Data Mining and Knowledge Discovery (2022) 36:1015–1061
https://doi.org/10.1007/s10618-022-00822-7

PETSC: pattern-based embedding for time series classification

Len Feremans[1] · Boris Cule[2,3] · Bart Goethals[1,4]

Received: 30 September 2020 / Accepted: 22 January 2022 / Published online: 24 March 2022

Abstract

Efficient and interpretable classification of time series is an essential data mining task with many real-world applications. Recently several dictionary- and shapelet-based time series classification methods have been proposed that employ contiguous subsequences of fixed length. We extend pattern mining to efficiently enumerate long variable-length sequential patterns with gaps. Additionally, we discover patterns at multiple resolutions thereby combining cohesive sequential patterns that vary in length, duration and resolution. For time series classification we construct an embedding based on sequential pattern occurrences and learn a linear model. The discovered patterns form the basis for interpretable insight into each class of time series. The pattern-based embedding for time series classification (PETSC) supports both univariate and multivariate time series datasets of varying length subject to noise or missing data. We experimentally validate that MR-PETSC performs significantly better than baseline interpretable methods such as DTW, BOP and SAX-VSM on univariate and multivariate time series. On univariate time series, our method performs comparably to many recent methods, including BOSS, cBOSS, S-BOSS, ProximityForest and ResNET, and is only narrowly outperformed by state-of-the-art methods such as HIVE-COTE, ROCKET, TS-CHIEF and InceptionTime. Moreover, on multivariate

Responsible editor: Panagiotis Papapetrou

✉ Len Feremans
len.feremans@uantwerpen.be

Boris Cule
b.cule@tilburguniversity.edu

Bart Goethals
bart.goethals@uantwerpen.be

1 Department of Computer Science, University of Antwerp, Antwerp, Belgium

2 Department of Cognitive Science and Artificial Intelligence, Tilburg University, Tilburg, The Netherlands

3 Department of Accountancy and Finance, University of Antwerp, Antwerp, Belgium

4 Faculty of Information Technology, Monash University, Melbourne, Australia

datasets PETSC performs comparably to the current state-of-the-art such as HIVE-COTE, ROCKET, CIF and ResNET, none of which are interpretable. PETSC scales to large datasets and the total time for training and making predictions on all 85 ‘bake off’ datasets in the UCR archive is under 3 h making it one of the fastest methods available. PETSC is particularly useful as it learns a linear model where each feature represents a sequential pattern in the time domain, which supports human oversight to ensure predictions are trustworthy and fair which is essential in financial, medical or bioinformatics applications.

Keywords Time series classification · Sequential pattern mining · SAX · Interpretable classification

1 Introduction

Time series classification is an important real-world problem as currently technology enables the collection of huge volumes of temporal data from users and devices. Existing time series classification methods often make *stringent assumptions* on the format of the time series and require that time series start at the same time, are univariate, are of equal length or only contain continuous values. However, in the real word, there is more variation in time series. For instance, IoT devices collect sensor values and operating system events at the same time and exhibit missing data and irregular sampling. Making accurate predictions on this large variety of datasets remains a hard problem despite recent advances. This variety of time series is illustrated in Fig. 1.

In the active research area of dictionary- and shapelet-based time series classification, patterns are *fixed-length continuous and contiguous subsequences* referred to as shapelets, motifs or words. Shapelet-based methods use subsequences of the raw numeric time series with elastic distance measures (Ye and Keogh 2011; Hills et al. 2014; Lucas et al. 2019). Dictionary-based methods convert continuous time series data to a symbolic sequence using Symbolic Aggregate Approximation (SAX) after which fixed-length contiguous subsequences are extracted for classification (Lin et al. 2003, 2012; Senin and Malinchik 2013). Alternatively, time series are first transformed using Symbolic Fourier Approximation (SFA) instead of SAX resulting in the best performing dictionary-based time series classification method, Bag of SFA Symbols (BOSS) (Schäfer 2015). Because of its good performance on the ‘bake off’ datasets of the UCR/UEA Archive (Bagnall et al. 2017), many extensions to BOSS have been proposed recently, such as WEASEL, cBOSS, S-BOSS and TDE (Schäfer and Leser 2017; Middlehurst et al. 2019; Large et al. 2019; Middlehurst et al. 2020b). However, since these methods combine the predictions of hundreds of individual BOSS classifiers and rely on the hard to interpret Symbolic Fourier Approximation representation, the resulting predictions are not interpretable. Finally, nearest neighbours with Dynamic Time Warping (DTW) have proven to be a remarkably strong baseline on both univariate and multivariate datasets (Shokoohi-Yekta et al. 2017; Bagnall et al. 2017).

Many more methods have been proposed. Heterogeneous ensemble methods, such as COTE (Bagnall et al. 2015), HIVE-COTE (Lines et al. 2018) and TS-CHIEF (Shi-

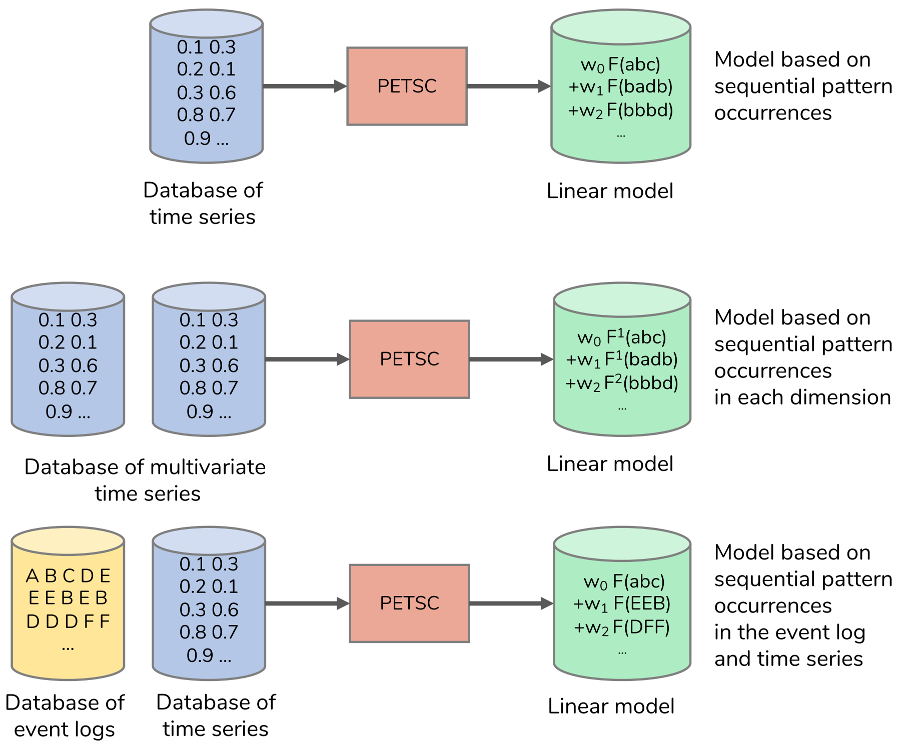
0.1 0.3
0.2 0.1
0.3 0.6
0.8 0.7
0.9 ...
Database of time series
PETSC
w_0 F(abc) $+w_1$ F(badb) $+w_2$ F(bbbd) ...
Linear model
Model based on sequential pattern occurrences
Database of multivariate time series
w_0 F^1(abc) $+w_1$ F^1(badb) $+w_2$ F^2(bbbd) ...
Model based on sequential pattern occurrences in each dimension
A B C D E
E E B E B
D D D F F
...
Database of event logs
w_0 F(abc) $+w_1$ F(EEB) $+w_2$ F(DFF) ...
Model based on sequential pattern occurrences in the event log and time series

Fig. 1 Illustration of different use cases where PETSC is applicable. PETSC can discriminate between different classes for univariate time series, multivariate time series and mixed-type time series consisting of both event logs and continuous time series

faz et al. 2020) combine strong shapelet-, dictionary- and distance-based baseline methods. High accuracy is also achieved by recent methods based on deep learning architectures, such as ResNet (Wang et al. 2017) and InceptionTime (Fawaz et al. 2019, 2020). All of the above mentioned methods mostly focus on increasing the accuracy on the UCR/UEA time series benchmark, but they require significant resources to train and make predictions and often do *not scale* to very large datasets and take days or weeks to complete on all 85 'bake off' datasets of the UCR archive (Dau et al. 2018). More recently, ROCKET (Dempster et al. 2020) achieves both high accuracy and efficiency, by completing on all 85 'bake off' datasets under 2 h. Another interesting new development is MR-SEQL (Le Nguyen et al. 2019) which is also accurate and, excluding the SFA representation, offers an *interpretable* model.

For the related task of sequence (or event log) classification, many algorithms have been proposed based on *frequent pattern mining* (Fan et al. 2008; Cheng et al. 2008; Zhou et al. 2016). Given the wealth of algorithms for efficiently enumerating sequential patterns (Aggarwal and Han 2014; Zaki and Meira 2014) and event log classification, one might wonder why it is scarcely mentioned within continuous time series classification. We argue this is due to the following causes. First, time series are highly autocorrelated, causing many repetitive symbols afters discretisation. That is, autocorrelated time series often contain long repetitive subsequences, e.g. *aaaaaaaaaa* or *bbbbbbbbb*, that occur frequently and as a consequence, all shorter subsequences of such sequences are also frequent causing a large output of less discriminative sequential patterns. Second, in contrast to irregular event logs, the cohesion (in terms of time) of matching pattern symbols is more important for regularly sampled continuous time series. For instance, sequential pattern *ab* matches with window *axxxxxxxxxb* and we show that, without temporal constraints on matching, this foils accurate classification. Thirdly, since frequency is anti-monotonic, shorter patterns are more frequent than longer ones, causing a long list of patterns of length 1, 2 or 3, while in dictionary-based methods discriminative subsequences are much longer, i.e., a length of 10 is more usual.

The traditional design choice of dictionary- and shapelet-based time series classifiers is to limit patterns to a fixed length, duration and resolution. This makes sense given that the space of sequential patterns grows exponentially with the length of the pattern and the size of the alphabet. Our method, however, adopt a different approach. First, we consider patterns of varying length, which is important, for instance, if pattern *aaa* is discriminative for class A and pattern *bbbccc* for class B, limiting the candidate patterns to fixed length inhibits optimally separating instances of both classes. Second, unlike related work into varying length patterns for time series classification (Le Nguyen et al. 2017; Raza and Kramer 2020), we do not focus only on contiguous subsequences, but on sequential patterns where the *duration of the sequential pattern occurrences varies* as we allow for gaps between consecutive symbols when matching the pattern to a discretised subsequence, e.g., *aaabbb* matches with sequence *aaaxbbb*. Third, we consider different resolutions when creating the representations, which is determined by the choice of the sliding window parameter. If this is set to a relatively high value, this smooths each segment and captures longer patterns, e.g., trends or seasonal patterns. If the window is set to a relatively low value, we smooth less and discover shorter and more diverse local shapes. We argue that choosing is losing when committing to a fixed length, duration or resolution of patterns.

We propose PETSC, a new dictionary-based method that discovers candidate sequential patterns of varying length and duration. The major steps of PETSC are illustrated in Fig. 2. Our method first transforms the time series dataset using a sliding window and SAX. For discovery of sequential patterns we extend principled techniques from frequent, discriminative and top-k pattern mining literature to reduce the number of candidates dramatically (Fan et al. 2008; Fradkin and Mörchen 2015). Since the frequency of the prefix is higher than or equal to the frequency of any sequential pattern starting with this prefix, i.e., frequency is anti-monotonic, we use this to efficiently prune candidates. That is, we enumerate candidate sequential patterns efficiently based on *prefix-projected pattern growth* (Pei et al. 2004) with respect to a novel constraint on *relative duration*. We propose two mining algorithms that discover the top-k most *frequent* and top-k most *discriminative* sequential patterns *directly*.

After discovering k sequential patterns we create an embedding matrix where each value represents the frequency of a sequential pattern for a time series. For multivariate time series we repeat this process and concatenate embedding vectors from each dimension independently. Additionally, we create an ensemble method that discovers

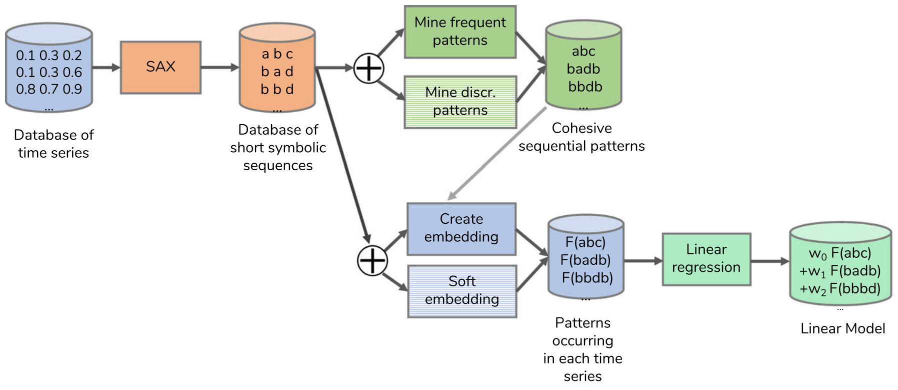

Fig. 2 Overview of the different steps in PETSC. First we transform each continuous time series to a symbolic sequence using SAX. Next we discover a set of varying-length frequent sequential patterns. Next we create an embedding based on matching pattern occurrences and finally train a linear model that assigns a different weight to each discovered pattern. An extension of PETSC mines discriminative patterns and another extension is based on near-matching patterns

the top-*k* sequential patterns in a limited number of SAX representations at multiple resolutions. Finally, for classification we train a *linear model* using an elastic net trained on the pattern-based embedding.

In summary, we make the following key contributions:

– We classify time series using a white-box linear model based on an embedding of varying length frequent sequential patterns. We show that this model is *easy to interpret* thereby supporting applications such as medical diagnosis or financial applications, where explainable predictions and a transparent model are essential for asserting fairness or building trust.
– We *scale to much larger datasets* and are on par with ROCKET (Dempster et al. 2020) concerning runtime performance by taking only 2.7 h to process all 85 'bake off' datasets (Bagnall et al. 2017).
  Moreover, our model is small and supports *real-time* applications, such as edge computation to handle the deluge of sensing data from IoT devices.
– We *outperform dictionary-based methods* such as BOP and SAX-VSM even with default hyperparameters and are competitive with BOSS. We also *outperform distance-based* approaches on both univariate and multivariate datasets (Shokoohi-Yekta et al. 2017; Bagnall et al. 2018). While out method performs slightly worse in terms of accuracy than the state-of-the-art method ROCKET on average, we still outperform ROCKET on 32 out of 109 univariate datasets and 10 out of 26 multivariate datasets.
– We explore and extend pattern mining research techniques to discover *long, cohesive and discriminative sequential patterns* in time series directly. Additionally, we present a novel temporal constraint on relative duration and an alternative to exact pattern matching to deal with discretisation errors.
– We *do not require stringent assumptions* on the format of the time series and naturally handle univariate, multivariate and mixed-type multivariate time series

subject to missing data or irregular sampling. This supports applications for classification of IoT devices that log both sensor values and discrete operating events.

The remainder of this paper is organised as follows. We present an overview of the related work in Sect. 2. In Sect. 3, we introduce the necessary preliminaries. In Sect. 4, we provide a detailed description of our method for sequential pattern mining, creating the pattern-based embedding and classification using a linear model. In Sect. 5 we present several optimisations that result in three additional variants of our algorithm. In Sect. 6, we present an experimental evaluation of our method and compare with state-of-the-art methods before presenting conclusions in Sect. 7.

2 Related work

In this section we present the relevant related work, starting off with various approaches to time series classification, before discussing the fields of frequent pattern mining, event log classification and explainability, which our work builds on.

2.1 Shapelet-based time series classification

Shapelet-based time series classification uses continuous subsequences to separate time series with different labels (Bagnall et al. 2017). Often Euclidean distance or an elastic distance measure, such as dynamic time warping, is used in combination with the nearest neighbour classifier (Ye and Keogh 2011) or other (binary and multiclass) algorithms such as decision trees (Hills et al. 2014). A disadvantage of shapelets is that the nearest neighbour search makes it relatively slow on larger datasets which has resulted in substantial work on optimisation (Rakthanmanon et al. 2012; Petitjean et al. 2014; Yeh et al. 2016).

2.2 Dictionary-based time series classification

For dictionary-based time series classification, Lin et al. (2012) proposed bag-of-patterns (BOP) where each time series is converted into segments using a sliding-window and then converted to discrete subsequences, or words, using SAX. Senin and Malinchik (2013) extended BOP in SAX-VSM thereby computing TF-IDF weights for each word and label. In both approaches, time series are assigned a label based on the nearest neighbour after transformation to a dictionary of words.

