IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 30 July 2024, accepted 16 September 2024, date of publication 20 September 2024, date of current version 2 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3464743

RESEARCH ARTICLE

Fast, Interpretable, and Deterministic Time Series Classification With a Bag-of-Receptive-Fields

FRANCESCO SPINNATO[1,2], RICCARDO GUIDOTTI[1,2], ANNA MONREALE[1], AND MIRCO NANNI[2]

[1]Department of Computer Science, University of Pisa, 56127 Pisa, Italy
[2]ISTI-CNR, 56124 Pisa, Italy

Corresponding author: Francesco Spinnato (francesco.spinnato@di.unipi.it)

This work was supported in part by the EU H2020 Program under the funding schemes ERC-2018-ADG G.A. 834756 "XAI: Science and Technology for the eXplanation of AI Decision Making," "SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics," G.A. under Grant 101120763 TANGO; in part by European Commission under the NextGeneration EU Program–National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza–PNRR) through the Project: SoBigData.it–Strengthening the Italian RI for Social Mining and Big Data Analytics (Prot. IR0000013–Avviso n. 3264 del 28/12/2021 and M4C2-Investimento 1.3, Partenariato Esteso)–Future Artificial Intelligence Research (FAIR) Spoke 1 "Human-Centered AI," under Grant PE00000013; and in part by Italian Project Fondo Italiano per la Scienza under Grant FIS00001966 MIMOSA.

ABSTRACT The current trend in the literature on Time Series Classification is to develop increasingly accurate algorithms by combining multiple models in ensemble hybrids, representing time series in complex and expressive feature spaces, and extracting features from different representations of the same time series. As a consequence of this focus on predictive performance, the best time series classifiers are black-box models, which are not understandable from a human standpoint. Even the approaches that are regarded as interpretable, such as shapelet-based ones, rely on randomization to maintain computational efficiency. This poses challenges for interpretability, as the explanation can change from run to run. Given these limitations, we propose the Bag-Of-Receptive-Field (BORF), a fast, interpretable, and deterministic time series transform. Building upon the classical Bag-Of-Patterns, we bridge the gap between convolutional operators and discretization, enhancing the Symbolic Aggregate Approximation (SAX) with dilation and stride, which can more effectively capture temporal patterns at multiple scales. We propose an algorithmic speedup that reduces the time complexity associated with SAX-based classifiers, allowing the extension of the Bag-Of-Patterns to the more flexible Bag-Of-Receptive-Fields, represented as a sparse multivariate tensor. The empirical results from testing our proposal on more than 150 univariate and multivariate classification datasets demonstrate good accuracy and great computational efficiency compared to traditional SAX-based methods and state-of-the-art time series classifiers, while providing easy-to-understand explanations.

INDEX TERMS Time series, classification, explainable AI, symbolic aggregate approximation.

I. INTRODUCTION

The availability of high-dimensional time series data has significantly expanded across critical domains such as finance, healthcare, and environmental science, capturing dynamic changes over time and presenting both opportunities and challenges for time series classification (TSC).

The current trend in this field, highlighted by the most recent "bake-off" survey paper [1], is to discover increasingly better-performing algorithms. This is achieved through various techniques, including combining multiple models in ensembles [2], [3], [4], representing time series in expressive feature spaces [5], [6], [7], and extracting features from different representations of the same time series signal, such as differencing and Fourier transform [8], as well as autoregressive coefficients [9].

As a consequence of this accuracy arms race, currently, the top-performing time series classifiers are black-box models. These models lack interpretability either due to their inherently complex structures, such as neural networks [10], or because,

The associate editor coordinating the review of this manuscript and approving it for publication was Jon Atli Benediktsson.

even when using relatively simple classifiers like tree ensembles or linear models, they leverage an uninterpretable feature space [3], [5]. Even the most well-known interpretable models for TSC, such as shapelet and interval-based algorithms [11], [12] rely on randomization to maintain reasonable computational efficiency, which poses challenges for interpretability as explanations can vary significantly between different runs [13].

A classical interpretable and deterministic approach for TSC is the so-called Bag-Of-Patterns (BOP) [14], analogous to the Bag-Of-Words approach [15] in text analysis, adapted for the time series domain. BOP utilizes the Symbolic Aggregate approXimation (SAX) [16] with a sliding window to transform time series data into symbolic subsequences, also called words. The counts of these words serve as input features for predicting categorical outputs. This window-wise discretization enhances data representation expressiveness [17], [18], [19], maintains interpretability, but imposes higher computational demands [20], particularly for longer sequences, thereby making it less feasible for very large datasets. In summary, to the best of our knowledge, the literature lacks a *fast*, *interpretable*, and above all, *deterministic* time series classifier capable of achieving good accuracy across a broad spectrum of tasks.

Given these challenges, our contributions are as follows. First, we formalize window-wise SAX, a technique widely adopted in experimental settings in the literature [17], [18], [19], but lacking systematic presentation and definition. We extend SAX with *dilation* and *stride* convolutional operators to extract *receptive fields*. Unlike standard subsequences, receptive fields capture the temporal evolution of a time series at multiple resolutions, emphasizing both local and global characteristics. Dilation enables skipping points within a subsequence, allowing for the detection of broader temporal patterns without additional computational overhead, while stride defines the overlap between consecutive receptive fields.

Second, we show that current Piecewise Aggregate Approximation (PAA) [21] and SAX-based classifiers exhibit quadratic worst-case time complexity with respect to the number of observations in a time series. We propose a deterministic speedup, enhancing the algorithm scalability over long time series.

Third, leveraging this speedup, we introduce an efficient and deterministic transform, named Bag-Of-Receptive-Fields (`BORF`), capable of converting both univariate and multivariate time series datasets into a human-understandable tabular representation. Unlike the original Bag-Of-Patterns approach [14], `BORF` represents subsequence counts as a sparse multivariate tensor rather than a dense matrix, offering greater flexibility for handling multivariate data.

Finally, we demonstrate that the integration of convolutional operators and multi-resolution symbolic patterns in `BORF` achieves competitive accuracy in TSC, while our optimization significantly enhances computational efficiency. We evaluate `BORF` across the complete UEA and UCR machine learning repositories, comprising over 150 datasets, benchmarking against SAX-based methods initially and subsequently against several state-of-the-art classifiers. To conclude, we assess the interpretability of our proposed approach, which is a combination of an interpretable feature space, i.e., the Bag-Of-Receptive-Fields, paired with a simple linear classifier.

The rest of this paper is structured as follows: Section II discusses related work and principal competitors, while Section III provides a background overview of our proposed methodology. The `BORF` approach is detailed in Section IV, followed by experimental results and analysis in Section V. Finally, conclusions are drawn in Section VI.

II. RELATED WORK

Time Series Classification (TSC) has been widely studied in recent years, with several so-called "bake-offs," [1], [20], [22] i.e., survey papers benchmarking the performance of the most famous time series classifier from different categories. Our proposal belongs to the dictionary-based family, which extracts features from time series by recording characteristics of discretized subpatterns [20]. These methods segment time series into subsequences, convert them into symbolic words, and create histograms of feature counts [14]. For this reason, they rely heavily on effective time series approximation methods such as the Symbolic Aggregate approXimation (SAX) [16] and Symbolic Fourier Approximation (SFA) [23]. SAX uses Piecewise Aggregate Approximation (PAA) [24] for segmentation and binning, while SFA focuses on spectral properties using Fourier coefficients [23].

SAX and SFA are often used in a window-based manner; that is, they are not applied to discretize the entire time series, but instead, a window is slid over the time series, converting each corresponding subsequence into a distinct word. This approach allows the extraction of scale-invariant local patterns in the sequence, given that each subsequence is normalized independently. The first method using this approach is Bag-of-Patterns (`BOP`) [25], an unsupervised transformation that uses SAX to generate discretized subsequences, counting their frequency. Other methods rely on supervised techniques. For example, `SAX-VSM` [17] introduces a tf-idf weighting of features to improve classification accuracy. `MR-SEQL` [18] uses the feature space of all subsequences in the training data to find useful features within SAX or SFA words. It employs greedy feature selection and a gradient bound to prune non-promising features. To improve time efficiency, `MR-SQM` [19] also concatenates multiple symbolic representations at multiple resolutions, but uses a Chi-squared bound and random sampling to filter subsequences extracted with SAX and SFA. There are several issues with these approaches. First, their performance is still subpar with respect to the state-of-the-art [1], [20]. Second, their implementation does not consider multivariate time series. Finally, as shown in Section IV, their worst-case time complexity, is quadratic with respect to the length of the time series, which negates the possibility of using them for very long time series. Better results in terms of accuracy are achieved by `WEASEL`, `WEASEL+MUSE`, and `WEASEL-V2` [8], [26], [27], which focus on the usage of SFA.

However, although informative for certain applications, SFA has higher computational complexity than SAX [19], [23], also losing temporal information, which makes it difficult to interpret from a human standpoint. The state-of-the-art for dictionary-based approaches is a black-box called HYDRA [2], which convolves the time series with kernels and organizes the resulting activation maps into groups, counting the number of best matches in terms of kernel activation with the time series.

The most famous interpretable TSC approaches are based on shapelets [13]. Shapelet-based approaches focus on finding subsequences that define a class independently of their position [28]. The discriminative features are distances, which can be used to convert the time series dataset into a tabular one through the so-called *shapelet transform* [29]. These approaches use different methods for extracting shapelets and building the subsequent classifier [30], [31]. Until recently, the Shapelet Transform (ST) [32] was the most accurate shapelet approach, with the drawback of high computational complexity resulting in rather long training and inference times. However, according to [1], the newer Random Dilated Shapelet Transform (RDST) achieves better performance at a faster runtime, with the drawback of more randomness in the approach, which can hinder its interpretability [13].

