Feature extraction by grammatical evolution for one-class time series classification

Stefano Mauceri¹ · James Sweeney² · Miguel Nicolau³ · James McDermott⁴

Received: 30 January 2020 / Revised: 28 October 2020 / Accepted: 13 March 2021 /
Published online: 23 April 2021
© The Author(s) 2021

Abstract

When dealing with a new time series classification problem, modellers do not know in advance which features could enable the best classification performance. We propose an evolutionary algorithm based on grammatical evolution to attain a data-driven feature-based representation of time series with minimal human intervention. The proposed algorithm can select both the features to extract and the sub-sequences from which to extract them. These choices not only impact classification performance but also allow understanding of the problem at hand. The algorithm is tested on 30 problems outperforming several benchmarks. Finally, in a case study related to subject authentication, we show how features learned for a given subject are able to generalise to subjects unseen during the extraction phase.

Keywords Evolutionary computation · One-class classification · Time series

Area Editor: Una-May O'Reilly.

✉ Stefano Mauceri
stefano.mauceri@ucdconnect.ie

James Sweeney
sweeneja@tcd.ie

Miguel Nicolau
miguel.nicolau@ucd.ie

James McDermott
james.mcdermott@nuigalway.ie

¹ Natural Computing Research and Applications Group (NCRA), University College Dublin, Dublin, Ireland

² University of Limerick, Limerick, Ireland

³ University College Dublin, Dublin, Ireland

⁴ National University of Ireland, Galway, Ireland

1 Introduction

Feature extraction should normally be customised to the problem at hand. Often, when modellers deal with a new problem they do not know which features could enable the best classification performance. Thus, a common approach is to construct an initial set of features, and then select the subset yielding best performance [34].

In contrast to the manual approach, there is growing interest in algorithms that enable the data-driven discovery of features, as made possible by deep learning methods [28]. The advantage is that modellers can redirect their efforts from the construction of the solution to the construction of the learning framework. While the former may be useful solely on a particular problem the latter may be effective on many.

In the present study, we investigate grammatical evolution (GE) [43], an evolutionary algorithm related to genetic programming (GP) [2, 27], as a means to achieve data-driven feature extraction from time series in the context of classification.

The feature-based approach to time series classification (TSC) is convenient for a variety of reasons [15, 39, 54]. (1) To reduce data to a manageable size. In particular, this can mitigate the curse of dimensionality, reduce computational requirements, and allow visualisation of time series data-sets. (2) To highlight properties of a class of time series enabling understanding of the problem at hand. (3) To reduce the impact of noise and missing values. (4) To deal with time series of different length.

Our algorithm sequentially extracts a user-defined number of features. At each step, a run of the algorithm returns a feature that is intended to minimise the classification error while being minimally correlated with features extracted during previous steps. By combining a set of primitive functions e.g. mean/standard deviation, GE creates more complex functions we refer to as feature-extractors. Each feature-extractor takes a time series as input and returns a single feature (a scalar). This can be seen as combining feature extraction and feature selection within the same algorithm.

Previous research on TSC proved that class membership may depend on features related to the whole time series, or on features related to one or more of its sub-sequences [5]. Thus, we enable our algorithm to search for both the features to extract and the sub-sequences from which to extract them. We show that these choices are central not only for classification performance, but also to allow understanding.

The classification performance of extracted features is evaluated using a one-class classifier [37] (Sect. 3.4). One-class classification is concerned with learning a classifier when all training samples belong to a single class. We are motivated by the requirements of a subject authentication problem we have recently investigated [36]. The aim of subject authentication is to confirm the identity of a person. Both binary and multi-class methods assume, in a sense, a fixed population of subjects well-represented in the data, which is not realistic for this scenario. Thus, the best option may be a semi-supervised one-class classifier tailored to the intended subject only.

Although we use a one-class classifier, our algorithm requires labelled data to evaluate the fitness function that drives the extraction process. This is different from a pure one-class classification scenario where only the samples of a single class are available for training a classifier. However, we use our subject authentication problem to show how features evolved using a one-class classifier are able to generalise to classes unseen during the extraction phase (Sect. 6.1).

We compare our algorithm against a 1-nearest neighbour classifier equipped with dynamic time warping (1NN-DTW) on raw data, considered as the standard benchmark in the literature [5] (Sect. 4.2). Finally, we conduct an experimental analysis to demonstrate the impact of interval/function selection on performance (Sect. 4.3).

The remainder of this work is organised as follows. In Sect. 2, we provide a brief introduction to feature-based TSC. In Sect. 3, we describe our GE-based algorithm. In Sect. 4, we describe the benchmark methods. In Sect. 5, we describe the experimental design. In Sects. 6–7, we analyse our results. Finally, in Sect. 8, we draw our conclusions, and discuss future work.

2 Related work

We briefly introduce feature extraction in Sect. 2.1. Works related to feature-based TSC are reviewed in Sect. 2.2, evolutionary approaches in Sect. 2.3.

The literature related to feature-based TSC reveals that there are only a few works that investigate the one-class assumption. One of the major contribution of the present study is to expand the discussion on one-class TSC.

The literature related to evolutionary approaches to TSC reveals a number of gaps that we seek to address. There is a lack of consistent comparison of proposed methods with relevant benchmarks. We address this issue by evaluating our approach on 30 problems, all but one from the UCR/UEA archive [10]. The UCR/UEA archive is the reference archive of problems for TSC researchers. Also, we compare our results against a 1NN-DTW on raw data considered as the standard benchmark in the literature [5]. It appears that it is not clear how to evolve multiple features that are not redundant. We tackle this problem by sequentially extracting the features, and requiring that their individual classification performance is maximised, while their average correlation with previously extracted features is minimised. Finally, we analyse evolved solutions to expand the discussion on problem/solution interpretability enabled by evolutionary techniques (Sect. 7).

2.1 Overview

Features are distinctive aspects of something. In machine learning, feature extraction aims at finding the most informative set of features for a certain task [22]. However, in most cases, the features to use are not known in advance. Thus, modellers go through time consuming trial/error procedures to extract and select most informative features.

A feature-based representation affects the choice and performance of the classifier and vice versa [14]. For instance, the quality of a feature-based representation may be overestimated if overfitting occurs, however, it may be underestimated if the classifier relies on assumptions violated by the feature-based representation (e.g. a normality assumption for a Gaussian-mixture classifier). In practice, expert practitioners first exploit their domain knowledge to extract a set of features. Then, they carry out a statistical analysis of the features before selecting an appropriate classifier. In contrast, our algorithm extracts features that are not only suited for the problem at hand, but are appropriate for the classifier used in the extraction process.

Finally, our algorithm allows modellers to extract features from time series without requiring a previous specific knowledge of the field. This is in line with an increasingly popular research area known as automated machine learning (AutoML) [19]. The goal of AutoML is to automate the typical machine learning pipeline (e.g. data pre-processing, feature extraction, model and hyper-parameter selection) reducing the amount of experts' work required to deploy a model.

2.2 Feature-based TSC

- **Description:** Time series are transformed into feature-vectors which are used to complete the classification task through any off-the-shelf classifier.
- **Pros:** Most feature-based approaches facilitate problem and solution interpretability, and ease computational complexity.
- **Cons:** It is not trivial to identify distinctive features especially when a new problem is addressed. Most feature-based approaches are expected to be weak on problems where features can be shifted along the time axis.

There are only few works investigating one-class TSC [37]. In a previous experimental study, we evaluate several approaches to derive dissimilarity-based representations for one-class TSC [37]. In dissimilarity-based representations, given a time series, each feature corresponds to its dissimilarity (distance) from one of a set of “reference” time series. Results show that the choice of the pair dissimilarity measure and set of “reference” time series is key to classification performance.