In Fig. 3 we show an illustrative example on a dataset provided by Lin et al. (2012) where we ran PETSC and highlight 2 sequential patterns with high support in each class. To make the visualisation clearer we removed overlapping pattern occurrences. In Fig. 4 we show another illustrative example and compare BOP with PETSC. Each time series of length 1000 is segmented with a sliding-window of length 50 and discretised using SAX into words of length 12 consisting of 4 symbols. Using BOP we create a dictionary that contains all unique subsequences of length 12. Using PETSC we mine the top-1000 most frequent sequential patterns with a length between 6 and 12 and a relative duration of 1.2, meaning that at most 2 gaps are allowed. We observe that

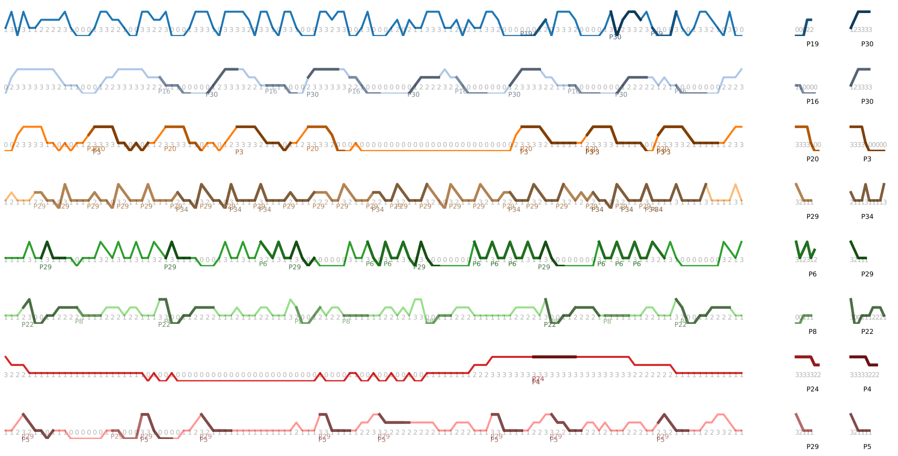

Fig. 3 Illustrative example where we show 8 time series of different classes and highlight occurrences for the top-2 sequential patterns with the highest support in each class. First, we segment the time series using a sliding-window of length $\Delta t = 120$ and discretise each segment using SAX with 4 symbols and a word length of 15. Next, we mine the top-200 frequent sequential patterns with a length between 5 and 15 and allow for 1 gap

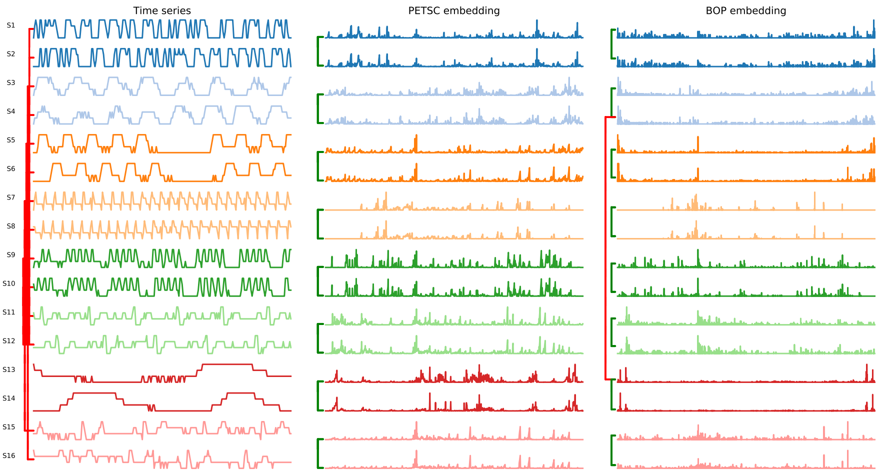

Fig. 4 Illustrative example where we show 16 times series of 8 different classes. In the second column, we show the corresponding PETSC embedding that represents the frequency of thousands of varying length sequential patterns. In the third column, we show the corresponding BOP embedding that represents the frequency of thousands of fixed-length sequences. In addition we connect each time series to its nearest neighbour in each column—if the line is green, the nearest neighbour is of the same class, and if the line is red, the nearest neighbour is of another class (Color figure online)

if we compute the nearest neighbour using Euclidean distance on the raw time series this is not an instance with the same label. The nearest neighbour in the embedding space of both BOP and PETSC is much more reliable for this dataset, however, BOP identifies S_4 as the nearest neighbour of S_{13}, even though the two time series belong to different classes.

Schäfer (2015, 2016) proposes BOSS which first transforms time series using Symbolic Fourier Approximation (SFA) and then creates a dictionary of patterns, in this case (unordered) sets of SFA symbols. BOSS offers high accuracy and has been extended to be more efficient. BOSS achieves an accuracy on the UCR Benchmark (Dau et al. 2018) that is within a critical distance of deep-learning based approaches, such as ResNet (Wang et al. 2017) and ensemble methods such as Proximity Forests (Lucas et al. 2019).

2.3 Time series classification with variable length patterns

Most related to our method is SEQL (Le Nguyen et al. 2017) which mines varying length sequences, after transformation using SAX. SEQL tackles the large search space by employing greedy search thereby mining the most discriminative subsequences directly. The authors further extend their method and propose an ensemble MR-SEQL that combines multiple resolutions of SAX and SFA representations which results in an accuracy competitive with recent deep learning and heterogeneous ensemble techniques (Le Nguyen et al. 2019). Unlike MR-SEQL, PETSC does not depend on SFA and uses non-contiguous patterns instead of subsequences by allowing pattern occurrences with varying duration. Another difference is that MR-SEQL combines $\sqrt{|S|}$ representations and linear models, while we propose a single linear model based on $\log(|S|)$ representations. Another related method based on varying length SAX patterns is MiSTiCl (Raza and Kramer 2020). MiSTiCl tackles the large subspace of varying length patterns using frequent (contiguous) string mining and creates an embedding based on different SAX representations similar to SEQL. Key differences with PETSC are that we mine non-contiguous sequential patterns and use a linear model for classification where MiSTiCl uses a random forest for classification making it uninterpretable. Finally, using pattern mining we naturally handle a larger variety of time series data sources, such as univariate time series with missing values, irregular sampling and multivariate time series (possibly of mixed type). We experimentally compare with MR-SEQL and MiSTiCl in Sect. 6.

2.4 Frequent pattern mining

Within the pattern mining community, there has always been a focus on interpretable association rules, based on itemsets and sequential patterns (Agrawal et al. 1994), also referred to as parallel and serial episodes for sequential data (Mannila et al. 1997). Early algorithms focused on how to generate many frequent patterns of varying length efficiently (Pei et al. 2004; Chen et al. 2007). Others have investigated how to efficiently incorporate various constraints (Pei et al. 2007). Others have adapted methods to produce less redundant patterns, leading to various condensed representations, i.e., closed, maximal or non-derivable (conjunctive) patterns. Finally, there have been attempts to directly report patterns ranked on non-frequency based interestingness measures such as leverage (Petitjean et al. 2016) and cohesion (Cule et al. 2019), or based on minimal description length to produce a set of patterns that compresses the sequence best (Lam et al. 2014). A known problem with frequent patterns is that they are often not interesting, which is even worse within the context of time series. We circumvent these issues, by first computing sliding windows and using SAX, which includes a local z-normalisation of each window, which is important in limiting the diversity of patterns. Additionally, we demonstrate that setting constraints on minimal length and relative duration is very important. We propose a new sequential pattern mining algorithm in Sect. 4 that is related to prefix-projected pattern growth such as PrefixSpan (Pei et al. 2004), mining with temporal constraints such as PG (Pei et al. 2007), algorithms that mine the top-*k* most frequent sequential patterns directly such as TSK (Fournier-Viger et al. 2013) and algorithms that mine the top-*k* most cohesive sequential patterns directly such as QCSP (Feremans et al. 2018).

2.5 Event log classification

Frequent and discriminative pattern mining has been proposed for discrete sequence or event log classification (Fan et al. 2008; Cheng et al. 2008; Zhou et al. 2016). In Sect. 5, we propose a new sequential pattern algorithm that discovers the most discriminative patterns directly where we rank patterns on *contrast*, defined as the difference in relative support in time series with and without a certain label. We also investigate the use of *sequential covering* to remove redundant patterns. In sequential covering we mine the best discriminative pattern and then remove all windows covered by this pattern. We repeat this process iteratively until all time series windows are covered by at least one discriminative pattern. In other works, authors have also proposed to create a representation based on occurrences of sequential patterns before classification, such as in BIDE-DC (Fradkin and Mörchen 2015) and SQN2VEC (Nguyen et al. 2018). Similar to our proposed method, they consider redundancy between patterns, gap constraints and a one-versus-all strategy for dealing with multiple classes. A key difference, however, is that only discrete sequential datasets, typically with a small alphabet, are considered, thereby ignoring challenges specific to time series classification.

2.6 Explainability

In practice, end-users of algorithms prefer the output to be explainable. Deep learning algorithms are often seen as black box methods, and practitioners in many fields prefer to use more interpretable methods, even if they produce lower accuracy. Recently, methods have been developed within the deep learning community for visualising attribution, i.e., quantifying the contribution of each time series interval (Hsieh et al. 2021). Other general algorithms have been constructed to explain individual decisions post hoc for black-box models (Molnar 2020). Most notably, algorithms such as LIME (Ribeiro et al. 2016) and SHAP (Lundberg and Lee 2017) attempt to provide explanations for each individual classification on a point-by-point basis. This enables *instance-based decision support*, i.e., highlight which parts of a test instance contribute the most to the assigned class label. However, this is still a weaker form of interpretability, as it only allows to explain a decision for an individual instance.

An *intrinsically interpretable model* (Molnar 2020) consists of a linear model or decision tree and a set of interpretable features (or patterns). With this type of white-box model, we look at the model internals, such as the weights of the linear model or decision tree branches, to trust decisions. That is, using an interpretable model it is possible for human experts to inspect both the model and the features in order to trust decisions for *any* instance. Note that it is always possible to train a global surrogate model post hoc thereby mimicking the behaviour of a black-box model at the cost of slightly worse accuracy, however training an interpretable model directly is far more straightforward and certainly preferable if high accuracy is achieved.

Another important aspect of a good explanation is *selectivity* or complexity of the model (Molnar 2020). That is, having thousands of features in a linear model hinders interpretation, and having fewer features or sparse linear models is better for providing decision support towards human end-users interpreting the model. In general selecting the top-k features with the highest information gain (Peng et al. 2005) can be employed to reduce the number of features. We remark that pattern redundancy has been extensively studied by the pattern mining community (Han et al. 2011; Aggarwal and Han 2014) and we can filter closed or maximal sequential patterns or patterns that are non-discriminatory, i.e., having low contrast using sequential covering (see Sect. 5.2).

Within the context of time series classification, there also exist techniques that are reducing overlapping patterns, such as numerosity reduction (Lin et al. 2012). Another technique is shapelet clustering, where the authors propose to reduce the number of shapelets using a clustering of (equal-length) shapelets to increase model interpretation at the cost of a slight drop in accuracy (Hills et al. 2014). However, we remark that most existing methods for time series classification are difficult to interpret and the employed models are unsuitable for human interpretation. That is, by adopting an SFA representation, random forests, deep learning or an ensemble thereof, existing methods are extremely hard to interpret. A notable exception is MR-SEQL (Le Nguyen et al. 2019) since its model consists of a linear model of features. However, MR-SEQL depends on SFA and since SFA symbols represent a complex-valued function of frequency this hinders interpretability.

In this work we discuss both instance-based decision support (by visualising attribution) and learn an intrinsically interpretable model directly. Additionaly we present a use-case where we reduce the number of patterns to further improve explanations in Sect. 6.3.

3 Preliminaries

In this section, we introduce the necessary concepts and notations, before formally defining our problem setting.

3.1 Time series data

A *time series* is defined as a time-ordered sequence of measurements $S = (\langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle)$, where each measurement has a timestamp t_k and $\forall\, i, j \in \{1..n\} : i < j \Rightarrow t_i \leq t_j$. For continuous time series $x_k \in \mathbb{R}$. For discrete time series $x_k \in \Omega$ where Ω denotes the finite domain of event types. We remark that for sequential pattern mining we do not require a single event at each timestamp (i.e. $t_i = t_j$) nor that the sample rate is regular (i.e. $\forall i : t_{i+1} - t_i$ is not constant). We use $|S|$ to denote the length of the time series. If time series are of different length we define $|S|$ to be the minimal length of any time series.

A *univariate time series dataset* consists of multiple labelled time series, i.e. $\mathcal{S} = \{S_i, y_i\}_{i=1}^{m}$. Each time series S_i may be of different length, however, in the univariate case we assume all measurements are of the same type. A class, or label, $y_i \in \mathcal{Y}$ where $\mathcal{Y}$ denotes the finite domain of classes, which is either binary or multi-class if it consists of more than two classes.

In a *multivariate time series dataset*, each labelled instance is composed of several time series, or dimensions: $\mathcal{S}^{1,\ldots,d} = \{S_i^1, \ldots S_i^d, y_i\}_{i=1}^{m}$. That is, there are m labelled instances and each instance consists of d time series, $S_i^1, \ldots, S_i^d$. Like in the univariate case, we assume that time series in each dimension d are of the same type, however, in different dimensions the types may vary. The sampling and length of each time series can also vary. In the remainder of this paper, we use the notation $\mathcal{S}^d$ (or $\mathcal{S}$) to refer to all time series in a single dimension of a multivariate time series dataset. We remark that PETSC reduces a multivariate time series dataset to d univariate time series datasets for extracting a sequential pattern-based embedding.

3.2 Segmentation

A *time series window* $S_k[t_a, t_b]$ is a contiguous subsequence of time series S_k and contains all measurements $\{\langle x_i, t_i\rangle \in S_k \mid t_a \leq t_i \leq t_b\}$ in the univariate case. In this work, we use *fixed-length sliding windows*. This means choosing a fixed time interval Δt (e.g. 5 s, 10 min or 1 h) and creating windows starting at $t_1, \ldots t_n$. For a time series of length $|S_i|$ there are $|S_i| - \Delta t + 1$ windows assuming an increment of 1. We denote the resulting set of windows as $\mathcal{S}^s$. For multivariate datasets, we apply a sliding-window over each dimension independently. It is known that the accuracy of time series classification is heavily dependent on the value of Δt, since segments must be long enough to contain the distinctive features, but also short enough to be informative[1] (Senin and Malinchik 2013; Le Nguyen et al. 2019). After transforming the time series data using a sliding-window, the dataset consists of many small segments and we ignore the order between segments. This enables us to mine local, phase-independent, patterns and is in contrast to interval-based time series classification methods. We remark that sequential pattern mining specific to intervals, i.e., by discovering and counting the

[1] We remark that alternatives to sliding-window based frequency for sequential patterns have been investigated that do not require choosing Δt (Cule et al. 2019). However, this is not compatible with window-based normalisation performed by SAX.

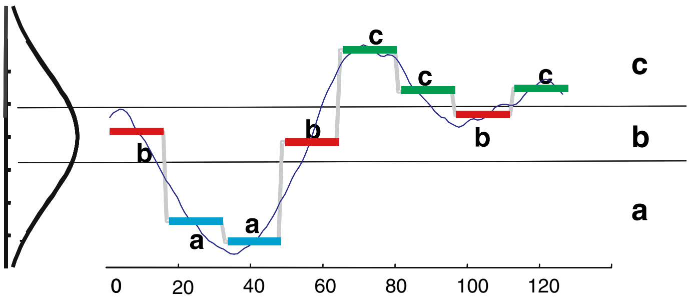

Fig. 5 Illustration of the SAX transform where the normalised continuous time series segment with length $|S| = 128$ is transformed using PAA with a word size of $w = 8$ and discretised into $\alpha = 3$ equal-density regions into the discrete sequence *baabccbc* (Lin et al. 2012)

frequency of sequential patterns in each interval separately, is not explored to date, but seems a logical direction for future work.

3.3 Symbolic aggregate approximation

After a sliding-window transformation, we reduce and discretise each continuous segment, or window, $S^s[t_a, t_b]$ using SAX (Lin et al. 2003). SAX has two parameters: the word size w and the alphabet size α. First, we z-normalise each window resulting in values distributed with a mean of 0.0 and a standard deviation of 1.0. If the standard deviation is below a certain threshold, typically set to 0.01, we do not z-normalise to prevent that noise is amplified. Next, the length of each window is reduced by computing the Piecewise Aggregate Approximation (PAA), e.g., with a word size, or PAA window, of 8 we divide the normalised segment into 8 equally sized subsegments and store the mean value for each subsegment (Keogh et al. 2001). Finally, we convert each mean value to a letter (or digit) using a lookup table into α equal-density regions assuming the Gaussian distribution. Figure 5 from Lin et al. (2012) illustrates the transformation of a single segment. We remark that parameters w and α have a large influence on the final classification accuracy. Lin et al. (2012) propose to set α to 3 or 4 for most datasets and w from 6 to 8. Since we mine variable length patterns, it makes sense to increase w, e.g., on some datasets setting w to 20 (and α as high as 12) resulted in best accuracy as we will discuss in Sect. 6.

3.4 Sequential pattern mining

A sequential pattern X is a sequence of one or more items, denoted as $X = (s_1, \ldots, s_l)$, where $s_k \in \Omega$ and Ω is the finite domain of symbols. For continuous time series $|\Omega| = \alpha$ after transformation using SAX. A sequential pattern may contain repeating items and we allow gaps (or unmatched symbols) between items. A sequential pattern $X = (s_1, \ldots, s_l)$ covers a segment $S^s_k[t_a, t_b]$ if:

$$X \prec S_k^s[t_a, t_b] \Leftrightarrow \exists t_1, \ldots, t_l \in [t_a, t_b] : t_1 < \cdots < t_l :$$
$$\forall j \in \{1, \ldots, l\} : \langle i_j, t_j \rangle \in S \wedge s_j = i_j.$$

The cover and support of a sequential pattern X in a segmented time series dataset $\mathcal{S}^s$ is defined as:

$$cover(X, \mathcal{S}^s) = \{S_k^s[t_a, t_b] \mid S_k^s[t_a, t_b] \in \mathcal{S}^s \wedge X \prec S_k^s[t_a, t_b]\},$$
$$support(X, \mathcal{S}^s) = |cover(X, \mathcal{S}^s)|.$$

A sequential pattern is *frequent* if its support is higher than a user-defined threshold on minimal support, or *min_sup*. Note that frequency based on a sliding-window is somewhat harder to interpret in the original time series since a sequential pattern often covers multiple overlapping windows.[2] For example, given discretised time series $S_k = xxxabcxxx$ we create the following windows using $\Delta t = 5$: $xxxab$, $xxabc$, $xabcx$, $abcxx$ and $bcxxx$. Subsequently, the support of sequential pattern $X = (a, b, c)$ in this instance is 3. A sequential pattern X is not *closed* if there exists a sequential pattern Z, such that X is a subsequence of Z and $support(X, \mathcal{S}) = support(Z, \mathcal{S})$, e.g., if (a, b) and (a, b, c) have the same support closed pattern mining would only keep (a, b, c). After segmentation and discretisation we can use any sequential pattern mining algorithm to efficiently mine all *frequent* sequential patterns in time series (Zaki and Meira 2014; Aggarwal and Han 2014). However, as discussed previously, existing pattern mining algorithms have issues concerning continuous time series.