Interval-based classifiers, such as Time Series Forest (TSF) [33] divide time series into random intervals and extract summary statistics like mean, standard deviation, and slope, training a random forest on the resulting dataset. One of the most famous methods in this category is the Canonical Interval Forest (CIF) [7] and its extension (DRCIF), which use the CATCH22 feature set [34] on randomly selected intervals from different transformations of the original time series, training a tree ensemble on this representation. The interpretability of these approaches depends on the feature set used, and, in general, explanations are in the form of highlighting the most relevant features extracted from discriminatory intervals [12]. Similar to the best-performing shapelet classifiers, these approaches heavily rely on randomization, causing variations in the areas of interest used by the classifier across runs. In contrast, our proposal is entirely deterministic, generating simple, repeatable representations.

As an interesting note, standard deep learning approaches often fall behind in general time series classification benchmarks [1], given that they usually require ad-hoc fine-tuning, often dependent on the specific task and dataset. An exception to this is H-INCEPTIONTIME [35], an extension of INCEPTIONTIME [10], which is an ensemble of deep Convolutional Neural Networks (CNNs). According to [1], it achieves state-of-the-art performance. Convolution is also used in ROCKET [5] and its numerous variants [3], [6], which are considered some of the best-performing models in TSC. They generate many random convolutional kernels to transform time series into feature maps, which are pooled and used in a linear classifier. Even with large and complex datasets, ROCKET often achieves high accuracy and processing speed [1].

Finally, the absolute top-performing models in TSC are ensemble hybrids of the aforementioned methods, such as MULTIROCKET-HYDRA [2], HIVE-COTE-2 [4], TS-CHIEF [36], and RIST [9]. By their nature, these models are very complex and thus not interpretable from a human standpoint.

In summary, the current trend in the literature is to build increasingly complex and diverse feature spaces extracted from the original time series signal and several signal transformations, heavily relying on randomization to achieve accurate and fast classification. With BORF, we go against this trend by demonstrating that it is still possible to build an efficient, completely deterministic, easy-to-interpret representation grounded only in the time domain, while achieving good classification performance.

III. BACKGROUND

This section provides all the necessary concepts to understand our proposal. We begin by defining time series data.[1]

Definition 1 (Time Series Data): A time series dataset, $\mathcal{X} = \{X_1, \ldots, X_n\} \in \mathbb{R}^{n \times c \times m}$*, is a collection of n time series. A time series, X, is a collection of c signals (or channels),* $X = \{x_1, \ldots, x_c\} \in \mathbb{R}^{c \times m}$*. A signal,* $\boldsymbol{x}$*, is a sequence of m real-valued observations sampled regularly at equal time intervals,* $\boldsymbol{x} = [x_1, \ldots, x_m] \in \mathbb{R}^m$.

When $c = 1$, the time series is *univariate*, for $c > 1$ it is *multivariate*. Time series datasets can be used in a variety of tasks. This work focuses on supervised learning, particularly Time Series Classification (TSC), through a Bag-Of-Patterns (BOP) representation [14].

Definition 2 (TSC): Given a time series dataset, $\mathcal{X}$*, Time Series Classification is the task of training a model, f, to predict a categorical output, y, for each input time series, X, i.e.,* $f(\mathcal{X}) = [f(X_1), \ldots, f(X_n)] = \boldsymbol{y} \in \mathbb{N}^n$.

Specifically, BOP tackles classification by extracting symbolic patterns (or words) from the time series, converting them into a tabular representation, where the rows represent the time series, and the columns are the extracted patterns. Values in this matrix correspond to the count of appearances of each pattern in each time series. Formally:

Definition 3 (Bag-of-Patterns): Given a time series dataset $\mathcal{X}$ *and a set of p patterns, a Bag-of-Patterns is a dataset* $Z \in \mathbb{N}^{n \times p}$*, where* $z_{i,j}$ *is the number of appearances of pattern j in the time series i.*

Any classifier can use this dataset for TSC. A common way of building a Bag-Of-Patterns is by time series discretization, which essentially converts signals into a symbolic form by approximating both the time and value axes. One of the most famous discretization algorithms is the Symbolic Aggregate Approximation (SAX) [16]. SAX uses the Piecewise Aggregate Approximation (PAA) [24] to segment a time series signal into equal-sized segments and then compute the mean value for each segment. Formally, the average of the

[1] A summary of the notation is available in Appendix VI.

i-th segment is:

$$\mu_i = \frac{l}{m} \sum_{j=\frac{m}{l}(i-1)+1}^{\frac{m}{l}i} x_j, \tag{1}$$

where l is defined as the word (or pattern) length, and $i \in [1, l]$ [16]. Therefore, the transformed signal is $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_l] \in \mathbb{R}^l$. This vector is then standardized into $\boldsymbol{\mu}^*$, with each element computed as $\mu_i^* = zscore(\mu_i, \mu^x, \sigma^x) = (\mu_i - \mu^x)/\sigma^x$, where μ^x and σ^x are the mean and standard deviation of the full signal.

The last step in SAX is discretization through equal-frequency binning. Values in $\boldsymbol{\mu}^*$ are quantized using a set of breakpoints, $\mathbf{b} = [b_1, \ldots, b_{\alpha-1}]$, obtained on quantiles of the standard Gaussian distribution, which bin values in $\alpha \geq 2$ equiprobable symbols. The final SAX approximation is stored in $\tilde{\boldsymbol{\mu}}$, where each element is computed as:

$$\tilde{\mu}_i = bin(\mu_i^*, \alpha, \mathbf{b}) = \begin{cases} 0 & \text{if } \mu_i^* \leq b_1, \\ k-1 & \text{if } b_{k-1} < \mu_i^* \leq b_k, \\ \alpha - 1 & \text{if } \mu_i^* > b_{\alpha-1}. \end{cases} \tag{2}$$

Since each signal element is accessed once, the cost of applying SAX to a time series signal of length m is $O(m)$ [16].

However, several TSC algorithms in the literature [14], [17], [18], [19] iterate the aforementioned process using a sliding window, thus converting each time series signal into a collection of SAX words, which become the patterns used to build of a Bag-Of-Patterns.

IV. BAG-OF-RECEPTIVE-FIELDS

In this section, we present BORF, Bag-Of-Receptive-Fields, our proposal to efficiently extract an interpretable feature space that any machine learning classifier can use for TSC. BORF extends the notion of Bag-Of-Patterns, representing time series as a sparse multivariate tensor instead of a simple matrix. We introduce here a connection between convolutional operators and pattern extraction by generalizing the concept of a window to that of a *receptive field*. In convolutional neural networks theory, the receptive field refers to the spatial extent of input data that influences the activation of a particular neuron in the network [37]. Applied to our setting, a receptive field can be viewed as a generalized time series subsequence.

Definition 4 (Receptive Field): Given a signal $\boldsymbol{x}$, the i-th receptive field of length $w \geq 1$ and dilation $d \geq 1$, is an ordered sequence of values, $[x_i, x_{i+d}, \ldots, x_{i+d\cdot(w-1)}]$.

The dilation, d, plays an essential role in convolution, offering control over receptive field size and determining the spacing elements, enabling the capture of long-range dependencies and patterns. By introducing gaps between observations, dilation expands the receptive field and facilitates the incorporation of distant information. By iterating over the signal $\boldsymbol{x}$, v receptive fields can be extracted [37], where:

$$v = 1 + \left\lfloor \frac{(m - w - (d-1)(w-1))}{s} \right\rfloor. \tag{3}$$

In this formulation, the stride, $s \geq 1$, is essentially the number of elements skipped when iterating over the signal, i.e., the distance between consecutive receptive fields. A receptive field simplifies to a subsequence obtained via sliding window when $d = 1$ and $s = 1$. Without loss of generality, we use the terms receptive field, pattern, subsequence, and window interchangeably.

In the following, first, we connect the notion of receptive fields to SAX discretization, proposing a way to lower the time complexity of extracting SAX words (Section IV-A). Then, in Section IV-A, we introduce the Bag-Of-Receptive-Fields, extending Definition 3, by considering receptive fields instead of standard subsequences for extracting symbolic patterns, and by building this representation as a sparse 3D tensor, which is able to represent multivariate time series data. This extension of SAX through dilation and stride allows for building a representative feature space while maintaining interpretability. In fact, dilation and stride only change the shape of a subsequence, such as by skipping time series observations, but they maintain their semantics. This ensures that these generalized subsequences remain meaningful while capturing different resolutions of the original time series data. Finally, in Section IV-C to IV-E, we discuss the complexity of the overall approach in detail and its applicability to TSC, also from an interpretability standpoint.

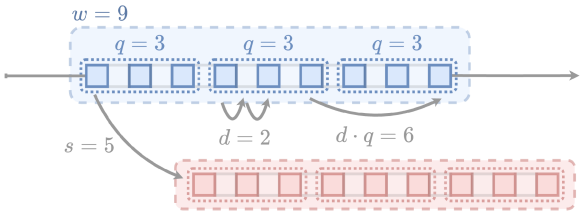


FIGURE 1. Two receptive fields (blue and red) with a length of $w = 9$, divided into segments of length $q = 3$. Dilation is the number of steps between consecutive observations within the receptive field. $d \cdot q$ is the segment hop, i.e., the number of steps between consecutive segments. The parameter $s = 5$ is the stride, i.e., the distance between consecutive receptive fields.

