2, \dots, c\}^n$ are used to compute the Fisher score to obtain the discriminatory quality of $\boldsymbol{f}$ as follows:

$$\text{FisherScore}\,(\boldsymbol{f}, \boldsymbol{y}) = \left(\sum\nolimits_{k=1}^{c} n_k (\mu_k^f - \mu^f)^2\right) / \left(\sum\nolimits_{k=1}^{c} n_k (\sigma_k^f)^2\right) \tag{3}$$

Here, μ^f is the overall mean of the elements in $\boldsymbol{f}$; μ_k^f and σ_k^f are the mean and standard deviation of the elements in $\boldsymbol{f}$ labelled with the k-th class; and n_k is the number of time series labelled with the k-th class.

Other feature ranking metric such as Laplacian score (He et al. 2005) or ReliefF (Robnik-Šikonja and Kononenko 2003) could also be used. We use Fisher score for its fast computation and the resultant high accuracy of r-STSF (see Sect. 6.2).

Aggregation functions: Standard interval-based classifiers such as TSF use *mean*, *standard deviation* (*std*), and *slope* (i.e., slope of the least-squares regression line) aggregation functions to obtain a representative value for the sub-series. Others interval-based methods such as STSF expand the set of aggregation functions by adding more robust statistics such as the *median* and the *interquartile range* (*iqr*). STSF also incorporates the *minimum* (*min*) and *maximum* (*max*) statistics as they can potentially detect discriminatory extreme values. According to our experiments, the *slope* is the aggregation function that contributes the most to the classification accuracy of STSF. This aggregation function captures the shape of the series, and it is the only one in the set of statistics used by STSF that can do so. *To reinforce capturing a higher level of details regarding the shape of the sub-series*, we further propose two aggregation functions: (i) *counts of mean-crossings* (*cmc*) (i.e., the number of intersections of a sub-series with the mean axis of the entire sub-series), and (ii) *counts of values above the mean* (*cam*) (i.e., the number of data points above the mean of a sub-series). Hence, r-STSF uses nine aggregations to compute the interval features: *cmc*, *cam* and the seven aggregation proposed in STSF.

4.3 Classification with randomized trees

Our algorithm creates *randomized binary trees* for classification (summarized in Algorithm 3). In such trees, the cut-point of each feature is randomly selected when looking for the feature that provides the best split (i.e., best random split). This perturbation strategy was originally proposed by Geurts et al. (2006) for the *extra-trees* (ET) algorithm. The idea is to create an ensemble of uncorrelated trees (i.e., different tree models), which decreases the estimation variance (i.e., variability of the predictions) and hence decreases the classification error. We use this strategy to decrease the similarities among the trees in our ensemble. We build our trees node-by-node recursively starting from the root node. At each node, we do not inspect all possible cut-points for each feature (to split that node). Instead, we *randomly* select one value *cp* as the cut-point (for each feature) (Line 7). This cut-point splits the learning sample *LS* and its class label vector y into two sets each. The rows of the learning sample and label vector where the values of a feature are less than or equal to its corresponding cut-point are sent to LS_L and y_L, respectively (Line 8). The remaining rows of *LS* and y are sent to LS_R and y_R, respectively (Line 9). Then, we compute the *information gain* (IG, Eq. (4)) for each split (Line 10) and use the split with the maximum gain (Lines 11 to 15) to expand our tree (Lines 17 and 18). If the node cannot be further split, i.e., all the samples have the same class label, the node becomes a leaf or a terminal node (Lines 1 to 4).

Algorithm 3: CreateRandomTree

```
Input: $LS$: learning sample of size $n \times v$, $v$ interval features, each feature (or column) with $n$
       values; $y$: class label vector of size $n \times 1$.
1  if cannot further split LS then
2      Designate this node as a terminal node;
3      return
4  end
5  $maxIG \leftarrow 0$;
6  for each $f \in$ columns of LS do
7      $cp \leftarrow$ Pick a value from feature $f$;
8      $LS_L \leftarrow LS(f \leqslant cp)$; $y_L \leftarrow y(f \leqslant cp)$;
9      $LS_R \leftarrow LS(f > cp)$; $y_R \leftarrow y(f > cp)$;
10     $currentIG \leftarrow \mathrm{IG}(y, y_L, y_R)$;
11     if $currentIG > maxIG$ then
12         $maxIG \leftarrow currentIG$;
13         $best_{LS_L} \leftarrow LS_L$; $best_{LS_R} \leftarrow LS_R$;
14         $best_{y_L} \leftarrow y_L$; $best_{y_R} \leftarrow y_R$;
15     end
16 end
17 CreateRandomTree($best_{LS_L}, best_{y_L}$);
18 CreateRandomTree($best_{LS_R}, best_{y_R}$);
```

For a given vector of class labels $\boldsymbol{y} = \{y^1, y^2, \dots, y^i, \dots, y^n\}$ where $y^i \in \{1, 2, \dots, c\}$, and its corresponding subsets after split, $\boldsymbol{y}_L$ and $\boldsymbol{y}_R$, the IG is given by:

$$\mathrm{IG}(\boldsymbol{y}, \boldsymbol{y}_L, \boldsymbol{y}_R) = \mathrm{H}(\boldsymbol{y}) - \left[\frac{|\boldsymbol{y}_L|}{|\boldsymbol{y}|}\mathrm{H}(\boldsymbol{y}_L) + \frac{|\boldsymbol{y}_R|}{|\boldsymbol{y}|}\mathrm{H}(\boldsymbol{y}_R)\right] \tag{4}$$

Here, H is the entropy (i.e., impurity measure), which is computed as $\mathrm{H}(\boldsymbol{y}_*) = -\sum_{k=1}^{c} \rho_k \log \rho_k$, where $\rho_k = |\{y^i | y^i = k; y^i \in \boldsymbol{y}_*\}| / |\boldsymbol{y}_*|$ is the relative frequency of class label y^i in $\boldsymbol{y}_*$.

4.4 r-STSF vs. STSF

r-STSF extends from STSF. r-STSF is redesigned to be extremely fast and competitive in terms of accuracy to SOTA TSC methods. Table 3 summarizes the main differences between r-STSF and STSF.

The additional time series representation (i.e., autoregressive) and aggregation functions (i.e., *cmc* and *cam*) and our novel perturbation scheme (i.e., supervised search through random partitions of sub-series and randomized binary trees to select discriminatory interval features) significantly improve the classification accuracy of r-STSF over its predecessor STSF (see Fig. 5). Using randomized binary trees (i.e., ET) instead of non-randomized trees (e.g., random forest (RF)) do not only improves significantly the classification accuracy (see Fig. 23), but using randomized trees also contributes to decrease the training time.

Randomized trees are known to be faster than non-randomized trees (Geurts et al. 2006) since in the former the cut-point of each feature is randomly selected when looking for the feature that provides the best split. Further, r-STSF's approach to train its tree-based ensemble is significantly faster than that of STSF. r-STSF requires an order of magnitude less computations when searching for candidate discriminatory interval features (see Sect. 5.3). It is worth noting that on STSF each tree node uses the same group of features (given as input to their corresponding tree) when looking for the best split whereas on r-STSF each tree node uses a different group of randomly selected features from $\mathscr{F}$ when looking for the best random split.

4.5 Computation complexity

The training time of r-STSF depends on two processes: (i) extracting candidate interval features and (ii) building a tree ensemble with a subset of the extracted features.

Candidate interval feature extraction: This process is summarized in Algorithm 1. The set of candidate interval features is found by a supervised search on each segment created after the initial random partition. The supervised search follows a binary-inspired search strategy (Algorithm 2). In a single run of Algorithm 1, the total number of candidate interval features extracted is $\mathscr{O}(z \cdot g \cdot \log m)$, where z is the number of time series representations, g is the number of aggregation functions, and m is the time series length. Since z and g are constants, the total number of candidate interval features extracted in a single run of Algorithm 1 is $\mathscr{O}(\log m)$.

For an ensemble with r trees, our experiments suggest that r-STSF requires $0.1 \times r$ runs of Algorithm 1 for classification accuracy optimization. r-STSF sets the number of runs of Algorithm 1 to the constant parameter $d = 50$. Hence, the total number of interval features computed to build an ensemble with r trees is $\mathscr{O}(d \cdot \log m)$. Overall, the total number of interval features computed is $\mathscr{O}(\log m)$. The time complexity for extracting such features from n (training) time series is $\mathscr{O}(n \cdot \log m)$. The space complexity when generating the sets of candidate interval features is $\mathscr{O}(n \cdot d \cdot m)$. Since d is a constant, the space complexity is thus $\mathscr{O}(n \cdot m)$.

Tree-based ensemble construction: The time complexity to train a single tree is $\mathscr{O}(n \cdot h \cdot v)$, where n is the number of training instances (i.e., time series), v is the number of interval features for node splitting, and h is the depth of the tree. For a balanced tree, h is in $\mathscr{O}(\log n)$. Also, as detailed above, the number of candidate interval features, v, is $\mathscr{O}(\log m)$. Hence, the time complexity of training a single tree is $\mathscr{O}(n \cdot \log n \cdot \log m)$. To compute r trees, the time complexity is

Table 3 Main differences between r-STSF and STSF. Changes in time series representations, aggregation functions and supervised search (i.e., columns 1, 2 and 3) contribute to improve the classification accuracy

	Time Series Representations	Aggregation Functions	Supervised Search	Tree Classifier ($\intercal$)	Train Ensemble of r Trees

STSF	Raw series, periodogram, derivative	Mean, std, slope, median, iqr, min, max	Fixed partition of sub-series	Binary tree using best learned split	**for** $i = 1$ **to** r **do**
					  feats $\leftarrow$ GetIntervalFeatures ()
					  $\intercal_i$.build(*feats*)
					end

r-STSF	Raw series, periodogram, derivative, *autoregressive*	Mean, std, slope, median, iqr, min, max, *cmc*, *cam*	*Random* partition of sub-series	*Randomized* binary tree using best *random* split	$\mathscr{F} \leftarrow \emptyset$
					for $j = 1$ **to** $0.1 \times r$ **do**
					  $\mathscr{F} \leftarrow \mathscr{F} \cup$ GetIntervalFeatures ()
					end
					for $i = 1$ **to** r **do**
					  feats $\leftarrow$ Pick $\sqrt{\lvert\mathscr{F}\rvert}$ features from $\mathscr{F}$
					  $\intercal_i$.build(*feats*)
					end

Changes in training the ensemble (i.e., column 5) contribute to reduce the training time. Changes in the tree classifier (i.e., column 4) contribute to both accuracy and training speed

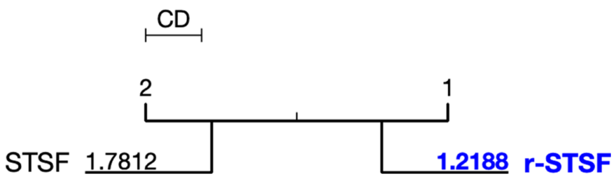


Fig. 5 Critical difference diagram of average ranks of r-STSF and STSF. The proposed r-STSF is significantly more accurate than STSF

$\mathcal{O}(r \cdot n \cdot \log n \cdot \log m)$. Similar to other interval-based approaches, the space complexity to build the ensemble is $\mathcal{O}(r \cdot m)$.

Computation complexity of r-STSF: The total training time for r-STSF is thus $\mathcal{O}(n \cdot \log m) + \mathcal{O}(r \cdot n \cdot \log n \cdot \log m) = \mathcal{O}(r \cdot n \cdot \log n \cdot \log m)$. The testing (or prediction) time complexity for r-STSF is $\mathcal{O}(r \cdot n \cdot \log n)$. The total space complexity for r-STSF is $\mathcal{O}(n \cdot m) + \mathcal{O}(r \cdot m)$.

5 Evaluation

We evaluate the classification effectiveness (i.e., accuracy) and computational efficiency (i.e., memory consumption and running time) of r-STSF. Our results show that r-STSF achieves classification accuracies competitive to those of SOTA methods, while it is orders of magnitude faster and requires less memory than most of them. By default, r-STSF is trained with 500 trees (similar to all interval-based methods) and uses four time series representations and nine aggregation functions as summarized in Table 3. We compare r-STSF against interval-based, shapelet-based, dictionary-based, kernel-based, deep learning-based and hybrid TSC methods as detailed in Sect. 5.1. All methods use their default settings and are tested on the 112 benchmark datasets from the UCR Time Series Classification Repository (Bagnall et al. 2019), i.e., UCR_{112}. We use the default train and test split of each dataset from the repository.

Classification effectiveness: To evaluate the effectiveness of r-STSF, we use the well-known *average ranking* metric. For a visual comparison according to the average rank, we use the critical difference diagram (Demšar 2006) which is widely used by the TSC community to assess the statistical differences of the ranks when comparing multiple classifiers over multiple datasets. Horizontal, bold lines show groups of statistically similar methods. A smaller rank indicates a better classifier. Ranks are based on classification accuracies. Accuracy results from each competitor TSC method are obtained from its original paper. In cases where results from a dataset are missing, we use the results reported in the UCR repository or in recent works (Middlehurst et al. 2021). We report the overall effectiveness comparison results against the competitors in Sect. 5.2. We also report the average ranks of r-STSF and competitors when evaluated on different application domains in Sect. 5.2.1.

Computational efficiency: To assess efficiency, we measure the algorithms' maximum memory consumption, and total training and testing times when running on UCR_{112}. r-STSF is implemented in Python. For a fair efficiency comparison with the competitor methods, we used their implementations in sktime Löning et al. (2019) (a library for time series analysis in Python). For the deep learning-based methods, ResNet and InceptionTime, we used their original Python implementations (we did not use sktime-dl because of errors found when running those methods). For gRSF (originally implemented in Java) we used a Python implementation from Wildboar (Samsten 2020), a Python module for temporal machine learning and fast distance computations. We report the overall efficiency comparison results against the competitors in Sect. 5.3. In addition, we evaluate the computational efficiency of TSC methods running on a set of eight large time series datasets from Bagnall et al. (2019) and report our results in Sect. 5.3.1.

All experiments are run on the High Performance Computing (HPC) system Spartan at The University of Melbourne. Each job runs on a single core and consists of a single dataset and TSC method. A job has a maximum run time of seven days and the maximum memory allowed is 500 GB.

5.1 Competitor TSC methods

To evaluate r-STSF, we compare it against other interval-based, shapelet-based, dictionary-based, kernel-based, hybrid and deep learning TSC methods. Table 4 details the classifiers used as competitors. We did not include distance-based methods such as EE (Lines and Bagnall 2015) or PF (Lucas et al. 2019) as they are not competitive to SOTA TSC methods (as shown in results from Middlehurst et al. (2020a) and Fawaz et al. (2020)), are very slow (Bagnall et al. 2020), and there is no record of their classification accuracies on the full UCR_{112} dataset. Similarly, we did not include mtSAX-SEQL+LR (Le Nguyen et al. 2019) because there is no result on UCR_{112} and its original implementation is in C++ (i.e., not in Python as the other competitor TSC methods). Other dictionary-based methods such as WEASEL (Schäfer and Leser 2017), S-BOSS (Large et al. 2019), cBOSS (Middlehurst et al. 2019), SMTS (Baydogan and Runger 2015) and Rand-TS (Görgülü and Baydogan 2021) were not included because they are

Table 4 Competitor TSC methods used in our evaluation. Default parameters of each competitor are set as suggested in their original paper

Interval-based	Hybrid	Deep learning
TSF (Deng et al. 2013)	TS-CHIEF (Shifaz et al. 2020)	ResNet (Fawaz et al. 2019)
RISE (Lines et al. 2018)	HC1 (Bagnall et al. 2020)	ITime (Fawaz et al. 2020)
STSF (Cabello et al. 2020)	HC2 (Middlehurst et al. 2021)	
CIF (Middlehurst et al. 2020a)		
DrCIF (Middlehurst et al. 2021)		
Shapelet-based	**Dictionary-based**	**Kernel-based**
gRSF (Karlsson et al. 2016)	BOSS (Schäfer 2015)	ROCKET (Dempster et al. 2020)
STC (Bagnall et al. 2020)	TDE (Middlehurst et al. 2020b)	

not significantly more accurate than the baseline BOSS, as detailed in Sect. 2. The deep learning-based method, FCN, was excluded as it is known to be outperformed by ResNet (Fawaz et al. 2019). The SOTA method TS-CHIEF (Shifaz et al. 2020) was not included in the computational efficiency evaluation because its original implementation is in Java and it has not been added to the sktime library.