3.5 Problem setting

Given a training dataset of time series $\mathcal{S}_{train}$ (or $\mathcal{S}_{train}^{1\ldots d}$) that is either univariate, multivariate or mixed-type the task is to predict the correct class $\hat{y}_q \in \mathcal{Y}$ for each test time series instance in $\mathcal{S}_{test} = \{S_q\}_{q=1}^t$ or $\mathcal{S}_{test}^{1\ldots d} = \{S_q^1, \ldots, S_q^d\}_{q=1}^t$ for multivariate time series. We evaluate PETSC on accuracy, execution time and interpretability and compare with state-of-the-art time series classification methods on univariate and multivariate benchmark from the UCR/UEA time series classification archive (Dau et al. 2018).

4 Pattern-based embedding for time series classification

In this section, we describe a method that constructs a pattern-based embedding for time series classification (PETSC). PETSC has four major steps. First, the time series is segmented and transformed using SAX. Second, a dictionary containing the top-k most frequent sequential patterns with respect to temporal constraints is mined from $\mathcal{S}$ (or in each dimension $\mathcal{S}^d$ for multivariate time series). Third, the sequential pattern

[2] This observation has led to adaptations for numerosity reduction in time series classification (Lin et al. 2012) or non-overlapping minimal windows in frequent pattern (or episode) mining (Zhu et al. 2010; Cule et al. 2019).

dictionary is used to compute the frequency of each pattern thereby creating a pattern-based embedding. Fourth, we train a linear model with L1 and L2 regularisation to separate each class based on the pattern-based embedding. In Sect. 5, we present three possible variants of our base PETSC classifier which we experimentally evaluate in Sect. 6.

The main steps of our method are shown in Algorithm 1. For brevity, we show only the version that takes as input a univariate time series. For multivariate (or mixed-type) time series, we repeat lines 1-9 for each time series and then merge the resulting embeddings, before constructing a classifier in line 10.

4.1 Preprocessing

Preprocessing is the first step in PETSC_TRAIN shown in Algorithm 1 (line 1-4). First, we segment each time series in $\mathcal{S}$ using a fixed-length sliding window of length Δt. After this transformation, we create equal length windows (in time). Note that these windows do not necessarily contain the same number of items if sampling is irregular. Next, each continuous window $S^s[t_a, t_b]$ is z-normalised because time series have widely different amplitudes. Frequent sequential patterns are often limited in length, i.e., traditionally a length of 5 is quite high, therefore we reduce the length of each window using the Piecewise Aggregate Approximation (PAA), such that a sequential pattern covers a large part of a window. For example, a continuous time series of length 1000 is transformed to $1000 - 120 + 1$ sliding windows of length 120 ($\Delta t = 2min$).

Algorithm 1: PETSC_TRAIN($\mathcal{S}$, Δt, w, α, k, min_len, $rdur$) Pattern-based embedding for time series classification

```
Input: Univariate time series $\mathcal{S} = \{S_i, y_i\}_{i=1}^m$, sliding window interval ($\Delta t$), SAX word size ($w$)
       and alphabet size ($\alpha$), number of patterns ($k$), minimum length pattern ($min\_len$) and temporal
       constraint ($rdur$)
Result: Set of patterns $\mathcal{P}$, coefficients linear model $\phi$
   // Preprocessing
1  $\mathcal{S}^s \leftarrow$ SEGMENT($\mathcal{S}$, $\Delta t$)
2  if $S$ is continuous then
3      foreach $S^s[t_a, t_b] \in \mathcal{S}^s$ do
4          $\mathcal{S}^s[t_a, t_b] \leftarrow$ DISCRETISE(PAA(Z- NORM($S^s[t_a, t_b]$), $w$), $\alpha$)
   // Mine sequential patterns
5  $\mathcal{P} \leftarrow$ MINE_FREQ_SP($\mathcal{S}^s$, $k$, $min\_len$, $rdur$)
   // Create embedding
6  $\mathcal{F} \leftarrow \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}^{|\mathcal{S}| \times |\mathcal{P}|}$
7  for $i \leftarrow 1$ to $|\mathcal{S}|$ do
8      for $j \leftarrow 1$ to $|\mathcal{P}|$ do
9          $\mathcal{F}_{i,j} = support_{rdur}(X_j, S_i^s)$
   // Train linear model
10 $\phi \leftarrow$ ELASTIC_NET($\mathcal{F}$, $\{y_i\}_{i=1}^m$)
11 return $\langle \mathcal{P}, \phi \rangle$
```

Each continuous window of length 120 is then reduced to a length of 16 using PAA ($w = 16$) and consequently, the mean amplitude is discretised to 4 symbols ($\alpha = 4$).

We remark that for larger datasets we could increase the increment of the sliding window to a larger value than 1, i.e., with a value of 2 (or 10) we decrease the number of windows $|\mathcal{S}^s| = |\mathcal{S}| \cdot (|S| - \Delta t + 1)$ by a factor of 2 (or 10) thereby conserving memory and decreasing execution time. Increasing the increment would mean skipping some overlapping segments, which would clearly result in a considerable reduction in runtimes, but it would potentially come at a cost of some important patterns not being discovered. In preliminary experiments we found that increasing the increment has a large influence on runtimes and a limited negative effect on accuracy, but since our algorithm is already very fast, we decided to set the default value of the window increment to 1, and not to skip any segments.

4.2 Discover top-*k* frequent cohesive sequential patterns

To create a dictionary for time series classification we propose a method based on *frequent pattern mining* to efficiently discover sequential patterns. Many different algorithms have been proposed to discover sequential patterns. However, continuous time series have their own challenges such as autocorrelation, the relatively small alphabet after the SAX transform and the goal to discover long and cohesive patterns similar to subsequences found by related dictionary-based methods. We propose the following goals for discovering patterns in time series:

- Mine sequential patterns above a specified minimum length.
- Enumerate the top-k most frequent sequential patterns *directly* instead of fine-tuning the threshold on minimal support (*min_sup* parameter) which is resource consuming.
- Only consider *cohesive* occurrences of sequential patterns which is known to be important in literature on episode mining (Zimmermann 2014; Cule et al. 2019).

Relative duration
Traditionally, temporal constraints bound the number of gaps or duration of a pattern regardless of the length of the pattern (Pei et al. 2007). We propose a constraint on the duration of a pattern occurrence relative to the length of the pattern (*rdur*). The cover and support of a sequential pattern X in a segmented time series dataset $\mathcal{S}^s$ with respect to *rdur* are defined as:

$$\begin{aligned} &cover_{rdur}(X, \mathcal{S}^s) = \{S_k^s[t_a, t_b] \mid S_k^s[t_a, t_b] \in \mathcal{S}^s \\ &\quad \wedge \exists t_{a'}, t_{b'} \in [t_a, t_b] : t_{b'} - t_{a'} \leq rdur \cdot |X| \wedge X \prec S_k^s[t_{a'}, t_{b'}]\}, \\ &support_{rdur}(X, \mathcal{S}^s) = |cover_{rdur}(X, \mathcal{S}^s)|. \end{aligned}$$

That is, X must cover a subsequence of a segment that is at most $rdur \cdot |X|$ long. For instance, if *rdur* is 1.2 this implies that we allow a maximal duration of $\lfloor 4 \cdot 1.2 \rfloor = 4$ for patterns of length 4, 6 for patterns of length 5 and 24 for patterns of length 20.
Algorithm

Algorithm 2: MINE_FREQ_SP($\mathcal{S}^s, k, min\_len, rdur$) Discover top-$k$ frequent cohesive sequential patterns in $\mathcal{S}^s$

```
Input: Segmented univariate time series $\mathcal{S}^s$, number of patterns ($k$), minimum length pattern
    ($min\_len$) and maximum duration relative to pattern length ($rdur$)
Result: Top-$k$ sequential patterns ranked on support with respect to rdur
1  $Q \leftarrow [\langle \emptyset, \mathcal{S}^s, \Omega \rangle]$
2  $\mathcal{P} \leftarrow$ MAKE_HEAP($k$)
3  $min\_sup \leftarrow 0$
4  while $Q \neq \emptyset$ do
5      $\langle X, P_X, Y \rangle \leftarrow$ POP($Q$)
6      if $Y = \emptyset$ then
7          if $support_{rdur}(X, \mathcal{S}^s) > min\_sup$ and $|X| \geq min\_len$ then
8              PUSH($\mathcal{P}$, $\langle X, support_{rdur}(X, \mathcal{S}^s) \rangle$)
9              if $|\mathcal{P}| > k$ then
10                 POP($\mathcal{P}$)
11                 $min\_sup \leftarrow$ MIN_HEAP($\mathcal{P}$)
12     else
13         if $support_{rdur}(X, \mathcal{S}^s) < min\_sup$ then
14             continue
15         $s_{l+1} \leftarrow$ FIRST($Y$);
16         PUSH($Q$, $\langle X, P_X, Y \setminus \{s_{l+1}\} \rangle$)
17         $Z \leftarrow (X, s_{l+1})$
18         $P_Z \leftarrow$ PROJECT($\mathcal{S}, Z, P_X, rdur$)
19         $Y_Z \leftarrow$ PROJ_CANDIDATES($\mathcal{S}, Z, P_Z$)
20         PUSH($Q$, $\langle Z, P_Z, Y_Z \rangle$)
21 return $\mathcal{P}$
```

The algorithm MINE_FREQ_SP, shown in Algorithm 2, discovers longer, frequent and cohesive sequential patterns. We start by creating an empty priority queue that stores all sequential pattern candidates sorted on support (line 1). During recursion we maintain candidate sequential patterns $X = (s_1, \ldots, s_l)$ and construct sequential patterns Z of length $l+1$ where X is the prefix and item s_{l+1} is from the set Y of (unvisited) symbols, or candidate items. For incrementally computing projections of supersequences Z we also maintain the projection of X on $\mathcal{S}^s$ denoted as P_X. Next, we create a heap of patterns (line 2). Like the queue, we also use a priority queue data structure, but this time sorted on descending support. Initially, *min_sup* is 0 (line 3). However, after k candidates are discovered, the current minimal support in the heap is used to prune future candidates (line 13) instead of a fixed value for *min_sup*. In the main loop, we first retrieve the most likely candidate sequential pattern, i.e., the one with the highest support, from the priority queue (line 5). Next, we check if the set of candidate items is empty (line 6) and we have a leaf in our search process. In this case, we add the current candidate pattern to the pattern heap if its support is higher than *min_sup* and the length of the pattern is greater than or equal to *min_len* (line 7-9). Since we want to return at most k sequential patterns, the pattern with the lowest support is removed (line 10) and *min_sup* is updated (line 11). Remark that we do not assume a maximum length, but since we depend on SAX, the length of any sequential pattern is limited by w. If the set of candidate items is not empty, we first check if the current candidate prefix is frequent (line 13). Next, we create two nodes in the search tree: the current

sequential pattern X without the first item s_{l+1} in the set of candidate items Y (line 16) and the sequential pattern $Z = (s_1, \ldots, s_l, s_{l+1})$ (Line 20).
Pseudo-projections
Crucial to performance of MINE_FREQ_SP is how to compute the support with respect to *rdur* and limit the number of candidates. We compute support (and candidates) using *incremental pseudo-projections*. A projection of X on $\mathcal{S}^s$ is denoted by P_X and consists of all windows $S^s[t_i, t_j]$ covered by X with respect to *rdur*. The support for sequential pattern X is then computed as $|P_X|$. In the projection data structure we maintain the index of each window and offsets $\langle t_1, t_l \rangle$ to the first (s_1) and last symbol (s_l) of $X = (s_1, \ldots, s_l)$ in the covered windows. The projection P_Z of supersequences $Z = (s_1, \ldots, s_l, s_{l+1})$ are computed incrementally by checking if s_{l+1} occurs in the suffix of each window $S^s[t_i, t_j]$, i.e., starting after the last offset of s_l for each window in P_X. For example, assume $S_k = xxxabcxxx$ and we create the following windows using $\Delta t = 5$: $w_1 = xxxab$, $w_2 = xxabc$, $w_3 = xabcx$, $w_4 = abcxx$ and $w_5 = bcxxx$. After projection on $X = (a, b)$ we keep references to windows and store $P_X = \{w_1 : \langle 4, 5 \rangle, w_2 : \langle 3, 4 \rangle, w_3 : \langle 2, 3 \rangle, w_4 : \langle 1, 2 \rangle\}$ in memory. For supersequence $Z = (a, b, c)$ we construct $P_Z = \{w_2 : \langle 3, 5 \rangle, w_3 : \langle 2, 4 \rangle, w_4 : \langle 1, 3 \rangle\}$.
Restricting projections using relative duration
Since we only consider cohesive occurrences, we use the relative duration *rdur* to further restrict the projection by translating this constraint to a constraint on the *absolute duration* and a *limit on the number of gaps*.

Firstly, there is limit on the maximum duration:

$$max\_duration = \lfloor max\_len \cdot rdur \rfloor.$$

As a consequence, we limit the search for s_{l+1} in each window in P_X with offsets $\langle t_1, t_l \rangle$ and range $[t_i, t_j]$ to indexes $t_k \in [t_l + 1, t_1 + max\_duration]$ which might be smaller than $[t_l + 1, t_j]$.

Secondly, the maximal number of remaining gaps is given by:

$$\begin{aligned} max\_gap\_total &= max\_duration - max\_len, \\ current\_gaps &= (t_l - t_1 + 1) - |X|, \\ max\_rem\_gap &= max\_gap\_total - current\_gaps. \end{aligned}$$

As a consequence we limit the search in each suffix to indexes $t_k \in [t_l + 1, t_l + max\_rem\_gap + 1]$. For example, assume we have $X = (a, b)$, $Z = (a, b, c)$, $max\_len = 5$, $rdur = 1.2$ and $w_1 = abxxxxxxxc$ and $w_2 = axbxc$. Z covers both windows, however for w_1 the duration would be 10, but *max_duration* of 6 is preventing a match with respect to *rdur*. For w_2 the duration would be 5, however, $max\_gap\_total = 1$ and $current\_gaps = (3 - 1 + 1) - 2 = 1$, so $max\_rem\_gap = 0$ and since c is at index 5 and not 4 this would also not be a match.

4.3 Constructing the pattern-based embedding

Having obtained a dictionary $\mathcal{P}$, i.e., a set of varying length patterns for the segmented time series $\mathcal{S}^s$, PETSC maps $\mathcal{S}^s$ to a pattern-based embedding. In the embedding space each time series S_i is represented by a vector of size $|\mathcal{P}|$, where each value is associated with the support of pattern $X_1, \ldots, X_k$ in S_i. This is shown in Algorithm 1 (line 6-9) where we first initialise the embedding matrix $\mathcal{F}$ to $|\mathcal{S}| \times |\mathcal{P}|$ dimensions. Next, we compute the support of each pattern in each time series, computed as the number of sliding windows covered by the pattern with respect to the constraint imposed by *rdur*. Note that by using the support as feature, we regard both the (cohesive) occurrence of a pattern and its frequency as important components for discriminating classes. For instance, a pattern might occur on average 10 times in each time series of class A, while occurring only once in class B.
Naïve algorithm
We implemented a naïve algorithm that computes the support of every pattern X in $\mathcal{P}$ in each time series S_i (in both the training and test database). The time complexity of constructing the pattern-based embedding of $\mathcal{S}$ is $\mathcal{O}(|\mathcal{P}| \cdot |\mathcal{S}|)$. Since the number of patterns $|\mathcal{P}|$ is controlled by the hyper-parameter k the total runtime is still reasonable since k is limited to a few thousand in practice for high accuracy. However, we remark that this results in a runtime that is an order of magnitude worse than the discovery of the top-k most frequent sequential patterns and might impede real-time applications. For example, in preliminary experiments on UCR datasets we found that computing the embedding takes minutes while mining the patterns only takes seconds.
Efficient algorithm
We can improve the naïve algorithm for computing the embedding based on the observation that the embedding matrix produced by PETSC is *sparse* and the density is often less than 1% on many datasets, similar to related dictionary-based methods such as BOP or SAX-VSM. For efficiently computing the sparse embedding we use prefix-projected pattern growth to identify all segments matching the pattern X without the cost of matching the full pattern $X = (s_1, \ldots, s_l)$. First, we compute the projection P_{X_p} for each prefix $X_p = (s_1, \ldots, s_p)$ for p from 1 to l incrementally. The number of segments matching in each successive projection, P_{X_p}, grows smaller (since support is anti-monotonic), thereby discarding segments that do not match prefix $(s_1, \ldots, s_p)$ before considering them for $(s_1, \ldots, s_p, s_{p+1})$. By matching the pattern X with respect to *rdur* using incremental prefix-projections and given that in practice most patterns cover only a few time series with respect to *rdur*, most windows are discarded early on without the computational cost of matching the complete pattern. In preliminary experiments on UCR datasets we found that constructing the embedding with this algorithm took seconds where the naïve algorithm would take minutes.