Two key questions that arise when dealing with feature-based TSC concern the features to extract, and the sub-sequence from which to extract them. Neither answer is obvious. The feature extraction process can be manual [35], or automated [36]. Features can be as general as statistical moments, or more carefully designed for time series like the time-reversal asymmetry statistic [49].

Assuming that the most effective features are not known in advance, some authors propose to extract several features and then retain only the best ones. Deng et al. [12] propose a tree-ensemble classifier called “time series forest” (similar to a random forest). In this case tree nodes calculate simple statistics on randomly selected sub-sequences. Building on this idea, Shifaz et al. [50] propose to exploit the strengths of successful TSC algorithms creating three novel splitting criteria. Their ensemble of decision trees achieves competitive performance. How to deploy their approach to the one-class classification domain is an interesting topic for future work.

Dempster et al. [11] apply thousands of random convolutional filters on time series. Resulting feature maps are summarised through two ad-hoc statistics, then concatenated in vectors, and used for classification. This approach exploits the success of convolutional neural networks for TSC, as reported in a recent comprehensive study on deep learning methods for TSC [16]. Finally, the method relies on the implicit feature selection enabled by a ridge regression classifier in order to avoid overfitting.

Lubba et al. [34] consider 4791 features for TSC. Of these, they retain only 22 features because of their classification performance on considered data-sets, and their minimal redundancy. This small subset named “catch22” represents the state of the art for feature-based TSC, and we use it to benchmark our algorithm.

Concluding, other approaches to feature-based TSC include dictionary-based methods, and graph features. In dictionary-based methods, also known as bag-of-patterns methods [32], time series are transformed into strings using SAX [33]. Then, each feature corresponds to the frequency of occurrence of a specific sub-sequence (or word) within a time series. Graph-based techniques first transform a time series into a graph, then features are extracted from this new representation [30].

2.3 Evolutionary approaches

The majority of prior research on evolutionary techniques for TSC has applied to ECG time series [18], and sensor time series for fault detection [31]. Some studies, discussed below, propose general purpose feature extraction algorithms [15, 23, 52].

Eads et al. [15] use GE [43] to evolve a single population of feature-extractors using a set of 25 primitives. Each feature-extractor is able to target any sub-sequence and return a single scalar. This hill-climbing algorithm does not make use of any crossover operator. However, it allows modifications to the current solution (addition, deletion, mutation of feature-extractors) only if changes increase the performance, or cause a negligible impact on performance but a decrease in run-time. Classification performance, tested on seven data-sets, is better than that of raw data.

Harvey and Todd [23] propose an algorithm based on GP [27] where 35 primitives are used to evolve sets of feature-extractors. Nearly all the primitives take a time series as input and output a transformed time series. To reduce time series to a single scalar each feature-extractor must end with a summation. The authors provide a number of rules to reduce redundancy of final feature-extractors and increase their interpretability. Performance is tested on simulated data only.

Finally, Virgolin et al. [52] propose a GP-based algorithm for sequential feature construction but not on time series. While feature extraction concerns the extraction of features from raw data, feature construction concerns the transformation of existing features. They focus on the interpretability of the GP trees emphasising interpretability of the original features, and limiting the height of the trees. As discussed in Sect. 7, we are also interested in interpretability. We point out that in TSC interpretability is aided by knowing which functions, and sub-sequences allow good classification performance.

3 Proposed method

In this section we describe our evolutionary algorithm for feature extraction from time series based on GE. The core components of our approach are the grammar (Sect. 3.2), and the fitness function (Sect. 3.3).

3.1 Overview

Our algorithm relies on GE, a grammar-based form of GP. We choose GE to implement our algorithm because, as opposed to GP, it allows us to handle a mixture of data types. While the type constraint could be handled with other approaches e.g. strongly typed GP [38], we believe that the grammar is a particularly convenient way to express the syntax of admissible solutions, and also we want solutions to be readable Python code as this can facilitate understanding.

The grammar allows the modeller to exploit her/his domain knowledge, and to impose syntactical constraints to guide the search of feasible solutions. In this study we have used our knowledge of the TSC domain to define the proposed grammar. We have designed the grammar to be *balanced* rather than *explosive* [41], giving a bias towards short solutions. In many cases, the primitives included in the grammar are high-level functions specifically defined for time series e.g. the complexity estimate [6]. However, a practitioner specialising in a particular sub-domain of TSC could add further domain knowledge, e.g. spectral features known to be useful in that domain. In addition, the grammar allows the selection of any sub-sequence. Sub-sequences may be key to class membership as demonstrated by TSC algorithms like shapelets [55], or bag-of-patterns [32].

The solution space of our algorithm consists of feature-extractors. Each feature-extractor F is a function which takes as input a specific sub-sequence $[a : b]$ of a time series T e.g. $T[20 : 50]$. The output is a single scalar i.e. a feature. In Eq. 1 is shown an example of a feature-extractor. Note that all the primitives included in a given feature-extractor are applied to the same sub-sequence.

$$F = \text{Mean}(T[20 : 50]) \times \text{Skewness}(T[20 : 50]) \quad (1)$$

Our algorithm outputs a single feature-extractor. Thus, to extract multiple features we have to run the algorithm multiple times. Sequential extraction allows control over the “quality” of all features. If we were to evolve a number of features all at once we would not know which ones are contributing to classification performance, or which ones are even harming it. Furthermore, by extracting one feature at a time we are oriented towards finding the minimum number of features for the problem at hand. In fact, we can extract one feature at a time and stop as soon as we meet the required level of performance, or we do not observe any meaningful improvement. On the other hand, our greedy approach to feature extraction may not find the global optimum. The alternative to evolve a set of feature-extractors at once, already considered by Eads et al. [15], warrants further investigation.

```

<F>          ::= Extract(T, <lb>, <ub>, <bool>, <fun>)
<fun>         ::= <primitive> | <primitive> | <op>
<primitive>  ::= Above0 | AbsoluteEnergy | AR | Autocorrelation |
                Below0 | CE | FFT | Kurtosis | LinearTrend |
                Max | Mean | MeanChanges | Median | Min |
                Skewness | Std | TRAS
<op>          ::= (<fun>+<fun>) | (<fun>-<fun>) |
                (<fun>×<fun>) | AQ(<fun>, <fun>)
<lb>          ::= 1 | 2 | ... | L
<ub>          ::= <lb> | None
<bool>        ::= True | False

```

Fig. 1 TSC grammar used to guide the search of feature-extractors

Concluding, the fitness function requires each feature to have good classification performance, while being minimally correlated with previous features. While the first component of the fitness function is defined to extract features of good predictive power, the second one weakens the redundancy of extracted features.

3.2 Grammar and primitives

The grammar guiding the search of feature-extractors is shown in Fig. 1. Details of the encoding of candidate solutions, and the mapping process can be found in previous works [43].

The general idea is to compose a feature-extractor $\langle F \rangle$ by selecting its individual components one at the time. Each component, enclosed in the angle brackets ($\langle . \rangle$), requires a choice from a set of predefined equally probable alternatives, separated by pipe symbols ($|$). Finally, the actual extraction is carried out by a wrapper function *Extract* that simply takes the input arguments and arranges them in the form shown in Eq. 1, returning a feature. Note that T (a time series) is not enclosed in the angle brackets. This is because T is an argument that we pass to the function at call time.

As said before, a feature-extractor consists of a function $\langle \text{fun} \rangle$ applied to a sub-sequence of a time series T . To select a specific sub-sequence we need to choose a lower bound $\langle \text{lb} \rangle$ and an upper bound $\langle \text{ub} \rangle$. These indices are chosen from the range $[1, L]$, where L corresponds to the time series length. Since a requirement is that the lower bound is below the upper bound, if this condition is violated $\langle \text{lb} \rangle$ and $\langle \text{ub} \rangle$ are swapped. The grammar allows $\langle \text{ub} \rangle$ to have a *None* value. In this case if $\langle \text{bool} \rangle$ is *True* we select the sub-sequence $[1 : \langle \text{lb} \rangle]$, if *False* the sub-sequence $[\langle \text{lb} \rangle : L]$.