A. WSAX

Although window-wise PAA and SAX are used in many works [14], [17], [18], [19], to the best of our knowledge, the formulation of [16], reported in Equation (1), was never formally extended for such a case. Thus, as a first step, we formalize window-wise PAA (wPAA), also incorporating stride and dilation operators. The output of wPAA can be represented as a matrix $M \in \mathbb{R}^{v \times l}$, where the rows correspond to the receptive fields, and the columns to the averaged segments:

$$M = \begin{bmatrix} \mu_{1,1} & \cdots & \mu_{1,l} \\ \vdots & \ddots & \vdots \\ \mu_{v,1} & \cdots & \mu_{v,l} \end{bmatrix} \in \mathbb{R}^{v \times l}, \tag{4}$$

$$\text{with } \mu_{i,j} = \frac{1}{q}\sum_{k=1}^{q} x_{1+(i-1)\cdot s+(j-1)\cdot d\cdot q+(k-1)\cdot d}, \tag{5}$$

where $q = w/l$ is the segment size and with $i \in [1, v]$, and $j \in [1, l]$. In summary, a time series signal is divided into (possibly) overlapping windows, which are then divided into non-overlapping segments. wPAA takes the average of each of these segments. In Figure 1, we show an example of a receptive field of length $w = 9$, containing $l = 3$ segments of length $q = 3$. Dilation, $d = 2$, is the number of steps between consecutive observations in the receptive field. $d \cdot q$ is the segment hop, i.e., the number of steps between consecutive segments. Finally, the stride, $s = 5$, is the window hop, i.e., the number of steps between consecutive receptive fields.

a: NAIVE APPROACH

With this formulation, it follows that the complexity of computing M, by repeating Equation (5) for each cell, as is currently done in the literature, is $O(v \cdot l \cdot q) = O(v \cdot w)$, given that for each cell of M, we have to aggregate q elements. The highest number of receptive fields we can have is when $d = 1$ and $s = 1$ [37]. Therefore, the complexity simplifies to $O((m - w + 1) \cdot w)$. This is mostly fine while $w \ll m$, but, in the worst case, when $w = {}^{(m+1)}/_{2}$, the complexity is $O(m^2)$, i.e., quadratic in the number of points of the time series signal.[2] This reduces the possibility of using bigger window sizes, as the time series signal gets longer.

In the following, we propose a solution for this long-standing problem regarding SAX-based methods and show that it is possible to compute wPAA efficiently, independently from the window size, with a complexity that depends only on the word length. This opens the possibility of using longer window sizes, which can capture long-range patterns, thus increasing the expressivity of words in the Bag-Of-Receptive-Fields.

b: SPEED UP

In Algorithm 1, we report the pseudo-code of wSAX describing our proposed speed-up. Given $q = w/l$ with $w \equiv 0 \,(\mathrm{mod}\, l)$, i.e., with the window size divisible by the word length, to optimize computation, we can precompute the averages for all the segments in the time series in a moving fashion (line 1), storing the result for later. Formally:

$$\boldsymbol{\mu}^{seg} = ma(x, q, d) = [\mu_1^{seg}, \ldots, \mu_{m+d-d\cdot q}^{seg}] \tag{6}$$

$$\text{with } \mu_i^{seg} = \frac{1}{q}\sum_{j=1}^{q} x_{i+(j-1)\cdot d}, \tag{7}$$

where μ_i is the mean of a single segment in x, with $i \in [1, m + d - d \cdot q]$, where $m + d - d \cdot q$ is the total number of segments. The main advantage of this formalization is that we do not have to independently calculate each μ_i^{seg} from Equation (7). Instead, we can employ moving algorithms, which compute $\boldsymbol{\mu}^{seg}$ in $O(m)$ [38]. In this context, a simple moving average

[2] A more detailed proof is reported in Appendix VI.

Algorithm 1 wSAX

Input: x - time series signal, w - window size, q - segment size, l - word length, α - alphabet size, $\mathbf{b}$ - breakpoints, d - dilation, s - stride
Output: $\tilde{M}$ - A matrix of SAX words

1 $\mu^{seg} = ma(x, q, d)$; // Moving avg for segments
2 $\mu^{win} = ma(x, w, d)$; // Moving avg for windows
3 $\sigma^{win} = mstd(x, w, d)$; // Moving std for windows
4 **init** $\tilde{M} \in \mathbb{N}^{v \times l}$; // Initialize empty matrix
5 **for** $1 \le i \le v$ **do** // For each window index
6 **for** $1 \le j \le l$ **do** // For each segment index
7 $\mu_{i,j} = \mu^{seg}_{1+(i-1)\cdot s+(j-1)\cdot d\cdot q}$; // wPAA
8 $\mu^*_{i,j} = zscore(\mu_{i,j}, \mu^{win}_{i\cdot s}, \sigma^{win}_{i\cdot s})$; // Standardize
9 $\tilde{\mu}_{i,j} = bin(\mu^*_{i,j}, \alpha, \boldsymbol{b})$; // Discretize
10 **return** $\tilde{M}$

algorithm suffices, where:

$$\mu_i^{seg} = \mu_{i-d}^{seg} + \frac{1}{q}(x_{i-d+d\cdot q} - x_{i-d}). \tag{8}$$

The vector μ^{seg} contains all the information necessary for wPAA. Instead of computing Equation (5) for each cell of M, we can iterate over i, j as in Algorithm 1 (lines 4-7), filling each entry of M as follows:

$$\mu_{i,j} = \mu^{seg}_{1+(i-1)\cdot s+(j-1)\cdot d\cdot q} \tag{9}$$

In Figure 2, we report this step for the two receptive fields depicted in Figure 1. Performing wPAA basically simplifies to iterating over each cell of M, collecting the pre-computed moving averages, based on Equation (9). For example, the blue receptive field is segmented into $[\mu_{1,1}, \mu_{1,2}, \mu_{1,3}] = [\mu_1^{seg}, \mu_7^{seg}, \mu_{13}^{seg}]$.

c: DISCRETIZATION

To compute wSAX, each segment average needs to be standardized based on the window to which it belongs. Therefore, we compute the mean of each window, $\boldsymbol{\mu}^{win} = ma(x, w, d)$ similarly to Equation (6), and their standard deviation (Alg. 1, lines 2-3):

$$\boldsymbol{\sigma}^{win} = mstd(x, w, d) = [\sigma_1^{win}, \ldots, \sigma_v^{win}] \tag{10}$$

$$\text{with } \sigma_i^{win} = \sqrt{\frac{1}{w}\sum_{j=1}^{w}(x_{i+(j-1)\cdot d} - \mu_i^{win})^2}, \tag{11}$$

which again, can be computed in $O(m)$, using a moving standard deviation algorithm such as [39]. Then, standardization

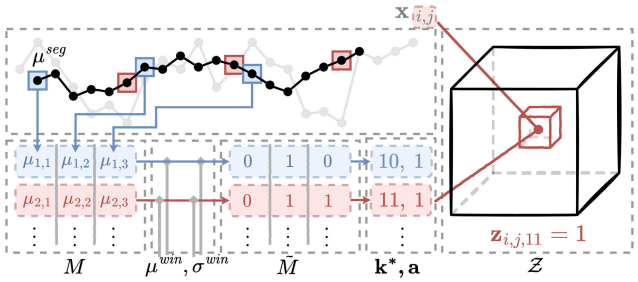


FIGURE 2. A simplified schema of Algorithm 2 for the two receptive fields of Figure 1, extracted from signal $x_{i,j}$. First, the moving average μ^{seg} is computed. Values in μ^{seg} are used to fill M as in Equation (9). The two receptive fields are then normalized, binned into the SAX words [0, 1, 0] and [0, 1, 1], and hashed into the integers and respective counts 10, 1 and 11, 1. This allows the update of $\mathcal{Z}$. E.g., for the red word, $z_{i,j,11} = 1$.

is applied to each element of M (Alg. 1, line 8):

$$\mu^*_{i,j} = zscore(\mu_{i,j}, \mu^{win}_{i\cdot s}, \sigma^{win}_{i\cdot s}) = \frac{\mu_{i,j} - \mu^{win}_{i\cdot s}}{\sigma^{win}_{i\cdot s}}. \tag{12}$$

Finally, discretization is performed as described in Equation (2): $\tilde{\mu}_{i,j} = bin(\mu^*_{i,j}, \alpha, b)$ (Alg. 1, line 9). The transformed segments, $\tilde{\mu}_{i,j}$, are collected in $\tilde{M} \in \mathbb{N}^{v\times l}$, which is the wSAX conversion of signal x. In Figure 2, the blue and red receptive fields are converted into the word vectors [0, 1, 0] and [0, 1, 1].

B. BAG-OF-RECEPTIVE-FIELDS

Once the SAX words are extracted, we store them in our proposed Bag-Of-Receptive-Fields, as described in Algorithm 2.

Definition 5 (Bag-Of-Receptive-Fields): Given a time series dataset $\mathcal{X} \in \mathbb{R}^{n\times c\times m}$ and a set of p patterns, a Bag-Of-Receptive-Fields is a tensor $\mathcal{Z} \in \mathbb{N}^{n\times c\times p}$, where $z_{i,j,k}$ is the number of appearances of SAX word k in the signal j of time series i.

In summary, to build $\mathcal{Z}$, we need to repeat wSAX for each time series signal in our dataset (Alg. 2, lines 2-3), storing the extracted words and respective counts. We view this problem as a way of hashing the extracted word vectors into integers. We propose an easy solution, which is to convert each row in $\tilde{M}$ into an integer obtained through concatenating the SAX symbols:

$$k_i = \sum_{j=1}^{l} \tilde{\mu}_{i,j} \cdot \alpha^{l-j}, \tag{13}$$

where α is the alphabe size. Thus, $k = [k_1, \ldots, k_v]$ (Alg. 2, line 5). Given $\mathbf{k}$, through a hashmap, we group-by and count the appearance frequencies of each SAX word, i.e., $k^*, a = unique(k)$, where k^* contains the unique hashed words, and $\mathbf{a}$ contains their count (Alg. 2, line 6). Basically, SAX words become integers, used directly as the feature indices in $\mathcal{Z}$. This operation is performed for each time series i, for each signal j. Therefore, $z_{i,j,k} = a$, where a represents how many times the SAX word k was found in $x_{i,j}$. In our example in Figure 2, the discretized red receptive field, [0, 1, 1], is converted to the integer 11 and appears only 1 time in our series signal. Its value and count are therefore stored in $\mathcal{Z}$, i.e., $z_{i,j,11} = 1$.