4.4 Constructing the time series classifier

The final step of PETSC (line 10 of Algorithm 1) is to construct the model for classifying time series. In principle we could use any classification algorithm such as a random forest or k-nearest neighbours, but given that we are interested in interpretable results and because of its simplicity we adopt a simple linear model. For each label $y \in \mathcal{Y}$ we learn a linear model that separates time series S_i based on its pattern-based embedding F_i (after normalisation):

$$\hat{y} = w_0 + w_1 \cdot F_{i,1} + \cdots + w_k \cdot F_{i,k}.$$

The model coefficients $W = (w_0, \ldots, w_k)$ are learned by minimising the following loss function:

$$\mathcal{L}(\lambda_1, \lambda_2, W) = |\mathbf{y} - \mathbf{F}^T W|^2 + \lambda_2 \sum_{i=1}^{k} w_i^2 + \lambda_1 \sum_{i=1}^{k} |w_i|.$$

The loss function combines regression with L1 and L2 regularisation referred to as an elastic net (Zou and Hastie 2005). L1, or LASSO, regularisation is particularly useful since frequent pattern-based features that are not discriminative have a high likelihood of a zero coefficient making the model more condensed and improving overall interpretability. Meanwhile, combining it with L2, or ridge, regularisation overcomes limitations by adding a quadratic part to the penalty which makes the loss function convex and a unique minimum is guaranteed. We remark that for multi-class time series datasets we use a one-vs-all approach.

5 Optimisations

In this section, we discuss possible optimisations of the PETSC algorithm presented above. Concretely, we develop three additional variants of the algorithm. First, we propose MR-PETSC, an ensemble method that combines patterns discovered in different resolutions. Second, we propose PETSC-DISC, a method that mines the top-k most discriminative patterns directly. Third, we present PETSC-SOFT that relaxes the requirement of an exact match to deal with discretisation errors. Next we discuss the use of common transformations to the original time series before applying dictionary-based methods. We conclude this section with an analysis of the time and space complexity of PETSC and variants.

5.1 Ensemble of multiresolution frequent sequential patterns

A disadvantage of PETSC is that, like related dictionary-based methods, its accuracy is highly dependent on the hyper-parameter Δt. Additionally, PETSC ignores discriminative patterns at different resolutions. Both of these issues are addressed by the MR-PETSC variant, shown in Algorithm 3 and illustrated in Fig. 6. In this approach, we still mine patterns with the same constraints and the same parameters for the SAX representation, but we do this recursively starting with a single segment equal to the length of the time series (or the minimum length in varying length time series) (line 3) and recursively divide the sliding window by two until the segment length is lower than the SAX word size (line 4 and 12). At each resolution, we join the embedding

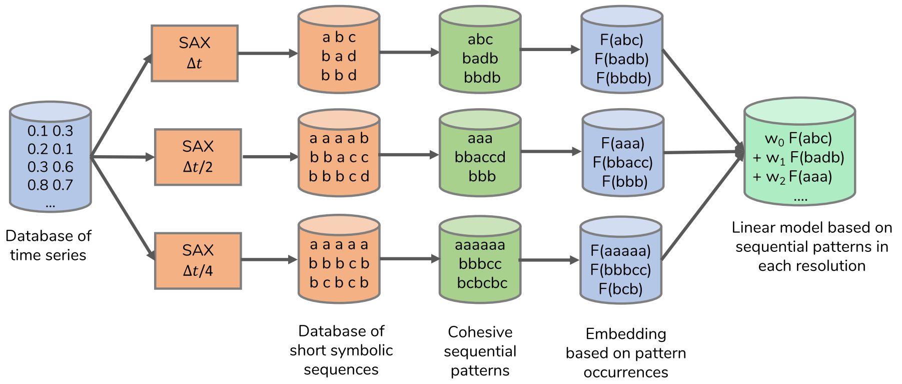

Fig. 6 Illustration of MR-PETSC where we first create different symbolic representations by varying the fixed-length sliding window (Δt) and then combine sequential patterns discovered in each representation in a single linear model

Algorithm 3: MR_PETSC_TRAIN($\mathcal{S}$, w, α, k, min_len, $rdur$) Multiresolution pattern-based embedding for time series classification

```
Input: Univariate time series $\mathcal{S} = \{S_i, y_i\}_{i=1}^{m}$, other parameters similar to PETSC
Result: Set of patterns $\mathcal{P}$, coefficients linear model $\phi$
1  $\mathcal{P} \leftarrow \emptyset$
2  $\mathcal{F} \leftarrow \emptyset$
3  $\Delta t \leftarrow min\{|S_i| \mid S_i \in \mathcal{S}\}$
4  while $\Delta t > w$ do
       // PETSC embedding at $\Delta t$
5      $\mathcal{S}^s \leftarrow$ SEGMENT($\mathcal{S}$, $\Delta t$)
6      foreach $S^s[t_a, t_b] \in \mathcal{S}^s$ do
7          $S^s[t_a, t_b] \leftarrow$ SAX($S^s[t_a, t_b], w, \alpha$)
8      $\mathcal{P}_{\Delta t} \leftarrow$ MINE_FREQ_SP($\mathcal{S}^s$, $k$, $min\_len$, $rdur$)
9      $\mathcal{F}_{\Delta t} \leftarrow$ CREATE_EMBEDDING($\mathcal{S}^s$, $\mathcal{P}_{\Delta t}$)
       // join
10     $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_{\Delta t}$
11     $\mathcal{F} \leftarrow \mathcal{F} \bowtie \mathcal{F}_{\Delta t}$
12     $\Delta t \leftarrow \Delta t \,/\, 2$
   // Train linear model
13 $\phi \leftarrow$ ELASTIC_NET($\mathcal{F}$, $\{y_i\}_{i=1}^{m}$)
14 return $\langle \mathcal{P}, \phi \rangle$
```

vectors (line 5-11) and train a linear model on the final embedding vector where we combine pattern-based feature values from different resolutions (line 13). The overhead of MR-PETSC is limited since we create at most $log(|S|)$ representations. For instance, if the original time series length is 128, we first create a single segment with $\Delta t = 128$, then Δt is equal to 64, 32 and finally 16 (assuming w is 15). In the embedding vector we will have $k \times 4$ pattern-based feature values from 4 different resolutions.

We remark that MR-PETSC produces high accuracy with *default parameters*, i.e., if we mine the top-200 sequential patterns with a minimum length of 5 and a constraint on relative duration of 1.1 on SAX strings consisting of 4 symbols ($\alpha = 4$)

and a fixed length of 15 ($w = 15$). However, for optimal performance we tune the hyper-parameters. We experimentally validate the accuracy of this ensemble method in Sect. 6.

5.2 Direct mining of the most discriminative patterns

In PETSC we mine all frequent patterns and rely on the elastic net classifier to select patterns that are frequent in one class rather than over different classes. We now propose PETSC-DISC, an alternative mining algorithm that discovers the most discriminative sequential patterns directly. Given a segmented time series dataset $\mathcal{S}^s$, label λ, time series windows with label λ (denoted by $\mathcal{S}^s_{\lambda+}$) and any other label (denoted by $\mathcal{S}^s_{\lambda-}$), we define discriminative support, or *contrast*, as

$$contrast(X, \mathcal{S}^s) = \Big| \frac{support_{rdur}(X, \mathcal{S}^s_{\lambda+})}{|\mathcal{S}^s_{\lambda+}|} - \frac{support_{rdur}(X, \mathcal{S}^s_{\lambda-})}{|\mathcal{S}^s_{\lambda-}|} \Big|.$$

Algorithm 4: MINE_DISCR_SP($\mathcal{S}^s$, λ, k, min_len, $rdur$, min_sup) Discover top-k discriminative sequential patterns in $\mathcal{S}^s$

```
Input: Segmented univariate time series $\mathcal{S}^s$, target class ($\lambda$), threshold on minimal support
       ($min\_sup$), other parameters similar to MINE_FREQ_SP
Result: Top-$k$ sequential patterns ranked on contrast
1  $Q \leftarrow \left[\langle \emptyset, \mathcal{S}^s_{\lambda+}, \mathcal{S}^s_{\lambda-}, \Omega \rangle\right]$
2  $\mathcal{P} \leftarrow$ MAKE_HEAP($k$)
3  $i \leftarrow 0$
4  while $Q \neq \emptyset$ or STOP($i$) do
5      $\langle X, P_{X,\lambda+}, P_{X,\lambda-}, Y \rangle \leftarrow$ POP($Q$)
6      if $Y = \emptyset$ then
7          if $|X| \geq min\_len$ then
8              PUSH($\mathcal{P}$, $\langle X, contrast(X, \mathcal{S}^s) \rangle$)
9              if $|\mathcal{P}| > k$ then
10                 POP($\mathcal{P}$)
11             $i \leftarrow i + 1$
12     else
13         if $support_{rdur}(X, \mathcal{S}^s) < min\_sup$ then
14             continue
15         $s_{l+1} \leftarrow$ FIRST($Y$);
16         PUSH($Q$, $\langle X, P_{X,\lambda+}, P_{X,\lambda-}, Y \setminus \{s_{l+1}\} \rangle$)
17         $Z \leftarrow (X, s_{l+1})$
18         $P_{Z,\lambda+} \leftarrow$ PROJECT($\mathcal{S}_{\lambda+}$, $Z$, $P_{X,\lambda+}$, $rdur$)
19         $P_{Z,\lambda-} \leftarrow$ PROJECT($\mathcal{S}_{\lambda-}$, $Z$, $P_{X,\lambda-}$, $rdur$)
20         $Y_Z \leftarrow$ PROJ_CANDIDATES($\mathcal{S}_{\lambda+}$, $Z$, $P_{Z,\lambda+}$) $\cup$ PROJ_CANDIDATES($\mathcal{S}_{\lambda-}$, $Z$, $P_{Z,\lambda-}$)
21         PUSH($Q$, $\langle Z, P_{Z,\lambda+}, P_{Z,\lambda-}, Y_Z \rangle$)
22 return $\mathcal{P}$
```

The algorithm for mining discriminative patterns directly is shown in Algorithm 4. A key difference with Algorithm 2 is that we employ *heuristic search* using *contrast* and visit candidates with high contrast first. However, since contrast is not anti-monotonic we cannot prune on contrast nor guarantee that after i iterations the current discovered top-k patterns have the highest contrast overall.

We start by creating an empty priority queue that contains all sequential pattern candidates sorted on contrast and a heap of discovered patterns sorted on descending contrast (line 1-2). Like before, during recursion we maintain candidate sequential patterns $X = (s_1, \ldots, s_l)$ and construct sequential patterns Z of length $l + 1$ where X is the prefix and item s_{l+1} is from the set Y. For efficiently computing contrast we maintain separate projections of X on time series instances with and without label λ, i.e., computed on $\mathcal{S}^s_{\lambda^+}$ and $\mathcal{S}^s_{\lambda^-}$, and denoted as P_{X,λ^+} and P_{X,λ^-}. In the main loop, we first retrieve the candidate sequential pattern with the highest contrast from the priority queue (line 5). If the set Y of candidate items is empty, we add the current candidate (if $|X| \geq min\_len$) to the pattern heap. As soon as k patterns have been found we remove the pattern with the lowest contrast from the heap (line 9-10). If the set of candidate items is not empty, we also remove discriminative patterns that are too infrequent overall, using the *min_sup* parameter (line 13). Next, we create two nodes in the search tree: the current sequential pattern X without the first item s_{l+1} in the set of candidate items Y (line 16) and the sequential pattern $Z = (s_1, \ldots, s_l, s_{l+1})$ (line 21). Here, we compute incremental pseudo projections on both $\mathcal{S}_{\lambda^+}$ and $\mathcal{S}_{\lambda^-}$ (line 18-20). The main loop stops if the queue is empty, meaning we have enumerated all frequent sequential patterns. Alternatively, we stop after a fixed number of iterations i (line 4) or dynamically, i.e., when no new patterns with high contrast are discovered in the last k iterations. For multiclass datasets we run MINE_DISCR_SP for each label and join all patterns resulting in a set of at most $|\mathcal{Y}| \cdot k$ patterns. We then return the top-k patterns with the highest contrast for any label. We remark that we also experimented with sequential covering to remove redundant patterns (Cheng et al. 2008). However, in preliminary experiments, we found that the increase in accuracy was too marginal to justify the additional complexity and runtime of sequential covering.

5.3 Relaxing exact pattern matching

Several alternatives have been proposed to create an embedding based on sequences, shapelets or sequential patterns. Laxman et al. (2007) proposes to compute frequency based on non-overlapping occurrences for discrete patterns. Hills et al. (2014) propose to compute the minimal Euclidean distance between each contiguous (and continuous) pattern and continuous segment in the time series and Senin and Malinchik (2013) propose to compute class-specific weights using TF-IDF. From a quality perspective we argue that for matching discrete patterns small differences might be ignored, e.g., given the pattern $X = (a, a, a, b, b, b, b)$ and the segment $S_1 = (a, a, a, b, b, b, c)$ the distance is small and the mismatch possibly due to noise. Especially as the discretisation error grows with long patterns, e.g., for a pattern of length 14 and an alphabet of size 3 the *average rounding error* accumulates to $14 \cdot 1/(2 \cdot 3)$. Based on this observation, we propose to count *near matching* occurrences as an alternative to exact matching and propose another variant of PETSC, namely PETSC-SOFT. For a pattern X and segmented time series $\mathcal{S}^s$ we define the *soft support* as:

$$\begin{aligned} support_{soft}(X, \mathcal{S}^s, \tau) = \Big|\big\{ S^s_k[t_a, t_b] \mid S^s_k[t_a, t_b] \in \mathcal{S}^s \wedge \\ min\_dist(X, S^s_k[t_a, t_b]) < \tau \cdot \frac{|X|}{2 \cdot \alpha} \big\}\Big|, \\ min\_dist(X, S^s_k[t_a, t_b]) = min(\{\, dist(X, S^s_k[t_{a'}, t_{b'}]) \mid \\ S^s_k[t_{a'}, t_{b'}] \sqsubseteq S^s_k[t_a, t_b] \wedge t_{b'} - t_{a'} = |X| \}) \end{aligned}$$

Here $\frac{|X|}{2 \cdot \alpha}$ returns the average rounding error and *dist* is the Euclidean distance.[3] For instance, in our previous example, with $\tau = 1.0$, X would match segment S_1 since $dist(X, S_1) = 1$ is lower than the expected rounding error of $7/(2 \cdot 3) = 1.166$. We introduce the parameter τ to allow user-defined control for soft matching, e.g. given $\alpha = 4$ and $\tau = 1.0$, we would allow a distance of 0 for patterns of length 7, 1 for patterns of length 8 and 2 for patterns of length 16. For τ equal to 2 or 3 matching is more relaxed. Remark that when using soft matching, we assume the relative duration to be 1.0, as the method's complexity would severely increase otherwise. In Sect. 6 we experimentally validate $support_{soft}$.

5.4 Basic time series transformations

While the original authors of both BOP and SAX-VSM report more wins over 1-NN DTW on their selection of datasets, the opposite is reported by Bagnall et al. (2017) on the complete UCR benchmark. We remark that both methods have their own biases and that performance will vary depending on the type of dataset, akin to the No Free Lunch Theorem (Wolpert and Macready 1997). Nearest neighbour with DTW will miss correlation of discriminative subsequences at different locations. Inversely, BOP misses overall correlation of two time series by extracting only local phase-independent patterns. Consequently, ensemble methods that combine both approaches such as DTW-F (Kate 2016) result in an overall increase in accuracy. We conclude that applying the following basic time series transformations (Hyndman and Athanasopoulos 2018) on $S_i = (x_1, \ldots, x_n)$ may result in improved performance:

- Derivative (or lag $n = 1$): $\hat{x}_t = x_t - x_{t-1}$
- Second derivative (or lag $n = 2$): $x_t = \hat{x}_t - \hat{x}_{t-1}$
- Logarithm: $x_t = \ln x_t$
- Logarithm of derivative: $x_t = \ln \hat{x}_t$

Especially, in the context of the UCR benchmark that contains many smaller datasets of synthetic and/or sine-like nature with a strong overall correlation, applying dictionary-based methods on both the original time series and after each transform might enhance the ability to discover local discriminative patterns. For instance, on the UCR *Adiac* dataset the error of DTW is 0.376 and the error of PETSC is 0.381. However, after

[3] We use the ordinal values for SAX symbols when computing Euclidean distance, that is $b - a$ is 1 and $c - a$ is 2.

transforming this dataset using the logarithm the error of PETSC drops to 0.350. However, to be fair to all methods, we did not apply any time series transformations on the data used in our experiments in Sect. 6.

5.5 Time and space complexity

Since PETSC mines sequential patterns on each dimension independently the execution time is linear in the number of dimensions d for multivariate time series. The total number of windows in the sequential database for mining in each dimension is dependent on the number of instances $|\mathcal{S}|$, the time series length $|S|$ and the sliding-window interval Δt, i.e. $|\mathcal{S}^s| = |\mathcal{S}| \cdot (|S| - \Delta t + 1)$. If we assume that top-k frequent cohesive sequential pattern mining is linear in $|\mathcal{S}^s|$ the complexity of MR-PETSC for multivariate datasets is approximately $\mathcal{O}(d \cdot |\mathcal{S}||S| \log(|S|))$.[4]

For univariate time series PETSC is thus linear in both the training set size and time series length while MR-PETSC is quasilinear in the time series length since we consider at most $log(|S|)$ SAX representations. Similar to MR-PETSC, MR-SEQL is also linear in the training set size and quasilinear in the time series length. However, for high accuracy MR-SEQL uses $\sqrt{|S|}$ different SAX and SFA representations, resulting in a slighly worse complexity of $\mathcal{O}(d \cdot |\mathcal{S}||S|^{\frac{3}{2}} log(|S|))$. Other scalable multivariate time series methods include Proximity Forests and TS-CHIEF that are based on an ensemble of decision trees are quasilinear in the training set size, but quadratic in the time series length. Finally, ROCKET is only linear in the time series length, i.e. the complexity on multivariate time series is given by $\mathcal{O}(d \cdot k|\mathcal{S}||S|)$. However the constant k that specifies the number of different kernels to train on is high, i.e. the suggested default is 10,000.