A function $\langle \text{fun} \rangle$ can use one or more elements of the $\langle \text{primitive} \rangle$ set. Multiple primitives are connected through the operators $\{+, -, *, AQ\}$ contained in $\langle \text{op} \rangle$. In $\langle \text{op} \rangle$, *AQ* (analytic quotient) is a function designed to perform division while avoiding division by 0 error [40]. This is achieved by transforming a divisor d into $\sqrt{a + d^2}$. We set $a = 1$ as suggested by the authors of this function.

Fig. 2 Fitness evaluation of a feature-extractor F . First the feature-extractor is applied on each training/validation sample deriving the respective feature-based representation. The classifier in use is a 1NN-ED. The variable x corresponds the number of the feature we are extracting. The algorithm minimises the classification error ($1 - \text{AUROC}$) and the average squared Pearson correlation coefficient with previous features on training data ($\overline{R^2}(F)$)

In $\langle \text{fun} \rangle$, $\langle \text{primitive} \rangle$ is repeated twice while $\langle \text{op} \rangle$ only once. This is done to make the selection of $\langle \text{primitive} \rangle$ twice as likely than $\langle \text{op} \rangle$ thus interrupting the recursion triggered by the $\langle \text{op} \rangle$ rule.

The set of 17 primitives includes functions able to take a sub-sequence as input and output a single scalar. These are among the simplest and most commonly used functions in feature-based time series classification: *mean*, *standard deviation*, *median*, *skewness*, *kurtosis*, *max*, *min*. *Above0* and *Below0* count the number of values above/below 0. *AbsoluteEnergy* returns the sum of the absolute value. *AR* returns the coefficient of an auto-regressive model of order 1. *Autocorrelation* returns the auto-correlation at lag 1. *CE* measures the shape complexity of a time series [6]. *FFT* returns the amplitude associated with the largest coefficient of a fast Fourier transform. *LinearTrend* returns the slope of a linear regression model. *MeanChanges* returns the mean of first order differences. *TRAS* is a measure of non-linearity known as time reversal asymmetric statistic [49].

3.3 Fitness evaluation

The fitness function, shown in Eq. 2, drives the search through the solution space. The fitness score is crucial because it influences the probability of a feature-extractor to be selected, breed with other feature-extractors, and thus pass on to future generations. The fitness evaluation process is illustrated in Fig. 2. Our goal is to sequentially extract features that are all able to achieve a good classification performance individually, while having minimum linear dependence between them.

As an example, suppose we want to extract two features for a certain classification problem. This means we have to run our algorithm $x = 2$ times with each run outputting a single feature-extractor. The first feature-extractor is evolved aiming at the minimisation of the classification error on a validation set. The classification error is defined as $1 - \text{AUROC}$. On the other hand, the second feature-extractor is evolved

aiming for both minimisation of the classification error on a validation set, and minimisation of the average squared Pearson correlation coefficient with previous features $\overline{R^2}(F)$. This coefficient is calculated as follows. We take the feature-based representation of training data according to a candidate feature-extractor. Then, we calculate its squared Pearson correlation with the feature-based representation of the same data according to each of the feature-extractors outputted from previous runs ($\hat{F}_i$). Finally, we calculate the average value. Both the classification error, and the average squared Pearson correlation coefficient have magnitude in the range $[0, 1]$ thus they are commensurate in scale, and have the same impact on the fitness score. This process is repeated three times using 3-folds cross-validation. Thus, the final fitness score of a feature-extractor corresponds to the average score of the 3-folds.

In summary, after all the feature-extractors of an initial population are assigned a fitness score, the crossover and the mutation operators are applied to create a new generation. The evolutionary process continues until a certain number of generations is reached. At that point the feature-extractor with the lowest fitness is considered the final solution of the algorithm.

$$Fitness(F) = \begin{cases} 1 - AUROC, & \text{if } x = 1 \\ 1 - AUROC + \overline{R^2}(F), & \text{otherwise} \end{cases} \quad (2)$$

$$\overline{R^2}(F) = \frac{1}{x-1} \sum_{i=1}^{x-1} R^2(F, \hat{F}_i) \quad (3)$$

$$R^2(F, \hat{F}_i) = \left[\frac{cov(F, \hat{F}_i)}{\sigma(F) \times \sigma(\hat{F}_i)} \right]^2 \quad (4)$$

In Eq. 4 $cov(\cdot)$ and $\sigma(\cdot)$ represent the covariance and the standard deviation functions respectively.

3.4 Classification framework

One-class classification [26], closely related to anomaly detection [8], is concerned with learning a classifier when only the data of a single class is available at training time, and/or at prediction time we might be exposed to classes that are unknown at training time. Usually, the class we learn is referred to as the positive (normal) class while all the samples that fall outside it are allocated to a “generic” negative (anomalous) class. Figure 3 is designed to demonstrate the usefulness of the one-class assumption. In part (1), a binary classifier trained to separate positive/negative samples fails when a new class of negative samples appears at prediction time. The same reasoning can be extended to a multi-class classifier. Conversely, in part (2) a one-class classifier trained on positive samples only is robust to negative classes whether they are known or not at training time.

There are several real world applications where data from the negative class are difficult or impossible to obtain. For instance, in biometric subject

Fig. 3 Binary vs. one-class classification. In part (1) a binary classifier (solid line) trained to separate positive samples from the negative ones (open circle/filled circle) fails when a new class of negative samples filled square appears at prediction time. In part (2) a one-class classifier (dashed line) trained on positive samples open circle only is robust against negative classes either known or not (filled circle/filled square) at training time

authentication problems [36] it is relatively easy to collect some data from the “target subject” class, however it is less straightforward to collect data that are representative of the “not target subject” class. Other related applications are: fraud detection [44], machine fault detection [51], medical diagnosis [48], network intrusion [7], and signature verification [21]. This problem is little investigated in the context of TSC [37].

The one-class classifier we use both in the feature extraction phase, and to evaluate the quality of extracted features is the 1NN classifier equipped with Euclidean distance (1NN-ED). The idea is to use a simple non-parametric classifier to emphasise the quality of extracted features, and make minimal assumptions about class distributions. During the prediction phase, a sample is assigned a score equal to the distance from its nearest training sample. By imposing a threshold on the distance, a sample can be classified as either positive or negative. A common way to set the threshold is to choose a value such that 95% of training data is correctly labelled as below the threshold, hence positive. However, this approach can require some fine tuning according to the specific problem at hand.

In order to avoid this and have a method that can be consistently applied across all data-sets we use classification scores to calculate the AUROC. The AUROC is obtained by computing the underlying area of a curve constructed by plotting the true positive rate against the false positive rate at various threshold settings. Threshold values are calculated as the midpoint between each pair of sorted classification scores.

We believe the AUROC is particularly suitable for one-class TSC experiments since it is insensitive to class imbalance. Furthermore, Gay and Lemaire [20] advocate the use of the AUROC for binary and multi-class TSC experiments too.

4 Benchmark methods

In this section we describe the benchmark methods we compare with. We consider a random search (Sect. 4.1), a distance-based approach (Sect. 4.2), and a feature-based approach with some variants (Sect. 4.3).

4.1 Random search

Random search (RS) is often used to evaluate whether the evolutionary process brings any benefit over the random generation of solutions. Thus, it disentangles the effect of the grammar from the effect of the search process.