5.2 Classification effectiveness

r-STSF is the only TSC method that that integrates explainability and achieves competitive classification accuracy comparable to SOTA methods. As Fig. 6 shows, despite having a larger average rank than SOTA TSC methods, r-STSF is *not* significantly different from them except for the parallel work HC2. However, in Sect. 5.3 we show the practical limits of HC2 when it is run on large datasets. r-STSF ranks the fourth (sixth), just behind the the other parallel (interval-based) work DrCIF. However, as shown in Sect. 5.3, r-STSF is on average two order of magnitude faster than DrCIF. It is worth noting the substantial improvements of recent interval-based methods. Compared to earlier interval-based methods such as TSF and RISE that were less competitive, current interval-based methods are in the group of best ranked TSC methods. As shown in Fig. 6, interval-based TSC methods such as r-STSF just rank behind time and memory expensive hybrid methods such as HC2 and TS-CHIEF, and kernel-based methods that do not integrate explainability such as ROCKET.

Table 9, Appendix A shows the classification accuracies (as reported) from the majority of competitors TSC methods mentioned above. We did not include the

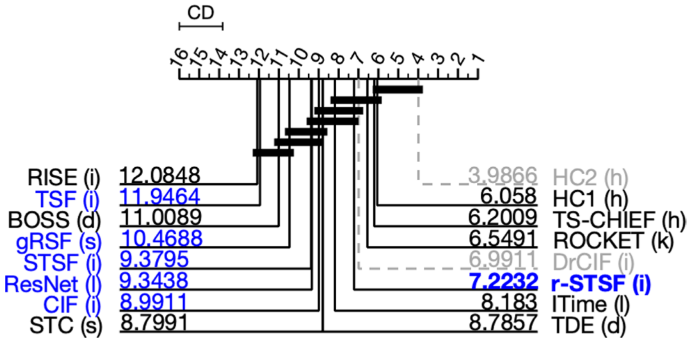


Fig. 6 Critical difference diagram of average ranks of r-STSF and competitors on UCR_{112}. TSC methods that integrate explainability are in blue. TSC methods that do not integrate explainability are in black. r-STSF is the only classifier competitive to SOTA TSC methods that integrates explanations. Interval-based methods (i) rank higher than deep learning-based (l), dictionary-based (d) and shapelet-based (s) TSC methods. Only highly accurate but time and memory expensive hybrid methods (h) and fast but non-explainable by design kernel-based methods (k) rank higher than the best interval-based methods

classification accuracies of every competitor approach due to space limitations, however, they are included in our accompanying website.

5.2.1 Comparing TSC methods with data from different application domains

It has been a common practice in the TSC community to use UCR_{112} for empirical studies. These datasets come from a variety of domains, with an emphasis on time series from the *image outline* domain, as shown in Fig. 7. We use the application domain as given in Bagnall et al. (2019). Domains with less than 4 representative datasets are labelled as *Others*. This is the case for *EOGHorizontalSignal*, *EOG-VerticalSignal* (originally part of the *EOG* domain); *InsectEPGRegularTrain*, *InsectEPGSmallTrain* (originally part of the *EPG* domain); *PigAirwayPressure*, *PigArtPressure*, and *PigCVP* (originally part of the *Hemodynamics* domain).

We argue that using an unbalanced distribution of times series domains may not be appropriate to assess the overall effectiveness of different TSC methods. As Fig. 8 shows, the average ranks of different TSC methods change when classifying datasets from different domains. The overall average ranks of the TSC methods on UCR_{112} (Fig. 6) are highly influenced by the classification accuracy in the over represented time series domain, i.e., *image outline* (Fig. 8a). The dominant or over represented domain could affect positively or negatively the "perceived" effectiveness of a classifier. For example, adding more time series datasets from *sensor readings* or *spectrograhps* domains could improve the average rank of r-STSF. Similarly, adding more datasets from *ECG* or *electric devices* domains could decrease the average performance of the SOTA method TS-CHIEF.

r-STSF is the best (third best) approach when classifying series from *sensor readings* domain (see Fig. 8b), which is the second largest domain on UCR_{112}. r-STSF, as well as other interval-based methods, focuses on *phase-dependent* discriminatory intervals, i.e., discriminatory features located at the same time regions over all time series. The high accuracy of r-STSF when classifying *sensor readings* series suggests that there are relevant features embedded in phase-dependent intervals. Nonetheless, these features cannot be found in the original (time-stamped) series but in additional representations of the time series. This is inferred from the rather poor performance (i.e., higher average ranks) of the interval-based methods TSF and CIF that only use the original series to extract

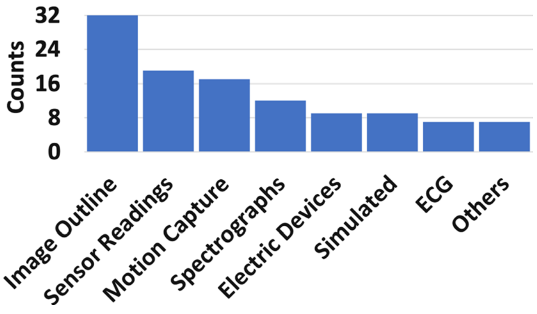


Fig. 7 Summary of dataset domains of UCR_{112}. There is an over representation of data from image outline classification problems

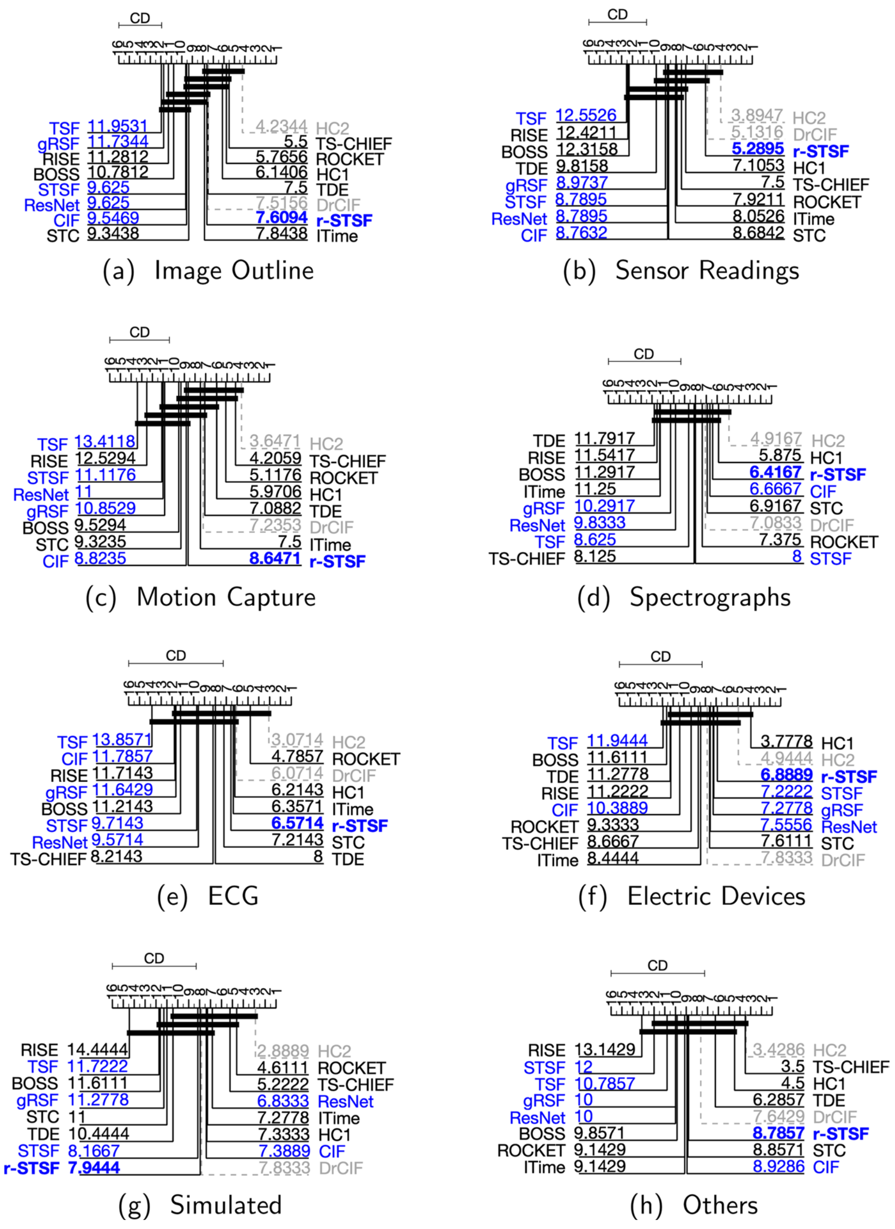


Fig. 8 Critical difference diagram of average ranks per time series domain in UCR_{112}. **a** Image Outline, **b** Sensor Readings, **c** Motion Capture, **d** Spectrographs, **e** ECG, **f** Electric Devices, **g** Simulated, **h** Others. TSC methods that integrate explainability are in blue color (Color figure online)

discriminatory features. Further, the additional parameters and design criteria make r-STSF to outperform STSF on this type of domain.

In series from the *spectrographs* domain, r-STSF also achieves a good performance. r-STSF is the second (third) best classifier according to the average ranks (see Fig. 8d). It is evident that on this type of domains there exists phase-dependent discriminatory features. This is the only domain where interval-based methods outperform most of the other SOTA classifiers. Series derived from the *spectrographs* domain show the spectrum of frequencies of a signal as it varies with time, i.e., show the time at which the signal's frequency changes. Thus, if series exhibit changes of their frequency at different times, interval-based TSC methods can detect those times and provide accurate classification accuracies.

According to Fig. 8 r-STSF usually ranks between the 1st (3rd) and 5th (7th) place, with the exception of the *simulated* and *motion capture* domains where r-STSF's effectiveness decreases. The *motion capture* datasets are prone to contain *phase-independent* discriminatory intervals, which pose an additional challenge for r-STSF (and other interval-based methods). This reflects on the higher average rank of r-STSF when classifying *motion capture* series (see Fig. 8c).

Among UCR_{112}, only nine are representative of the *simulated* domain, which are BME, CBF, ChlorineConcentration (ChlCon), Mallat, ShapeletSim, SmoothSubspace, SyntheticControl (SynthCon), TwoPatterns, and UMD. As shown in Table 9, Appendix A, most of classifiers (including r-STSF) achieve accuracies above 95% for the majority of the *simulated* datasets. Consider the case of CBF dataset where the accuracy of most of the classifiers is above 99%, and r-STSF ranks in 9th (11th) place (among the fourteen (sixteen) evaluated TSC methods, but 8th if considering the ten classifiers shown in Table 9, Appendix A) despite having an accuracy of 99.17%. In this kind of scenarios, the ranks amplify the differences among TSC methods and may not reflect their actual performance difference.

It is worth noting that r-STSF outperforms STSF in the majority of domains. They have a similar performance only on datasets from *electric devices* (Fig. 8f) and *simulated* (Fig. 8g) domains. This suggests that, on these types of datasets/domains, the number of relevant features is small. On such datasets, randomized methods are less likely to identify the relevant features (Geurts et al. 2006).

5.3 Computational efficiency

Training time: In Fig. 9, we present the normalized training time of each TSC method. r-STSF is two orders of magnitude faster than the majority of the evaluated TSC methods (and almost three orders of magnitude faster than HC2), except for TSF, RISE, STSF, and ROCKET. r-STSF is an order of magnitude faster than STSF and has a similar training time to that of ROCKET. As discussed in Sect. 5.1, we did not evaluate the computational efficiency of TS-CHIEF as it was implemented in Java. However, r-STSF is estimated to be three orders of magnitude faster than TS-CHIEF in training based on the results reported by Middlehurst et al. (2021).

Testing time: For the majority of datasets from UCR_{112}, the number of testing instances is much larger than the number of training instances. This affects the performance of classifiers such as ROCKET, which is fast on training but may become slow at testing. ROCKET uses a large number of kernels (10,000) to transform the

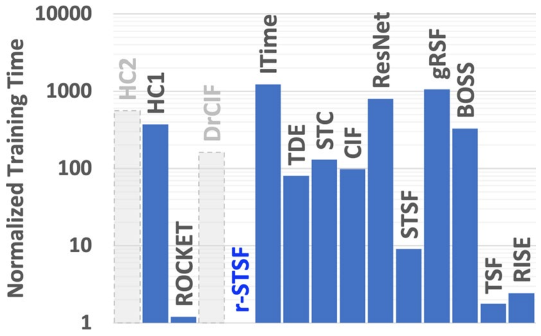


Fig. 9 Normalized training times of HC2, HC1, ROCKET, DrCIF, InceptionTime (ITime), TDE, STC, CIF, ResNet, STSF, gRSF, BOSS, TSF, RISE, and r-STSF when training on UCR_{112}. The TSC methods are ordered by their accuracy ranks from the left to the right based on Fig. 6. All training times are relative to the training time of r-STSF, i.e., the training time of r-STSF is considered to be 1. r-STSF is the fastest TSC method at training and is part of the top 5 most accurate TSC methods

time series into a feature-based representation. The computation of each representation is related not only to the number of kernels but also to the characteristics of the datasets (i.e., the number of instances and the length of the time series). For large datasets with very long series, ROCKET becomes computationally expensive. This difference in the number of instances between training and testing sets is reflected in ROCKET's testing time, which is an order of magnitude slower than r-STSF and even InceptionTime (Fig. 10).

The SOTA methods HC2 and HC1 are both expensive at testing. HC2 is almost three orders of magnitude slower than r-STSF when testing on UCR_{112}. HC1 is two orders of magnitude slower than r-STSF. Similarly, DrCIF, TDE, CIF, and BOSS are also two orders of magnitude slower than r-STSF at testing. InceptionTime (ITime)

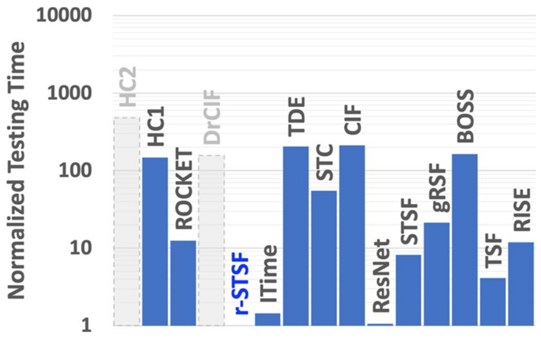


Fig. 10 Normalized testing times of HC2, HC1, ROCKET, DrCIF, InceptionTime (ITime), TDE, STC, CIF, ResNet, STSF, gRSF, BOSS, TSF, RISE, and r-STSF when testing on UCR_{112}. The TSC methods are ordered by their accuracy ranks from the left to the right based on Fig. 6. All testing times are relative to the testing of r-STSF, i.e., testing time of r-STSF is considered to be 1. r-STSF is the fastest TSC method at testing and is part of the top 5 most accurate TSC methods

and ResNet are comparable to r-STSF in terms of testing time. We could not estimate the testing time of TS-CHIEF for the reasons provided above.

Memory consumption: As shown in Fig. 11, among the SOTA TSC methods, r-STSF is the most memory efficient on UCR_{112}. HC2 requires five times more memory than r-STSF, whereas HC1 requires almost three times more memory than r-STSF. Methods such as STC, ResNet, and gRSF are the most memory efficient, but they are not competitive to SOTA TSC methods in terms of accuracy.

The results reported in Figs. 9, 10, and 11 suggest that, among the SOTA group, r-STSF is the only method that is fast at both training and testing, and it is also the most memory efficient within the SOTA group. ROCKET is fast at training, but much slower at testing (specially if the testing datasets have a large number of instances), and it requires more memory than r-STSF. HC2, HC1, and DrCIF are two orders of magnitude slower at training and testing, and are more memory expensive than r-STSF. It is worth mentioning that when running on GPU, ResNet and InceptionTime are an order of magnitude faster than when running on CPU. Similarly, results for gRSF presented above are only based on 111 datasets. gRSF could not complete a run (i.e., training and testing) for *HandOutlines* dataset within the seven-day time limit. gRSF's original Java implementation is approximately an order of magnitude faster than its Python implementation. The large training times of gRSF can be related to differences on its implementations.