From a space complexity perspective, the proposed pattern mining algorithms relies on a depth-first search strategy to enumerate candidate sequential patterns and as such has manageable memory requirements, i.e. the mining algorithm only has to keep at most max_len candidate sequential patterns in memory, which is limited by w. Likewise the size of heap is limited by the maximal number of patterns k. Furthermore, by relying on incremental pseudo-projections of the sequential database, the space complexity is bounded by the representation of the sequential database itself which is linear in the training set size and linear in the time series length. The embedding data structure is linear in the training set size and in the number of patterns. Since the number of patterns k is limited to 1000 in practice and by adopting sparse data structures (since most pattern occurrences will be 0) this is efficient. Overall, for MR-PETSC, the space complexity is $\mathcal{O}(d \cdot k \cdot log(|S|)|\mathcal{S}|)$ since we concatenate $|\mathcal{S}|$ embedding vectors of length k for each dimension d in different resolutions $log(|S|)$ in memory.

[4] Note that pattern mining has a worst-case time complexity which is exponential in the size of the pattern and the alphabet size. That is, with a pattern size (or word size) of w and α different symbols, there are α^w possible sequential patterns of length w. However, we assume parameters such as w, α, k and $rdur$ are constants. That is, we argue that in the context of time series classification, and not pattern mining, it is less relevant to perform a detailed analysis of the efficiency of our method for large values of k or $rdur$, since we do not observe an increase in time series classification accuracy for large values of both k and $rdur$.

For large multivariate datasets, we see that in practice out-of-memory errors can occur, since a space complexity of $\mathcal{O}(|\mathcal{S}| \cdot |S|)$ for the sequential database can be problematic for very large datasets. In this case, we suggest to limit memory consumption by selecting a larger for the window increment i. For instance, for *EigenWorms* $|\mathcal{S}^s| = 128 \cdot (17{,}984 - 20 + 1) \approx 2.2 \cdot 10^6$ for $\Delta t = 20$. By selecting a larger window increment (or stride) than 1, we can reduce the size of the sequential database by a constant factor. For instance, given an increment i equal to Δt (i.e. non-overlapping sliding windows), the number of windows in each time series drops with $|S|/i$. For instance, for *Eigenworms* $|\mathcal{S}^s| = 128 \cdot (17{,}984)/20 = 115{,}098$ for $i = \Delta t = 20$. For large multivariate datasets this can lead to a speed-up proportional to i, however the frequency of patterns is then computed only approximately, as we require overlapping sliding windows to compute the exact count of pattern occurrences.

6 Experiments

In this section, we compare PETSC and its variants with existing state-of-the-art methods.

6.1 Experimental setup

Datasets
We compare methods against univariate and multivariate datasets from the UCR/UEA benchmark (Dau et al. 2018). In Table 1 we show the details on a subset of 19 univariate datasets from the UCR archive (Senin and Malinchik 2013). In Table 2 we show the details on 19 multivariate datasets from the UCR archive (Bagnall et al. 2018). We selected multivariate time series where $|S|$ was at least 30 and the total file size was not too large (i.e., $<$ 300 MB). We also compare totals against the 85 'bake off' univariate datasets (Bagnall et al. 2017; Dau et al. 2018), and 26 'bake off' equal-length multivariate datasets (Bagnall et al. 2018; Ruiz et al. 2021). Time series are from different domains and applications, i.e., time series extracted from images (such as shapes of leaves), resulting from a spectrograph or medical devices (ECG/EEG) and various sensors, simulations, motion detection and human activity recognition (HAR).
State-of-the-art methods
We compare the accuracy of PETSC and its variants with base learner such as 1-NN with DTW and comparable interpretable dictionary-based methods such as BOP, SAX-VSM and MR-SEQL (Le Nguyen et al. 2019) for univariate datasets. We also compare with non-interpretable dictionary-based methods based on a Bag of SFA Symbols, such as BOSS (Schäfer 2015), cBOSS (Middlehurst et al. 2019), S-BOSS (Large et al. 2019) and TDE (Middlehurst et al. 2020b), methods based on deep learning such as ResNet (Wang et al. 2017) and InceptionTime (Fawaz et al. 2020) and heterogeneous ensemble methods such as Proximity Forests (Lucas et al. 2019), ROCKET (Dempster et al. 2020), HIVE-COTE (Lines et al. 2018) and TS-CHIEF (Shifaz et al. 2020).

Table 1 Details on 19 UCR univariate datasets

Dataset	$\|\mathcal{Y}\|$	$\|\mathcal{S}_{train}\|$	$\|\mathcal{S}_{test}\|$	$\|S\|$	Type
Adiac	37	390	391	176	Image
Beef	5	30	30	470	Spectro
CBF	3	30	900	128	Simulated
Coffee	2	28	28	286	Spectro
ECG200	2	100	100	96	ECG
FaceAll	14	560	1690	131	Image
FaceFour	4	24	88	350	Image
Fish	7	175	175	463	Image
GunPoint	2	50	150	150	Motion
Lightning2	2	60	61	637	Sensor
Lightning7	7	70	73	319	Sensor
OliveOil	4	30	30	570	Image
OSULeaf	6	200	242	427	Image
SyntheticControl	6	300	300	60	Simulated
SwedishLeaf	15	500	625	128	Image
Trace	4	100	100	275	Sensor
TwoPatterns	4	1000	4000	128	Simulated
Wafer	2	1000	6164	152	Sensor
Yoga	2	300	3000	426	Image

Table 2 Details on 19 UCR multivariate datasets

Dataset	$\|\mathcal{Y}\|$	$\|\mathcal{S}_{train}\|$	d	$\|S\|$	Type
ArtWordRec	25	275	9	144	Motion
AtrialFibr	3	15	2	640	ECG
BasicMotions	4	40	6	100	HAR
CharTraject	20	1422	3	60+	Motion
Cricket	12	108	6	1197	HAR
EigenWorms	5	128	6	17,984	Motion
ERing	6	30	4	65	HAR
Epilepsy	4	137	3	206	EEG
EthanolConc	4	261	3	1751	Spectro
FingerMov	2	316	28	50	EEG
HandMovDir	4	160	10	400	EEG
LSST	14	2459	6	36	Simulated
Libras	15	180	2	45	HAR
NATOPS	6	180	24	51	HAR
RacketSports	4	151	6	30	HAR
SelfRegSCP1	2	268	6	896	EEG
SelfRegSCP2	2	200	7	1152	EEG
SWalkJump	3	12	4	2500	ECG
UWaveGestLib	8	120	3	315	HAR

For multivariate datasets we compare with base learner such as 1-nearest neighbour classification using either Euclidean distance (1-NN ED), dimension independent dynamic time warping (DTW$_I$) and dimension dependent dynamic time warping (DTW$_D$) (Shokoohi-Yekta et al. 2017). We also compare with ensembles of state-of-the-art univariate classifiers that train a separate classifier over each dimension independently, such as ROCKET, HIVE-COTE, ResNet, cBOSS, Shapelet Transform Classifier (STC) (Hills et al. 2014) and methods based on random forests using shapelet-based and other time series features, such as Time Series Forest (TSF) (Deng et al. 2013), Generalized Random Shapelet Forest (gRSC) (Karlsson et al. 2016) and Canonical Interval Forest (CIF) (Middlehurst et al. 2020a). In univariate and multivariate experiments we use the publicly reported results from the UCR/UEA benchmark and 'bake off' studies (Bagnall et al. 2017, 2018; Ruiz et al. 2021).
Parametrisation of methods
All dictionary-based methods have the same preprocessing steps which include setting an appropriate window size for segmentation (Δt) and setting the word (w) and alphabet (α) length for the SAX (or SFA) representation. BOP and SAX-VSM have no additional parameters. BOSS has an additional parameter that controls if normalisation should be applied. For mining patterns PETSC has three additional parameters, namely the number of patterns (k), minimum length of a pattern (min_len) and a constraint on the duration (or cohesion) of pattern occurrences ($rdur$). PETSC-SOFT has an additional parameter τ to control soft matching. PETSC-DISC has an additional *min_sup* parameter to control the minimal support for discriminative patterns which we set to 3. For MR-PETSC we report the results with default parameters (i.e., $w = 15$, $\alpha = 4, k = 200, min\_len = 5$ and $rdur = 1.1$) and varying parameters. For all PETSC variations we set the regularisation parameters λ_1 and λ_2 for linear regression to default values (i.e. . We remark that the implementation and experimental scripts for PETSC are implemented using Java and Python and are *open source*.[5]
Hyper-parameter optimisation
For optimising preprocessing parameters Δt, w and α we use *random search* on a validation set (Bergstra and Bengio 2012). That is, we iterate through a fixed number of randomly sampled parameter settings and keep the parameter setting with the best accuracy after 100 iterations. Here, Δt is randomly sampled between 10% and 100% of the time series length. For SAX, w is between 5 and 30 and α between 3 and 12. For all PETSC variants, k is between 500 and 2500, min_len in $\{0.1w, 0.2w, 0.3w, 0.4w\}$ and $rdur$ in $\{1.0, 1.1, 1.2, 1.5\}$. For PETSC-SOFT, τ is in $\{1/2\alpha,\ 2/2\alpha,\ 3/2\alpha\}$.

6.2 Effect of hyper-parameters on the accuracy of PETSC

We begin our experimental analysis by having a closer look at the effect various hyper-parameters have on the accuracy of our algorithm.
Window and SAX parameters
In this experiment we run PETSC on a grid with varying window, SAX word and alphabet length on *Adiac* and *Beef*. Figure 7 shows the test error for varying w and α for PETSC on *Adiac* when Δt is $0.5 \cdot |S|$ and *Beef* when Δt is $0.1 \cdot |S|$. Default

[5] Source code of PETSC: https://bitbucket.org/len_feremans/petsc.

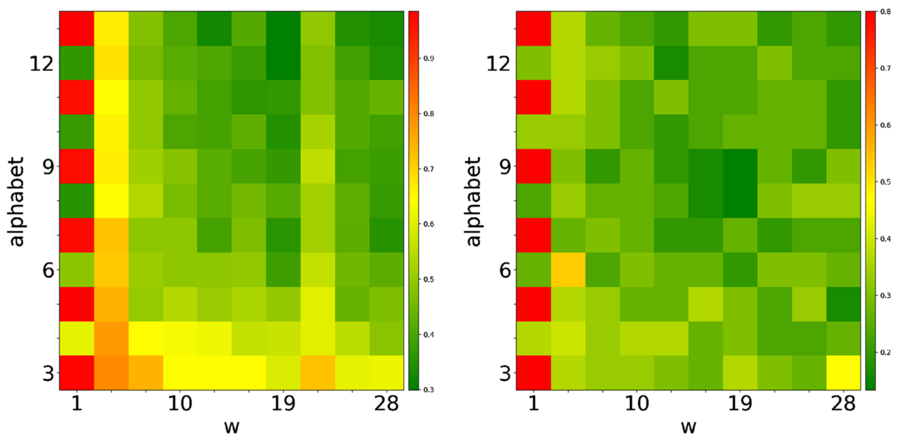

Fig. 7 Impact of varying w and α on the classification error of PETSC on *Adiac* (left) and *Beef* (right). The minimal error on *Adiac* is 0.299 ($w = 19$, $\alpha = 12$ and $\Delta t = 0.5 \cdot |S|$) and 0.133 on *Beef* ($w = 19$, $\alpha = 8$ and $\Delta t = 0.1 \cdot |S|$)

values for mining are $k = 1000$, $min\_len = 0.3w$ and $rdur = 1.0$. First, we observe that accuracy deteriorates for small values of either w or α. Next, we observe that for *Adiac* and *Beef* the error is minimal for high values of α and medium values for w, or the other way around with medium values for α and high values for w, but slightly worse if both parameters are set to high values.

In *Adiac* the optimal parameters result in frequent patterns of length 5 to 19 consisting of 12 distinct symbols and cover almost half the time series ($\Delta t = 0.5 \cdot |S|$), which is sensible since in *Adiac* time series are overall similar long sine-like waves, where subtle differences result in a different label. In *Beef* the optimal parameter for Δt is 10% and for $\Delta t = 0.25 \cdot |S|$ and $\Delta t = 0.5 \cdot |S|$ the minimal error (for any combination of w and α) increases with 6% and 10%. This suggests that short patterns are far more discriminative for this dataset and explains the high accuracy of PETSC on this dataset.

Pattern mining parameters
In this experiment we run PETSC on a grid with varying k, minimum length and relative duration on *Adiac* and *Beef*. Figure 8 shows the test error for varying k and *min_len* when *rdur* is 1.0 with optimal window and SAX parameters. First, we observe that setting k too low or *min_len* too high (i.e. more than $w/2$) results in lower accuracy. Second, we observe that the error (for any combination of k and *min_len*) increases with 4% and 10% with a relative duration of 2.0. We remark that it only took a couple of seconds to run PETSC for each parameter setting with a maximum of 30s for high values of *rdur* and k and low values of Δt.

Based on these and further experiments we conclude that:

– The parameter Δt is domain-specific and ranges from $0.1 \cdot |S|$ to $0.5 \cdot |S|$ and has a severe impact on accuracy. For short time series (i.e. $|S| < 50$), we set $\Delta t = |S|$, but this invalidates the use-case of discovering local phase-independent patterns.

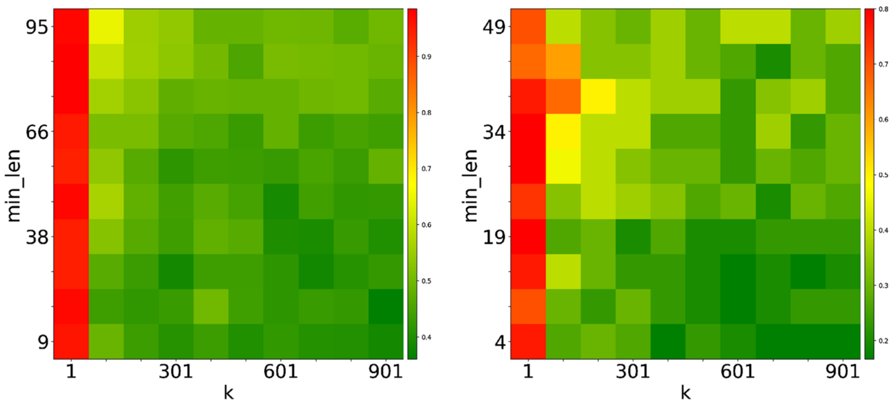

Fig. 8 Impact of varying k and *min_len* on the classification error of PETSC on *Adiac* (left) and *Beef* (right). The minimal error on *Adiac* is 0.360 ($k = 901$, *min_len* $= 19$ and *rdur* $= 1.0$) and 0.166 on *Beef* ($k = 701$, *min_len* $= 4$ and *rdur* $= 1.0$)

- SAX parameters α and w should be sampled together and have a severe impact on accuracy. Too low values might degrade performance, while setting both values high causes too much entropy.
- The parameter *rdur* is domain-specific and has impact on accuracy. On *Adiac* and *Beef* a value of 1.0 works best, however, we see that on most datasets the error is lowest if *rdur* is 1.1 or 1.5. If *rdur* is set too high (i.e., 2.0 or more) this results in degraded performance.
- For mining, k has less influence on accuracy but should be set sufficiently high, i.e., 1000 is a good default value. We experimented with k up to 5000, but this did not improve results. Finally, *min_len* should be lower than $w/2$, but not lower than 3.

6.3 Explaining PETSC and MR-PETSC

In this section we first discuss how we can use our model for instance-based decision support by visualising attribution and how to visualise and learn from the intrinsically interpretable model. We also provide a use-case where we discuss techniques to reduce the number of patterns to further improve explanations.

6.3.1 Instance-based decision support

For interpretability, we can compute the influence, or *attribution* of each sequential pattern occurrence weighted by the coefficient of the linear model. We use this attribution to highlight regions of the time series that lead to predicting each label y or not. First, we determine for each sequential pattern X_k its occurrences in the segmented SAX representation of a time series S_i. Next, we compute the inverse transform that maps each occurrence back to the original raw time series based on the window offset and the scaling factor $\Delta t/w$. Next, we compute the attribution of a pattern at each location in S_i which is $w_k/support_{rdur}(X_k)$ where w_k is the corresponding coefficient of pattern X_k in the linear model. We remark this only includes patterns that occur at least once in S_i and have a non-zero coefficient w_k. For interpretability, we make sure the intercept of the linear model (w_0) is 0 and that the embedding vector is mean-centred, such that the weights multiplied by the feature values (feature effects) explain the contribution to the predicted outcome (Molnar 2020). The sum of the attribution of each pattern at each location indicates the relative importance of a location and the sign indicates if is a positive or negative for class y. For MR-PETSC the sum of attributions is composed of patterns at each resolution, but this does not impede interpretation compared to PETSC, since we still employ a single linear model of patterns. In Fig. 9 we show the first two time series with different label for the *Gun-Point*, *ECG200* and *TwoPatterns* dataset where we highlight discriminative regions by visualising the sum of pattern attributions at each location. In the visualisation, the colour and the thickness of the line are determined by the sign and the absolute value of the sum of attributions. We observe that for discriminating between the Gun-drawn and Point gestures in the *GunPoint* dataset, the small dip after the main movement is discriminative for the Point gesture which confirms existing work (Ye and Keogh 2011; Le Nguyen et al. 2019).