Following the rules of our grammar, the RS creates a number of feature-extractors ($\# \text{ generations} \times \text{population size}$, Sect. 5.2) with no duplicates allowed. Feature-extractors are evaluated through our fitness function, and the best one is selected for comparison with our algorithm. As per our algorithm, the RS is repeated multiple times in order to extract multiple features.

4.2 1NN with DTW

The 1NN-DTW classifier (with warping window set through cross-validation [47]) is considered the standard benchmark in the TSC literature [5]. Unlike all the other approaches used in this study 1NN-DTW does not involve feature-vectors but raw time series. This is a distance-based classification approach that compares two time series using DTW. In the one-class classification framework each test sample receives a classification score equal to the DTW distance to its nearest training sample.

For the warping window size we have tested all the values in the range $[1, 0.1 \times L]$ (where L is the time series length) using a 10-fold cross-validation on validation data.

4.3 Analysis of function/sub-sequence selection

As discussed in Sect. 3, our algorithm can dynamically decide both the features to extract and the sub-sequences from which to extract them. A parallel can be drawn with the manual approach where features and sub-sequences can be selected according to fixed user-defined strategies. To investigate the behaviour of our data-driven algorithm we evaluate a Cartesian product of the alternatives mentioned below.

In terms of features to be extracted we consider two sets. (1) The 17 primitives used in our algorithm (Sect. 3.2) and (2) the C22 features discussed in Sect. 2.2. Overall, C22 features are different and more complex with respect to our primitives.

In terms of segmentation strategies we consider three approaches. (1) We extract the features from the whole time series. (2) We segment time series into a number of (approximately) equally sized sub-sequences. Then, features are extracted from each sub-sequence and concatenated into a feature-vector. (3) We segment time series using a change point model. Generally speaking, change point models are used to divide a

time series into distinct homogeneous sub-sequences [4]. We use the bottom-up segmentation algorithm proposed by Keogh et al. [25]. In order to define the number of sub-sequences and their boundaries for a given data-set we implement the following algorithm. Given a training set we select 50% of the samples at random. We segment each sample using the bottom-up algorithm and we record the number of sub-sequences found. Then, we compute the average number of sub-sequences (rounding to the nearest integer). Continuing, we re-run the segmentation algorithm on the same samples imposing as stopping criterion the average number of sub-sequences found before. Finally, we average resulting indices.

5 Experimental design

We provide an overview of the data-sets used in Sect. 5.1. A summary of the main components of the evolutionary framework is provided in Sect. 5.2. We report some implementation details in Sect. 5.3.

5.1 Time series data

We test our algorithm on 30 data-sets. Of these, 29 are selected from the UCR/UEA archive, while one is a proprietary data-set [36]. All the data-sets are partitioned into labelled training and test sets and have previously been examined in several binary and multi-class TSC experiments [5]. All the time series are univariate, contain only real numbers, have a fixed length within a given data-set, and are z-normalised [46].

We adapt the data-sets to the one-class classification framework as follows. Given a data-set, while maintaining the original split between training/test data, each of the classes is considered in turn as the positive class (and so it is used for training) and all the others become the negative class. Classification performance for a given data-set corresponds to the average performance over the classes.

When carrying out a TSC experiment it is good practice to consider all the data-sets of the UCR/UEA archive [10]. This is to avoid spurious results, and allow a thorough comparison between different studies. However, we consider only 29 data-sets selected as follows: from the original body of 85 data-sets of the archive, first we filter out those where there is at least one class with less than 17 training samples. Then we sort the remaining ones according to their total number of samples and select the smallest 29. The rationale is that our algorithm requires a validation set to evaluate the quality of candidate feature-extractors during the evolutionary process. Thus, we want to ensure that there is always a sufficient amount of samples to allow a meaningful training/validation split. Specifically, we use a 2:1 split of training data. Furthermore, we prefer smaller data-sets as our current implementation is very expensive in terms of runtime.

5.2 GE configuration

The evolutionary process is configured with parameters that are common across the field [45]. The algorithm follows a generational approach with a population

of 500 individuals (feature-extractors) for 40 generations. In the GP literature it is most common to use 50 generations [27], however 40 is also often used [9]. In their review of the GP literature, Poli et al. [45] state that the number of generations typically falls in the range [10, 50], where the most productive search is usually performed.

The population is randomly initialised with uniform probability. Genomes are initialised with length 200, while the max length is set to 500. Individuals are recombined with each other using two-point crossover with probability 0.8, and no wraps are allowed. Given two individuals, the crossover operator creates two new individuals by randomly selecting two different points on each genome. Then, the initial and final sections of one genome are merged with the mid section of the other. We increase the selection pressure setting the tournament size to five. After the crossover phase, the resulting population is subject to the mutation operator. We use the int-flip mutation operator that may change with probability 0.01 each gene of a given individual. Elitism is used to preserve the best individual through the generations.

5.3 Implementation details

The experiment is implemented in the Python programming language¹ and relies on the PonyGE2 library [17].

The experiment took about 5000 CPU hours. Our current implementation is expensive especially if compared with the benchmark methods which take just a few hours to run. However, it is not straightforward to draw conclusions about the computational complexity, and scalability of our algorithm. First, for each data-set we run the algorithm 30 times for statistical purposes. However, this would not be required in real-world use. Continuing, runtime depends on several implementation details (e.g. the primitives included in the grammar) which could be reconsidered in future research. Finally, the expensive feature extraction phase is balanced by savings during the prediction phase. In fact, during the prediction phase our algorithm allows the classifier to work with low-dimensional feature-based representations. In contrast, 1NN-DTW always requires the whole time series to work.

6 Results

Classification performance for the data-sets of the UCR/UEA archive is shown in Tables 1, 2. Results related to our subject authentication problem are discussed in Sect. 6.1. For each data-set we extract 10 features and we repeat this process for 30 independent runs. We standardise features to have zero mean and unit variance (with reference to the training set). Algorithm limitations are discussed in Sect. 6.2.

We compare the performance of the features evolved through our algorithm with that of two benchmark sets of features. Of these, PR is the set of primitives used in

¹ https://github.com/spaghetix/FE_GE_TSC.

the grammar, while C22 is the set of features selected by Lubba et al. [34]. These features are used to derive feature-based representations of time series according to three different strategies. (1) Features are extracted from the whole time series (columns 1-PR/1-C22). (2) Time series are broken down into 5 adjacent and equally sized sub-sequences. Then, features are extracted from each sub-sequence and grouped together in a feature-vector (columns 5ES-PR/5ES-C22). (3) Time series are broken down in a number of adjacent sub-sequences using a change point model [25]. Then, features are extracted from each sub-sequence and grouped together in a feature-vector (columns CPM-PR/CPM-C22). We observe that the average number of sub-sequences per data-set found through the change point model is 5 and this is why we use this number in item (2). Concluding, we also compare against a distance-based classification approach (column 1NN-DTW), a strong baseline [5].

Results show that on average our algorithm is superior to any benchmark method. Our algorithm gives better results than 1NN-DTW. It also improves on the C22 features considered the state of the art for feature-based TSC [34].

We consider column 1-3 our best result (in the trade-off between best average performance subject to the lowest dimensionality achievable, which is our aim), while 1NN-DTW is the best benchmark method for comparison. We find that the difference between column 1-3 and 1NN-DTW is statistically significant using a Wilcoxon signed-rank test with a p-value threshold of $\alpha = 0.05$ [53]. We observe that the p-values decrease monotonically as additional features are added, as we would expect. If a Bonferroni-Holm correction for multiple testing [1] is applied, statistical significance is achieved for all tests from 1-2 onward. As 1-3 is the point with the largest reduction in p-value relative to the preceding feature number choice (1-3, p -value=0.007 vs. 1-2, p -value=0.024), we select this as our optimal result in the performance/dimensionality reduction trade-off.