5.3.1 Comparing TSC methods on larger time series datasets

In UCR_{112}, *ElectricDevices* is the dataset with the largest training size (8,926 training instances), and *HouseTwenty* is the one with the longest time series (3,000 data points per time series). However, *ElectricDevices* has only 96 data points per time series, while *HouseTwenty* has only 34 training instances.

To assess TSC methods under a big data scenario, we report the classification accuracy, training time, testing time and memory consumption of r-STSC and competitors when running on eight larger time series datasets from Bagnall et al. (2019). Such datasets were selected based on (i) being univariate (i.e., 1 dimensional), and (ii) having a training set size larger than that of *ElectricDevices* or a series length longer than that of *HouseTwenty*. Although the *AbnormalHearbeat* dataset with

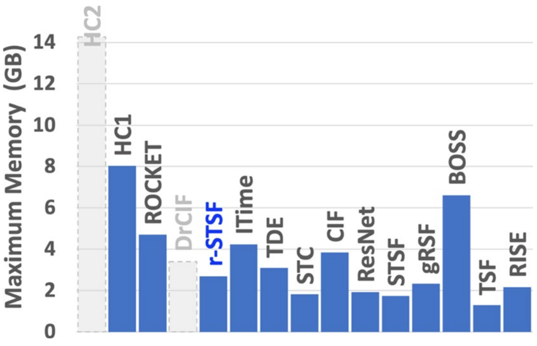


Fig. 11 Maximum memory used (in GB) of HC2, HC1, ROCKET, DrCIF, InceptionTime (ITime), TDE, STC, CIF, ResNet, STSF, gRSF, BOSS, TSF, RISE, and r-STSF when testing on UCR_{112}. The TSC methods are ordered by their accuracy ranks from the left to the right based on Fig. 6. r-STSF is memory efficient and is part of the top 5 most accurate TSC methods

Table 5 Largest Time Series Datasets from Bagnall et al. (2019)

Type	Dataset	Training set size	Test set size	Length	Number of classes
Largest	RightWhaleCalls	10,934	1,962	4,000	2
	FruitFlies	17,259	17,259	5,000	3
	InsectSound	25,000	25,000	600	10
	MosquitoSound	139,883	139,883	3,750	6
Longest	CatsDogs	138	137	14,773	2
	BinaryHeartbeat	204	205	18,530	2
	UrbanSound	2,713	2,712	44,100	10
	DucksAndGeese	50	50	236,784	5

Table 6 Classification Accuracy of TSC methods in the four largest datasets from Bagnall et al. (2019)

	r-STSF	STSF	TSF	RISE	ROCKET	CIF	DrCIF
RightWhaleCalls	84.00	81.35	77.42	82.98	78.85	83.23	**86.44**
FruitFlies	92.03	91.35	90.13	83.19	**96.33**		
InsectSound	76.05	75.75	45.83	74.63	**80.39**	69.27	77.66
MosquitoSound	80.69	79.99	56.98	75.97	**87.84**		

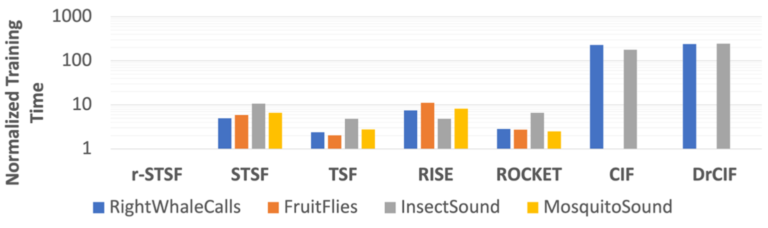
Bold classification accuracies are used to highlight the most accurate methods for each time series dataset

time series of length 3,053 meets these selection criteria, we did not include it into this set of experiments for its small training set size (303 training instances).

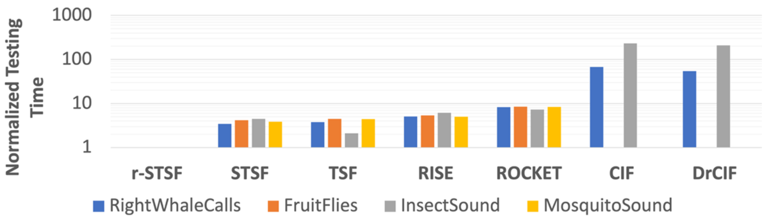
Table 5 summarizes the eight datasets. In particular, *FruitFlies* has twice the number of training instances of that of *ElectricDevices*. *InsectSound* and *MosquitoSound* have approximately three and 15 times more training instances than *ElectricDevices* has, respectively. *RightWhaleCalls* has a training set size similar to that of *ElectricDevices* but significantly longer time series.

CatsDogs, *BinaryHeartbeat*, and *UrbanSound* have approximately five, six and 15 times longer series than those of *HouseTwenty*, respectively. Despite having a small training set size, *DuckAndGeese* is the dataset with the longest time series (79 times longer than those of *HouseTwenty*). Hence, we consider it relevant to assess the efficiency of TSC methods under a very long series setting.

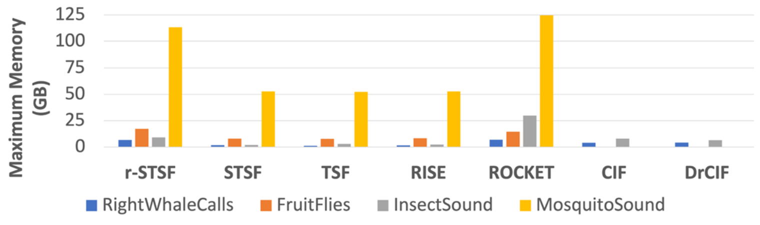
Largest training size datasets: Table 6 and Fig. 12 summarize our results on the four time series datasets with the largest training set sizes shown in Table 5. We report classification accuracy (Table 6), normalized training time, i.e., relative the training times of r-STSF (Fig. 12a), normalized testing time (Fig. 12b), and memory consumption (Fig. 12c). The only TSC methods to complete a run (i.e., training and testing) within the seven-day time limit for each of the four datasets are the interval-based methods r-STSF, STSF, TSF and RISE, and the



(a) Training Time



(b) Testing Time



(c) Memory Consumption

Fig. 12 Computational efficiency comparison of TSC methods in the four largest datasets from Bagnall et al. (2019). **a** Normalized training times of r-STSF and competitor TSC methods. All training times are normalized to the training time of r-STSF, i.e., training time of r-STSF is considered to be 1. **b** Normalized testing times of r-STSF and competitor TSC methods. All testing times are normalized to the testing time of r-STSF, i.e., testing time of r-STSF is considered to be 1. **c** Maximum memory used (in GB) of r-STSF and competitor TSC methods. r-STSF is the fastest TSC method at training and testing and is memory efficient

kernel-based method ROCKET. The other interval-based methods, CIF and DrCIF, complete a run just for *RightWhaleCalls* and *InsectSound*. This suggests that the competitors TSC methods HC2, HC1, STC, BOSS, TDE, ResNet, ITime and gRSF are impractical to train their models on datasets with a large number of training time series, where the time series are of medium length.

We make the following observations:

– In terms of effectiveness, ROCKET achieves an average classification accuracy of 85.85% followed by r-STSF with 83.19% and STSF with 82.11%.

- In terms of training speed, r-STSF takes around 12 h to train on the four datasets, ROCKET takes 32 h, whereas TSF takes 33 h. CIF and DrCIF take 8 and 10 days, respectively, to train on the two datasets mentioned above.
- r-STSF is also the fastest at testing, which takes only 3 h on the four datasets, whereas STSF, TSF and RISE take 11, 13 and 15 h, respectively. ROCKET takes 25 h which is the slowest at testing. CIF and DrCIF take 20 and 18 h to test, respectively, but only finish on two datasets.
- In terms of memory efficiency, ROCKET is overall the most memory expensive approach, while r-STSF is the second. On datasets with a large number of medium-length training instances such as *MosquitoSound*, ROCKET and r-STSF require approximately twice the memory of interval-based methods such STSF, TSF and RISE. On datasets with a large number of short training series such as *InsectSound*, ROCKET is memory-inefficient, which consumes approximately three times more memory than r-STSF does. On the other two large datasets *RighWhaleCalls* and *FruitFlies*, r-STSF and ROCKET consume approximately the same amount of memory.

Our results suggest that, among the SOTA TSC methods, only r-STSF and ROCKET are practical for time series datasets with large training set sizes. r-STSF is the fastest TSC method at training, and it is approximately an order of magnitude faster at testing time than ROCKET. ROCKET and r-STSF are the most accurate TSC method (that could complete their runs on all four large datasets) but also are the most memory expensive, where r-STSF is slightly more memory efficient than ROCKET. Ideas similar to Random Patches (Louppe and Geurts 2012) that build each individual model of the ensemble not only using random subsets of features but also randomly sampled training instances, can be further explored and integrated into r-STSF to improve its memory efficiency without decreasing its classification accuracy. We take this as a future work.

Longest time series: Table 7 and Fig. 13 summarize our results on the four time series datasets with the longest series shown in Table 5. Like above, we report classification accuracy (Table 7), normalized training time (Fig. 13a), normalized testing time (Fig. 13b), and memory consumption (Fig. 13c). gRSF is the only TSC method that could not complete a run (within the seven-day time limit) for any of these four datasets. However, this can be related to its Python implementation as discussed in Sect. 5.3. InceptionTime just completed a run for *CatsDogs*, which is the least lengthy of the four dataset. CIF, DrCIF, HC2, STC, BOSS, TDE, and ResNet could complete a run just for *CatsDogs* and *BinaryHeartbeat* datasets. HC1 could not run for *UrbanSound* which is the only dataset with medium training set size (all other three datasets have a small training set size). r-STSF, STSF, TSF, RISE and ROCKET are the only TSC methods that can run on all four lengthy datasets. This suggests that such TSC methods are practical to classify datasets with very long series.

Summary: We make the following observations:

Table 7 Classification Accuracy of TSC methods in the four longest datasets from Bagnall et al. (2019)

	r-STSF	STSF	TSF	RISE	ROCKET	CIF	DrCIF	HC1	HC2	STC	BOSS	TDE	ResNet	ITime
CD	70.73	69.51	62.80	68.90	**73.17**	68.29	69.51	70.73	71.95	68.29	65.24	64.63	60.98	60.37
BH	72.20	72.20	65.37	74.63	74.63	73.66	72.20	73.66	74.63	**76.59**	68.78	75.61	74.15	
US	**83.30**	79.32	50.52	72.81	76.12									
DG	52.00	52.00	28.00	56.00	58.00			**74.00**						

CD Cats Dogs, *BH* Binary Heartbeat, *US* Urban Sound, *DG* Ducks And Geese

Bold classification accuracies are used to highlight the most accurate methods for each time series dataset

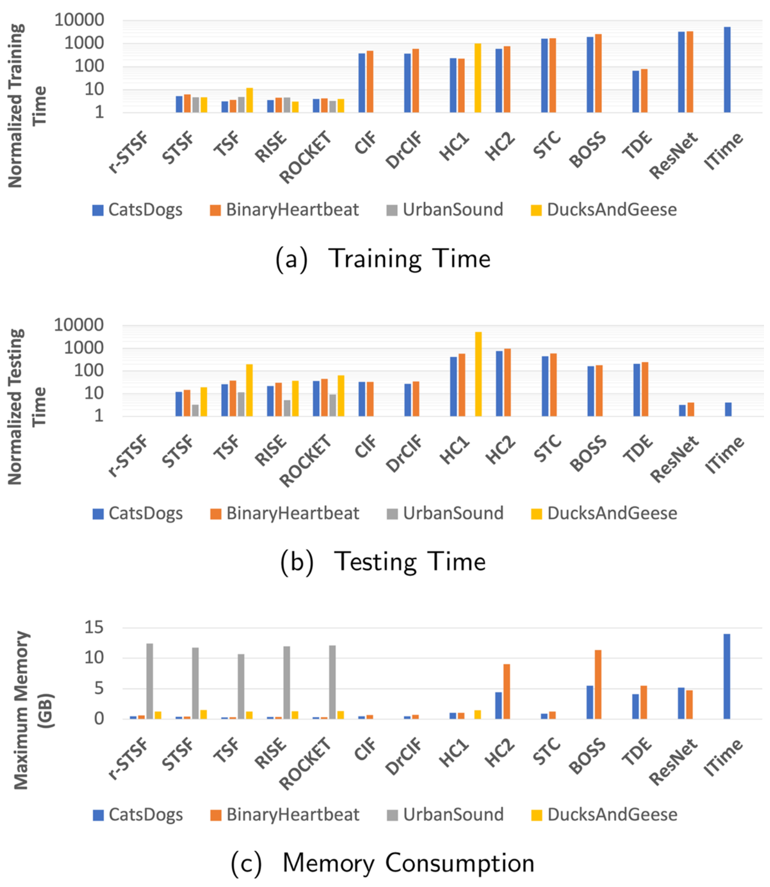


Fig. 13 Computational efficiency comparison of TSC methods in the four longest datasets from Bagnall et al. (2019). **a** Normalized training times of r-STSF and competitor TSC methods. All training times are normalized to the training time of r-STSF, i.e., training time of r-STSF is considered to be 1. **b** Normalized testing times of r-STSF and competitor TSC methods. All testing times are normalized to the testing time of r-STSF, i.e., testing time of r-STSF is considered to be 1. **c** Maximum memory used (in GB) of r-STSF and competitor TSC methods. r-STSF is the fastest TSC method at training and testing and is memory efficient

- Among the TSC methods that could complete a run on all four datasets, ROCKET is the most accurate, with a 70.48% accuracy on average. r-STSF follows closely with an accuracy of 69.56% and STSF with 68.25%.
- r-STSF is the fastest at training on the four datasets, taking less than 2 h in total. ROCKET is the second fastest, completing all runs in 6 h. From the group of TSC methods that could only run on two datasets (*CatsDogs* and *BinaryHeartbeat*), TDE is the fastest, with 4 h in total for training. This is followed by CIF and DrCIF, which take 25 and 28 h, respectively. HC1 takes more than 5 days to train on three datasets, among which 5 days are spent on just *DucksAndGeese*.

Table 8 Run times in seconds (i.e., training plus testing time) of r-STSF and ROCKET variants when running on eight large time series datasets detailed in Table 5

Dataset	r-STSF	ROCKET	MiniROCKET	MultiROCKET
CatsDogs	93	664	51	188
BinaryHeartbeat	137	1,051	77	271
UrbanSound	7,313	34,869	2,407	8,804
DucksAndGeese	450	3,336	782	941
RightWhaleCalls	2,333	7,287	1,318	2,860
FruitFlies	6,415	26,008	2,972	25,508
InsectSound	1,719	11,558	1,366	5,150
MosquitoSound	44,294	162,577	34,408	Failed after 42,604
Total time in seconds	62,754	247,350	43,381	86,326
Total time in hours	17	69	12	24

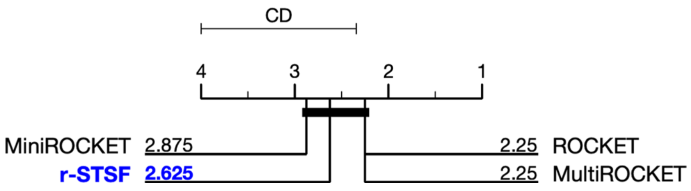


Fig. 14 Critical difference diagram of average ranks of r-STSF and competitor ROCKET variants on the eight largest time series datasets detailed in Table 5

– In terms of testing time, r-STSF significantly faster than the other (also fast) TSC methods. It takes only 30 min to classify all four datasets. Other fast approaches such as ROCKET and TSF take 5 and 7 h, respectively. HC1 needs more than 1 day just to test on *DucksAndGeese*.
– In terms of memory efficiency, HC2, BOSS, TDE, and InceptionTime are most memory expensive when running on datasets with long series. r-STSF, STSF, TSF, RISE, and ROCKET have a similar performance in this regard.