6.3.2 Global explanations

PETSC learns an intrinsic sparse interpretable model, consisting of k weights $w_1, \ldots w_k$ of the linear model and the corresponding k sequential patterns. We can inspect this model, visually or otherwise, and provide decision support instance-based as explained previously, on a feature level, i.e., by rendering all occurrences of a single pattern, or globally, for instance by rendering the most discriminative patterns directly as shown in Fig. 3.

In Table 3 we show the top-5 patterns with the highest absolute weight w_k in each resolution on the *Gunpoint* dataset where MR-PETSC has an error of 0.04 ($w = 20$, $\alpha = 12$, $k = 250$, $min\_len = 6$ and $rdur = 1.0$). We remark that there is *redundancy* between the patterns and we will discuss techniques to handle this in the next section. A positive weight and a large absolute value indicate that the first pattern is specific to class *Gun*, meaning the gesture of drawing a gun and holding it, instead of pointing a finger. Note that our model is sparse, which is beneficial for explanations (Molnar 2020), and that the first pattern only occurs in 8% of time series windows. We conclude that PETSC and MR-PETSC offer a transparent model (Molnar 2020) and enable human experts to inspect each pattern specific to class A or B and trust decisions for *any* possible instance.

6.3.3 Interpretation use-case

While the SAX representation is interpretable, as are the resulting sequential patterns, it is not obvious how to explain real-world use-cases, especially since MR-PETSC discovers thousands of patterns in multiple resolutions. Therefore we further illustrate this based on a use-case. We consider the *BeetleFly* dataset from the UCR/UEA

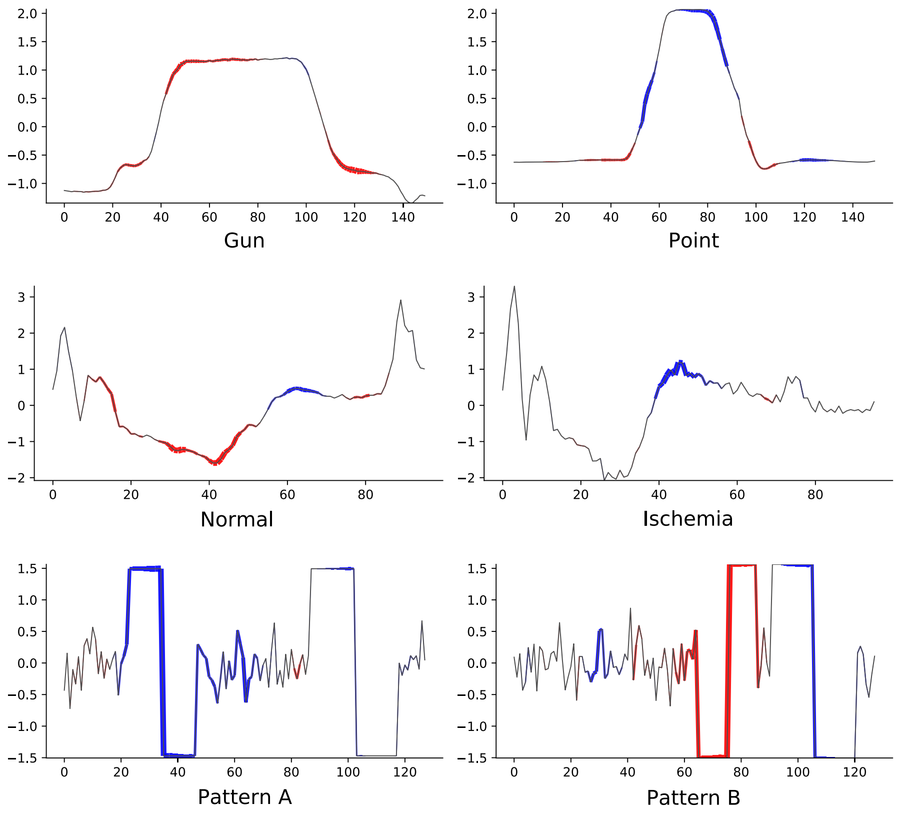

Fig. 9 The first two time series with different class from the *GunPoint*, *ECG200* and *TwoPatterns* dataset. We use MR-PETSC to discover patterns in different resolutions and highlight regions that are more important for making predictions with MR-PETSC

benchmark. The *BeetleFly* dataset consists of 20 training images of either a beetle or fly transformed to a time series as illustrated in Fig. 10.

Identifying optimal hyper-parameters

We used a random search to identify that the optimal SAX representation consists of 12 bins ($\alpha = 12$) and words of size 30 ($w = 30$). For pattern mining the optimal parameters are $k = 1000$, $min\_len = 8$ and $rdur = 1.0$. Note that using random search we identified multiple parameter settings at which the error on the validation set was 0.0. The total time for running MR-PETSC 100 times with random parameters was 10 min using 8 cores. Note that using our open-source code it is easy to identify optimal parameters using random search on a validation set (using multiple threads) and then run the method with the optimal parameters on the test set.

Accuracy

We find that MR-PETSC achieves state-of-the-art performance with a large relative increase over ROCKET on many datasets originating from the shapelet tranformation method (Hills et al. 2014), such as *BeetleFly* where the accuracy is 100%. Shapelets and sequential patterns are both based on rotational-invariant subsequences, so it is not

Table 3 Top-5 most discriminative patterns for each resolution of MR-PETSC for *Gunpoint*

Δt	Weight	Seq. pattern	Support
	0.715	ejllll	0.08
	−0.656	dddddddfklll	0.04
$\|S\|$	−0.656	dddddddf	0.04
	−0.656	dddddddfkll	0.04
	−0.656	dddddddfkl	0.04
	−0.796	hjkkkkkkk	0.02
	−0.794	bbbbbcccc	0.02
$\|S\|/2$	−0.764	bbbbbcc	0.03
	−0.742	bbbbbc	0.04
	0.730	bbbbbbbb	0.04
	−1.000	aaaabc	0.02
	−0.926	bbbbbc	0.01
$\|S\|/4$	0.799	eeeeeeeeee	0.01
	0.793	cccccccccc	0.04
	−0.749	ddcccc	0.01

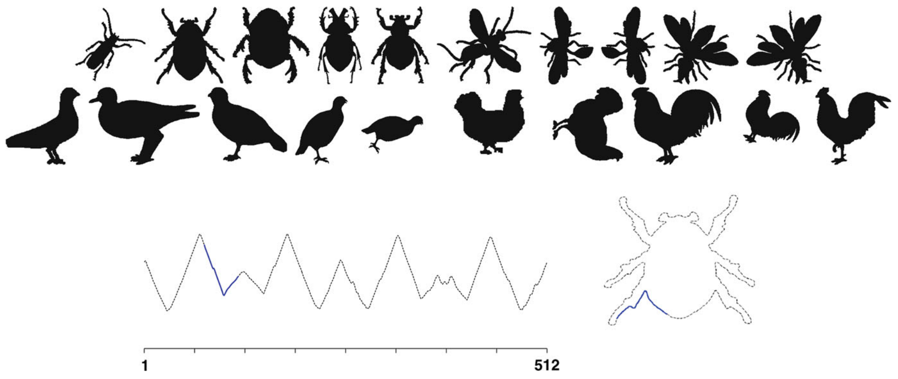

Fig. 10 Top figure shows instances of the *BeetleFly* and *BirdChicken* datasets. Bottom figure illustrates the outline of a beetle represented as a time series where a subsequence is highlighted in blue (Hills et al. 2014). With optimised pre-processing parameters and after removing redundant patterns based on Jaccard similarity PETSC achieves an accuracy of 80% and the model consists of a single subsequence which can subsequently be highlighted in each image outline to facilitate trivial interpretation

surprising that both approaches work well on similar datasets. The relative improvements achieved by MR-PETSC over ROCKET on such datasets is significant, i.e., *BeetleFly* (+13%), *Herring* (+12%), *Lightning2* (+12%), *CinCECGTorso* (+10%), *FaceFour* (+7%) and *BirdChicken* (+2%).

Multi-resolution pattern visualisation

The benefit of our model is that at each resolution we have the sequence of symbols of each pattern. Moreover, we can match each sequential pattern to the time series and inspect the individual occurrences. In Fig. 11 we show 4 resolutions of the same time series of a fly and a beetle. With a window of 512 the window size is equal to

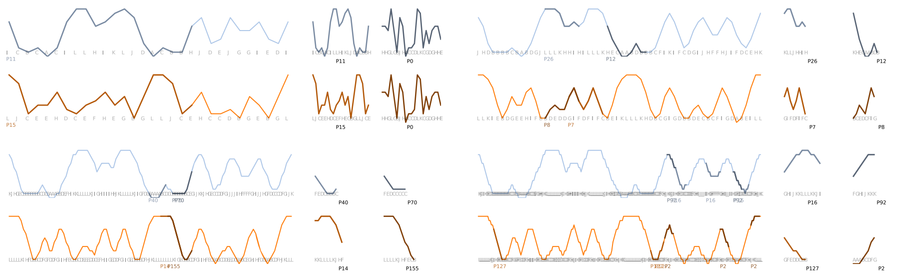

Fig. 11 Example of two time series of a fly and a beetle using a window of 512 (= $|\mathcal{S}|$), 256, 128 and 64. Note that by using the same SAX parameters on different windows results in more fine-grained patterns matching more closely a shorter piece of the time series

the time series length. In the last subfigure each window has size 64 or 1/8 of the time series. Remark that by using the same SAX parameters on different windows results in more fine-grained patterns matching more closely a shorter piece of the time series. For visualisation purposes we normalise the entire time series instead of applying window-level normalisation.

Reducing the number of patterns

A problem is that by having 4 resolutions with 1000 patterns each, interpretation becomes difficult. We propose to reduce the number of varying length patterns in each resolution significantly by removing redundant patterns, thereby removing patterns while maintaining high accuracy. Pattern redundancy has been extensively studied in the pattern mining community (Han et al. 2011; Aggarwal and Han 2014). We chose to remove redundant patterns based on Jaccard similarity, defined as:

$$jaccard(X_1, X_2, \mathcal{S}^s) = \frac{cover(X_1, \mathcal{S}^s) \cap cover(X_2, \mathcal{S}^s)}{cover(X_1, \mathcal{S}^s) \cup cover(X_2, \mathcal{S}^s)}.$$

Next we create a non-redundant pattern set $\mathcal{P}'_\gamma$ such that no two patterns in the set have Jaccard similarity higher than γ, that is we add X_j to $\mathcal{P}'_\gamma$ if and only if $\nexists X_i \in \mathcal{P}'_\gamma : i \neq j \wedge jaccard(X_i, X_j, \mathcal{S}^s) > \gamma$. For instance, with $\gamma = 0.9$ we remove patterns where 90% of occurrences overlap. The Jaccard similarity is of particular interest because of the sliding window-based preprocessing which results in many patterns that overlap within a small delay, i.e., given the SAX sequences (a, a, a, b, b) and (a, a, b, b, b) we have frequent sequential patterns $X_1 = (a, a)$, $X_2 = (a, b)$, $X_3 = (b, b)$ and $X_4 = (a, a, b, b)$ which all make each other redundant, and it is safe to remove all but one of them.

The number of patterns after filtering on *BeetleFly* are shown in Table 4. We also show the number of patterns with a non-zero weight after training the linear model using an elastic net (see Sect. 4.4). We observe that, when the window size is set to 512, 256 or 32 the number of patterns that are non-redundant w.r.t. $\gamma = 0.9$ becomes very small. In Table 5 we show the corresponding error of PETSC for each setting for different values of γ. We observe that on *BeetleFly* using the complete series with a window size of 512 results in an error of 0.5, which is meaningless since we only have two classes (we also observe this when plotting the attribution since the mean

Table 4 Number of non-redundant patterns in *BeetleFly* in each resolution

Window	$\|\mathcal{P}\|$	Non-zero weight	$\|\mathcal{P}'\|_{\gamma=1.0}$	$\|\mathcal{P}'\|_{\gamma=0.9}$	$\|\mathcal{P}'\|_{\gamma=0.5}$
512	1000	1000	20	20	20
256	1000	998	73	73	26
128	1000	997	714	641	70
64	1000	995	477	286	<u>10</u>
32	1000	998	142	75	**1**

The setting for γ that results in the lowest number of patterns is considered best and shown in bold, the second best setting is underlined

Table 5 Error after filtering non-redundant patterns in *BeetleFly* for different window sizes and values of γ and by combining patterns in all windows

Window	$\mathcal{P}$	$\mathcal{P}'_{\gamma=1.0}$	$\mathcal{P}'_{\gamma=0.9}$	$\mathcal{P}'_{\gamma=0.5}$
512	0.500	0.500	0.500	0.500
256	0.450	0.400	0.400	0.250
128	0.250	0.200	0.200	0.250
64	<u>0.100</u>	0.150	**0.050**	0.250
32	0.150	<u>0.100</u>	<u>0.100</u>	0.200
Combined	0.05	0.05	0.100	0.250

The setting with the lowest error is shown in bold, the second best settings are underlined
The best result misclassifies only 1/20 test instances using 286 patterns
However, using just a single pattern we have a setting that misclassifies only 4/20 test instances

attribution accumulates to zero). The lowest error is 0.05 when the window is 64 and γ is 0.9, meaning that 1/20 time series is misclassified. Using a window of 32 and γ equal to 0.5 we have a single pattern and only misclassify 4/20 time series, which is interesting. We remark that we can further reduce the number of patterns by increasing the regularisation weights $\lambda_{\{1,2\}}$ (see Sect. 4.4).

Interpretation of outline images

For interpretation of the *BeetleFly* patterns, we created a version of the dataset starting from the original images enabling us to render pattern occurrences in the original 2D space for easy interpretation.[6] We applied the *radial distance* method to convert the original MPEG-7 images to time series (Bober 2001; Adamek and O'Connor 2003). That is, we extract the outline and then compute the distance between the centre and each point in the outline.

First we run PETSC with parameters as discussed previously ($window = 64$, $w = 30$, $\alpha = 10$, $k = 1000$, $min\_len = 8$, $rdur = 1$) thereby discovering 315 frequent patterns with a non-zero weight. Using this setting 15/20 images are correctly classified. Next, we remove redundant patterns using a Jaccard similarity threshold of 0.5 resulting in 119 patterns of which only 38 have a non-zero weight. Surprisingly, the accuracy increases to 17/20 in this case. Finally we render each pattern in the

[6] We remark that there are differences in our creation of *BeetleFly* dataset compared to the UCR version due to small changes in the pre-processing of the original MPEG-7 source images.

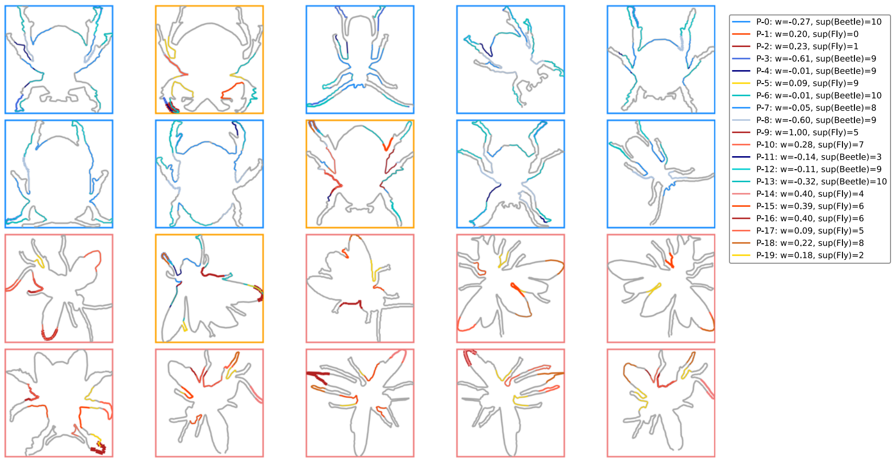

Fig. 12 Visualisation of the top-20 patterns on the *BeetleFly* dataset where we have an accuracy of 0.85 meaning 3/20 test images are misclassified. We show the top-20 patterns with the highest weight, where blue patterns are discriminative for beetles (i.e., part of legs) and red patterns for flies (i.e., part of wings). Here, each image is first converted to a time series using the *radial distance* method. For classification we learn a linear model using PETSC and subsequently remove redundant patterns resulting in 38 patterns. For visualisation we map each pattern occurrence from the SAX representation to the original time series and then back to the original 2D outline

original representation. First, we compute occurrences of each pattern in the SAX-based sliding window representation. Next, we map each occurrence to the original time series representation and then back to the outline. The top-20 patterns with the highest weight are shown in Fig. 12. We observe that patterns correspond to important discriminative features such as the legs of a beetle or the wings of a fly. We remark that a similar approach can be used to visualise the top-20 (non-redundant) patterns discovered in multiple resolutions using MR-PETSC.

We find that by reducing the number of patterns using a threshold on Jaccard similarity we can drastically filter the number of patterns and make the model extremely comprehensible. However, often a larger amount of patterns may result in higher accuracy, so there a *trade-off* to be made between model complexity and accuracy as is expected. We find that by removing redundant patterns it is straightforward to make a trade-off between model comprehension and accuracy.

6.4 Comparing PETSC variants

6.4.1 Comparing accuracy of PETSC variants

In this experiment, we compare the accuracy of PETSC with PETSC-SOFT, PETSC-DISC and MR-PETSC. We also compare with BOP, since the fixed-length SAX sequences used by BOP are a special case of the patterns in PETSC, i.e., if we set *rdur* to 1.0, *min_len* to *w* and *k* to a large number (to include all patterns with a support greater than 0), PETSC and BOP produce the same patterns and embedding. To avoid that results are biased by the random search for hyper-parameter optimisation for BOP, PETSC and its variants we employ cross-validation and report the mean error on the publicly available train/test split pairs after 5 runs.