Algorithm 2 BORF

Input: $\mathcal{X}$ - time series dataset, w - window size, q - segment size, l - word length, α - alphabet size, $\mathbf{b}$ - breakpoints, d - dilation, s - stride
Output: $\mathcal{Z}$ - Bag-Of-Receptive-Fields

1 **init** $\mathcal{Z} \in \mathbb{N}^{n\times c\times p}$; `// Initialize empty tensor`
2 **for** $1 \le i \le n$ **do** `// For each time series index`
3 **for** $1 \le j \le c$ **do** `// For each signal index`
4 $\tilde{M} = \text{wSAX}(x_{i,j}, w, q, l, \alpha, \boldsymbol{b}, d, s)$; `// wSAX`
5 $\boldsymbol{k} = hash(\tilde{M})$; `// Hash SAX words`
6 $\boldsymbol{k}^*, \boldsymbol{a} = unique(\boldsymbol{k})$; `// Group-by and count`
7 **for** $(k, a) \in \{(\boldsymbol{k}^*, \boldsymbol{a})\}$ **do** `// For each word, count`
8 $z_{i,j,k} = a$; `// Store the word and count`
9 **return** $\mathcal{Z}$

C. COMPLEXITY

a: TIME

The time complexity of computing the averages for the segments and window means is $O(m)$ and can be calculated using moving average algorithms [38]. The same applies to the moving standard deviation [39]. Once these quantities are computed, they need to be accessed a number of times equal to the dimension of M. Therefore, the complexity is $O(v \cdot l) = O((m - w + 1) \cdot l)$. The worst case would be when $l \in O(w)$ and $w \in O(m)$, e.g., $w = m/2$, and $l = w/2$, thus leading to an hypothetical overall complexity of $O(m^2)$. However, it is important to stress that this becomes a pathological parameter choice as the signal length increases. In fact, in the literature of SAX-based predictors, w is increased as m grows, whereas the word length, l, is always kept very low to avoid the combinatorial explosion in possible words [18], [19]. For this reason, compared to the naive one, the advantage of our implementation is that, within a reasonable parameter choice, the complexity does not increase with larger window sizes. Hence, the final time complexity of repeating this process for each time series signal in our dataset is $O(n \cdot c \cdot m \cdot l)$, while the naive approach is $O(n \cdot c \cdot m^2)$. To further prove this point, in Section V, we show that a word length of 8 is sufficient to achieve good classification performance, comparable to the state-of-the-art, and empirically benchmark the scalability of the approach.

b: SPACE

Space complexity is tricky when dealing with SAX patterns, as their number grows fast as the word length and alphabet size increase. The complexity is dominated either by the total number of theoretically possible words, or the number of receptive fields, and is $O(c \cdot \min(\alpha^l, n \cdot m))$. This seems to require a lot of memory. However, in our case, $\mathcal{Z}$ is mostly sparse, given that the majority of entries are 0, i.e., only a few patterns appear in each time series. Choosing a sparse coordinate format representation, where each entry in $\mathcal{Z}$ is stored as a 4-valued tuple (i, j, k, a), the total complexity is the number of non-zero elements in the sparse representation, which can be represented as a function of the dataset size $(n \cdot c \cdot m)$, but also depend on the intrinsic characteristics of the data. We investigate this relationship in Section V, and provide a heuristic for estimating the number of non-zero elements, given the total dataset size.

D. TIME SERIES CLASSIFICATION

BORF is an unsupervised method, however we benchmark it on Time Series Classification to assess its performance in extracting meaningful features for this task. Given that classification algorithms usually have a 2D input, the Bag-Of-Receptive field can be reshaped naively by concatenating the second and third dimension, leaving the first unchanged, going from a tensor $\mathcal{Z} \in \mathbb{N}^{n \times c \times p}$ to a standard matrix dataset $Z \in \mathbb{N}^{n \times (c \times p)}$. Having previous knowledge of the dataset, an alternative could be to sum the counts of patterns extracted from different groups of time series signals. In general, the flexibility in aggregation could be of great use when domain experts know which signal to aggregate and which signal to keep separate based on their semantics. We explore this possibility in a case study in Section V.

As highlighted in many recent works [5], [18], [19], building multiple representations of the same input data is extremely important for classification performance. In our setting, this is equivalent to building many Bag-Of-Receptive-Fields, each generated with different parameters (e.g., dilation, window size, word length), and then stacking each resulting Z_i horizontally: $Z = [Z_1|Z_2|Z_3|\dots]$. Any classification model that takes sparse matrices as input can then use the resulting dataset for training and inference.

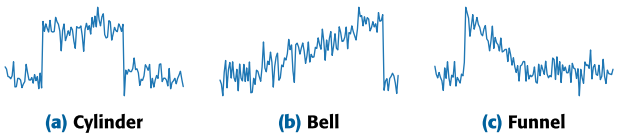


FIGURE 3. Example instances from the three classes of the "Cylinder-Bell-Funnel" dataset.

E. EXPLAINABILITY

The interesting characteristic of a Bag-Of-Receptive-Fields is that, contrary to many state-of-the-art time series classifiers, the meaning of a feature value is very simple to interpret from a human standpoint. Given $z_{i,j,k} = a$, this simply translates into "The pattern k is contained a times into signal j of time series i". Therefore, paired with an interpretable classifier, this feature space can provide easy-to-understand explanations. Interpretability is achieved through both an easily comprehensible feature representation and a simple classifier that is utilized to perform predictions. For example, using a decision tree, one could extract the decision paths, i.e., rules with conditions based on the number of appearances of discriminative patterns. However, in this work, we focus on simple linear models, which are also used in many state-of-the-art competitors as they achieve very good performance.

In particular, we pair a ridge classifier with SHAP [40] to highlight the contribution, $\phi_{i,j,k}$, of each pattern count, $z_{i,j,k} = a$, towards the classification. $\phi_{i,j,k}$ is called SHAP value, and if positive indicates a contribution to the positive class, a negative SHAP value contributes toward the negative class (or, in general, all the other classes if the dataset is multiclass), while a SHAP value close to zero indicates that the feature value is irrelevant. Given this semantic, in a Bag-Of-Receptive-Field, $\phi_{i,j,k}$ is the contribution of $z_{i,j,k} = a$ toward a given class y, which translates to "The fact that pattern k of signal j of time series i is contained a times, pushes the classification toward class y". This allows for building both *global* and *local* explanations, which can give several insights into the classifier behavior. In the following subsections, we show examples of these kinds of explanations on the classical "*Cylider-Bell-Funnel*" (CBF) dataset [41], containing time series having three distinct shapes, as shown in Figure 3.

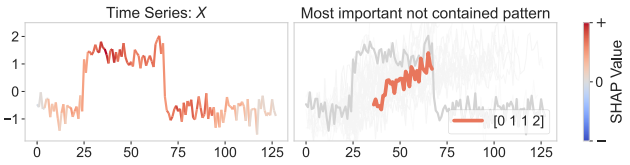


FIGURE 4. Local explanation for a *Cylinder* instance of the CBF dataset. On the left, the original time series colored based on the saliency map, highlighting the most relevant observations for the classification. On the right, the medoid of the most important pattern that is not contained in the time series. The grey area represents all the possible alignments of the pattern '0,1,1,2' found in the dataset.

a: LOCAL EXPLANATION

The local explanation can be built by treating the feature importance of contained and not-contained patterns in different ways. Given a time series, $X \in \mathbb{R}^{c \times m}$, its BORF conversion, $Z \in \mathbb{N}^{c \times p}$, and a feature importance matrix, $\Phi \in \mathbb{R}^{c \times p}$, obtained through some local explainer, such as SHAP, we propose a way to build a saliency map, $\Psi \in \mathbb{R}^{c \times m}$. A single feature importance, $\phi_{j,k}$, for a single SAX word, $z_{j,k}$, can be mapped back to the original timesteps only if $z_{j,k}$ is contained in the time series, i.e., $z_{j,k} > 0$. If that is the case, we can retrieve the set $T_{j,k} = \{(j, i_1), (j, i_2), \dots\}$, containing

all the unique alignment indices[3] for $z_{j,k}$ on X, and in particular, on signal, x_j. The saliency matrix is then defined as:

$$\Psi = \begin{bmatrix} \psi_{1,1} & \cdots & \psi_{1,m} \\ \vdots & \ddots & \vdots \\ \psi_{c,1} & \cdots & \psi_{c,m} \end{bmatrix} \in \mathbb{R}^{c \times m}, \tag{14}$$

$$\text{with } \psi_{j,i} = \sum_{k=1}^{p} \phi_{j,k} \cdot \mathbb{1}[(j,i) \in T_{j,k}]. \tag{15}$$

Basically, the importance of each pattern is "spread" on the original indices from which it was extracted. For this reason, the saliency map has to be scaled, in order to sum to the original contribution, and is therefore multiplied by a scaling factor:

$$\Psi^* = \Psi \cdot \frac{\sum_{j=1}^{c} \sum_{k=1}^{p} \phi_{j,k} \cdot \mathbb{1}[z_{j,k} > 0]}{\sum_{j=1}^{c} \sum_{i=1}^{m} \psi_{j,i}} \tag{16}$$

In practice, the sum of the saliency values is scaled to equal the sum of the feature importances of contained patterns.