These results suggest that r-STSF is the best TSC method when classifying time series datasets with very long series. It is competitive to ROCKET in terms of accuracy, while it is faster at both training and testing than ROCKET, STSF, TSF, and RISE. r-STSF is also more memory efficient comparing with the other SOTA TSC methods and uses the same amount of memory as that by ROCKET.

Additional results on comparing with faster ROCKET variants: When our paper was under review, two faster variants of ROCKET, MiniROCKET (Dempster et al. 2021) and MultiROCKET (Tan et al. 2022), have been proposed. These two methods are significantly faster than ROCKET while remaining competitive to SOTA TSC methods in classification accuracy. We compare r-STSF against ROCKET and its faster variants on the eight large time series datasets above. As shown in Table 8, MiniROCKET and r-STSF are the fastest approaches, requiring 12 and 17 h, respectively, to train and classify the large time series datasets.

Although MiniROCKET is very fast, it is the least accurate among the four TSC methods (see Fig. 14). MultiROCKET is slighlty slower than r-STSF but more accurate. None of the faster ROCKET variants integrate explainability in their classifications. We consider r-STSF relevant in cases of large time series data where explainability is required. It is worth noting that MultiROCKET could not finish its run in *MosquitoSound* due to memory issues. Hence, its total running time is expected to increase. In Table 8 we are reporting the running time before failure. For completion, in Fig. 14 we set the accuracy of MultiROCKET as same as ROCKET when run on *MosquitoSound*.

6 Ablation study

To provide a comprehensive analysis of r-STSF, we present experimental results to support (i) our decision for a supervised selection of intervals features instead of an unsupervised one (i.e., random selection) and (ii) our decision to use Fisher score as the feature ranking metric in the supervised search. Further, we explore the impact of different parameters of r-STSF on its classification accuracy. The parameters to consider are: (i) number of candidate discriminatory interval feature sets, (ii) time series representations, and (iii) aggregation functions. The number of trees in the ensemble is not considered in this analysis. It is set to 500 following the other tree-based TSC methods (Deng et al. 2013; Shifaz et al. 2020). Moreover, we present experimental results to support our claims regarding the improvements to the average classification accuracy of r-STSF when (i) using random partitions (instead of middle-point partitions) of intervals when searching for candidate discriminatory interval features, and (ii) building ensembles of randomized binary trees (i.e., *extra-trees*) instead of non-randomized binary trees (i.e., *random forest*). All the results presented in this section are based on the average classification accuracy over three runs of r-STSF when classifying on UCR_{112}. r-STSF uses by default (unless said otherwise) the four time series representations and the nine aggregation functions mentioned in Sect. 4.

6.1 Supervised vs unsupervised selection of interval features

Our proposed r-STSF selects (in a supervised manner) a set of candidate discriminatory interval features and then uses such features to build an ensemble of binary trees (see Fig. 3). We compare r-STSF against different “unsupervised" versions of r-STSF, i.e., random selections of candidate discriminatory interval features. We set the number of randomly selected interval features to compute according to $sqrt(m) \times f$, where m is the time series length and $f \in \{1,2,3,4,5,6,7,8,9,10\}$. Hence, unsupervised versions of r-STSF with larger f values will compute a larger set of candidate discriminatory interval features. As shown in Fig. 15, r-STSF is not significantly different in accuracy from the majority of the unsupervised versions.

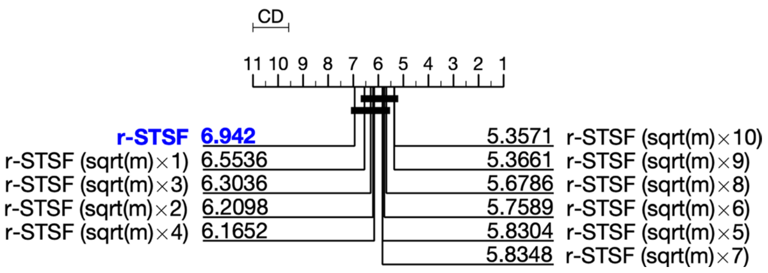

CD
11 10 9 8 7 6 5 4 3 2 1
r-STSF 6.942
r-STSF (sqrt(m)×1) 6.5536
r-STSF (sqrt(m)×3) 6.3036
r-STSF (sqrt(m)×2) 6.2098
r-STSF (sqrt(m)×4) 6.1652
5.3571 r-STSF (sqrt(m)×10)
5.3661 r-STSF (sqrt(m)×9)
5.6786 r-STSF (sqrt(m)×8)
5.7589 r-STSF (sqrt(m)×6)
5.8304 r-STSF (sqrt(m)×5)
5.8348 r-STSF (sqrt(m)×7)

Fig. 15 Critical difference diagram of average ranks of our default r-STSF, i.e., supervised selection of intervals (blue color) and different unsupervised versions of r-STSF, i.e., random selection of intervals (black color). Unsupervised versions differ in the number of interval features to select, which is given by $sqrt(m) \times f$. The default r-STSF is not significantly different in accuracy from the majority of unsupervised versions of r-STSF. It takes a large number of random intervals ($f \geqslant 9$) to make this difference significant (Color figure online)

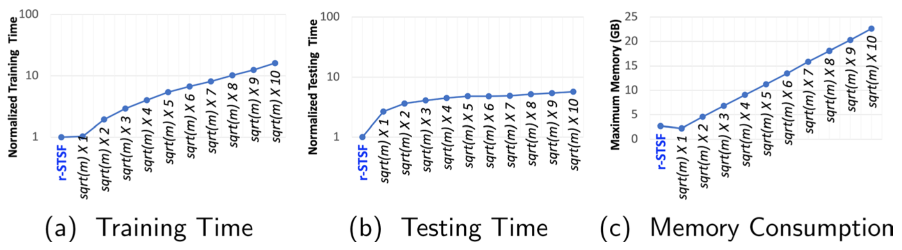


Fig. 16 **a** Normalized training time, i.e., training time of r-STSF = 1, **b** normalized testing time, i.e., testing time of r-STSF = 1 and **c** maximum memory consumed in GB of our default r-STSF (color blue) and unsupervised versions of r-STSF (color black). The default (supervised) r-STSF is faster in training and testing than all of the unsupervised versions. It is also more memory efficient than most of them (Color figure online)

Only when $f \geqslant 9$, r-STSF is significantly less accurate than its corresponding unsupervised versions. i.e., r-STSF ($sqrt(m) \times 9$) and r-STSF ($sqrt(m) \times 10$). However, as shown in Fig. 16, such unsupervised versions are an order of magnitude slower at training (Fig. 16a) and approximately five times slower at testing (Fig. 16b) than r-STSF. Besides, they also require five times more memory (Fig. 16c).

The results in Figs. 15 and 16 suggest that our default r-STSF achieves the best trade-off among accuracy, training and testing times, and memory consumption. Unsupervised versions either require large training time and substantially more memory space to be significantly more accurate than our default r-STSF, or they take a larger testing time to achieve a slightly higher but not significantly different accuracy.

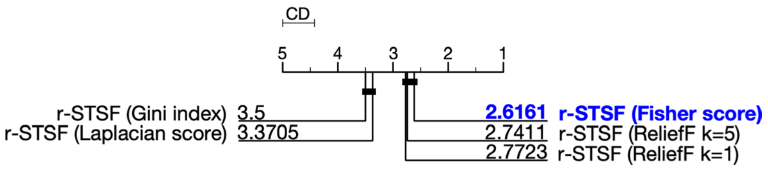


Fig. 17 Critical difference of average ranks of r-STSF with different feature ranking metrics. r-STSF with Fisher score is the most accurate approach

6.2 Feature ranking metric

As discussed in Sect. 4.2, r-STSF requires training an ensemble of trees with a group of highly discriminatory set of interval features. The supervised selection of these intervals is guided by a feature ranking metric that scores the “quality” of the intervals. Intervals with a high score are more likely to maximize class separability. We compare our default feature ranking metric, Fisher score, against the well-known Gini index and other commonly-used ranking metrics such as Laplacian score (He et al. 2005) and ReliefF (Robnik-Šikonja and Kononenko 2003). For the competitor metrics, we use an implementation by Li et al. (2018). As shown in Fig. 17, r-STSF with Fisher score is significantly more accurate than with Gini index or Laplacian score. r-STSF with (different settings of) ReliefF is statistically as accurate as r-STSF with Fisher score. ReliefF estimates the relevance of features based on how well their values distinguish the instances (of the same or different classes) that are near to each other. It uses the k-nearest neighbour search to compute similar instances. ReliefF with $k = 5$ is slightly more accurate than ReliefF with $k = 1$, but both still rank behind Fisher score. Although r-STSF with ReliefF is competitive in terms of accuracy against r-STSF with Fisher score, ReliefF is known to be slow on large training sets (Urbanowicz et al. 2018), and hence we have used Fisher score by default.

6.3 Number of candidate discriminatory interval feature sets

The extraction of d sets of candidate discriminatory interval features may become expensive on large datasets with very long series. This parameter has a direct impact on the efficiency of r-STSF. The process to extract a single set of candidate intervals (i.e., $d = 1$) is summarized in Algorithm 1. r-STSF’s predecessor, STSF, extracts a set of intervals for each tree of the ensemble, e.g., for an ensemble with 500 trees, STSF extracts 500 sets of candidate interval features. In r-STSF, we propose to extract 50 sets of interval features (i.e., $d = 50$) regardless of the ensemble size. The training process of r-STSF does not only differ from that of STSF in the number of extracted sets of intervals, but also in how such features are used to build each of the trees in the ensemble. As detailed in Sect. 4.4, STSF uses each set of intervals to build each of the r trees in the ensemble, whereas r-STSF merges all sets into a superset $\mathscr{F}$. Then, each trees is built by using a set of randomly selected interval features from $\mathscr{F}$.

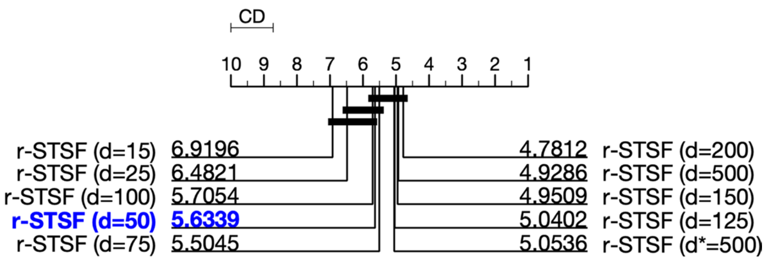


Fig. 18 Critical difference diagram of average ranks of r-STSF with varying numbers of sets of candidate discriminatory interval features, i.e., d. When $d \geq 50$, the average ranks are close and statistically similar. This validates our decision of just computing a small number of sets of candidate intervals. The special case of training the ensemble of trees as in STSF (i.e., one set per tree – r-STSF (d*=500)) is an order of magnitude slower and not significantly different from r-STSF (d=50). This validates our decision to build each tree from features randomly selected from a larger super set

We study the classification accuracy of r-STSF when setting d to $\{15, 25, 50, 75, 100, 125, 150, 200, 500\}$. We also include a special case, $d^* = 500$, where each of the 500 trees of the ensemble is build as in STSF (i.e., each set of intervals builds one tree at a time). As shown in Fig. 18, after extracting 50 sets of candidate interval features ($d = 50$), there is no significant difference in terms of classification accuracy. Thus, for r-STSF, we set d to 50 by default for efficiency considerations. Using $d = 50$ instead of $d = 500$ or $d^* = 500$ makes r-STSF an order of magnitude faster in the feature extraction process without significantly affecting its effectiveness.

6.4 Time series representations

r-STSF uses four time series representations to extract discriminatory interval features, including the original (raw), i.e., X_O, periodogram, i.e., X_P, derivative, i.e., X_D, and autoregressive representations, i.e., X_G, of the training time series. As presented in Fig. 19, the best rank is achieved by using all four time series

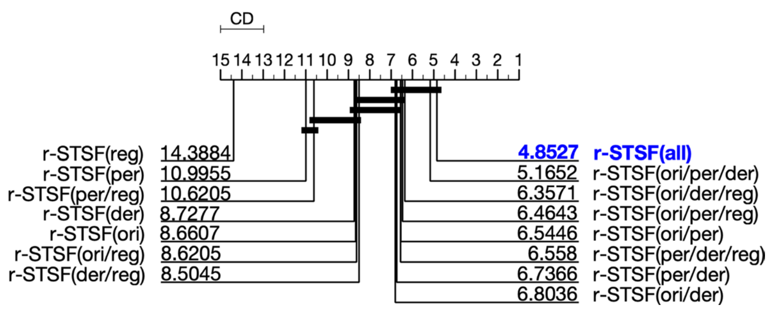


Fig. 19 Critical difference diagram of average ranks of r-STSF with different combinations of time series representations: original (ori), periodogram (per), derivative (der), and autoregressive (reg). r-STSF is more effective when using all four time series representations, i.e., r-STSF (all)

representations. Individually, X_O is the most effective (i.e., r-STSF (ori)). Using only X_G (i.e., r-STSF (reg)) makes r-STSF perform poorly (in terms of accuracy).

X_G is relevant when past values have an effect on current values of a given time series. If a time series does not hold this premise, it is challenging to classify such series by just using this representation. Moreover, even for a series holding this premise it is necessary to estimate the correct lag order which allows to identify the lagged relationships within a series. If a large number of time series classification problems from UCR_{112} cannot be modeled by an autoregressive process, then using just the autoregressive representation of the series is not enough to achieve accurate classifications. However, when comparing the classification accuracy of r-STSF (all) and r-STSF (ori/per/der) (Fig. 37, Appendix B) we find that the addition of X_G improves the classification accuracy in many datasets. In some datasets such as Car, Ham, OSULeaf, Phoneme, ProximalPhalanxOutlineCorret, ScreenType, SonyAIBORobotSurface2, Wine, and Worms, this improvement is between 2 and 6%.

6.5 Aggregation functions

We evaluate the impact of the aggregation functions (aggregation statistics) on the classification accuracy of r-STSF, which by default uses nine statistics to compute the interval features (detailed in Sect. 4.2). For each statistic, we compute r-STSF by (i) using that statistic and (ii) removing it from the set of (nine) aggregation functions. As shown in Fig. 20, generic statistics such as the mean, median, and min contribute slightly to the effectiveness of r-STSF. The most important statistics are slope and max, which yield the individual best ranks (i.e., r-STSF (slope) and r-STSF (max)). Their importance is also reflected when they are removed from the set of aggregation functions (i.e., r-STSF (no slope) and r-STSF (no max)), which leads to large drops in the average rank. Besides, although individually cmc and cam are the least important, if they are removed from the aggregation functions, r-STSF has

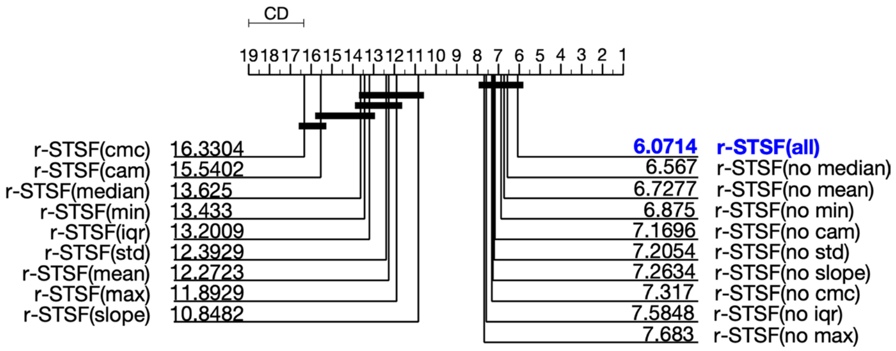


Fig. 20 Critical difference diagram of average ranks of r-STSF with different combinations of aggregation functions (or statistics). r-STSF (*statistic*) means using r-STSF just with the respective *statistic*. r-STSF (no *statistic*) means using all aggregation functions but *statistic*. r-STSF (all) uses all the aggregation functions and is the most effective approach

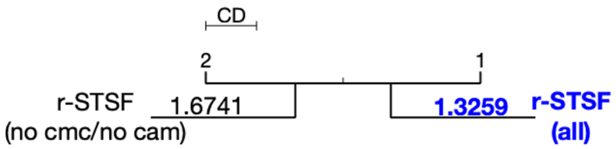


Fig. 21 Critical difference diagram of average ranks of r-STSF with and without cmc and cam aggregation functions. r-STSF (no cmc/cam) means using r-STSF without cmc and cam statistics. r-STSF (all) uses all the aggregation functions and is the most effective approach

a substantial drop in its average rank (similar to removing the slope statistic). This suggests that cmc and cam are more effective when combined with other statistics.