Table 6 shows the error on 19 univariate datasets. We see that some methods work better on some datasets. However, overall MR-PETSC produces the lowest average error. Moreover, even with default parameters MR-PETSC has an average error of only 0.139, which is lower than other variants even with optimised parameters. By considering varying length patterns PETSC (and all its variants) outperforms BOP by +3%. By considering patterns of both varying length and resolution, MR-PETSC outperforms BOP by +6.5%. When comparing PETSC with PETSC-SOFT we observe that near matching patterns results in best results on 5/19 datasets which is promising. The results for PETSC-DISC are on par with PETSC, however on datasets such as *GunPoint* and *TwoPatterns* where the presence of clear local patterns is well known, PETSC-DISC performs slightly better. However, on other datasets PETSC is better which would suggest that frequent cohesive sequential patterns are as useful as sequential patterns with high contrast. To limit the number of experiments, we only compare MR-PETSC with the state-of-the-art methods. We decided not to study all possible combinations, i.e., using multiresolution or discriminative patterns in combination with $support_{soft}$ for constructing the embedding, or combining discriminative patterns mined in different resolutions.

6.4.2 Comparing execution time of PETSC variants

In this experiment, we compare the execution time of BOP, PETSC and its variants. We included time for both training and testing using random search with 100 iterations and report the mean and total time after 5 runs. The execution times are reported in Table 7 where we report individual results on the 4 largest datasets. We observe that PETSC is faster than BOP. Considering that PETSC creates variable length patterns thereby covering, in theory, an exponentially large search space, this is surprising. However, BOP requires the nearest neighbour search at test time which explains why it starts to slow down on datasets with many instances, e.g., on *TwoPatterns* which consists of 1000 training instances. Compared to PETSC, PETSC-SOFT is a slower because we must use the naïve algorithm for computing the embedding using $support_{soft}$. In contrast, PETSC uses a more efficient algorithm for computing the embedding based on $support_{rdur}$ thereby leveraging sparse optimisations. Finally, MR-PETSC runs PETSC at most $log(|S|)$ times and is about 7 times slower than PETSC. We conclude that all PETSC variants are extremely efficient in absolute time. MR-PETSC takes on average 1.1h to run 100 random parameter settings and the total time on all 19 datasets is 3.2h for PETSC and 20.9 h for MR-PETSC using a single core.

6.5 Comparing against state-of-the-art methods

In this section, we compare our algorithm to a variety of existing classifiers in terms of accuracy and runtime on both univariate and multivariate datasets. To avoid congested

Table 6 Classification error of BOP, PETSC and its variants on 19 univariate time series from the UCR

Dataset	BOP	PETSC	PETSC- SOFT	PETSC- DISC	MR- PETSC
Adiac	0.403	0.398	0.391	0.420	<u>0.375</u>
Beef	0.367	0.360	<u>0.287</u>	0.373	<u>0.287</u>
CBF	0.033	0.026	<u>0.014</u>	0.019	0.017
Coffee	0.064	0.057	0.064	0.079	<u>0.043</u>
ECG200	0.198	0.184	<u>0.153</u>	0.184	0.178
FaceAll	0.252	0.257	0.240	0.265	<u>0.225</u>
FaceFour	0.123	0.030	0.050	0.032	<u>0.005</u>
Fish	0.112	0.064	0.069	0.097	<u>0.026</u>
GunPoint	<u>0.027</u>	0.076	0.028	0.059	0.063
Lightning2	0.298	0.266	0.256	0.295	<u>0.203</u>
Lightning7	0.466	0.301	0.356	0.332	<u>0.266</u>
OSULeaf	0.328	0.207	0.146	0.188	<u>0.036</u>
OliveOil	0.167	0.142	0.313	0.120	<u>0.047</u>
SwedishLeaf	0.250	0.140	0.154	0.158	<u>0.108</u>
SyntheticControl	0.031	<u>0.022</u>	0.027	0.034	0.029
Trace	0.014	0.006	<u>0.004</u>	<u>0.004</u>	<u>0.004</u>
TwoPatterns	0.034	0.018	<u>0.006</u>	0.015	0.024
Wafer	0.008	0.014	0.008	0.011	<u>0.005</u>
Yoga	<u>0.178</u>	0.217	0.205	0.219	0.188
avg. error	0.177	0.147	0.146	0.153	<u>0.112</u>
avg. rank	4.0	3.1	2.4	3.6	<u>1.6</u>

The settings with the lowest error are underlined
PETSC and its variants outperform BOP on most datasets

Table 7 Runtime in seconds of BOP, PETSC and its variants on the four largest datasets, and average and total runtimes on all 19 univariate datasets from the UCR archive (average over 100 runs using random search)

Dataset	BOP	PETSC	PETSC- SOFT	PETSC- DISC	MR- PETSC
FaceAll	2047	<u>1071</u>	1279	1859	6400
TwoPatterns	6313	<u>1272</u>	1957	1961	4683
Wafer	1350	<u>1076</u>	2135	1578	3920
Yoga	<u>1108</u>	1709	1927	2120	10,052
avg. time	786	<u>619</u>	836	1065	3976
total. time	14,938	<u>11,763</u>	15,886	20,240	75,548

The settings with the lowest runtime are underlined figures and huge tables, we report a selection of comparisons here, while the complete raw experimental results are available at our website.[7]

6.5.1 Comparing accuracy on univariate datasets

In this experiment we compare MR-PETSC against comparable interpretable base learners and current state-of-the-art methods, such as HIVE-COTE, ROCKET, TS-CHIEF and InceptionTime. Hyper-parameters for MR-PETSC were optimised using random search with 100 iterations on a validation set. Since MR-PETSC often finds multiple parameter settings at which the error is 0.0 on the validation set, we ran the parameter optimisation 3 times and report the best of 3 runs thereby assuming an Oracle. For all state-of-the-art methods we use the reported results available from the UCR archive (Bagnall et al. 2017; Dau et al. 2018). If no data was available, we ran experiments ourselves using sktime (Löning et al. 2019).

The results are shown in Fig. 13 where we compare the average rank of MR-PETSC with other methods on 85 'bake off' datasets of the UCR archive using a critical difference diagram. Tests are performed with the Wilcoxon signed-rank test using the Holm correction (Demšar 2006). We see that MR-PETSC outperforms baseline methods such DTW, BOP and SAX-VSM. MR-PETSC performs comparably to MR-SEQL, cBOSS, BOSS, S-BOSS, ProximityForest and ResNet, but ranks significantly lower than the current state-of-the-art methods Inceptiontime, TS-CHIEF, ROCKET and HIVE-COTE, none of which are interpretable. In Fig. 14 we compare the accuracy of MR-PETSC with ROCKET, ResNET, BOSS and MR-SEQL on a subset of UCR/UEA datasets where both methods produced results. MR-PETSC outperforms ROCKET on 32/109 datasets (ROCKET does better on 64/109 datasets, while the others were ties), ResNET on 42/112 datasets, BOSS on 50/112 datasets and MR-SEQL on 31/85 datasets. This shows that while ROCKET ranks significantly higher on average, MR-PETSC still outperforms ROCKET on 29% of datasets, which is excellent for an interpretable method. Finally, in Table 8 we report details on 19 univariate datasets from the UCR archive. On 9 out of 19 datasets MR-PETSC reports the best results compared to DTW, BOP, SAX-VSM and BOSS. Even with suboptimal default parameters MR-PETSC has an average error of 0.129 which is lower than other methods except BOSS.

[7] Full experimental results: https://bitbucket.org/len_feremans/petsc/src/master/Results.html.

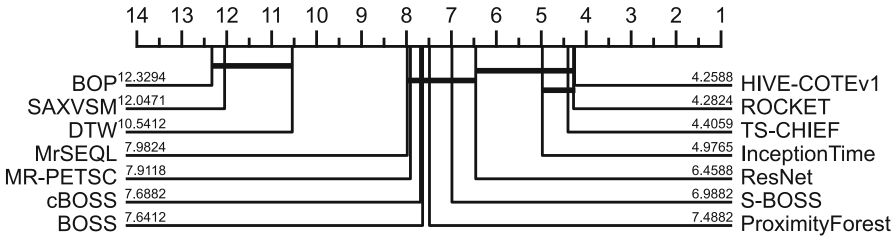

Fig. 13 Average rank of MR-PETSC compared to 13 time series classification methods on a subset of the 85 'bake off' univariate datasets. A solid bar indicates there is no significant difference in rank. The top clique of four classifiers represent the current state-of-the-art (Dau et al. 2018)

Table 8 Classification error of MR-PETSC compared to DTW, BOP, SAX-VSM and BOSS on 19 univariate time series from the UCR archive

Dataset	MR- PETSC	DTW	BOP	SAX- VSM	BOSS
Adiac	0.350	0.376	0.408	0.543	<u>0.251</u>
Beef	<u>0.200</u>	0.533	0.497	0.504	0.385
CBF	0.013	0.007	0.037	0.042	<u>0.002</u>
Coffee	<u>0.000</u>	0.214	0.056	0.062	0.011
ECG200	0.110	0.120	0.214	0.165	<u>0.110</u>
FaceAll	0.225	0.189	0.061	0.035	<u>0.026</u>
FaceFour	<u>0.000</u>	0.024	0.052	0.057	0.004
Fish	<u>0.017</u>	0.051	0.109	0.059	0.031
GunPoint	0.020	0.013	0.030	0.041	<u>0.006</u>
Lightning2	0.230	0.213	0.303	0.256	<u>0.190</u>
Lightning7	<u>0.247</u>	0.247	0.447	0.404	0.334
OliveOil	<u>0.067</u>	0.167	0.153	0.154	0.130
OSULeaf	<u>0.029</u>	0.248	0.300	0.140	0.033
SyntheticControl	<u>0.020</u>	0.023	0.074	0.131	0.032
SwedishLeaf	0.112	0.102	0.216	0.294	<u>0.082</u>
Trace	<u>0.000</u>	0.050	0.023	0.008	0.000
TwoPatterns	0.021	<u>0.001</u>	0.056	0.111	0.009
Wafer	0.008	0.004	0.003	0.004	<u>0.001</u>
Yoga	0.176	0.130	0.138	0.164	<u>0.090</u>
avg. error	0.097	0.142	0.167	0.167	<u>0.090</u>
avg. rank	2.3	3.1	3.9	4.1	<u>1.6</u>

The settings with the lowest error are underlined

We observe that MR-PETSC improves on both BOP and SAX-VSM and is not significantly worse than BOSS

6.5.2 Comparing execution time on univariate datasets

In a first experiment, we compare the execution time of MR-PETSC with DTW and BOSS. The average execution time for training and making predictions on the 19

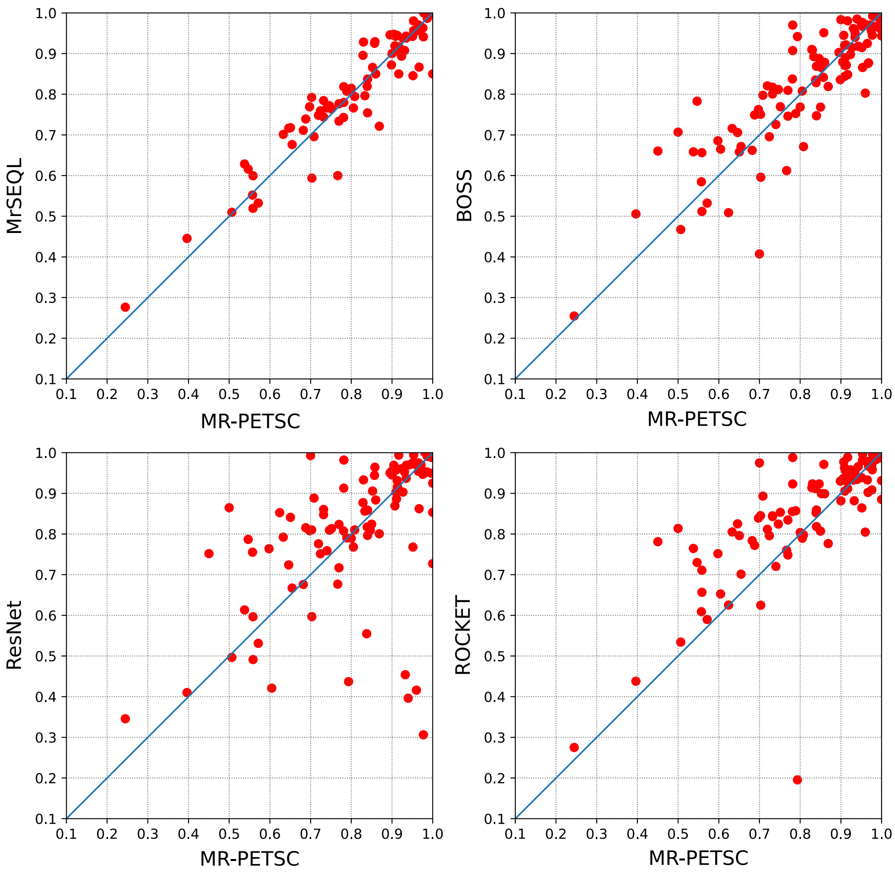

Fig. 14 Comparison of accuracy between MR-PETSC and 4 univariate methods. Each dot is one of the (at most) 112 univariate datasets from the UCR/UEA repository. A dot below the diagonal line indicates MR-PETSC is more accurate

UCR datasets is 16s for DTW, 34s for MR-PETSC and 2218s for BOSS.[8] BOSS takes minutes or hours where both MR-PETSC and DTW require seconds or minutes to complete. DTW is overall faster, but MR-PETSC is 30% faster than DTW on the two largest datasets consisting of a 1000 training instances, that is, on *TwoPatterns* and *Wafer* DTW took 94s and 151s, while MR-PETSC took 69s and 107s. We remark that our implementation is partially in Java and Python and that we do no take advantage of low-level optimisations available in C++.

In a second experiment we ran MR-PETSC with default parameter settings on the 85 ‘bake off’ datasets of the UCR archive one by one using a single core. We compare the total execution time of both training and testing and compare with recent state-of-the-art methods focusing on faster training, namely ROCKET, cBOSS which is an optimisation of BOSS, MR-SEQL, MiSTiCl, InceptionTime, Proximity Forest and TS-CHIEF. In Table 9 we show the results. We see that ROCKET is faster since it took

[8] We use the implementations available in the sktime library (Löning et al. 2019).

Table 9 Total execution time in hours for training and predictions on all 85 'bake off' UCR/UEA datasets for MR-PETSC and state-of-the-art methods (Dempster et al. 2020)

	ROCKET	MR-PETSC	cBOSS	MiSTiCL	MR-SEQL	INCEPTION TIME	PROXIMITY FOREST	TS-CHIEF
Total time	**1.4 h**	<u>2.7 h</u>	19.5 h	20.2 h	23.7 h	6 d	11 d	11 d

The setting with the best result is shown in bold, the second best result is underlined

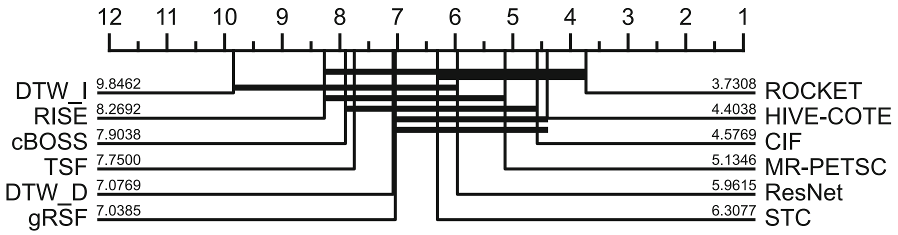

Fig. 15 Average rank of MR-PETSC compared to 11 time series classification methods on 26 'bake off' equal-length multivariate datasets. A solid bar indicates there is no significant difference in rank. In previous studies, the state-of-the-art methods were ROCKET, HIVE-COTE, CIF and ResNet (Ruiz et al. 2021)

1.4 h to complete, while MR-PETSC took 2.7 h to complete. However, MR-PETSC is substantially faster than other aforementioned methods.

6.5.3 Comparing accuracy on multivariate datasets

In this experiment we compare MR-PETSC with baseline distance-based methods and current state-of-the-art multivariate time series classification methods such as HIVE-COTE, ROCKET, CIF and ResNet (Ruiz et al. 2021). For optimising hyper-parameters for MR-PETSC we used grid search and report the best results after evaluating 27 parameter settings, w in {10, 15, 20}, α in {4, 8, 12} and *rdur* in {1.0, 1.1, 1.5} and assume an Oracle that selects the best parameters settings. On large datasets we chose default parameters. For all state-of-the-art methods we used the reported results available from Ruiz et al. (2021) or ran experiments ourselves using sktime (Löning et al. 2019) if no data was available.

The results are shown in Fig. 15 where we compare the average rank of MR-PETSC with other methods on 26 'bake off' multivariate datasets using a critical difference diagram. We observe that MR-PETSC is ranked above the state-of-the-art method ResNet and performs significantly better than DTW$_I$. ROCKET is ranked first and is significantly better than DTW$_I$, RISE, cBOSS, TSF, DTW$_D$ and gRSF, but not significantly better than CIF, HIVE-COTE, MR-PETSC, ResNET and STC. In Fig. 16 we compare the accuracy of MR-PETSC with ROCKET, ResNET, DTW$_D$ and DTW$_I$ on 26 datasets. We observe that MR-PETSC outperforms ROCKET 10/26 datasets (ROCKET does better on 15/26 datasets with one tie), ResNET on 16 datasets, DTW$_D$ on 17 datasets and DTW$_I$ on 23 datasets. We conclude that our interpretable method produces comparable results to the best-performing state-of-the-art methods, all of which are non-interpretable.