Comparing the average performance of columns 1–3 and 1-PR in Table 1, we can see that our algorithm is able to exploit and combine the functions included in the set of primitives in a way that allows +17% AUROC relative to the use of the functions alone. Furthermore, our algorithm achieves the highest average performance using only 3 features. This gives the time series representation with lowest dimensionality. In fact, PR and C22 require at least 17 and 22 features respectively while 1NN-DTW requires the whole time series. Dimensionality has an impact on storage requirements, and computational complexity at prediction time.

To extract features from a segmented time series has a positive impact on performance. We note a +11/10% AUROC comparing the average performance of column 1-PR with that of columns 5ES-PR/CPM-PR. Conversely, the same does not hold for C22 features. In that case segmentation has little or no impact on performance. This seems to depend on the nature of features. The PR set mainly includes simple statistical moments like mean, standard deviation etc. Although these features are easy to compute, their descriptive power is weak when they are extracted from the whole time series. Moreover, all time series of the UCR/UEA archive are z-normalised so they all have 0 mean and a standard deviation of 1. When a time series is segmented, PR features have more descriptive power.

On the other hand, C22 features are more complex, thus they are able to capture some distinctive characteristics even when they are extracted from the whole time

Fig. 4 Average AUROC for all data-sets of our algorithm (filled circle). (1) AUROC per feature. (2) AUROC per sequence of features. Secondly, RS average AUROC for all data-sets (■)

series. In order to evaluate if 1-C22 features are overfitting we measure their performance on validation data. The average performance across all data-sets on validation data is equal to 66% AUROC. Considering that the performance of 1-C22 features on test data is 70% AUROC we can conclude that these features are not overfitting.

On average, the performance achieved by segmenting a time series in equally sized sub-sequences is nearly the same as that achieved by segmenting through a more complex model, as we can see comparing columns 5ES-PR/5ES-C22 with columns CPM-PR/CPM-C22. It is difficult to explain such a result within this experimental study. Investigating this result in more detail is the focus of further research. However segmentation in equally sized segments can be considered as a strong baseline.

Our method converges to the maximum performance with only three features. In Fig. 4 we show the average AUROC for all data-sets of our algorithm (filled circle). As shown in part (1), the “quality” of features decreases by 4% from the first to the third feature. The performance remains steady from the third to the tenth feature (when rounded to the nearest integer). This demonstrates that individually features maintain a similar level of performance. In part (2), we can see that by combining multiple features the AUROC increases by 2% between the first and the third feature. Then we can observe a weak but positive trend. Features extracted by our algorithm are not explicitly required to improve the overall classification performance when grouped with those that have been extracted before. This objective is pursued indirectly by minimizing the linear correlation of a feature with those extracted before.

Secondly, Fig. 4 shows RS average AUROC for all data-sets (■). In order to speed-up the computation we do not repeat the RS for 30 runs, as done for our algorithm. The variance between RS runs is expected to be low because the sample size is large, and there is no danger of an “unlucky” initial population, as in GE itself. Also, as we have a large enough number of data-sets, the variance across data-sets makes our results statistically significant. Our algorithm is better than RS. However, RS performance rivals 1NN-DTW, a strong baseline. This demonstrates that both our grammar and the search contribute strongly to performance.

Table 1 Average AUROC for the data-sets of the UCR/UEA repository rounded to the nearest integer. (a) Performance of every sequence of features, e.g. 1-10 means all 10 features. (b) Benchmark methods. Columns in bold relate to our best result (1-3) (best performance with lowest dimensionality), and the best benchmark method (1NN-DTW)

Data-set	(a)									
	1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9	1-10
Coffee	91	98	99	99	99	99	99	99	98	98
Computers	65	67	66	65	65	64	64	64	64	64
DistalPhalanxOutlineAgeGroup	81	80	80	80	80	79	79	79	78	78
DistalPhalanxTW	81	81	81	81	82	82	82	81	82	82
ECG200	80	82	83	84	85	86	86	86	87	87
Earthquakes	67	68	68	68	68	68	69	69	69	70
Fish	80	85	87	88	89	90	90	90	90	90
GunPoint	91	87	86	87	88	89	92	92	92	93
Ham	60	65	67	68	69	69	69	70	70	69
Haptics	63	63	63	63	64	64	64	64	64	65
Herring	51	53	53	53	53	54	53	53	53	53
LargeKitchenAppliances	75	78	79	79	78	78	78	77	77	77
Lightning2	64	70	72	74	75	75	74	74	74	74
Meat	95	97	98	97	97	97	97	96	96	96
MiddlePhalanxOutlineAgeGroup	59	58	58	58	58	57	57	57	56	57
MiddlePhalanxTW	71	72	72	72	72	72	72	72	73	73
OSULeaf	78	80	81	81	81	81	80	80	80	80
ProximalPhalanxOutlineAgeGroup	91	90	87	86	85	82	81	80	80	80
ProximalPhalanxTW	91	92	92	93	93	93	93	93	93	93
RefrigerationDevices	64	64	64	64	64	64	64	64	64	65
ScreenType	58	60	61	61	61	61	60	60	60	60
SmallKitchenAppliances	79	81	81	80	80	80	80	80	80	80
SyntheticControl	99	99	99	99	99	99	100	100	100	99
ToeSegmentation1	64	66	68	69	69	70	71	72	72	72
ToeSegmentation2	74	77	78	79	80	79	79	79	79	80
Trace	100	100	100	100	100	100	100	100	100	100
Wine	56	58	58	59	59	58	59	59	60	61
Worms	70	73	73	74	75	75	75	75	76	75
WormsTwoClass	63	65	64	64	63	63	63	64	64	65
Average	74	76	77	77	77	77	77	77	77	77

6.1 Subject authentication

This proprietary data-set, which we refer to as “AccelerometerData”, concerns a subject authentication problem [36]. To collect this data a group of nine subjects wore a watch-like tri-axial accelerometer for about a month in free living conditions. Each time series corresponds to the average acceleration per minute over an entire day.

Table 1 continued

(b)	1-PR	5ES-PR	CPM-PR	1-C22	5ES-C22	CPM-C22	INN-DTW
	61	93	92	95	91	92	95
	51	52	53	56	55	51	55
	59	60	66	70	60	61	66
	73	80	79	83	74	75	82
	56	80	76	72	80	75	82
	50	56	54	56	53	55	64
	61	85	85	84	87	82	89
	59	78	70	80	75	82	96
	51	53	58	55	55	56	54
	51	63	62	60	56	60	61
	50	50	49	51	55	51	53
	51	62	58	71	61	59	71
	56	64	67	66	57	69	77
	74	98	98	85	86	92	97
	48	54	62	58	57	57	53
	72	74	72	73	71	71	71
	65	81	83	83	81	78	86
	65	68	81	79	77	80	67
	79	85	87	85	85	80	88
	50	57	56	63	61	60	65
	50	53	51	57	55	53	51
	52	66	68	73	68	64	59
	96	99	99	99	76	82	99

Table 1 (continued)

(b)						
1-PR	5ES-PR	CPM-PR	1-C22	5ES-C22	CPM-C22	INN-DTW
59	63	64	65	62	61	63
63	62	62	57	62	65	70
96	98	96	96	92	94	98
53	56	61	47	50	54	59
53	74	76	84	80	81	77
49	60	61	70	71	70	67
60	70	71	71	69	69	73

Fig. 5 Average AUROC “AccelerometerData” data-set, subjects 1 to 6. solid line Only subjects 1-6 are used during the feature-extraction phase. dashed line All subjects 1-9 are used during the feature-extraction phase

Features extracted by our algorithm enable good classification performance when tested on data that include subjects not available during the feature extraction phase. In other words, features learned to distinguish subject 1 from subjects 2-6 are also effective to distinguish subject 1 from subjects 7-9.