As shown in Fig. 21, not including the cmc and cam aggregation functions makes r-STSF significantly less accurate when comparing with the version of r-STSF which includes them into the set of statistics. Moreover, as shown in Fig. 38, Appendix C, the addition of cmc and cam statistics improves the classification accuracy in many datasets. In some datasets such as Beef, Ham, RefrigerationDevices, ToeSegmentation1, Wine, and WormsTwoClasses, this improvement is between 2 and 5%.

6.6 Impact of the perturbation scheme

r-STSF uses a novel perturbation scheme to create an ensemble of uncorrelated trees for higher accuracy. Our scheme employs (i) random partitions when assessing the discriminatory quality of sub-series and (ii) randomized trees to build the ensemble of trees for classification. We evaluate the impact of these steps in this section.

Random partitions: r-STSF's predecessor, STSF, uses the middle-point of the sub-series as a cut point and builds fixed partitions when searching for candidate discriminatory interval features. In Sect. 4.2, we showed that such a technique may not be the best if the goal is to create an ensemble of uncorrelated trees. In r-STSF, we rely on random partitions when assessing the discriminatory quality of sub-series, which increases the average accuracy of r-STSF by 1% compared to using fixed partitions. Moreover, as shown in Fig. 22, r-STSF with random partitions is significantly more accurate than r-STSF with fixed partitions.

Randomized trees: The Extra-trees (ET) algorithm builds an ensemble of randomized binary trees to decrease the variance of the ensemble (by creating uncorrelated trees) and thus achieves high classification accuracy. In randomized trees, the cut-point of each feature is randomly selected when looking for the feature that provides the best split. These trees are different from those in a random forest (RF),

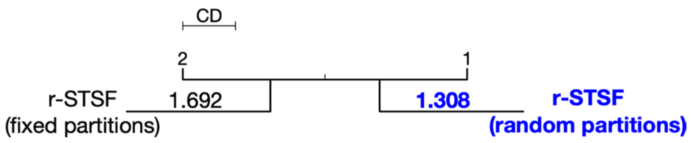


Fig. 22 Critical difference diagram of average ranks of r-STSF with fixed and random partitions. r-STSF with random partitions is significantly more accurate than r-STSF with fixed partitions

where for each feature only the best cut-point (i.e., split with the lowest entropy) is selected. r-STSF's predecessor, STSF, follows RF and builds non-randomized binary trees. For r-STSF, we create an ensemble of randomized binary trees.

As detailed in Table 3, r-STSF first extracts a set of candidate discriminatory interval features, i.e., $\mathscr{F}$. Then, (at node level) it randomly selects $\sqrt{\mathscr{F}}$ interval features from $\mathscr{F}$ when looking for the best random split. In this section, we denote r-STSF as r-STSF (ET) and compare it against two variants of r-STSF which use non-randomized trees and are denoted as r-STSF (RF) and r-STSF (RF all), respectively. r-STSF (RF) selects $\sqrt{\mathscr{F}}$ interval features to split the tree nodes and builds an ensemble of non-randomized trees, whereas r-STSF (RF all) also builds non-randomized trees but uses all features from $\mathscr{F}$ for the splits. Thus, r-STSF (RF) builds “more" randomized trees than r-STSF (RF all), but “less" randomized than r-STSF (ET). We also include in our comparisons a variant of r-STSF, denoted as r-STSF (ET 1), that randomly selects a single interval feature from $\mathscr{F}$ to split each node and builds randomized trees. Hence, on r-STSF (ET 1), the feature and its cut-point are randomly selected. Trees built in this manner are known as *totally randomized trees* (Geurts et al. 2006) and are “more" randomized than those of r-STSF (ET).

As shown in Fig. 23, r-STSF (ET) is significantly more accurate than the versions of r-STSF using non-randomized trees. Moreover, the poor performance of r-STSF (ET 1) suggests that there is a limit on the level of perturbation that can be applied when building uncorrelated trees. Trees that are “too” uncorrelated negatively impact the classification accuracy.

7 Explainable classification

To provide further insights to the classification decisions of r-STSF, we use the discriminatory intervals computed by it. Each discriminatory interval has information on its interval boundary (i.e., starting and ending indices), the aggregation function (e.g., mean), and the time series representation (e.g., periodogram) that was used to generate the corresponding feature. To highlight the regions of the

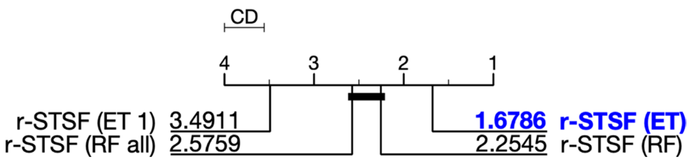


Fig. 23 Critical difference diagram of average ranks of r-STSF at different levels of perturbation. r-STSF (ET 1) is the most extreme case where a single feature and cut-point are randomly selected to split the nodes. It follows r-STSF (ET) where the nodes are split according to the random cut-point (from a group of randomly selected features) that provides the best split. Next, r-STSF (RF) which is similar to r-STSF (ET), selects a group of features but looks for the best cut-point instead of selecting one at random. r-STSF (RF all) also looks for the best split, but among all the available features, i.e., not from a selected group. r-STSF (ET) is significantly more accurate than the other versions of r-STSF

time series which are discriminatory, i.e., maximizing class separability, we count the number of times that each data point of a testing series is contained in the discriminatory intervals. We normalize such counts and use a heatmap to visualize the most interesting regions.

As discussed in Sect. 4, r-STSF uses different *time series representations* for classification. These representations allow for explanations directly, i.e., it is not necessary to rely only on relevant features from the original series. In this section, we present cases where the original time series does not support the identification of discriminatory features and show how our additional representations provide more discriminatory features while enabling explainability. More importantly, since r-STSF uses simple statistics to compute its features, *our explanations are not only limited to highlighting discriminatory regions, i.e., we can also provide explanations for the classifier's decision*. For example, if a discriminatory interval was extracted with the slope, it can be explained that sub-series from one class may exhibit a different trend from that of sub-series of another class.

7.1 Integrating explainability on time series classification

TSC methods that integrate explainability into their classifications do so by identifying and highlighting relevant time-stamps that can potentially provide further insights to the classification decision. TSF and CIF capture discriminatory features embedded in sub-series, whose location in the time series (i.e., time-stamps) is highlighted by the *temporal importance curve* mechanism (Deng et al. 2013). RSF and gRSF adapt ideas from the temporal importance curve and propose an importance measure for the shapelet scenario. They identify time-stamps and shapelet lengths relevant for the classifier's decision. However, discriminatory or relevant features are not always presented in the time series in their original form (i.e., time-stamped data), but can be found in additional representations/transformations of the time series (Bagnall et al. 2012). For example, mtSAX-SEQL+LR (Le Nguyen et al. 2019) captures discriminatory features in symbolic representations of the time series. Given that symbol-based representations do not directly support explainability, mtSAX-SEQL+LR maps the features back to their locations in the (original) series to integrate explainability. ResNet and FCN use class activation map (CAM) (Wang et al. 2017) to highlight the sub-series that contribute the most for a given class identification.

The mechanism used by mtSAX-SEQL+LR, ResNet and FCN to highlight revelant sub-series is known as *saliency maps*. Saliency maps are often regarded as "less explainable" mechanisms (Rudin 2019). They show which sub-series are relevant for a class, but cannot provide more information to explain what the model does with that sub-series, i.e., what characteristics of the data are relevant for the classifier's decision. For instance, as shown in Fig. 24, ResNet only highlights the relevant sub-series to each class when explaining the model's decision to classify the *Meat* time series dataset. r-STSF's explainability mechanism not only highlights relevant sub-series but provides the most important time series representation and the most important aggregation functions (detailed in Sect. 7.2) to identify discriminatory

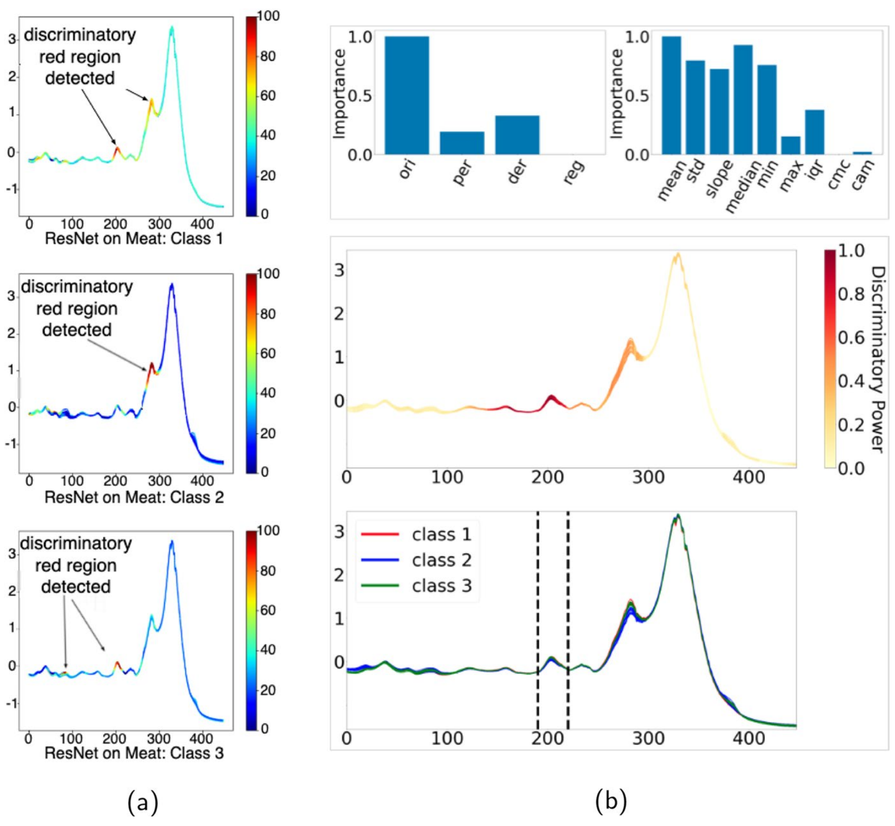


Fig. 24 **a** ResNet's explainability mechanism using CAM (figure adapted from Wang et al. (2017)). **b** r-STSF's explainability mechanism using relevant intervals computed from original time series representation and the mean aggregation function to explain for the classification of the *Meat* dataset. ResNet highlights the sub-series relevant to each class but cannot provide any other information to explain for the classifier's decision. r-STSF uses the time series in their original form (i.e., time-stamped data) to highlight relevant sub-series to all classes, and the mean aggregation function suggests that the classifier could identify series from different classes due to a difference in their mean values in the highlighted regions

sub-series when classifying the dataset. This extra level of information can be used to explain with more details what characteristics of the data is considered by the model when making its decision.

7.2 Computing importance of time series representations and aggregation functions

Using different time series representations increases the chances to find discriminatory interval features. Depending on the time series dataset, some representations may be more important than the others. Similarly, depending on the time series representation, there are aggregation functions which may contribute more than the others when extracting discriminatory intervals.

Discriminatory interval features are located in the nodes of our tree-based ensemble, i.e., $\mathscr{F}^*$ in Fig. 3. Each of these interval features keeps a record of the time series representation and aggregation function used for its computation. To identify which time series representation and which aggregation functions contribute the most over a given dataset, we estimate the importance of their corresponding features by computing the *mean decrease impurity* (MDI) (Louppe et al. 2013) of such features. At each tree node θ, the interval feature f that achieves the largest impurity decrease is selected as the best split S_θ. When using *entropy* as the impurity measure, the information gain (IG) computes the impurity decrease (see Eq. (4)). Thus, the importance of an interval feature f is estimated by adding up the weighted impurity decrease $\rho_\theta \mathrm{IG}(f, \theta)$ of every node θ where f is used (to split that node), averaged over all r trees in the forest:

$$\mathrm{MDI}(f) = \frac{1}{r} \sum_{\top} \sum_{\theta \in \top : S_\theta = f} \rho_\theta \mathrm{IG}(f, \theta) \tag{5}$$

where $\top$ is a tree classifier and ρ_θ is the probability of reaching node θ. Probability ρ_θ is calculated as the number of time series instances that reach the node, divided by the total number of time series instances n. The higher the MDI, the more important the interval feature is. Time series representations and aggregation functions used to compute *important* interval features are also considered to be more important. To estimate the importance of a representation (or aggregation function), we average the importance of their corresponding features. For example, to estimate the importance of the periodogram representation, we compute the average of the MDI (i.e., importance) of all the features extracted from this representation. After estimating the importance of the remaining representations (i.e., original, derivative and autoregressive), the importance values are normalized. A similar process is used to estimate the importance of each aggregation function.

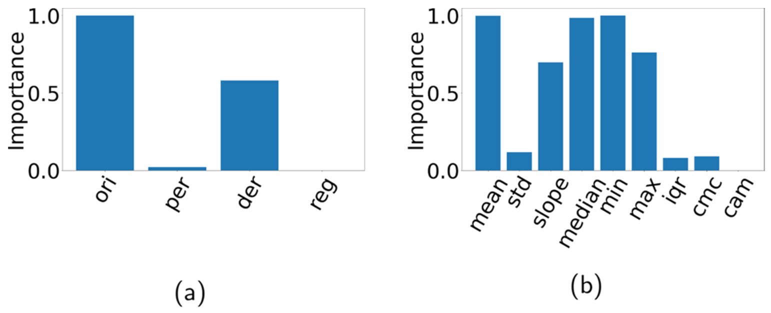


Fig. 25 **a** Importance of time series representations and **b** importance of aggregation functions when classifying the ItalyPowerDemand dataset with r-STSF. The original (ori) representation is the most important. The mean and min aggregation functions are the most important when extracting discriminatory intervals from the original representation of the dataset

7.3 Explaining classifications with the original time series

r-STSF uses different time series representations for classification. When the time series contain intervals with discriminatory time-stamped data (e.g., class A series have values above threshold *T*, whereas class B series have values below *T*), the original representation of the series is enough to achieve an accurate classification. As a case study, we use the ItalyPowerDemand dataset (Bagnall et al. 2019), which consists of series of daily household energy consumption. When classifying this dataset with r-STSF, the original time series representation is the most important (Fig. 25a). Besides, as shown in Fig. 25b, in this representation, the mean and min aggregation functions contribute the most to the classification accuracy. We select the min aggregation to continue with our case (the mean shows similar results and is omitted).

The most discriminatory interval (according to the min aggregation function) is located between 19:00 and 20:00 (Fig. 26a). As shown in Fig. 26b, between 19:00 and 20:00 (highlighted between the pair of dashed black lines), the majority of winter series (red color) are above the summer series (blue color). This interval shows that, in winter, by 20:00, residents reach a peak in the energy consumption of their households appliances, whereas in summer, at that hour of the day, the energy consumption is smaller, which can be explained by residents' tendency to spend earlier times indoors in winter due to shorter days and cooler temperatures.

7.4 Explaining classifications with the periodogram representation

In datasets such as LargeKitchenAppliances (Bagnall et al. 2019) which contains energy consumption readings from three home appliances (dishwasher, tumble dryer, and washing machine), the variability in the time of using each appliance makes it difficult to identify discriminatory intervals in the original time series (Fig. 27).