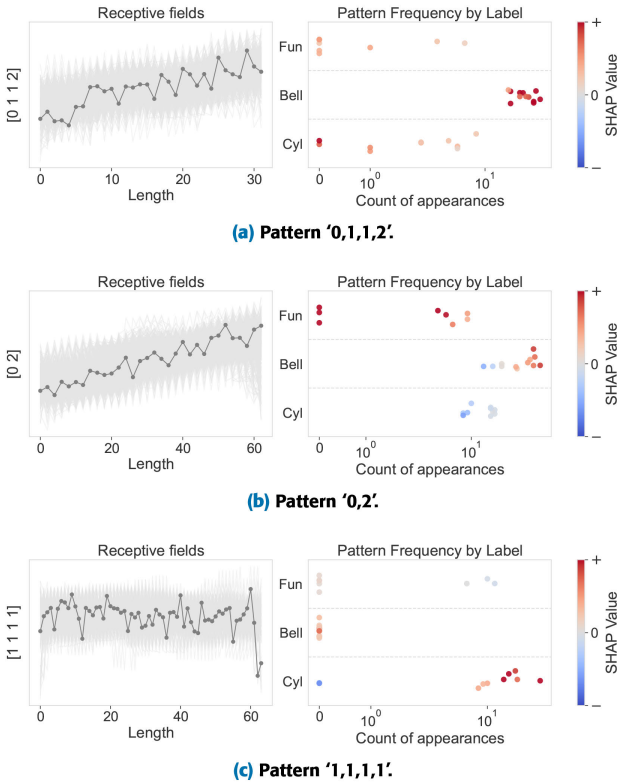
An example of these saliency maps can be viewed in Figure 4 (left), plotted on a *Cylinder* time series. The more important the observation, the more intense the color will be. In this case, we can easily see that the classifier is "looking" mostly at the central part of the time series (from the 30*th* to the 90*th* timestep). These are the most relevant parts of the time series, as the extremes usually contain noise, and thus tend towards a grey color. On the other hand, when $z_{j,k} = 0$, i.e., when pattern $z_{j,k}$ is not contained in the time series, it cannot be mapped back to the time series. Therefore, we show its general shape, by retrieving the indices $T_{j,k}$, and its respective observations, $X_{T_{j,k}}$, from several time series in the dataset. In practice, we use a background dataset of sorts to map SAX words onto the time series to see the real shape that these symbolic words assume in our dataset. In Figure 4, we show the alignments of pattern '0,1,1,2' with a light gray color, and the medoid of these alignments is colored based on its SHAP value. This pattern closely resembles the increasing part of a *Bell* instance, and its absence is a strong indicator for the model towards the *Cylinder* class.

In summary, the local explanation comprises a saliency map explaining the contained patterns, and a feature importance vector, explaining the contribution of not-contained patterns.

b: GLOBAL EXPLANATION

The local explanation highlights the importance of patterns for single time series. However, it does not tell anything about the general relevance of a given pattern. From the local explanation, a natural question could arise, i.e., if the pattern '0,1,1,2' is important in general or only for the given time series. In general, the explanation that arises from the Bag-Of-Receptive-Fields is the importance of frequency counts of the extracted patterns. To find the most important patterns in the dataset, we compute the average rank in terms of absolute SHAP values, i.e., we highlight the patterns that are

[3]We take the unique indices, as $z_{j,k}$ can have overlapping alignments on x_j.



(a) Pattern '0,1,1,2'.

(b) Pattern '0,2'.

(c) Pattern '1,1,1,1'.

FIGURE 5. Global explanation for three of the most important patterns for the CBF dataset. On the left, in dark gray, the medoid of all alignments of the pattern in the CBF dataset. On the right, the count of appearances of each pattern in each time series in the dataset, divided by label. The color represents the importance of the pattern count in terms of SHAP values.

the most important for most time series. We summarize this information for CBF in Figure 5 for three of the most relevant patterns in this dataset: '0,1,1,2', '0,2', and '1,1,1,1'. On the left, we depict their medoid in dark gray, while in light gray, we depict all the possible mappings of the same word in the given time series dataset. This part of the plot can help give an idea of what the classifier considers the most important shapes in our data. In our case, the first two patterns are increasing, resembling a *Bell* instance of our dataset, while the latter is flatter, more similar to a *Cylinder* instance.

In the right scatterplots of Figure 5, each point represents a time series. Its x-axis position represents how many times a given pattern is contained in that time series, the y-axis position is the label of that time series, while the color is the SHAP value: red if it contributes toward its own class, gray if it is irrelevant, and blue if it contributes toward other classes. With this plot, we can easily see in which kind of time series the pattern is contained or not, and how many times, while also assessing the importance of its presence/absence.

Let's take, for example, pattern '0,1,1,2' from Figure 5. This pattern is contained in *Bell* time series many times (more than 10), as shown by the cluster of red points on the right. This translates to: "*In general, the fact that pattern '0,1,1,2' is contained many times in the time series pushes*

the prediction toward Bell.''. This makes a lot of sense as the pattern resembles the slowly increasing shape of a *Bell*. '0,1,1,2' is sometimes also contained in *Cylinder* and *Funnel* instances, but the importance is lower (more toward the gray color), suggesting that if the pattern is contained only a few times, it is not really discriminative toward the class. Finally, there are a few bright red instances of the class *Cylinder* in which this pattern is contained zero times, which translates to: ''*In general, the fact that pattern '0,1,1,2' is not contained in the time series pushes the prediction toward Cylinder.*''. The instance depicted in the local explanation is one of them.

The blue points in Figure 5b are also very interesting, as they indicate that '0,2' has a somewhat peculiar behavior in some instances. In fact, similarly to '0,1,1,2', the '0,2' pattern is normally contained many times (more than 10) in *Bell* instances. However, if '0,2' is contained around 10 times, this actually contributes toward the class *Funnel*. Therefore, the blue point instances indicate that if '0,2' is contained around 10 times in *Bell* or *Cylinder* instances, this pushes the prediction toward the class *Funnel*, i.e., the model recognized that this pattern frequency is atypical in *Bells* and *Cylinders*.

Finally, pattern '1,1,1,1' in Figure 5c represents a flat part of the time series, that usually appear around 10 times in *Cylinder* instances. The blue point represents an atypical *Cylinder* instance, where '1,1,1,1' is not contained given that, in general, the model considers the absence of '1,1,1,1' as a strong indicator toward *Bells* or *Funnels*.

V. EXPERIMENTS

In this section, we benchmark BORF in terms of classification performance and runtime against state-of-the-art baselines on the UEA/UCR repositories [45], [46]. We provide a comparison against different families of classifiers, also studying in which kind of datasets BORF perform best. Further, we assess scalability and provide an ablation study to understand which parameters affect BORF more. Finally, we show the flexibility of BORF for multivariate time series on a case study, and address interpretability with qualitative examples.

a: DATASETS

The UCR and UEA TSC repositories comprise 158 datasets, 128 univariate and 30 multivariate. Some datasets contain time series with variable observations, and some contain missing values. We apply missing value imputation by linear interpolation and last value padding. Further, we concatenate all signals in a single axis for algorithms that do not support multivariate time series. For all tests, we adopt the default training and test splits.

b: BORF HEURISTIC

Following the current trend in TSC [1], [20], [22], instead of fine-tuning BORF to maximize performance for specific applications, we propose a model that achieves good results in a wide range of problems. Since several hyperparameters exist, we provide a heuristic that computes a vector of window sizes, dilations, and word lengths, as follows: $\boldsymbol{w} = [2^2, 2^3, \ldots, 2^{\lfloor \log_2 m \rfloor}]$, $\boldsymbol{d} = [2^0, 2^1, \ldots, 2^{\lfloor \log_2 (\log_2 m) \rfloor}]$, $\boldsymbol{l} = [2, 4, 8]$, $\alpha = 3$, $s = 1$. Each valid combination of these parameters is used as a configuration of BORF, resulting in a multi-resolution representation of the time series. As commonly done in the literature [18], to avoid blowing up noise with standardization, we set a minimum standard deviation ratio $\theta = \sigma^{win}/\sigma^{x}$, between the standard deviation of a window, σ^{win}, and the standard deviation of full signal, σ^{x}. If $\theta \leq 0.15$, we consider the window constant (all zeros). Tuning is performed on the same 42 *development dataset* used to tune ROCKET [5], and is discussed in the ablation study. After the transformation, we apply an inverse hyperbolic sine function element-wise and use an interpretable ridge classifier as a predictive model.[4]

TABLE 1. List of competitor classifiers, divided by classifier family, with information about feature space and classifier interpretability, and stochasticity. Models that have an interpretable feature space and classifier, and are also deterministic are highlighted in gray.

Model		interpretable features	interpretable classifier	deterministic
Dictionary-Based				
BORF	(ours)	✓	✓	✓
BOP	[14]	✓	✓	✓
HYDRA	[2]	✗	✓	✗
MRSEQL	[19]	✓*	✓	✓
MRSQM	[18]	✓*	✓	✗
REDCOM	[42]	✗	✓	✗
SAXVSM	[17]	✓	✓	✓
W+MUSE	[27]	✗	✓	✗
Interval-based				
DRCIF	[4]	✗	✓	✗
RSTSF	[12]	✓	✓	✗
TSF	[33]	✓	✓	✗
Shapelet-based				
RDST	[11]	✓	✓	✗
ST	[29]	✓	✓	✗
Kernel-based				
MINIROCKET	[6]	✗	✓	✗
ROCKET	[5]	✗	✓	✗
Neural Networks				
CNN		✗	✗	✗
IT	[10]	✗	✗	✗
MCDCNN	[43]	✗	✗	✗
TAPNET	[44]	✗	✗	✗
Distance-based				
KNN-DTW		✓	✓	✓
Hybrid				
MR-HYDRA	[2]	✗	✗	✗

* Without SFA features.

c: COMPETITORS

We compare BORF against several state-of-the-art approaches presented in Section II. A summary of these baselines is reported in Table 1. First, we assess the performance of BORF against SAX-based methods, namely BOP [14],

[4]Code is available at: https://github.com/fspinna/borf

SAX-VSM [17], MR-SQM [18], and MR-SEQL [19]. Subsequently, we evaluate the performance against many different competitors. As a baseline, we use KNN with DTW as the distance metric and a Sakoe-Chiba band of 0.1 (KNN-DTW). For dictionary-based approaches, we benchmark WEASEL+MUSE (W+MUSE) [27], RedComets (REDCOM) [42], and HYDRA [2]. For interval-based models, we compare Time Series Forest (TSF) [33], Random Supervised Time Series Forest (RSTSF) [12], and the Diverse Representation Canonical Interval Forest (DRCIF) [4]. For shapelet-based classifiers, we assess the Shapelet Transform (ST) [29] and its random dilated version (RDST) [11]. For deep learning-based models, we compare with a standard 1D Convolutional Neural Network (CNN), TAPNET [44], a multi-channels deep convolutional neural networks (MCDCNN) [43], and INCEPTIONTIME (IT) [10]. Additionally, we evaluate ROCKET [5], MINIROCKET [6], and the current absolute state-of-the-art in terms of accuracy/speed tradeoff, Multirocket-Hydra (MR-HYDRA) [2].