Figure 5 shows the average classification performance achieved by our algorithm on a subset of subjects (1-6). The solid line (solid line) represents the performance achieved when, during the feature-extraction phase, we use a validation set that includes only subjects 1-6. The dashed line (dashed line) represents the performance achieved when, during the feature-extraction phase, we use a validation set that includes all subjects (1-9). In both cases, extracted features are evaluated on a test set including all subjects (1-9). The two lines follow almost the same trajectory. Excluding the first feature, the average absolute difference between the two lines is equal to 0.3% AUROC.

6.2 Limitations

Our algorithm reveals a number of limitations. The first limitation concerns the runtime required by our current implementation. Although, as discussed in Sect. 5.3, there are several aspects to consider in this regard, overall the algorithm would greatly benefit from any substantial improvement of its runtime.

Continuing, we observe that the algorithm tends to overfit. As shown in Fig. 6 part (1), average AUROC of features 1 to 10 on validation data is approximately 11% higher than on test data. This value drops to 6% if we consider sequences of features, as shown in part (2). The performance per feature seems to follow the same pattern on both validation and test data. Conversely, the performance per group of features exhibits a weak negative trend on validation data, and a weak positive trend on test data showing that together multiple features generalise better. Also, the performance of our algorithm remains better than baselines on test data. While overfitting is a

Fig. 6 (1) Average validation (■) and test (filled circle) AUROC for all data-sets per feature. (2) Average validation and test AUROC for all data-sets per sequence of features

problem it also presents an opportunity to improve our results through regularisation strategies, as we plan to do in future work.

Finally, one possible weakness in our fitness function is that as the number of extracted features increases the correlation penalty is averaged over an increasing number of features as well. While this could make the penalty very small, after a large number of feature-extractors have been created, we observe that a small number of features (e.g. 3 in Table 1) achieves best average performance.

7 Feature-extractors and interpretability

In this section we show how the evolutionary process leads to the selection of specific features and sub-sequences according to the problem at hand. This not only enables good classification performance but also understanding.

The development of interpretable machine learning models is an important research topic [13]. Interpretability is not only essential to machine learning practitioners, e.g. understanding a model allows understanding of its limitations, but also legislation can require model interpretability. For instance, the General Data Protection Regulation act² introduced in the European Union gives to citizens a “right to explanation” with respect to decisions taken by algorithms using their data.

Several researchers have shown that GP can be used to enable interpretability of so-called black-box models (models that are not easy to interpret) [52]. Others have shown that evolutionary algorithms can be as effective as black-box models with the advantage of being more interpretable [29].

Our algorithm allows interpretability of the classification outcome in several ways. In Sects. 7.1, 7.2, we expand the discussion about feature-extractors and interpretability in the context of TSC, however here we provide some more general

² <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679&qid=1603268249651>

Fig. 7 A time series per each of the 6 classes of the “SyntheticControl” data-set

considerations. First, as the search for feature-extractors is driven by the classification performance they allow through a 1NN classifier, it is expected that our approach considers a sample to be normal if it is “close enough” to other samples we already know to be normal (i.e. training data). This type of decision by analogy resembles human thinking [42]. Continuing, extracted features are immediately interpretable as they are expressed as readable Python code. Also, as extracted features are composed of high-level functions e.g. mean, standard deviation, etc. predictions made by our algorithm can be explained in the context of the problem at hand. Finally, as our algorithm gives good classification performance with two or three features, it allows useful visualisation of time series data-sets (Figure 9), and data visualisation is considered key to interpretability [29]. On the other hand, as the number of primitives included in a feature-extractor increases they become more difficult to interpret (Appendix A). As suggested by Lensen et al. [29] in the context of data visualisation, future work may investigate multi-objective fitness functions able to trade-off between classification performance and complexity of the solutions found.

7.1 The features to extract

We use the “SyntheticControl” data-set [3] as a case study to illustrate the ability of our algorithm to discover the right features for the problem. As shown in Fig. 7 this data-set contains 6 different classes of simulated time series. In (1) time series are generated by sampling from a Normal distribution. In (2) time series are generated by sampling from a Normal distribution and adding a cyclic component through a sine function. In (3) and (4) time series are generated by sampling from a Normal distribution and adding/subtracting a trend component. In (5) and (6) time series are generated by sampling from a Normal distribution and adding/subtracting a trend component only after/before a certain time point.

We count the selection frequency of each function/operator used in the first two feature-extractors evolved for each class of the “SyntheticControl” data-set. Without

Fig. 8 Selection frequency of the grammar primitives for the first two features extracted from each of the 6 classes of the “SyntheticControl” data-set. The horizontal lines represent the bar heights we would observe if feature-extractors were generated at random i.e. without regard to fitness

loss of generality we focus on the first two feature-extractors because this eases our discussion and enables visualisation as shown in Fig. 9. Feature-extractors used to generate Fig. 9 are detailed in Appendix A.

In Fig. 8 we can see that the algorithm selects specific groups of functions/operators for each class of the data-set. As reflected by the bar heights, the algorithm finds that class 1 is mainly characterised by the coefficient of an auto-regressive model (AR), and a measure of complexity (CE). The cyclic component of class 2 is captured through the FFT and the auto-correlation functions. Classes 3 and 4 requires similar features like max/min, and the mean. In addition, class 4 is characterised by the linear trend. Finally, classes 5 and 6 require functions like linear trend, max/min, mean, and number of values above/below zero. Finally, the algorithm most often relates multiple functions through the AQ operator for all classes.

Concluding, as mentioned before, features are standardised (with reference to the training set) hence as shown in Fig. 9 our algorithm tends to concentrate the samples of the normal class around the origin even though this is not explicitly required. This aspect shows that our algorithm could also work well using “simple” classifiers, for instance a radial basis function classifier which has the advantage of requiring only a single hyper-parameter, i.e. a distance threshold from the origin.

7.2 The sub-sequences from which to extract

To offer insight into which sub-sequences are most useful we consider the “Gun-Point” data-set [55], and our “AccelerometerData” data-set [36] (Figs. 10–12).

Fig. 9 2D feature-based representation of each class of the “SyntheticControl” data-set. open square Training samples. open circle Normal test samples. filled circle Anomalous test samples. Shading represents distance from training set

Fig. 10 Selection frequency of each point within a time series of the “GunPoint” data-set. solid line Average time series from the data-set. dashed line Frequencies we would observe if feature-extractors were generated at random i.e. without regard to fitness. dotted line Frequencies as per our algorithm

Figure 10 is derived by averaging over the two classes of the “GunPoint” data-set. In this case, our algorithm over-selects two specific sub-sequences. The most frequently selected one corresponds to the sub-sequence underlying the large peak at the begin of the dotted line. The other corresponds to the sub-sequence underlying the small peak towards the end of the dotted line. Ye and Keogh [55] demonstrate that the time series of this data-set can be classified with high performance using “shapelets”. Shapelets are sub-sequences that are maximally representative of a class according to a predefined criterion [55]. In another study related to shapelets Hills et al. [24] show that there are two important sub-sequences in the “GunPoint” problem, the same are identified by

Fig. 11 Sub-sequences of the “GunPoint” data-set related to the time interval [0 : 40]. Part (1) and (2) of the figure show the sub-sequences related to class 1 and 2 respectively

our algorithm. Also, they report that the top five most important shapelets extracted by their algorithm are related to the end of the time series. However, our algorithm over-selects the sub-sequences at the beginning of the time series. This means that our algorithm is able to target sub-sequences that are relevant for the problem, but overall it works in a way that is different from shapelets. Thus, the representation generated by our algorithm could be a good addition to a representation based on shapelet-transform [24].