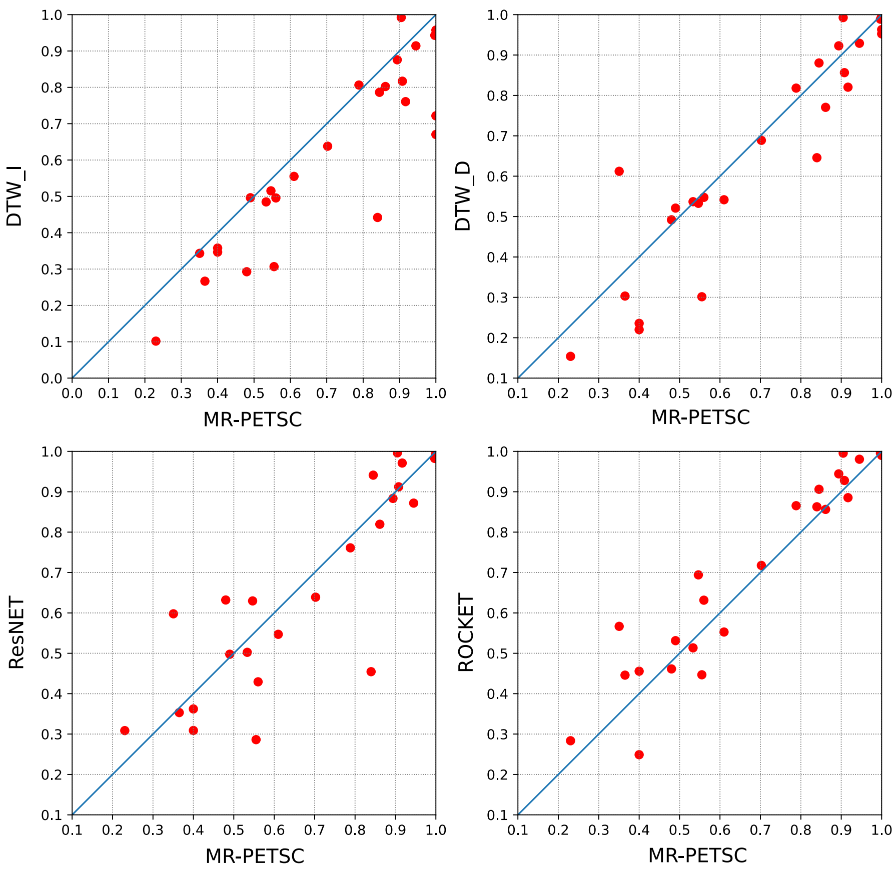

Fig. 16 Comparison of accuracy between MR-PETSC and 4 multivariate methods. Each dot is one of the 26 'bake off' multivariate datasets. A dot below the diagonal line indicates MR-PETSC is more accurate

Table 10 shows the error on 19 multivariate datasets for both MR-PETSC and distance-based baselines on 19 multivariate datasets. We observe that distance-based methods work better on some datasets, e.g., on *LSST* the error is 1.5% lower and on *Libras* the error is 4.9% lower, possibly since for these datasets $|S|$ is quite small. However, on 14/19 datasets the inverse is true and on *ERing* the error drops by more than 80% and on *EthanolConc* by more than 20% compared to the best distance-based approach. This suggests that for these datasets using whole time series distances is less useful as a feature and there is a clear benefit in using local sequential pattern occurrences as features. We conclude that MR-PETSC improves on distance-based baselines on multivariate datasets and is not significantly worse than current state-of-the-art methods. We remark that even with default parameters MR-PETSC has an average error of 0.329 and is still on par with distance-based approaches.

Table 10 Classification error of MR-PETSC compared to distance-based baseline methods 1-NN ED, 1-NN DTW$_I$ and 1-NN DTW$_D$ on 19 multivariate time series from the UCR

Dataset	MR- PETSC	1- NN ED	1- NN DTW$_I$	1- NN DTW$_D$
ArtWordRec	0.003	0.030	0.020	0.013
AtrialFibr	0.600	0.733	0.733	0.800
BasicMotions	0.000	0.325	0.000	0.025
CharTraject	0.059	0.036	0.031	0.010
Cricket	0.000	0.056	0.014	0.000
ERing	0.055	0.867	0.867	0.867
EigenWorms	0.229	0.450	0.450	0.382
Epilepsy	0.000	0.333	0.022	0.036
EthanolConc	0.445	0.707	0.696	0.677
FingerMov	0.390	0.450	0.480	0.470
HandMovDir	0.662	0.721	0.694	0.769
LSST	0.440	0.544	0.425	0.449
Libras	0.155	0.167	0.106	0.128
NATOPS	0.083	0.150	0.150	0.117
RacketSports	0.092	0.132	0.158	0.197
SelfRegSCP1	0.212	0.229	0.235	0.225
SelfRegSCP2	0.467	0.517	0.467	0.461
SWalkJump	0.600	0.800	0.667	0.800
UWaveGestLib	0.200	0.119	0.131	0.097
avg.error	0.245	0.388	0.334	0.343
avg.rank	1.5	3.1	2.5	2.4

MR-PETSC outperforms distance-based baseline methods

6.5.4 Comparing execution time on multivariate datasets

In this experiment we compare MR-PETSC with distance-based approaches on multivariate datasets. On most of the 19 multivariate UCR datasets MR-PETSC requires a couple of seconds or minutes and is slower than DTW if the number of instances is low. We remark that on *SelfReg(ulation)SCP1* we ran out of memory using a limit of at most 8GB, using the hardest parameters setting during grid search, i.e. $w = 15$, $\alpha = 12$, $rdur = 1.5$ and $min\_len = 5$. On *EigenWorms* MR-PETSC took about 3 h to process 262MB of data in sktime format. *EigenWorms* consists of 128 instances, but each instance consists of 6 dimensions of length 17,984. Segmenting long time series with a small window results in large sequential databases making MR-PETSC slower. However, since the pattern mining algorithms employ a depth-first strategy, they are memory efficient. To validate MR-PETSC performance on large datasets we ran it on *FaceDetection* ($|\mathcal{S}| = 5890$, $d = 144$, $|S| = 62$, *file_size* $= 804$MB), *PEMS-SF* ($|\mathcal{S}| = 267$, $d = 963$, $|S| = 144$, *file_size* $= 420$MB) and *MotorImagery* ($|\mathcal{S}| = 278$, $d = 64$, $|S| \approx 1000$, *file_size* $= 537$MB) and the total execution time was respectively 1.5h, 1.8h and 7.5h using a single core. We remark that Bagnall et al.

(2018) did not report results of DTW on *EigenWorms* or *FaceDetection* since it did not complete in time for publication. Ruiz et al. (2021) also report missing values for HIVE-COTE, DTW$_D$, gRSF, RestNet, MR-SEQL and RISE due to out-of-memory issues or failing to finish within a copious 7 day limit. We conclude that MR-PETSC successfully scales to very large multivariate time series datasets.

7 Conclusions

PETSC leverages decades of research into pattern mining to discover long cohesive sequential patterns. We have shown that varying length sequential patterns with gaps are a new type of important feature in time series classification. Additionally, we studied soft support of patterns to deal with discretisation errors and direct mining of sequential patterns with the highest contrast. We find that MR-PETSC, that combines PETSC on different resolutions, is the best performing variant. On univariate datasets, MR-PETSC is more accurate than related interpretable dictionary-based methods, such as BOP and SAX-VSM and is on a par with MR-SEQL. More importantly, MR-PETSC achieves comparable performance to recent non-interpretable methods, such as BOSS, cBOSS, S-BOSS, ProximityForest and ResNet, but is narrowly outperformed by the current state-of-the-art methods InceptionTime, TS-CHIEF, ROCKET and HIVE-COTE, none of which are interpretable. On multivariate datasets MR-PETSC does even better, and achieves performance that is not significantly different than that of non-interpretable state-of-the-art such as ROCKET, CIF and HIVE-COTE.

In the design of MR-PETSC we use the same parameters for the SAX representation and pattern mining in its ensemble and combine only a small number of base PETSC learners in different resolutions (typically fewer than 10). In contrast to state-of-the-art methods our predictions are easy to interpret, enabling us to highlight important patterns for predicting time series or to inspect local discriminative patterns. We have shown how our model enables instance-based decision support by visualising attribution and presented a use-case where we reduce the number of patterns and enable human oversight of our intrinsically interpretable model before making any decision.

Moreover, MR-PETSC consists of efficient algorithms to discover patterns, create the embedding and train a linear model for making final predictions. On univariate time series the runtime performance of MR-PETSC is slightly slower than that of ROCKET by taking 2.7 h to complete on the 85 ‘bake off’ datasets, but still orders of magnitude faster than BOSS, TS-CHIEF, HIVE-COTE and about 5 times faster than cBOSS and MR-SEQL. MR-PETSC scales to large multivariate time series between 200 and 800MB where it completes training and predictions in hours using a single core while improving on accuracy compared to baseline distance-based methods and performing comparably to state-of-the-art in terms of accuracy.

Since PETSC builds upon pattern mining, it naturally handles a large variety of time series formats, including both univariate, multivariate and mixed-type time series of varying length and containing missing data or non-contiguous sampling. Additionally, the linear model and sequential patterns discovered in the time domain, allow for predictions that are fully interpretable and explainable, which is essential in many domains and a cause of growing concern in recent times. This is in contrast with existing methods with comparable accuracy and speed that rely on difficult to interpret techniques such as random forests, SFA-based representations, deep learning or large heterogeneous ensembles thereof. For future work we want to investigate applications, such as resource-constrained classification of time series from IoT devices that contain both continuous sensor values and discrete events.

References

Adamek T, O'Connor N (2003) Efficient contour-based shape representation and matching. In: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval, pp 138–143

Aggarwal CC, Jiawei H (2014) Frequent pattern mining. Springer, Berlin

Agrawal R, Srikant R et al (1994) Fast algorithms for mining association rules. In: Proceedings 20th international conference on very large databases, vol 1215, pp 487–499

Bagnall A, Lines J, Hills J, Bostrom A (2015) Time-series classification with cote: the collective of transformation-based ensembles. IEEE Trans Knowl Data Eng 27(9):2522–2535

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, Dau HA, Lines J, Flynn M, Large J, Bostrom A, Southam P, Keogh E (2018) The UEA multivariate time series classification archive. arXiv preprint arXiv:1811.00075

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(Feb):281–305

Bober M (2001) Mpeg-7 visual shape descriptors. IEEE Trans Circuits Syst Video Technol 11(6):716–719

Chen Y, Nascimento MA, Ooi BC, Tung AKH (2007) Spade: on shape-based pattern detection in streaming time series. In: 2007 IEEE 23rd international conference on data engineering. IEEE, pp 786–795

Cheng H, Yan X, Han J, Philip SY (2008) Direct discriminative pattern mining for effective classification. In: 2008 IEEE 24th international conference on data engineering. IEEE, pp 169–178

Cule B, Feremans L, Goethals B (2019) Efficiently mining cohesion-based patterns and rules in event sequences. Data Min Knowl Discov 33(4):1125–1182

Dau HA, Keogh E, Kamgar K, Yeh C-CM, Zhu Y, Gharghabi S, Ratanamahatana CA, Yanping HB, Begum N, Bagnall A, Mueen A, Batista G, Hexagon ML (2018) The UCR time series classification archive, October 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

Dempster A, Petitjean F,Webb GI (2020) Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. Data Min Knowl Discov 34(5):1454–1495

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

Fan W, Zhang K, Cheng H, Gao J, Yan X, Han J, Yu P, Verscheure O (2008) Direct mining of discriminative and essential frequent patterns via model-based search tree. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 230–238

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 P-A, Petitjean F (2020) Inceptiontime: finding alexnet for time series classification.Data Min Knowl Discov 34(6):1936–1962

Feremans L, Cule B, Goethals B (2018) Mining top-k quantile-based cohesive sequential patterns. In: Proceedings of the 2018 SIAM international conference on data mining. SIAM, pp 90–98

Fournier-Viger P, Gomariz A, Gueniche T, Mwamikazi E, Thomas R (2013) Tks: efficient mining of top-k sequential patterns. In: International conference on advanced data mining and applications. Springer, pp 109–120

Fradkin D, Mörchen F (2015) Mining sequential patterns for classification. Knowl Inf Syst 45(3):731–749

Han J, Pei J, Kamber M (2011) Data mining: concepts and techniques. Elsevier

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

Hsieh T-Y, Wang S, Sun Y, Honavar V (2021) Explainable multivariate time series classification: a deep neural network which learns to attend to important variables as well as time intervals. In: Proceedings of the 14th ACM international conference on web search and data mining, pp 607–615

Hyndman RJ, Athanasopoulos G (2018) Forecasting: principles and practice. OTexts

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

Kate RJ (2016) Using dynamic time warping distances as features for improved time series classification. Data Min Knowl Discov 30(2):283–312

Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001) Dimensionality reduction for fast similarity search in large time series databases. Knowl Inf Syst 3(3):263–286

Lam HT, Mörchen F, Fradkin D, Calders T (2014) Mining compressing sequential patterns. Stat Anal Data Min ASA Data Sci J 7(1):34–52

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

Laxman S, Sastry PS, Unnikrishnan KP (2007) A fast algorithm for finding frequent episodes in event streams. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, pp 410–419

Le Nguyen T, Gsponer S, Ifrim G (2017) Time series classification by sequence learning in all-subsequence space. In: 2017 IEEE 33rd international conference on data engineering (ICDE). IEEE, pp 947–958

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

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. ACM, 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, 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):1–35

Löning M, Bagnall A, Ganesh S, Kazakov V, Lines J, Király FJ (2019) sktime: A unified interface for machine learning with time series. In: Workshop on systems for ML at NeurIPS

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

Lundberg SM, Lee S-I (2017) A unified approach to interpreting model predictions. In: Proceedings of the 31st international conference on neural information processing systems, pp 4768–4777

Mannila H, Toivonen H, Inkeri Verkamo A (1997) Discovery of frequent episodes in event sequences. Data Min Knowl Discov 1(3):259–289

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

Middlehurst M, Large J, Bagnall A (2020) The canonical interval forest (CIF) classifier for time series classification. arXiv preprint arXiv:2008.09172

Middlehurst M, Large J, Cawley G, Bagnall A (2020) The temporal dictionary ensemble (TDE) classifier for time series classification. In: The European conference on machine learning and principles and practice of knowledge discovery in databases

Molnar C (2020) Interpretable machine learning. Lulu.com

Nguyen D, Luo W, Nguyen TD, Venkatesh S, Phung D (2018) Sqn2vec: learning sequence representation via sequential patterns with a gap constraint. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 569–584

Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu M-C (2004) Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Trans Knowl Data Eng 16(11):1424–1440

Pei J, Han J, Wang W (2007) Constraint-based sequential pattern mining: the pattern-growth methods. J Intell Inf Syst 28(2):133–160

Peng H, Long F, Ding C (2005) Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans Pattern Anal Mach Intell 27(8):1226–1238

Petitjean F, Forestier G, Webb GI, Nicholson AE, Chen Y, Keogh E (2014) Dynamic time warping averaging of time series allows faster and more accurate classification. In: 2014 IEEE international conference on data mining. IEEE, pp 470–479

Petitjean F, Li T, Tatti N, Webb GI (2016) Skopus: mining top-k sequential patterns under leverage. Data Min Knowl Discov 30(5):1086–1111

Rakthanmanon T, Campana B, Mueen A, Batista G, Westover B, Zhu Q, Zakaria J, Keogh E (2012) Searching and mining trillions of time series subsequences under dynamic time warping. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 262–270

Raza A, Kramer S (2020) Accelerating pattern-based time series classification: a linear time and space string mining approach. Knowl Inf Syst 62(3):1113–1141

Ribeiro MT, Singh S, Guestrin C (2016) "Why should i trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1135–1144

Ruiz AP, Flynn M, Large J, Middlehurst M, Bagnall A (2021) The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Discov 35(2):401–449

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 (2016) Scalable time series classification. Data Min Knowl Discov 30(5):1273–1298

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

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

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(3):742–775

Shokoohi-Yekta M, Bing H, Jin H, Wang J, Keogh E (2017) Generalizing DTW to the multi-dimensional case requires an adaptive approach. Data Min Knowl Discov 31(1):1–31

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). IEEE, pp 1578–1585

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Ye L, Keogh E (2011) Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min Knowl Discov 22(1–2):149–182

Yeh C-CM, Zhu Y, Ulanova L, Begum N, Ding Y, Dau HA, Silva DF, Mueen A, Keogh E (2016) Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: 2016 IEEE 16th international conference on data mining (ICDM). IEEE, pp 1317–1322

Zaki MJ, Meira W (2014) Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press, Cambridge

Zhou C, Cule B, Goethals B (2016) Pattern based sequence classification. IEEE Trans Knowl Data Eng 28(5):1285–1298. https://doi.org/10.1109/TKDE.2015.2510010

Zhu H, Wang P, He X, Li Y, Wang W, Shi B (2010) Efficient episode mining with minimal and non-overlapping occurrences. In: 2010 IEEE international conference on data mining. IEEE, pp 1211–1216

Zimmermann A (2014) Understanding episode mining techniques: benchmarking on diverse, realistic, artificial data. Intell Data Anal 18(5):761–791

Zou H, Hastie T (2005) Regularization and variable selection via the elastic net. J R Stat Soc Ser B (Stat. Methodol.) 67(2):301–320