All models are trained with the default library hyperparameters or values proposed in the respective papers.[5] Each model is allowed one day (1440 minutes) for training and inference on each dataset and is allocated 32 cores and 64 GB of memory.[6] A run is considered failed if it exceeds the maximum time or crashes due to out-of-memory errors, and is assigned the lowest rank.

d: CLASSIFICATION

In this section, we compare the predictive and runtime performance of BORF against competitor approaches. We report predictive performance in terms of accuracy, as it is the most commonly used metric, and runtime performance as the sum of training and inference runtime (in seconds). Performance in terms of F1-measure is reported in Appendix VI, as it produces results very similar to accuracy. Appendix VI also presents the full accuracy table on all datasets for the top 13 best-performing baselines.

To assess the statistical significance of these results, in Figure 6, we first report the Critical Difference (CD) plot comparing the pairwise rankings of all benchmarked methods. Two methods are considered tied if the null hypothesis that their performance is the same cannot be rejected using the Nemenyi test at α=0.05. The best models are reported on the right. In terms of accuracy (Figure 6a), BORF has the 9th average rank and is statistically tied to the top-performing models, such as ROCKET and MINIROCKET, with the only statistically better methods being RDST and MR-HYDRA. Regarding runtime, BORF has the 2nd best rank, statistically tied with HYDRA. BORF and HYDRA are statistically faster than all other approaches, with the closest ones being MR-HYDRA and MRSQM. This plot shows that our approach achieves good performance that is close to,

[5]We used the Python libraries pyts [47] for BOP and SAX-VSM, and aeon [48] and sktime [49] for all the other methods.

[6]System: Lenovo SD650 nodes, with Intel Xeon Platinum 8268 CPUs.

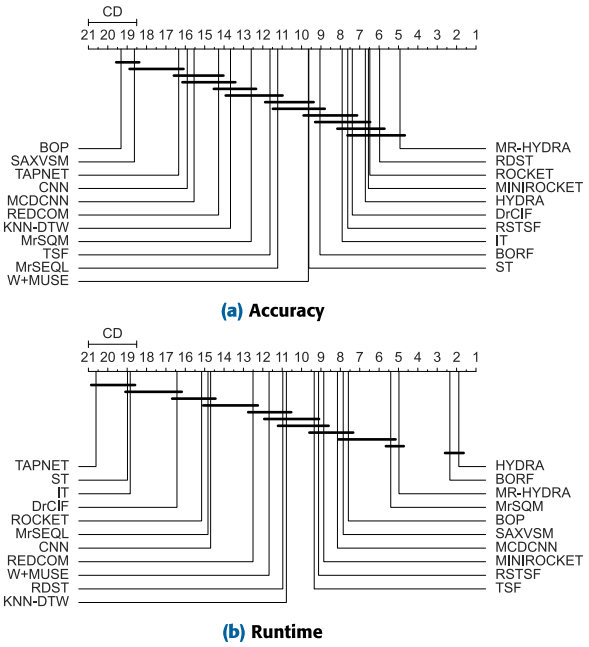


FIGURE 6. CD Plots for all benchmarked methods. Best models to the right.

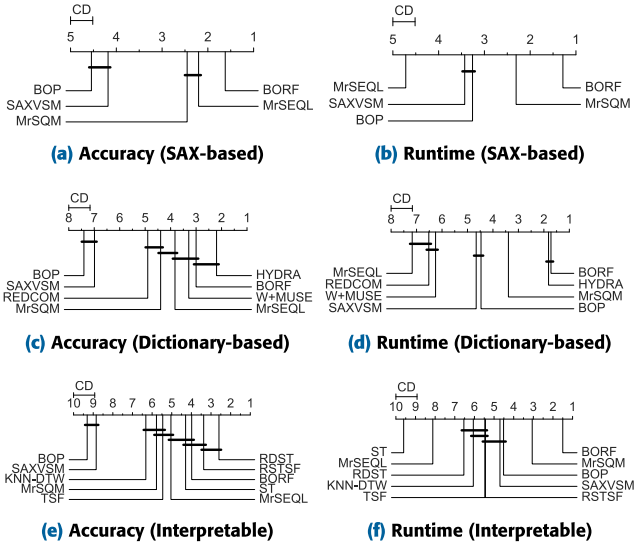


FIGURE 7. CD Plots for SAX-based, dictionary-based, and interpretable methods. Best models to the right.

even if slightly below, the general state-of-the-art in terms of accuracy, and state-of-the-art speed. As state-of-the-art accuracy against all methods is not the claim of this work, but rather competitive performance against SAX-based, dictionary-based, and interpretable methods, we also report the CD plots for subsets of models belonging to different classifier families.

Figure 7a and 7b show a comparison against all SAX-based methods. For this subset, BORF is the clear winner, both in terms of accuracy and runtime, surpassing MRSEQL in terms of accuracy, while being even faster than MRSQM. In Figure 7c and 7d, we compare against dictionary-based methods. HYDRA

is the most accurate model, but it is statistically tied with BORF. W+MUSE and MRSEQL are close to BORF in terms of accuracy rank, but they are significantly slower. Finally, in Figure 7e and 7f, we compare BORF against interpretable approaches, i.e., approaches that are widely regarded as interpretable or discuss interpretability in their respective papers. For this subgroup, BORF is still the fastest approach, while achieving a 3rd rank in accuracy, behind RDST, and statistically tied to RSTSF and ST. Note that RDST and RSTSF, while considered interpretable, are not deterministic. Therefore, while their predictions can be interpreted, they can also change from run to run given the same exact hyperparameters. On the contrary, BORF is interpretable and also stable across runs. These experiments on subsets of classifier families show that BORF is always first in terms of runtime, while being the most accurate SAX-based approach and among the best dictionary-based and interpretable models.

TABLE 2. Comparison Matrix between BORF and competitor approaches. Models are sorted by median difference in accuracy with BORF. Wins, ties, and losses are to be read "against BORF". Best models on top. Grey rows are statistically tied with BORF using the Wilcoxon signed-rank test.

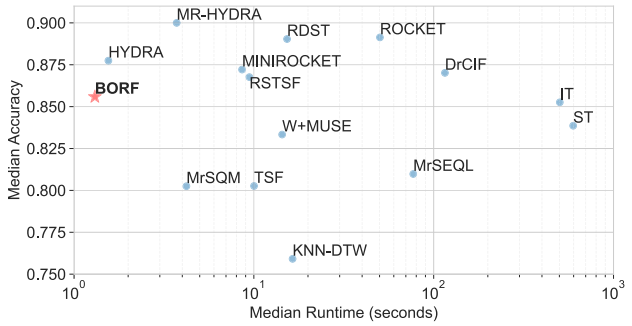
	Δaccuracy					
	median	*mean*	*win*	*tie*	*loss*	*p-val*
MR-HYDRA	+0.02	+0.04	109	22	27	<0.001
RDST	+0.02	+0.03	103	20	35	<0.001
ROCKET	+0.02	+0.02	100	17	41	<0.001
HYDRA	+0.01	+0.02	95	19	44	<0.001
MINIROCKET	+0.01	+0.02	96	17	45	<0.001
DRCIF	+0.01	+0.02	91	8	59	0.001
RSTSF	+0.01	0.00	89	12	57	0.016
IT	+0.01	+0.01	86	9	63	0.066
BORF	0.86	0.80				
ST	0.00	−0.01	71	12	75	0.604
W+MUSE	0.00	−0.04	75	15	68	0.171
MRSEQL	−0.01	−0.04	45	9	104	<0.001
TSF	−0.02	−0.04	50	7	101	<0.001
MRSQM	−0.03	−0.05	34	11	113	<0.001
KNN-DTW	−0.06	−0.08	32	6	120	<0.001
REDCOM	−0.06	−0.09	28	6	124	<0.001
CNN	−0.10	−0.21	22	5	131	<0.001
MCDCNN	−0.11	−0.19	21	3	134	<0.001
TAPNET	−0.15	−0.25	14	6	138	<0.001
SAXVSM	−0.38	−0.39	5	1	152	<0.001
BOP	−0.43	−0.44	3	1	154	<0.001

While the CD-plot is widely used for performance comparison, it was criticized in [50] for being prone to manipulation, as the average rank of a model depends on the performance of other comparators. For this reason, in Tables 2 and 3, we report the Comparison Matrices [50] in terms of accuracy and runtime. This representation, proposed in [50], computes the Wilcoxon Signed Rank Test, evaluating the median difference by ranking absolute differences and summing signed ranks. A significant p-value ($\leq$ 0.01 or $\leq$ 0.05) indicates a statistically significant difference between samples. We report both the mean and median difference, as well as the number of wins, ties, and losses against BORF. Models are ranked based on the median performance, with the best models on top. Again, in terms of accuracy, BORF has the 9th rank, statistically tied to RSTSF, IT, ST, and W+MUSE. As for median runtime, BORF is the best model, statistically tied to HYDRA. Against any competitor, BORF wins more than 100 times out of the total, showing its superior median runtime performance. However, the mean runtime performance indicates that there are some outlier datasets in which BORF does not perform as well. For example, BORF has a median difference with ROCKET that is 42.2 seconds faster, but 126.1 seconds slower on average. This difference is due to only four losses overall, among which the only ones with significant runtime deltas are on the "*FaceDetection*" and the "*InsectWingbeat*" datasets. Discarding these two datasets would improve the average runtime tenfold, from 298.7 to only 27.9 seconds. "*FaceDetection*" and "*InsectWingbeat*" are highly multivariate ($\geq$ 100 signals) and short ($\leq$ 65 points) datasets. We believe this difference in performance on these datasets is due to two reasons. First, 1D convolution is extremely well-optimized for multivariate time series datasets, so models like ROCKET, MINIROCKET, HYDRA, and MULTIROCKET-HYDRA greatly benefit from this. Second, BORF is much more efficient on longer datasets, where $l \ll m$, i.e., when the word length is much lower than the number of points of the time series.