In Fig. 11, we show all the sub-sequences of the “GunPoint” data-set related to the time interval [0 : 40]. This is the most selected time interval according to the analysis related to Fig. 10. Sub-sequences are divided per class: part (1) and (2) show the sub-sequences related to class 1 and 2 respectively. The figure provides insights into the sub-sequences that are important for classification according to our algorithm. Visual assessment of the sub-sequences allows us to claim that most time series could be correctly classified if we were able to catch shape dissimilarities between the two classes in the considered time interval. As shown in Table 1, our algorithm achieves approximately 90% AUROC on this data-set. Thus, not only our algorithm is able to detect important sub-sequences, but also it is able to extract useful features.

Finally, Figure 12 considers all the nine classes of the “AccelerometerData” data-set. We can see that each person is characterised by her/his activity during a specific part of the day. In the time axis 0 corresponds to 0am and 1440 to 11.59pm. Most subjects are characterised by their activity between 6.30–8.30am, presumably the time they wake up and go to work. Two subjects are characterised by their activity during 0–6.30am (1,6). Also, for three subjects (6,7,9) the activity between 9–11pm seems to be important for their classification.

8 Conclusions

We propose a data-driven evolutionary algorithm for feature extraction from time series. The goal is to find a low-dimensional feature-based representation that enables good one-class classification performance with minimum human intervention.

Fig. 12 Selection frequency of each point within a time series of the “AccelerometerData” data-set. The number in brackets in the top left corner of each plot corresponds to a different class of the data-set

The search for suitable solutions is guided by our grammar specifically defined for time series. In fact, our algorithm can select both the features to extract, and the sub-sequences from which to extract them. Both aspects can be key to TSC problems, as revealed by several related studies.

Evolved features are composed of high-level functions e.g. mean, standard deviation, etc. By choosing the number of features to be extracted a high-dimensional time series can be reduced to a feature-vector of arbitrary dimensionality. This not only lowers computational complexity at prediction time, but also

allows the visualisation of time series data-sets. Furthermore, the analysis of the functions/sub-sequences that are most frequently selected during the evolutionary process enables understanding about the problem at hand. Finally, our algorithm enables better classification performance than considered benchmark methods.

Future work could investigate further the generalization capabilities of our algorithm in a one-class classification scenario where new classes appear at prediction time. Another avenue of research could investigate other primitives to be included in the grammar. A remaining issue of our algorithm is the tendency to overfit. This suggests that our results could be improved, an interesting topic for future work.

Appendix A examples of feature-extractors

Below are listed the feature-extractors used to produce Fig. 9. In the text T represents a time series and L its length.

• Feature 1

- Class (1) $\text{Min}(T[1 : 59]) + \text{CE}(T[1 : 59])$
- Class (2) $\text{AQ}(\text{FFT}(T[1 : 57]), \text{Kurtosis}(T[1 : 57]))$
- Class (3) $\text{AQ}(\text{Mean}(T[9 : L]) * \text{STD}(T[9 : L]) - \text{Max}(T[9 : L]), \text{STD}(T[9 : L]))$
- Class (4) $\text{Min}(T[4 : 34])$
- Class (5) $\text{Mean}(T[1 : 50]) * \text{AQ}(\text{Min}(T[1 : 50]), \text{Mean}(T[1 : 50]))$
- Class (6) $\text{AQ}(\text{Below0}(T[36 : L]), \text{LinearTrend}(T[36 : L]) * \text{Min}(T[36 : L]) * \text{Median}(T[36 : L]) - \text{TRAS}(T[36 : L]))$

• Feature 2

- Class (1) $\text{Above0}(T[1 : 53]) - \text{AQ}(\text{AR}(T[1 : 53]), \text{Max}(T[1 : 53]))$
- Class (2) $\text{AQ}(\text{FFT}(T[1 : 34]), \text{AQ}(\text{AR}(T[1 : 34]) + \text{FFT}(T[1 : 34])), \text{AbsSum}(T[1 : 34]) * \text{AQ}(\text{FFT}(T[1 : 34]), \text{AQ}(\text{AR}(T[1 : 34]) + \text{LinearTrend}(T[1 : 34]) * \text{LinearTrend}(T[1 : 34]), \text{TRAS}(T[1 : 34]))$
- Class (3) $\text{AQ}(\text{Max}(T[1 : 34]), \text{AQ}(\text{Min}(T[1 : 34]), \text{AQ}(\text{Mean}(T[1 : 34]) * \text{Skewness}(T[1 : 34]) + \text{Max}(T[1 : 34]) * \text{AR}(T[1 : 34]), \text{Median}(T[1 : 34])))$
- Class (4) $\text{Above0}(T[22 : L]) - \text{AQ}(\text{AQ}(\text{AQ}(\text{Above0}(T[22 : L]), \text{Autocorrelation}(T[22 : L])), \text{AbsSum}(T[22 : L]) - \text{LinearTrend}(T[22 : L]) + \text{Min}(T[22 : L]), \text{Mean}(T[22 : L])) - \text{AQ}(\text{Max}(T[22 : L]), \text{Min}(T[22 : L]) + \text{LinearTrend}(T[22 : L]))$
- Class (5) $\text{Max}(T[7 : L]) * \text{Mean}(T[7 : L])$
- Class (6) $\text{Below0}(T[7 : 22]) - \text{AQ}(\text{AR}(T[7 : 22]) * \text{Max}(T[7 : 22]), \text{AQ}(\text{AQ}(\text{AQ}(\text{Min}(T[7 : 22]) * \text{Autocorrelation}(T[7 : 22]), \text{Below0}(T[7 : 22])), \text{AQ}(\text{MeanChanges}(T[7 : 22]), \text{CE}(T[7 : 22])), \text{CE}(T[7 : 22]) - \text{AbsSum}(T[7 : 22]) + \text{Mean}(T[7 : 22]))$

Acknowledgements This work is funded by ICON plc. We would like to thank the reviewers for their thoughtful comments and efforts towards improving our manuscript.

Funding Open Access funding provided by the IReL Consortium.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. H. Abdi, Holm's sequential bonferroni procedure. *Encycl. Res. Des.* **1**(8), 1–8 (2010)
2. A. Agapitos, R. Loughran, M. Nicolau, S. Lucas, M. O'Neill, A. Brabazon, A survey of statistical machine learning elements in genetic programming. *IEEE Trans. Evol. Comput.* **23**(6), 1029–1048 (2019)
3. R.J. Alcock, Y. Manolopoulos, et al., Time-series similarity queries employing a feature-based approach. In: 7th Hellenic conference on informatics, pp 27–29 (1999)
4. S. Aminikhanghahi, D.J. Cook, A survey of methods for time series change point detection. *Knowl. Inf. Syst.* **51**(2), 339–367 (2017)
5. A. Bagnall, J. Lines, A. Bostrom, J. Large, E. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.* **31**(3), 606–660 (2017)
6. G.E. Batista, E.J. Keogh, O.M. Tataw, V.M. De Souza, Cid: an efficient complexity-invariant distance for time series. *Data Min. Knowl. Discov.* **28**(3), 634–669 (2014)
7. V.L. Cao, M. Nicolau, J. McDermott, Learning neural representations for network anomaly detection. *IEEE Trans. Cybern.* **99**, 1–14 (2018)
8. V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey. *ACM Comput. Surv. (CSUR)* **41**(3), 15 (2009)
9. W. Cui, A. Brabazon, M. O'Neill, Evolving efficient limit order strategy using grammatical evolution. In: *IEEE Congress on Evolutionary Computation*, IEEE, pp 1–6 (2010)
10. H.A. Dau, A. Bagnall, K. Kamgar, C.C.M. Yeh, Y. Zhu, S. Gharghabi, C.A. Ratanamahatana, E. Keogh, The ucr time series archive. *IEEE/CAA J. Automatica Sinica* **6**(6), 1293–1305 (2019)
11. A. Dempster, F. Petitjean, G.I. Webb, Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Min. Knowl. Discov.* **34**(5), 1454–1495 (2020)
12. H. Deng, G. Runger, E. Tuv, M. Vladimir, A time series forest for classification and feature extraction. *Inf. Sci.* **239**, 142–153 (2013)
13. F. Doshi-Velez, B. Kim (2017) Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:170208608*
14. R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification* (Wiley, NewJersey, 2012)
15. D. Eads, K. Glocer, S. Perkins, J. Theiler, Grammar-guided feature extraction for time series classification. In: *Proceedings of the 9th Annual Conference on Neural Information Processing Systems*, Citeseer, pp 1–8 (2005)
16. H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.A. Muller, Deep learning for time series classification: a review. *Data Min. Knowl. Discov.* **33**(4), 917–963 (2019)
17. M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg, M. O'Neill, Ponyge2: Grammatical evolution in python. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, pp 1194–1201 (2017)
18. E. Fernández-Blanco, D. Rivero, M. Gestal, J. Dorado, Classification of signals by means of genetic programming. *Soft Comput.* **17**(10), 1929–1937 (2013)