As shown in Fig. 28a, the original representation has a low importance in the classification of the LargeKitchenAppliances dataset whereas the periodogram representation is much more important. Moreover, mean is the aggregation function

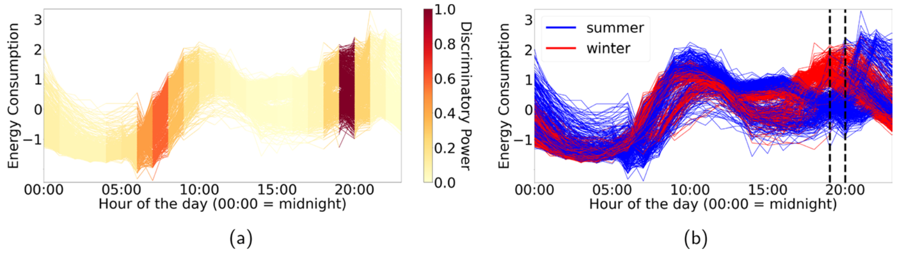


Fig. 26 Original representations of all time series in the ItalyPowerDemand dataset. **a** Location (in time) of discriminatory intervals according to the min aggregation function. **b** The two types of series can be differentiated according to their minimum energy consumption between 19:00 and 20:00. Summer days are in blue color; winter days are in red color (Color figure online)

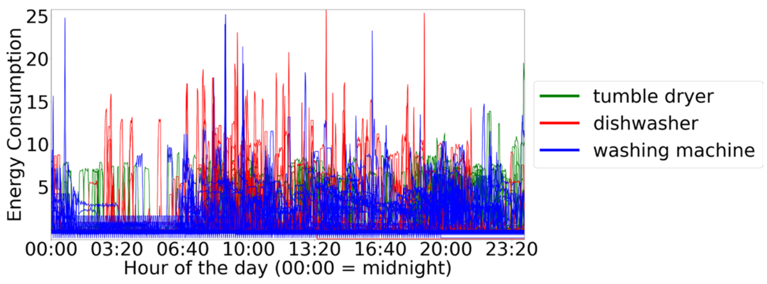


Fig. 27 LargeKitchenAppliances (original) time series. It is difficult to identify discriminatory intervals in the original representation of the series. The few cases of high energy consumption for the dishwasher (red color) around 03:20 are outliers and hence are not discriminatory for r-STSF. Most of dishwasher series have (in that hour) small energy consumption values, similar to those of washing machine (blue color) and tumble dryer (green color) series (Color figure online)

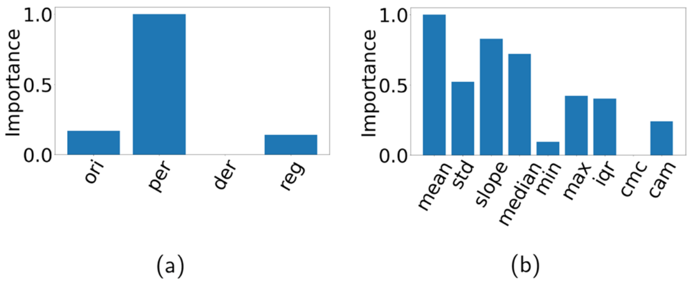


Fig. 28 **a** Importance of time series representations and **b** importance of aggregation functions when classifying the LargeKitchenAppliances dataset with r-STSF. The periodogram (per) representation is the most important representation when classifying this dataset. The mean aggregation function is the most important when extracting discriminatory intervals from the periodogram representation of the dataset

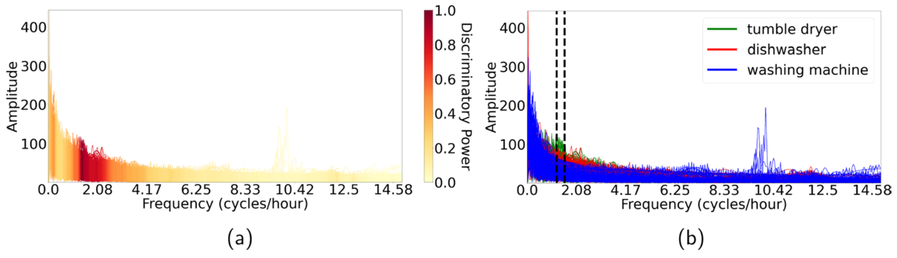


Fig. 29 Periodogram representations of all time series in LargeKitchenAppliances dataset. **a** Location (in the frequency-domain) of discriminatory intervals according to the mean aggregation function. **b** The three types of appliances can be differentiated according to their amplitude around the frequency of 1 cycle/hour. Tumble dryer series are in green color, dishwasher series in red color and washing machine series in blue color (Color figure online)

that contributes the most to the classification accuracy when extracting features from the periodogram representation (Fig. 28b).

The periodogram representation shows how the signal's amplitude is distributed over a range of frequencies. The frequencies with higher amplitude reveal repeated patterns at such frequencies. As shown in Fig. 29a, r-STSF uses the periodogram representation of LargeKitchenAppliances series to identify discriminatory intervals (using the mean aggregation function) located approximately at the frequency of 1 cycle/hour. From Fig. 29b, we see that, around this frequency (highlighted between the pair of dashed black lines), there is a difference in the amplitude of the three types of appliances. The energy consumption signal of each appliance belongs to one operation cycle. The discriminatory frequency of 1 cycle/hour reveals that all three appliances have the most different energy consumption around the one-hour mark on average while running these appliances. The higher amplitude of this frequency for the tumble dryer than for the other two appliances suggests that householders use the tumble dryer more often during the day (e.g., one hour in the morning and one hour in the evening) or that it takes more than one hour to complete the drying process (when overloaded, the tumble dryer usually requires two hours). Similarly, this discriminatory frequency also suggests that householders use the dishwasher more often than the washing machine.

7.5 Explaining classifications with the first-order derivative representation

To demonstrate how r-STSF allows for explainable classifications in the derivative representation, we discuss the case of SonyAIBORobotSurface2 dataset (Bagnall et al. 2019) in this subsection. This dataset contains readings from an X-axis accelerometer attached to a quadruped Sony AIBO dog robot while walking on two different surfaces: cement and carpet. Each time series has a length of 65 data points (we assume 65 s of walking). The classification task is to detect the surface being walked on. Although an inspection to the original series of this dataset may lead to potential discriminatory intervals (Fig. 30), the derivative representation has a much more important role when classifying the series (Fig. 31a).

The derivative representation is useful when time series can be differentiated by their trends or changes in the series values. The derivative representation is the first-order difference of a time series, and hence, every data point of this representation

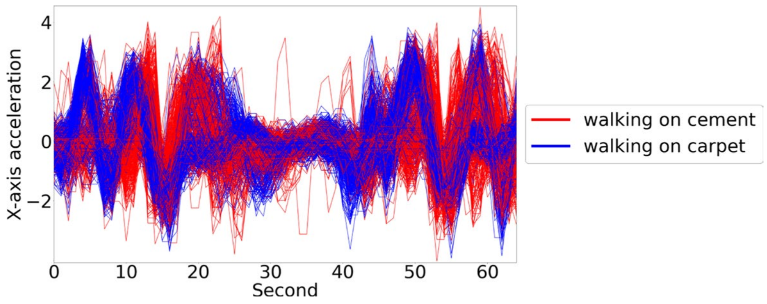


Fig. 30 SonyAIBORobotSurface2 (original) time series. Potential discriminatory intervals are located at second 10 and at second 50. At both locations, walking on cement (red color) and walking on carpet (blue color) signals are out-of-phase (Color figure online)

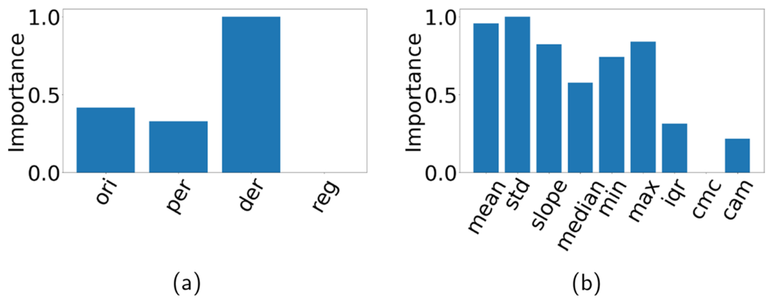


Fig. 31 **a** Importance of time series representations and **b** importance of aggregation functions when classifying the SonyAIBORobotSurface2 dataset with r-STSF. The derivative (der) representation is the most important representation, and std is the most important aggregation function when extracting discriminatory intervals from the derivative representation

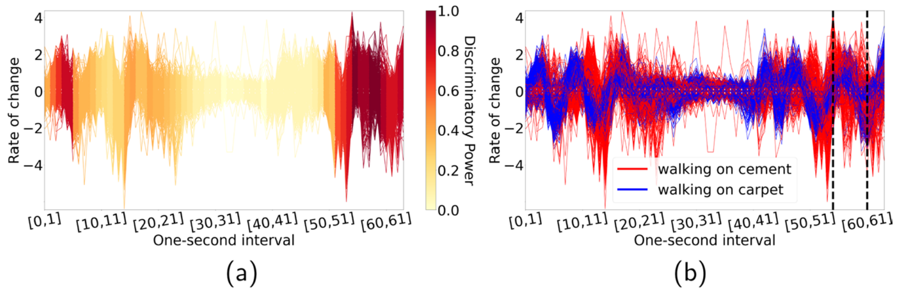


Fig. 32 Derivative representations of all time series in the SonyAIBORobotSurface2 dataset. **a** Location of discriminatory intervals according to the std aggregation function. **b** The two types of surfaces can be differentiated for the high variability of the AIBO robot acceleration when walking on cement in the time interval between second 55 and second 60 of the original time series. Walking on cement series are in red color; walking on carpet series are in blue color

can be mapped back to a pair of original data points (i.e., interval) of the original representation. As stated by Kertész (2014): "When a robot walks on a rigid surface, it produces vertical body oscillations while soft surfaces absorb these anomalies". Hence, walking on cement increases the variability (or rate of change) of the X-axis acceleration readings. The derivative representation is thus more useful to classify the SonyAIBORobotSurface2 series. Similarly, the std aggregation function, which measures the spread of the values, is also important to detect discriminatory interval in this representation (Fig. 31b).

As shown in Fig. 32a, std detects discriminatory intervals at the beginning (first 5 s of walking) and at the end (last 10 s of walking) in the derivative representations. A higher difference in the variability of the acceleration values between walking on cement and on carpet can be found approximately in the time interval between seconds 55 to 60 (highlighted between the pair of dashed black lines in Fig. 32b). We have no further information of the data collection process for the SonyAIBORobotSurface2 dataset, which limits our explanations for the higher variability (of walking on cement) at the beginning and the end of the walking sessions. One explanation could be that due

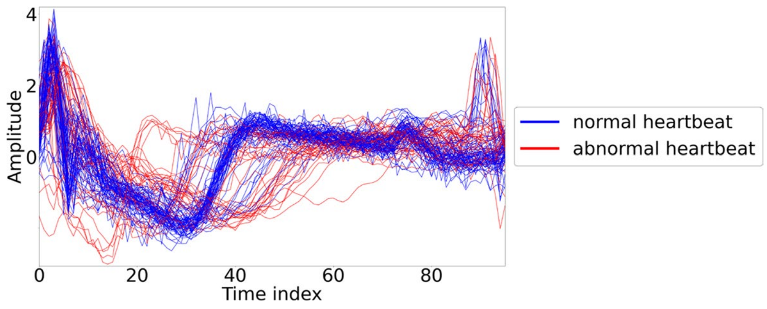


Fig. 33 ECG200 (original) time series. There is not a specific interval where normal (blue color) and abnormal (red color) series are easily differentiable (Color figure online)

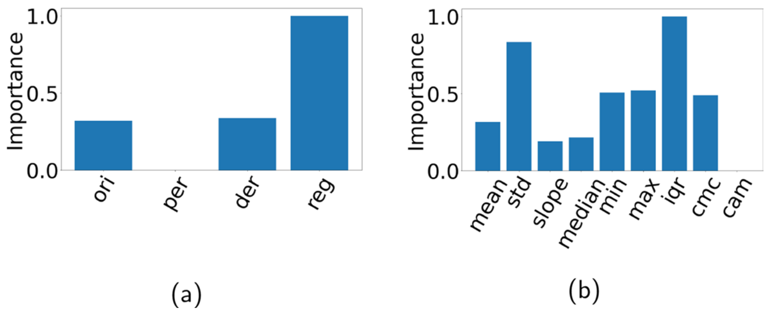


Fig. 34 **a** Importance of time series representations and **b** importance of aggregation functions when classifying the ECG200 dataset with r-STSF. The autoregressive (reg) representation is the most important representation. The iqr aggregation function is the most important aggregation function when extracting discriminatory intervals from the derivative representation

to the inertia (i.e., tendency of a body to resist a change in motion or rest), the acceleration varies when the robot starts its movement and also when it is about the stop.

7.6 Explaining classifications with the autoregressive representation

We use the ECG200 dataset (Bagnall et al. 2019; Olszewski 2001) to show how r-STSF uses the autoregressive representation for classification and how to explain such classification. Each series in ECG200 contains the measurements recorded by one electrode during one heartbeat. The heartbeats are labeled as normal and abnormal. The original representation of ECG200 series (Fig. 33) may not be the most informative to capture discriminatory intervals. Normal and abnormal series follow a similar pattern in time, which makes it difficult to identify discriminatory intervals.

As shown in Fig. 34a, the autoregressive representation is identified as the most important representation by r-STSF. Moreover, std and iqr, which capture dispersion or variability of the data points, extract the most discriminatory intervals in the autoregressive representation (Fig. 34b). An autoregressive (AR) model holds the premise

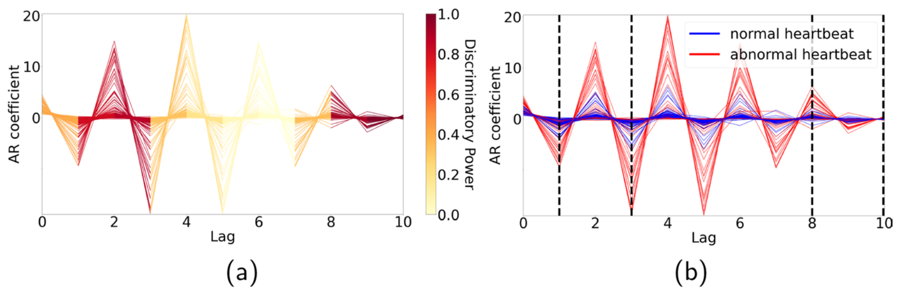


Fig. 35 Autoregressive representations of all time series in the ECG200 dataset. **a** Location of discriminatory intervals according to the iqr aggregation function. **b** The variability of the AR coefficients of the abnormal heartbeats (red color) is higher than those of the normal heartbeats (blue color) between lags 1 and 3 and between lags 8 and 10 (Color figure online)

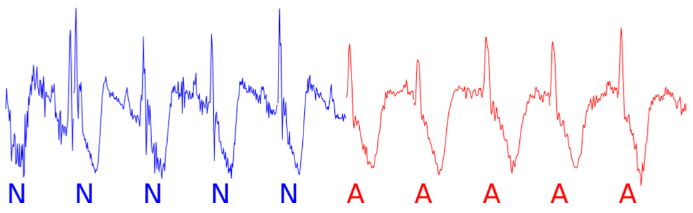


Fig. 36 Example of ten ECG200 series. N: normal heartbeats (blue), A: abnormal heartbeats (red) (Color figure online)

that past values have an impact on current values. For example, in an AR process with lag order of 1 (denoted by "AR(1)"), the current value is predicted based on the immediate preceding value using a linear model. The coefficients of such predictors are the AR coefficients, which can be used as features for time series classification (Lines et al. 2018). r-STSF does not use the AR coefficients directly as features, but it uses them as another representation of the original time series, i.e., AR representation. Thus, r-STSF extracts discriminatory interval features from the AR representation.

As shown in Fig. 35a, the most discriminatory interval features are located between lags 1 and 3 and also between lags 8 and 10. The variability of the AR coefficients of series from abnormal heartbeats is much higher than that from series of normal heartbeats between these lags. r-STSF uses this difference in the variability to differentiate normal and abnormal heartbeats.

The AR coefficients of abnormal heartbeats are usually larger than those of normal heartbeats (Fig. 35). AR coefficients do not provide specific information on the relationship of the variables, i.e., AR coefficients cannot tell to which extent current and past value are correlated. Nonetheless, a high AR coefficient implies that past values have some impact on current values whereas a low AR coefficient implies a small or no impact. Hence, for ECG200, past values from abnormal series have some impact on current values. On the contrary, in normal heartbeats there is a small or no impact of past values on current ones. As shown in Fig. 36, normal heartbeats are more irregular or noisy than abnormal heartbeats. Therefore, it is difficult to establish a correlation between past values and current ones.