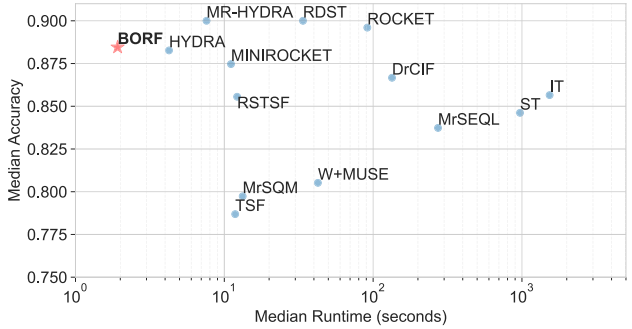
TABLE 3. Comparison Matrix between BORF and competitor approaches. Models are sorted by median difference in runtime with BORF. Wins, and losses are to be read "against BORF". Best models on top. Grey rows are statistically tied with BORF using the Wilcoxon signed-rank test.

	Δruntime (sec)				
	median	*mean*	*win*	*loss*	*p-val*
BORF	1.3	298.7			
HYDRA	+0.2	−294.5	55	103	0.415
MRSQM	+1.4	+1718.0	16	142	<0.001
MR-HYDRA	+1.8	−286.2	25	133	<0.001
SAXVSM	+2.0	+8465.7	14	144	<0.001
BOP	+2.1	+7918.0	14	144	<0.001
MCDCNN	+5.2	+3067.0	8	150	<0.001
MINIROCKET	+6.6	−264.9	15	143	<0.001
RSTSF	+7.2	+1977.3	3	155	<0.001
TSF	+7.6	−247.1	13	145	<0.001
KNN-DTW	+9.6	+628.1	11	147	<0.001
RDST	+10.8	−157.1	7	151	<0.001
W+MUSE	+12.1	+3112.7	2	156	<0.001
REDCOM	+21.0	+304.8	12	146	<0.001
ROCKET	+42.2	−126.1	4	154	<0.001
CNN	+51.7	+1516.3	5	153	<0.001
MRSEQL	+74.9	+4104.1	1	157	<0.001
DRCIF	+102.2	+69.2	4	154	<0.001
IT	+488.2	+1275.1	4	154	<0.001
ST	+571.4	+4489.2	0	158	<0.001
TAPNET	+3293.0	+17 514.3	0	158	<0.001

We further prove this point in Figure 8. In Figure 8a, we show median runtime performance against median accuracy for all datasets. The best models are on the top left. Again, BORF is in the top-10 regarding accuracy, while being the fastest in terms of runtime, with the closest one being HYDRA. In Figure 8b, we show the same plot, but on the subset of 45 longest univariate datasets, having more than 500 observations. Overall, for this subset, BORF is in the top 5,



(a) All datasets



(b) Long ($m \geq 500$), univariate ($c = 1$) datasets

FIGURE 8. Comparison of median accuracy performance against median runtime (seconds). Best models are on the top left.

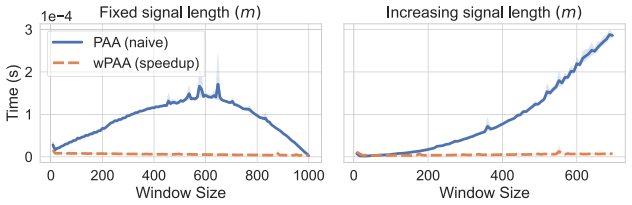


FIGURE 9. Runtime comparison between naive PAA and our proposal (wPAA). On the left: runtime when changing the window size, and keeping time series length fixed. On the right: runtime when changing the time series length and setting the window size to half the number of points.

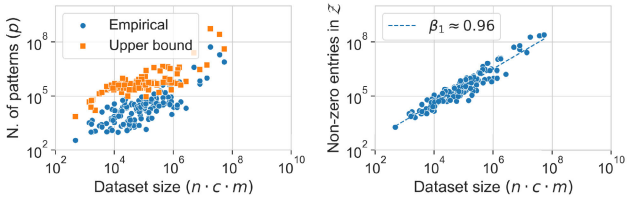


FIGURE 10. Space complexity. On the left: dataset size against the number of extracted patterns. Each blue point represents the empirical number of extracted features for each dataset, and each orange point is the upper bound on the number of features for that specific dataset. On the right, the dataset size against non-zero elements in $\mathcal{Z}$ and fitted log-log regression line.

outperforming even MINIROCKET and HYDRA in terms of accuracy and speed. This further proves that BORF tends to be more efficient on longer datasets, which are arguably the most common in real-world scenarios.

e: SCALABILITY

Given the state-of-the-art speed that BORF achieves, we specifically analyze the impact of our proposed algorithmic speedup. That is, we assess the scalability of wPAA against the classical naive PAA (used by methods such as MRSQM and MRSEQL), when varying the window size, w. First, in Figure 9 (left), we conduct experiments on randomly sampled signals, by fixing their length to 1000, while varying the window size from 8 to 1000 and measuring average runtime on 10 trials. The word length is kept fixed at 8. The runtime of naive PAA, as expected given its theoretical complexity of $(m - w + 1) \cdot w$, grows until $w \approx \frac{m+1}{2}$, and then decreases. Conversely, wPAA, with the exception of a small overhead at the beginning due to the moving average computation, constantly decreases as the window size increases, and is superior to naive PAA at each point of the curve. On the right, we generate random signals at increasing lengths m, setting the window size at half the signal length. This represents the worst-case scenario for PAA, and its quadratic behaviour clearly evidences it. Meanwhile, the runtime of wPAA increases very slowly, as the complexity depends not on the window size but (almost) linearly on the number of observations.

Regarding space complexity, in Figure 10, we compare the dataset size ($n \cdot c \cdot m$) with the number of extracted patterns (left) and the number of non-zero elements in $\mathcal{Z}$ (right) for all datasets in the repositories. Specifically, the plot on the left displays the number of extracted features for each dataset in blue, and the number of theoretically possible features in orange. The upper bound, $O(c \cdot \min(\alpha^l, n \cdot m))$, is significantly higher than the number of extracted patterns, more than an order of magnitude in most cases. On the right, we illustrate the number of non-zero elements in $\mathcal{Z}$, representing the overall empirical space complexity of the algorithm against the dataset size. A linear relationship between the dataset size and the number of non-zero elements becomes evident when visualizing this scatterplot on a log-log scale. To better assess this relationship, we perform a linear regression analysis. The result, $nnz \approx 5.74 \cdot (n \cdot c \cdot m)^{0.96}$, highlights a nearly proportional relationship, suggesting that the non-zero elements scale almost linearly with the dataset size.

f: ABLATION STUDY

We present here an ablation study on BORF to assess which heuristic parameters affect most accuracy and total runtime. Starting from our baseline configuration, we change one parameter at a time. Specifically, we benchmark BORF without the parameter θ (w/o θ), without the inverse hyperbolic sine function (w/o ash), and without performing dilated discretization (w/o dilations). We then test by setting different maximum word lengths, ($\max l = 2$, $\max l = 4$, $\max l = 16$). Additionally, we test limiting the maximum number of window sizes ($\max l = 8$, $\max l = 16$, $\max l = 32$).

TABLE 4. Performance delta in median Accuracy (higher is better) and median runtime (lower is better) for various alternative hyperparameter configurations with respect to the BORF baseline heuristic presented in Section V. The two best-performing models for each metric are in bold.

	θ	d	ash	l	w	Acc	Time (s)
baseline	✓	✓	✓	2,4,8	all	**0.820**	18.0
w/o θ	✗	✓	✓	2,4,8	all	−0.013	+8.0
w/o ash	✓	✓	✗	2,4,8	all	−0.020	−0.1
w/o dilations	✓	✗	✓	2,4,8	all	−0.010	−10.1
max $l = 2$	✓	✓	✓	2	all	−0.185	**−17.2**
max $l = 4$	✓	✓	✓	2,4	all	−0.050	**−14.3**
max $l = 16$	✓	✓	✓	2,4,8,16	all	**+0.003**	+27.5
max $w = 8$	✓	✓	✓	2,4,8	4,8	−0.043	−11.9
max $w = 16$	✓	✓	✓	2,4,8	4,8,16	−0.014	−7.5
max $w = 32$	✓	✓	✓	2,4,8	4,8,16,32	−0.001	−2.5

The results are reported in Table 4. On the top row, we report the baseline's median accuracy and median running time, on the 42 *development datasets* [5]. In all other rows, we report the performance delta between the alternatives and the baseline. Regarding accuracy, almost all alternatives perform worse than the baseline, with the exception of increasing the word length to 16, which slightly improves accuracy. However, we deemed the runtime increase to be too significant to consider this configuration in the heuristic. Decreasing the word length significantly impacts performance negatively; accuracy decreases by 0.05 if we remove $l = 8$ and 0.185 if we also remove $l = 4$, but it positively impacts runtime. However, we believe the performance drop is too severe to ignore. The parameter θ is particularly interesting because its removal deteriorates both the accuracy and runtime. This parameter impacts the *unique* function in Algorithm 1, which receives fewer unique words to hash, while simultaneously removing noisy subsequences. Overall, the proposed baseline appears to be a good compromise between accuracy and runtime performance. The inverse hyperbolic sine has limited effect, but is still noticeable, at a very small performance gap. The only parameter that could be worth limiting is the window size (max $l = 32$). However, the performance drop is still present, with minimal runtime improvement. Moreover, the development datasets tend to be short in terms of length, so we postulate we can obtain better generalizability with our proposed heuristic.