19. M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning. In: *Advances in neural information processing systems*, pp 2962–2970 (2015)
20. D. Gay, V. Lemaire, Should we Reload Time Series Classification Performance Evaluation? (a position paper). [ArXiv:1903.03300](https://arxiv.org/abs/1903.03300) [stat.ML] (2019)
21. Y. Guerbai, Y. Chibani, B. Hadjadj, The effective use of the one-class svm classifier for handwritten signature verification based on writer-independent parameters. *Pattern Recognit.* **48**(1), 103–113 (2015)
22. I. Guyon, A. Elisseeff, *An Introduction to Feature Extraction In Feature Extraction* (Springer, Berlin, 2006)
23. D.Y. Harvey, M.D. Todd, Automated feature design for numeric sequence classification by genetic programming. *IEEE Trans. Evol. Comput.* **19**(4), 474–489 (2015)
24. J. Hills, J. Lines, E. Baranauskas, J. Mapp, A. Bagnall, Classification of time series by shapelet transformation. *Data Min. Knowl. Discov.* **28**(4), 851–881 (2014)
25. E. Keogh, S. Chu, D. Hart, M. Pazzani, Segmenting time series: A survey and novel approach. In: *Data mining in time series databases*, World Scientific, pp 1–21 (2004)
26. S.S. Khan, M.G. Madden, One-class classification: taxonomy of study and review of techniques. *Knowl. Eng. Rev.* **29**(3), 345–374 (2014)
27. J.R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol 1. MIT press (1992)
28. Y. LeCun, Y. Bengio, G. Hinton, Deep Learn. *Nat.* **521**(7553), 436 (2015)
29. A. Lensen, B. Xue, M. Zhang, Genetic programming for evolving a front of interpretable models for data visualization. *IEEE Transactions on Cybernetics* (2020)
30. D. Li, J. Lin, T.F.D. Bissyande, J. Klein, Y. Le Traon, Extracting statistical graph features for accurate and efficient time series classification. In: *21st International Conference on Extending Database Technology* (2018)
31. L. Liao, Discovering prognostic features using genetic programming in remaining useful life prediction. *IEEE Trans. Ind. Electron.* **61**(5), 2464–2472 (2014)
32. J. Lin, Y. Li, Finding structural similarity in time series data using bag-of-patterns representation. In: *International conference on scientific and statistical database management*, Springer, pp 461–477 (2009)
33. J. Lin, E. Keogh, S. Lonardi, B. Chiu, 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*, ACM, pp 2–11 (2003)
34. C.H. Lubba, S.S. Sethi, P. Knaute, S.R. Schultz, B.D. Fulcher, N.S. Jones, catch22: canonical time-series characteristics. *Data Min. Knowl. Discov.* **33**(6), 1821–1852 (2019)
35. S. Mauceri, L. Smith, J. Sweeney, J. McDermott, Subject recognition using wrist-worn triaxial accelerometer data. In: *International Workshop on Machine Learning, Optimization, and Big Data*, Springer, pp 574–585 (2017)
36. S. Mauceri, J. Sweeney, J. McDermott, One-class subject authentication using feature extraction by grammatical evolution on accelerometer data. In: *Heuristics for Optimization and Learning*, Springer, pp 393–407 (2018)
37. S. Mauceri, J. Sweeney, J. McDermott, Dissimilarity-based representations for one-class classification on time series. *Pattern Recognit.* **100**, 107122 (2020)
38. D.J. Montana, Strongly typed genetic programming. *Evol. comput.* **3**(2), 199–230 (1995)
39. A. Nanopoulos, R. Alcock, Y. Manolopoulos, Feature-based classification of time-series data. *Int. J. Comput. Res.* **10**(3), 49–61 (2001)
40. J. Ni, R.H. Drieberg, P.I. Rockett, The use of an analytic quotient operator in genetic programming. *IEEE Trans. Evol. Comput.* **17**(1), 146–152 (2012)
41. M. Nicolau, A. Agapitos, Understanding grammatical evolution: Grammar design. In: *Handbook of Grammatical Evolution*, Springer, pp 23–53 (2018)
42. R.M. Nosofsky, Attention, similarity, and the identification-categorization relationship. *J. Exp. Psychol: General* **115**(1), 39 (1986)
43. M. O'Neill, C. Ryan, Grammatical evolution. In: *Grammatical evolution*, Springer, pp 33–47 (2003)
44. C. Phua, V. Lee, K. Smith, R. Gayler, A comprehensive survey of data mining-based fraud detection research. *arXiv preprint [arXiv:10096119](https://arxiv.org/abs/10096119)* (2010)
45. R. Poli, W.B. Langdon, N.F. McPhee, J.R. Koza, *A field guide to genetic programming*. Lulu. com (2008)

46. T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh, Addressing big data time series: mining trillions of time series subsequences under dynamic time warping. *ACM Trans. Knowl. Discov. Data (TKDD)* **7**(3), 10 (2013)
47. C.A. Ratanamahatana, E. Keogh, Making time-series classification more accurate using learned constraints. In: *Proceedings of the 2004 SIAM international conference on data mining*, SIAM, pp 11–22 (2004)
48. T. Schlegl, P. Seeböck, SM. Waldstein, U. Schmidt-Erfurth, G. Langs, Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In: *International Conference on Information Processing in Medical Imaging*, Springer, pp 146–157 (2017)
49. T. Schreiber, A. Schmitz, Discrimination power of measures for nonlinearity in a time series. *Phys. Rev. E* **55**(5), 5443 (1997)
50. A. Shifaz, C. Pelletier, F. Petitjean, G.I. Webb, Ts. Chief, Webb GI, Chief Ts, A scalable and accurate forest algorithm for time series classification. *Data Min. Knowl. Discov.* **34**(3), 742–745 (2020)
51. H.J. Shin, D.H. Eom, S.S. Kim, One-class support vector machines-an application in machine fault detection and classification. *Comput Ind. Eng.* **48**(2), 395–408 (2005)
52. M. Virgolin, T. Alderliesten, P.A. Bosman, On explaining machine learning models by evolving crucial and compact features. *Swarm Evol. Comput.* **53**, 100640 (2020)
53. F. Wilcoxon, Individual comparisons by ranking methods. *Biometrics Bulletin* **1**(6), 80–83 (1945)
54. S.J. Wilson, Data representation for time series data mining: time domain approaches. *Wiley Interdiscip. Rev: Comput. Stat.* **9**(1), e1392 (2017)
55. L. Ye, E. Keogh, Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data Min. Knowl. Discov.* **22**(1–2), 149–182 (2011)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.