Further, from Fig. 35, we can infer that the impact of past 2, 3, 8, and 9 values on current ones is much higher for abnormal heartbeats than for normal heartbeats. Although for abnormal heartbeats there are also high AR coefficients at lags 4 and 5; the iqr aggregation function considers such values as extreme or outliers, thus the interval between such lags is not considered as discriminatory.

8 Conclusions and future work

We proposed r-STSF, an extremely fast and highly efficient interval-based algorithm for time series classification. r-STSF integrates explainability into its classification and achieves SOTA classification accuracy. To achieve competitive classification accuracies, r-STSF builds an ensemble of randomized trees for classification. It uses four time series representations, nine aggregation functions, and a supervised search strategy combined with a feature ranking metric when searching for highly discriminatory sets of interval features. The discriminatory interval features enable explainable classification results. Extensive experiments on real-world datasets validate the accuracy and efficiency of our proposed method – r-STSF is as accurate as SOTA TSC methods but orders of magnitude faster, enabling it to classify large datasets with long series.

While randomized trees have shown to improve the classification accuracy (when compared to non-randomized trees) for a large number of datasets, they are less likely to identify relevant features in datasets with a small number of relevant features as they might miss those. As future work, we plan to make r-STSF adaptive. When a dataset is expected to have a high percentage of relevant features – estimated with techniques such as *permutation importance* (Louppe et al. 2013) – r-STSF trains an ensemble of randomized trees for classification. Otherwise, r-STSF uses non-randomized trees. Moreover, the intrinsic multivariate nature of time series signals (e.g., 3-axis accelerometer) leads us to extend r-STSF towards multivariate or multidimensional scenarios. Other methods such as CIF extend to multivariate scenarios by searching for discriminatory interval features in different dimensions of a series. However, extracting interval features per dimension (i) could potentially miss discriminatory features only found when data from every dimension is combined, e.g., an interval from series of the x, y, and z-axis of the accelerometer may not be discriminatory when assessed by separate, but it is discriminatory when data from each axis is combined, and (ii) may hinder the explainability when classifying datasets with a high number of dimensions due to such explainability has to be done per dimension (by highlighting discriminatory regions in the series from each dimension). We plan to transform the individual (per dimension) time series into a single (unified) representation from where to extract discriminatory interval features.

We consider r-STSF highly relevant to enable further studies on interval-based classifiers. Results from an extensive experimental study on TSC methods (Bagnall et al. 2017) reported that interval-based TSC methods were the least competitive family of TSC methods. However, more recent studies including ours have positioned the interval-based methods among the best ranking TSC methods. As shown in Fig. 6, r-STSF ranks just behind time and memory expensive hybrid methods such as HC2 and TS-CHIEF, and non-explainable by design kernel-based approaches such as ROCKET. Our proposed r-STSF achieves similar classification accuracy to that of DrCIF while being two orders of magnitude faster.

Appendices

A Results on 112 UCR benchmark time series datasets

See Table 9

Table 9 Average classification accuracy over 10 runs of r-STSF for each dataset of UCR_{112}. Other methods are as reported in their papers or accompanying websites

Datasets	HC2	DrCIF	HC1	CHIEF	RKET	Itime	STC	TDE	STSF	r-STSF
ACSF1	93.00	88.00	**94.00**	84.00	87.80	89.60	89.00	92.00	81.00	90.40
Adiac	81.33	82.10	81.07	79.80	78.47	82.97	81.59	78.01	81.94	**83.55**
ArrowHead	86.29	80.57	82.29	83.27	80.51	84.69	69.14	**88.57**	68.00	74.06
Beef	86.67	76.67	**90.00**	70.61	83.33	68.67	83.33	83.33	83.67	87.00
BeetleFly	95.00	90.00	95.00	91.36	90.00	80.00	80.00	**100.00**	90.00	92.00
BirdChicken	90.00	90.00	**95.00**	90.91	90.00	**95.00**	90.00	**95.00**	90.00	90.50
BME	**100.00**	**100.00**	95.33	99.33	**100.00**	99.33	94.00	87.33	99.33	99.87
Car	**91.67**	88.33	83.33	85.45	89.17	89.00	85.00	85.00	80.00	87.50
CBF	**100.00**	99.89	99.89	99.79	99.99	99.78	96.56	99.89	97.94	99.17
Chinatown	98.25	**98.84**	97.96	96.81	98.02	98.31	97.38	96.23	97.96	98.57
ChlConcentration	77.14	73.85	73.88	71.67	81.30	**87.28**	75.21	70.39	77.99	77.82
CinCECGTorso	**100.00**	**100.00**	99.49	98.32	83.49	84.22	99.13	99.71	98.89	99.93
Coffee	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
Computers	76.40	73.60	77.60	70.51	76.00	**78.56**	67.20	68.40	76.40	73.20
CricketX	82.56	74.62	80.77	81.38	82.23	**84.05**	79.23	80.77	67.95	74.51
CricketY	**85.13**	80.51	82.31	80.19	85.03	83.90	77.69	81.54	74.92	77.21
CricketZ	**86.41**	81.03	82.31	83.40	85.77	84.92	81.03	81.54	71.54	77.23
Crop	76.51	**78.19**	77.26	76.42	75.02	75.10	73.29	71.90	76.46	77.70
DiatomSizeReduc	95.75	86.27	93.46	**97.30**	97.03	93.46	79.74	88.89	96.11	92.52
DisPhaOutAgeGro	74.82	74.82	**75.54**	74.62	75.47	73.38	**75.54**	74.82	72.66	73.02
DisPhaOutCorr	77.54	78.62	77.17	78.23	76.78	76.81	77.90	73.91	**80.07**	78.12
DisPhaTW	70.50	69.78	67.63	67.04	**71.87**	66.47	71.22	67.63	67.63	68.06
Earthquakes	74.82	74.82	74.82	74.82	74.82	74.24	74.82	74.82	74.10	**75.25**
ECG200	89.00	89.00	86.00	86.18	90.60	**91.80**	84.00	87.00	86.60	89.70
ECG5000	94.71	94.29	**94.73**	94.54	94.70	93.93	94.36	94.47	94.36	94.39
ECGFiveDays	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.88	**100.00**	98.01	99.48
ElectricDevices	**75.76**	73.56	74.59	75.53	73.05	70.86	73.97	70.90	73.83	74.03
EOGHorizSignal	**65.75**	61.88	61.05	65.47	64.09	58.78	58.84	55.52	57.46	57.15

Table 9 (continued)

Datasets	HC2	DrCIF	HC1	CHIEF	RKET	Itime	STC	TDE	STSF	r-STSF
EOGVertiSignal	56.91	58.56	54.70	**59.67**	54.23	46.41	48.90	46.96	51.10	52.82
EthanolLevel	69.60	59.80	68.40	52.80	58.20	**80.36**	72.40	45.60	61.00	61.84
FaceAll	86.80	76.39	77.87	84.14	**94.75**	80.08	74.62	77.81	90.00	92.64
FaceFour	**100.00**	**100.00**	98.86	**100.00**	97.50	95.68	73.86	**100.00**	97.95	98.86
FacesUCR	96.34	90.73	95.90	**96.63**	96.16	96.40	91.32	96.05	88.65	89.51
FiftyWords	83.30	78.24	77.58	**84.50**	83.05	80.66	74.95	78.46	76.20	76.99
Fish	98.86	94.29	**99.43**	**99.43**	97.89	97.60	93.14	**99.43**	89.49	92.91
FordA	95.38	96.89	95.08	94.10	94.49	95.73	92.95	93.33	96.67	**97.68**
FordB	83.09	81.60	83.21	82.96	80.63	**84.89**	81.36	81.98	81.11	83.01
FreezRegTrain	**99.96**	**99.96**	99.75	99.75	99.76	99.65	99.93	99.02	99.93	99.92
FreezSmTrain	99.89	**99.96**	98.56	99.79	95.19	86.57	99.79	94.98	99.82	98.14
GunPoint	**100.00**	99.33	**100.00**	**100.00**	**100.00**	**100.00**	99.33	**100.00**	92.27	97.00
GunPAgSpan	99.68	99.37	99.68	**100.00**	99.68	98.73	98.10	**100.00**	96.84	98.99
GunPMalVsFem	**100.00**	**100.00**	**100.00**	**100.00**	99.78	99.56	97.47	99.68	99.37	**100.00**
GunPOldVsYng	**100.00**	**100.00**	**100.00**	**100.00**	99.05	96.19	96.83	99.68	**100.00**	**100.00**
Ham	69.52	73.33	70.48	71.52	72.57	70.48	75.24	59.05	70.48	**76.95**
HandOutlines	93.78	91.89	93.24	93.22	94.16	**94.65**	92.16	90.81	92.16	91.57
Haptics	54.55	51.95	53.90	51.68	52.50	**54.87**	53.90	51.95	50.32	51.56
Herring	67.19	60.94	67.19	58.81	68.59	66.56	67.19	59.38	**68.75**	60.47
HouseTwenty	96.64	91.60	98.32	97.48	96.39	97.48	98.32	**99.16**	96.64	91.93
InlineSkate	55.27	54.91	48.55	52.69	45.82	48.51	43.64	50.18	55.75	**66.75**
InsEPGRegTrain	**100.00**	**100.00**	**100.00**	**100.00**	99.96	99.84	99.60	**100.00**	**100.00**	**100.00**
InsEPGSmTrain	**100.00**	**100.00**	**100.00**	**100.00**	98.15	94.14	97.99	**100.00**	96.79	**100.00**
InsWngbeatSnd	66.26	**68.84**	65.96	64.29	65.66	63.04	63.99	59.49	67.73	66.80
ItaPowerDem	97.18	97.28	96.02	97.06	96.91	96.42	94.95	95.43	97.08	**97.31**
LargeKitApp	**90.67**	82.40	85.33	80.68	90.00	90.03	82.40	74.67	78.67	80.64
Lightning2	**78.69**	77.05	77.05	74.81	76.39	**78.69**	68.85	77.05	70.98	76.72

Table 9 (continued)

Datasets	HC2	DrCIF	HC1	CHIEF	RKET	Itime	STC	TDE	STSF	r-STSF
Lightning7	**82.19**	75.34	72.60	76.34	**82.19**	80.27	75.34	69.86	75.89	76.85
Mallat	**97.78**	91.86	97.27	97.50	95.60	94.06	93.90	93.39	96.86	96.57
Meat	93.33	93.33	91.67	88.79	94.50	93.33	**98.33**	88.33	95.00	95.00
MedicalImages	80.79	78.95	73.16	79.58	79.75	78.66	71.45	75.92	78.46	**81.67**
MidPhaOutAgeGro	59.74	**61.04**	59.09	58.32	59.55	52.34	57.14	60.39	58.44	59.35
MidPhaOutCorr	85.22	83.16	83.51	**85.35**	84.12	81.65	81.44	75.95	81.79	83.61
MidPhaTW	55.84	58.44	58.44	55.02	55.58	50.78	59.74	**61.69**	59.09	59.68
MixShapRegTrain	**97.57**	95.63	96.95	97.11	97.04	96.62	96.00	97.48	93.69	94.90
MixShapSmTrain	**95.88**	92.12	95.38	94.93	93.86	91.16	93.86	95.05	88.74	89.59
MoteStrain	**97.04**	94.41	96.81	94.75	91.42	88.61	94.49	94.25	92.19	94.48
NonInvFetECGTho1	94.71	93.49	92.42	91.13	95.14	**95.62**	93.59	83.41	93.20	93.64
NonInvFetECGTho2	96.74	94.10	94.50	94.50	**96.88**	95.79	94.71	87.18	93.82	94.60
OliveOil	**93.33**	**93.33**	86.67	88.79	92.67	82.00	90.00	86.67	**93.33**	90.00
OSULeaf	96.28	85.95	98.35	**99.14**	93.80	92.48	95.45	95.04	79.71	84.88
PhalangesOutCorr	83.10	83.68	82.05	**84.50**	83.00	83.75	82.63	76.92	83.10	84.06
Phoneme	35.55	**40.61**	38.08	36.91	27.96	32.81	35.28	36.81	32.01	39.76
PigAirwayPressure	93.27	28.37	96.15	97.60	8.85	53.17	**96.63**	93.27	19.23	37.50
PigArtPressure	**99.52**	90.38	99.04	98.08	95.29	99.33	98.08	**99.52**	50.00	92.40
PigCVP	**96.63**	64.90	96.15	96.15	93.27	95.29	94.23	**96.63**	28.37	68.32
Plane	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
PowerCons	98.33	99.44	**100.00**	98.89	93.11	94.78	96.11	92.22	**100.00**	**100.00**
ProPhaOutAgeGro	85.37	84.39	83.90	84.97	85.51	84.49	85.37	**85.85**	84.88	85.41
ProPhaOutCorr	90.03	89.69	89.00	88.82	89.90	91.75	90.72	86.94	91.41	**92.06**
ProPhaTW	**82.93**	79.51	82.44	81.86	81.61	78.15	80.00	80.98	77.56	79.12
RefrigDev	53.87	**61.07**	57.87	55.83	53.47	52.27	53.87	53.60	57.60	59.04
Rock	92.00	88.00	**96.00**	90.00	89.80	75.20	**96.00**	82.00	76.00	75.80
ScreenType	57.87	54.13	55.73	50.81	48.56	57.97	**62.93**	45.33	54.13	54.61

Table 9 (continued)

Datasets	HC2	DrCIF	HC1	CHIEF	RKET	Itime	STC	TDE	STSF	r-STSF
SeHndGendCh2	95.67	94.17	**97.00**	92.33	92.30	80.23	93.67	86.33	**97.00**	96.60
SeHndMovCh2	85.56	85.56	87.11	**87.78**	64.44	42.00	78.67	68.22	81.33	83.49
SeHndSubCh2	90.22	92.22	**95.33**	92.44	88.36	78.71	90.00	84.89	88.89	88.93
ShapeletSim	**100.00**	99.44	**100.00**	**100.00**	**100.00**	91.67	**100.00**	**100.00**	98.89	97.89
ShapesAll	92.17	85.50	**93.17**	93.00	90.82	91.83	83.00	93.00	83.83	86.12
SmallKitApp	83.73	81.87	83.47	82.21	82.13	75.57	81.33	78.13	**84.27**	82.35
SmoothSubspace	98.67	98.00	98.00	**100.00**	97.93	98.13	92.67	86.00	98.00	98.00
SonAIBORobSurf1	90.85	89.18	76.71	82.64	**92.41**	86.39	79.03	60.23	91.68	89.58
SonAIBORobSurf2	92.97	90.24	94.23	92.48	91.64	94.61	**94.33**	90.77	83.44	87.40
StarLightCurves	98.22	98.08	98.06	**98.24**	98.11	97.78	97.52	97.62	97.82	97.94
Strawberry	97.57	96.49	97.03	96.63	98.19	**98.27**	96.22	97.03	95.95	96.84
SwedishLeaf	96.48	96.16	95.36	96.55	**96.59**	96.35	93.44	94.08	94.26	95.54
Symbols	97.59	96.38	97.99	97.66	97.46	**98.03**	96.58	96.18	88.56	97.24
SyntheticControl	**100.00**	99.67	99.67	99.79	99.70	99.60	99.33	99.33	99.03	99.00
ToeSegm1	96.49	92.54	**97.37**	96.53	97.02	96.14	96.05	96.49	76.75	84.74
ToeSegm2	93.85	87.69	**96.15**	95.38	92.62	94.31	94.62	94.62	88.46	87.92
Trace	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	98.90	**100.00**
TwoLeadECG	99.91	99.74	99.82	99.46	99.91	99.67	**100.00**	**100.00**	98.94	98.44
TwoPatterns	**100.00**	**100.00**	99.93	**100.00**	**100.00**	**100.00**	98.10	99.88	99.66	99.69
UMD	99.31	97.92	97.92	98.61	99.24	98.19	93.75	97.92	99.31	**99.79**
UWaveGestLibAll	97.43	97.26	96.79	96.89	**97.57**	94.37	95.59	94.61	95.20	95.59
UWaveGestLibX	**85.46**	83.67	82.89	84.11	**85.46**	81.38	80.65	82.64	81.27	82.88
UWaveGestLibY	**77.47**	76.55	74.65	77.23	77.29	75.46	71.19	76.21	73.78	75.75
UWaveGestLibZ	**79.73**	78.11	77.72	78.44	79.17	75.02	76.30	77.95	75.92	76.81
Wafer	**100.00**	99.90	99.97	99.91	99.83	99.85	**100.00**	**100.00**	99.97	99.97
Wine	87.04	79.63	79.63	**89.06**	80.74	65.93	70.37	66.67	64.81	77.78
WordSynonyms	75.24	69.28	69.75	**78.74**	75.52	73.23	65.99	76.33	63.61	65.39

Table 9 (continued)

Datasets	HC2	DrCIF	HC1	CHIEF	RKET	Itime	STC	TDE	STSF	r-STSF
Worms	75.32	76.62	64.94	**80.17**	72.73	76.88	72.73	72.73	76.62	79.22
WormsTwoClass	79.22	77.92	76.62	81.58	79.87	78.18	**81.82**	76.62	79.22	80.52
Yoga	**93.40**	87.67	91.17	83.47	90.85	89.06	88.07	92.60	82.90	85.59
Average Accuracy	**87.67**	**84.89**	**86.51**	**86.34**	**85.47**	**84.44**	**84.56**	**84.01**	**82.62**	**84.94**

Bold classification accuracies are used to highlight the most accurate methods for each time series dataset

B Comparison of r-STSF with and without the autoregressive representation

See Fig. 37

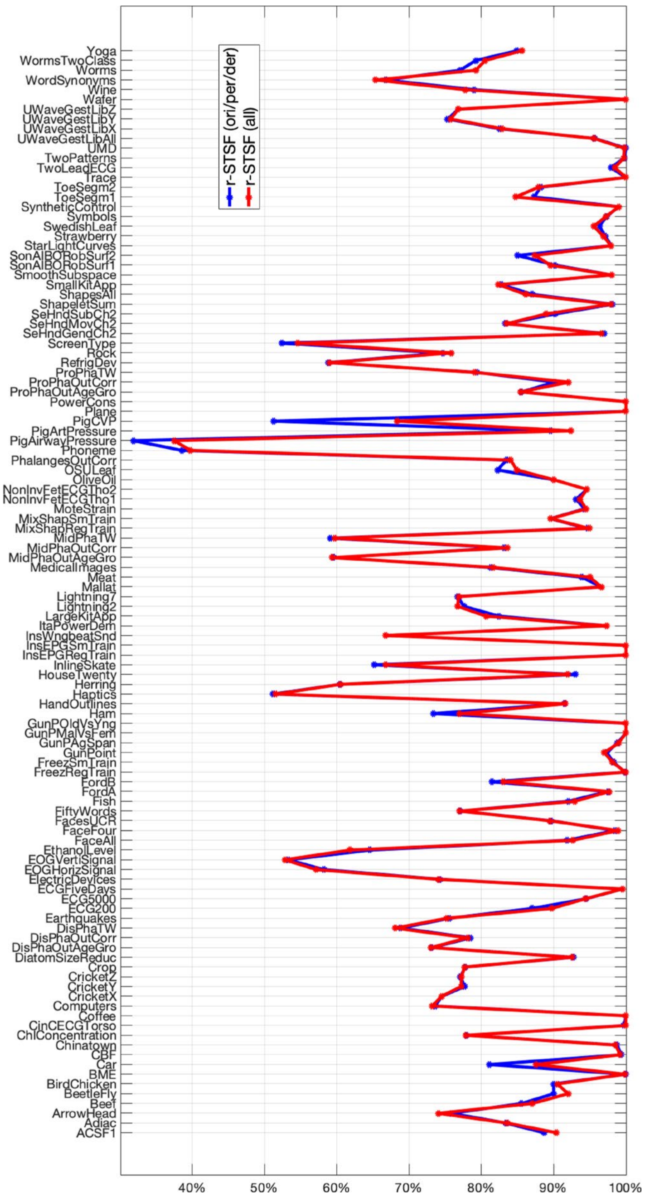


Fig. 37 Comparison of average accuracy (x-axis) of r-STSF when using three time series representations (r-STSF (ori/per/der)) and when using four time series representations (i.e., including autoregressive representation - r-STSF (all))

C Comparison of r-STSF with and without the *counts of mean-crossings* (cmc) and *counts of values above the mean* (cam) aggregation functions

See Fig. 38

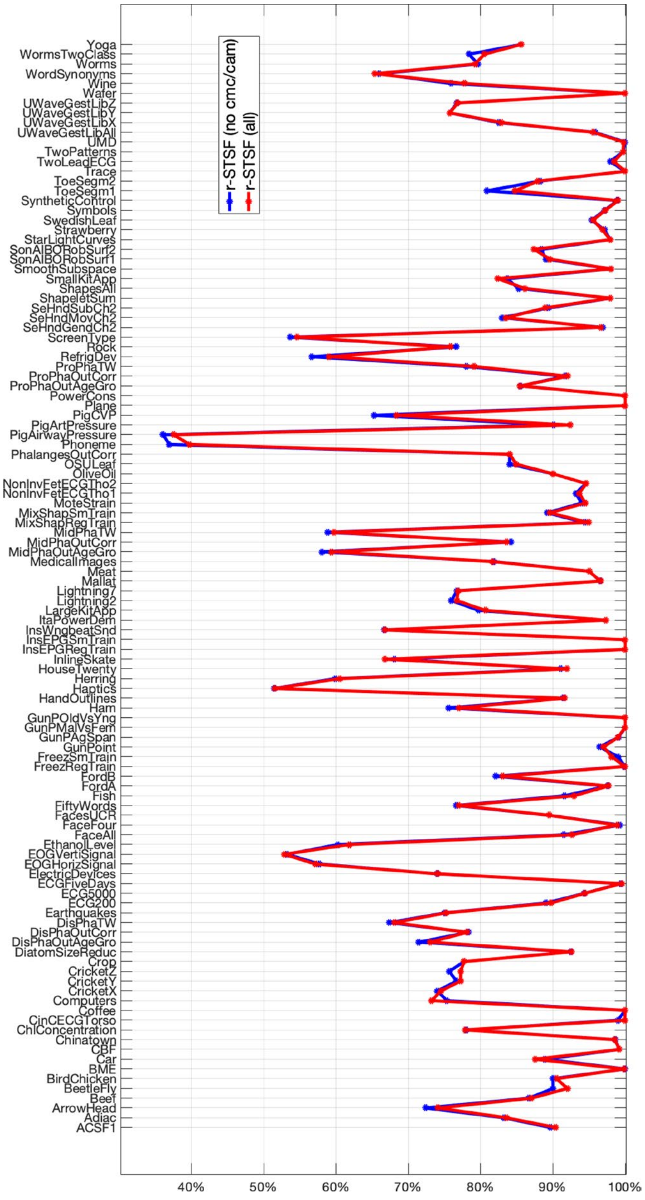


Fig. 38 Comparison of average accuracy (x-axis) of r-STSF when using all nine aggregation functions (r-STSF (all)) and when using seven aggregation functions (i.e., without cmc and cam aggregation functions - r-STSF (no cmc/cam))

Acknowledgements This research was partially supported under Australian Research Council's Discovery Projects funding scheme (project number DP170102472). The experiments were supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. This research was partially supported under Australian Research Council's Discovery Projects funding scheme (project number DP170102472).

Availability of data and materials Not applicable.

Code availability We provide our source code at https://github.com/stevcabello/r-STSF.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest.

Ethical approval Not applicable.

References

Bagnall A, Davis L, Hills J, Lines J (2012) Transformation based ensembles for time series classification. In: Proceedings of the 2012 SIAM international conference on data mining (SDM), pp 307–318

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 Min Knowl Discov 31(3):606–660

Bagnall A, Lines J, Vickers W, Keogh E (2019) The UEA & UCR time series classification repository. www.timeseriesclassification.com

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 1.0). In: International workshop on advanced analytics and learning on temporal data (AALTD), pp 3–18

Bailly A, Malinowski S, Tavenard R, Chapel L, Guyet T (2016) Dense bag-of-temporal-SIFT-words for time series classification. In: International workshop on advanced analytics and learning on temporal data (AALTD), pp 17–30

Baydogan MG, Runger G (2015) Learning a symbolic representation for multivariate time series classification. Data Min Knowl Discov 29:400–422

Baydogan MG, Runger G (2016) Time series representation and similarity based on local autopatterns. Data Min Knowl Discov 30(2):476–509

Baydogan MG, Runger G, Tuv E (2013) A bag-of-features framework to classify time series. IEEE Trans Pattern Anal Mach Intell 35(11):2796–2802

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Brockwell PJ, Davis RA, Calder MV (2002) Introduction to time series and forecasting. Springer, Berlin

Cabello N, Naghizade E, Qi J, Kulik L (2020) Fast and accurate time series classification through supervised interval search. In: 2020 IEEE 20th international conference on data mining (ICDM), pp 948–953

Dempster A, Petitjean F, Webb GI (2020) ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Min Knowl Discov 34:1454–1495

Dempster A, Schmidt DF, Webb GI (2021) Minirocket: A very fast (almost) deterministic transform for time series classification. In: Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining (KDD), pp 248–257

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Inf Sci 239:142–153

Duda RO, Hart PE, Stork DG (2012) Pattern classification. John Wiley and Sons

Fawaz HI, Forestier G, Weber J, Idoumghar L, Muller PA (2019) Deep learning for time series classification: a review. Data Min Knowl Discov 33(4):917–963

Fawaz HI, Lucas B, Forestier G, Pelletier C, Schmidt DF, Weber J, Webb GI, Idoumghar L, Muller PA, Petitjean F (2020) InceptionTime: finding AlexNet for time series classification. Data Min Knowl Discov 34:1936–1962

Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42

Górecki T, Łuczak M (2013) Using derivatives in time series classification. Data Min Knowl Discov 26(2):310–331

Görgülü B, Baydogan MG (2021) Randomized trees for time series representation and similarity. Pattern Recognit 120:108097

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 (KDD), pp 392–401

He X, Cai D, Niyogi P (2005) Laplacian score for feature selection. In: Proceedings of the 18th international conference on neural information processing systems (NIPS), pp 507–514

Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014) Classification of time series by shapelet transformation. Data Min Knowl Discov 28(4):851–881

Ifrim G, Wiuf C (2011) Bounded coordinate-descent for biological sequence classification in high dimensional predictor space. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining (KDD), pp 708–716

Karlsson I, Papapetrou P, Boström H (2015) Forests of randomized shapelet trees. In: Proceedings of statistical learning and data sciences (SLDS), pp 126–136

Karlsson I, Papapetrou P, Boström H (2016) Generalized random shapelet forests. Data Min Knowl Discov 30(5):1053–1085

Karpagachelvi S, Arthanari M, Sivakumar M (2012) Classification of electrocardiogram signals with support vector machines and extreme learning machine. Neural Comput Appl 21(6):1331–1339

Keogh EJ, Pazzani MJ (2001) Derivative dynamic time warping. In: Proceedings of the 2001 SIAM international conference on data mining (SDM), pp 1–11

Kertész C (2014) Exploring surface detection for a quadruped robot in households. In: IEEE international conference on autonomous robot systems and competitions (ICARSC), pp 152–157

Large J, Bagnall A, Malinowski S, Tavenard R (2019) On time series classification with dictionary-based classifiers. Intell Data Anal 23(5):1073–1089

Le Nguyen T, Gsponer S, Ilie I, O'Reilly M, Ifrim G (2019) Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. Data Min Knowl Discov 33(4):1183–1222

Li J, Cheng K, Wang S, Morstatter F, Trevino RP, Tang J, Liu H (2018) Feature selection: a data perspective. ACM Comput Surv 50(6):94

Lin J, Keogh E, Lonardi S, Chiu B (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery (DMKD), pp 2–11

Lin J, Khade R, Li Y (2012) Rotation-invariant similarity in time series using bag-of-patterns representation. J Intell Inf Syst 39(2):287–315

Lines J, Bagnall A (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Discov 29(3):565–592

Lines J, Taylor S, Bagnall A (2018) Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles. ACM Trans Knowl Discov Data 12(5):52

Löning M, Bagnall A, Ganesh S, Kazakov V, Lines J, Király FJ (2019) sktime: A unified interface for machine learning with time series. arXiv preprint arXiv:1909.07872

Louppe G, Geurts P (2012) Ensembles on random patches. In: Joint european conference on machine learning and knowledge discovery in databases (ECML PKDD), pp 346–361

Louppe G, Wehenkel L, Sutera A, Geurts P (2013) Understanding variable importances in forests of randomized trees. In: Proceedings of the 26th international conference on neural information processing systems (NIPS), pp 431–439

Lubba CH, Sethi SS, Knaute P, Schultz SR, Fulcher BD, Jones NS (2019) catch22: canonical time-series characteristics. Data Min Knowl Discov 33(6):1821–1852

Lucas B, Shifaz A, Pelletier C, O'Neill L, Zaidi N, Goethals B, Petitjean F, Webb GI (2019) Proximity forest: an effective and scalable distance-based classifier for time series. Data Min Knowl Discov 33(3):607–635

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

Middlehurst M, Large J, Bagnall A (2020a) The canonical interval forest (CIF) classifier for time series classification. In: 2020 IEEE international conference on big data (Big Data), pp 188–195

Middlehurst M, Large J, Cawley G, Bagnall A (2020b) The temporal dictionary ensemble (TDE) classifier for time series classification. In: Joint european conference on machine learning and knowledge discovery in databases (ECML PKDD), pp 660–676

Middlehurst M, Large J, Flynn M, Lines J, Bostrom A, Bagnall A (2021) HIVE-COTE 2.0: a new meta ensemble for time series classification. Mach Learn 110(11):3211–3243

Olszewski RT (2001) Generalized feature extraction for structural pattern recognition in time-series data. PhD thesis, Carnegie Mellon University

Pattarin F, Paterlini S, Minerva T (2004) Clustering financial time series: an application to mutual funds style analysis. Comput Stat Data Anal 47(2):353–372

Proakis JG, Manolakis DG (2014) Digital signal processing: principles algorithms and applications. Pearson

Rakthanmanon T, Keogh E (2011) Fast-shapelets: A fast algorithm for discovering robust time series shapelets. In: Proceedings of 11th SIAM international conference on data mining (SDM), pp 668–676

Robnik-Šikonja M, Kononenko I (2003) Theoretical and empirical analysis of ReliefF and RReliefF. Mach Learn 53(1–2):23–69

Rodriguez JJ, Kuncheva LI, Alonso CJ (2006) Rotation forest: a new classifier ensemble method. IEEE Trans Pattern Anal Mach Intell 28(10):1619–1630

Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1(5):206–215

Samsten I (2020) wildboar. https://github.com/wildboar-foundation/wildboar

Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Min Knowl Discov 29(6):1505–1530

Schäfer P, Högqvist M (2012) SFA: A symbolic Fourier approximation and index for similarity search in high dimensional datasets. In: Proceedings of the 15th international conference on extending database technology (EDBT), pp 516–527

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

Schwert GW (1989) Tests for unit roots: a Monte Carlo investigation. J Bus Econ Stat 7:147–159

Shifaz A, Pelletier C, Petitjean F, Webb GI (2020) TS-CHIEF: a scalable and accurate forest algorithm for time series classification. Data Min Knowl Discov 34:742–775

Tan CW, Dempster A, Bergmeir C, Webb GI (2022) MultiRocket: multiple pooling operators and transformations for fast and effective time series classification. Data Min Knowl Discov 36:1623–1646

Urbanowicz RJ, Meeker M, La Cava W, Olson RS, Moore JH (2018) Relief-based feature selection: introduction and review. J Biomed Inform 85:189–203

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

Zhou B, Khosla A, Lapedriza A, Oliva A, Torralba A (2016) Learning deep features for discriminative localization. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp 2921–2929

Authors and Affiliations

Nestor Cabello[1] · **Elham Naghizade**[2] · **Jianzhong Qi**[1] · **Lars Kulik**[1]

✉ Nestor Cabello
ncabello@student.unimelb.edu.au

Elham Naghizade
e.naghizade@rmit.edu.au

Jianzhong Qi
jianzhong.qi@unimelb.edu.au

Lars Kulik
lkulik@unimelb.edu.au

1 The University of Melbourne, Melbourne, Australia

2 RMIT University, Melbourne, Australia