g: CASE STUDY: FINGERMOVEMENTS

We showcase the explainability of the BORF representation on the multivariate *FingerMovements* dataset [51]. This dataset contains EEG sessions of a subject seated with their arms resting on a table, fingers positioned on a keyboard. The task required self-paced typing using the index and little fingers in any chosen order. Each sample comprises recordings from 28 EEG channels, representing signals from distinct brain regions, as depicted in Figure 11. The goal, as defined in the UEA repository, is binary classification:

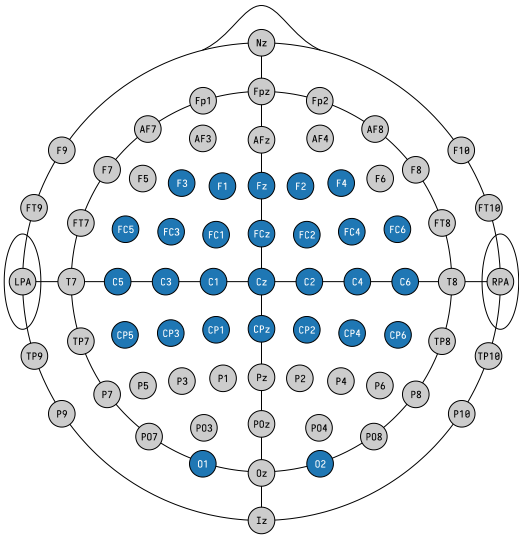


FIGURE 11. Electrode locations based on the International 10-20 system for encephalography recording. The FingerMovements dataset contains recordings from the blue regions.

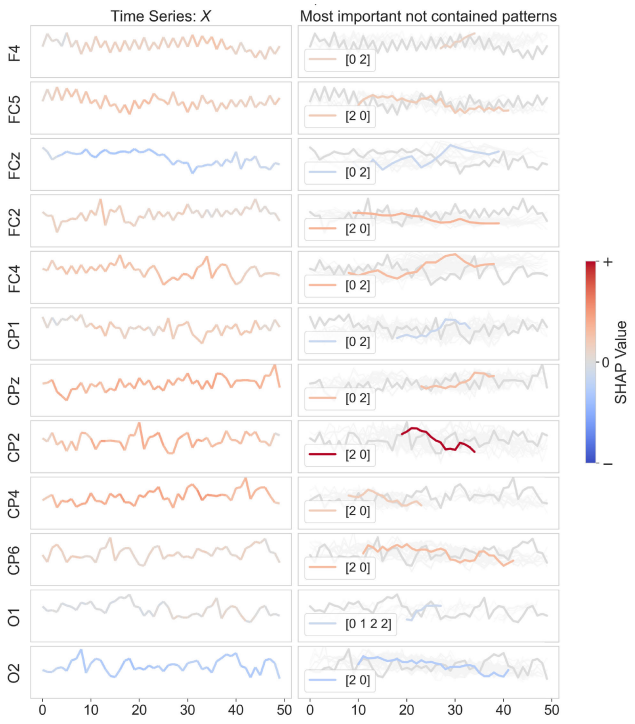
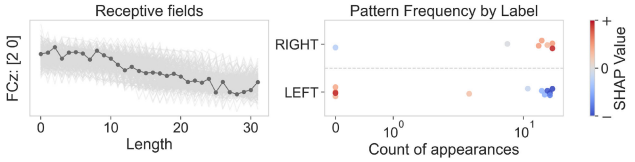


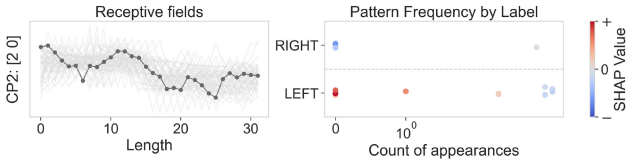
FIGURE 12. Local explanation for a *Left-Hand* instance of the FingerMovements dataset. On the left, 12 out of the 28 signals of the original time series colored based on the saliency map, highlighting the most relevant observations for the classification. On the right, for each signal, the medoid of the most important pattern that is not contained in that signal of the time series.

0 for imminent *Left-Hand* movements and 1 for *Right-Hand* movements.

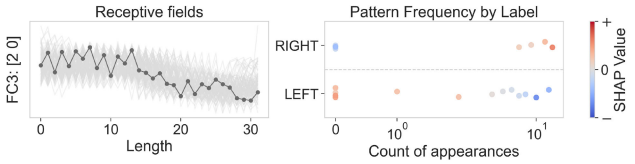
For the local explanation, we choose the *Left-Hand* instance reported in Figure 12. On the left side of the picture, we show



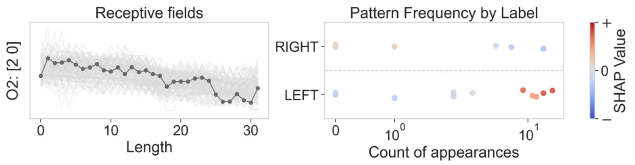
(a) Pattern '2,0' from FCz.



(b) Pattern '2,0' from CP2.



(c) Pattern '2,0' from FC3.



(d) Pattern '2,0' from O2.

FIGURE 13. Global explanation for four of the most important '2,0' patterns for the FingerMovements dataset. On the left, in dark gray, the medoid of all alignments of the pattern in the FingerMovements dataset. On the right, the count of appearances of each pattern in each time series in the dataset, divided by label. The color represents the importance of the pattern count in terms of SHAP values.

12 of the most relevant signals out of the 28, colored based on the importance of each observation toward the classification. For this instance, the saliency map is mostly spread and does not indicate that any part of the time series is particularly important by itself for the classification. However, it is still useful, as it highlights the most relevant signals with a more intense color, in this case, CPz, CP2, and CP4 for positive contribution and O2 for negative contribution. In this case, the right plot, showing the most important not contained patterns, is more interesting, as it shows that the most relevant pattern is '2,0' for half of the depicted signals. This seems to be a very important shape for many signals in the dataset.

Indeed, by using the global explanation, we can easily see that '2,0' is relevant for many instances of the *FingerMovements* dataset, as shown in Figure 13. Analyzing the explanation, it seems, however, that this pattern behaves differently in various signals. For example, if this pattern is not contained in the FCz, CP2, or FC3 signals, it seems to push toward the *Left-Hand* class, while the opposite is true for the '2,0' pattern contained in the O2 signal.

In this sense, a significant advantage of BORF is the flexibility, after transforming the time series, in deciding which patterns have the same semantics, i.e., whether the counts of patterns from certain channels should be summed together or kept separate. In particular, after using BORF for transforming the dataset into $\mathcal{Z} \in \mathbb{N}^{n \times 28 \times p}$, we can ask ourselves if patterns for certain brain regions should be summed together or not, i.e., *"Does it make sense to sum the frequency of pattern '2,0' extracted from the occipital lobe with the same pattern '2,0' extracted from the frontal lobe?"*. From the global explanation, it seems like frequencies from some signals have the same semantics and some do not.

Conventionally, pattern count in each signal is maintained separately, i.e., BORF preserves the counts of each pattern for each signal independently. In this setting, the accuracy of BORF is 0.56, which ranks in the top 5 among the competitor classifiers tested earlier. An alternative method involves summing patterns across all channels, effectively treating pattern frequencies from various parts of the brain as semantically equivalent, obtaining $\mathcal{Z}' \in \mathbb{N}^{n \times 1 \times p}$, where $z'_{i,1,k} = \sum_{j=1}^{c} z_{i,j,k}$. This method has improved performance on this dataset, achieving an accuracy of 0.58, which is the best, tied with W+MUSE. With BORF, it is possible to supervise pattern aggregation. For instance, as illustrated in Figure 11, the two occipital regions O1 and O2 (at the bottom of the picture) are spatially remote from the 26 recordings of the parietal and frontal lobes, which cluster together in this dataset. Therefore, we can sum pattern counts from these different regions separately, i.e., $z''_{i,1,k} = \sum_{j=1}^{26} z_{i,j,k}$ (for the parietal and frontal lobes), and $z''_{i,2,k} = \sum_{j=27}^{28} z_{i,j,k}$ (for the occipital regions), obtaining $\mathcal{Z}'' \in \mathbb{N}^{n \times 2 \times p}$. Using this dataset as our classifier input, we achieve a superior accuracy of 0.61, which is better than that of any classifiers tested earlier. This experiment shows that the interpretability, paired with the capability of defining aggregation after the Bag-Of-Receptive-Fields transformation, offers many insights and substantial flexibility, allowing an expert user to use the explanation as a qualitative and quantitative tool to answer research questions.

VI. CONCLUSION

In this work, we proposed BORF, a fast, interpretable, and deterministic transformation for univariate and multivariate time series data. We formalized the window-wise Symbolic Aggregate Approximation (wSAX) and augmented it with dilation and stride convolutional operators. Our approach significantly reduces the computational complexity traditionally associated with window-wise SAX-based classifiers and extends the Bag-Of-Patterns framework to a more flexible Bag-Of-Receptive-Fields. We have shown that the proposed approach is statistically superior in runtime and predictive performance compared to SAX-based methods while being very competitive among dictionary-based and interpretable classifiers. Moreover, the proposed heuristic is scalable both

in time and memory, allowing the accurate analysis of long datasets. We have assessed the interpretability and flexibility of our approach in a case study, showcasing it as a very promising tool for answering research questions.

A limitation of our proposal is that it is still subpar compared to ensemble hybrids such as Multirocket-Hydra, which achieve better predictive accuracy but are uninterpretable. Beating these kinds of models was never the claim of this work, as they use multiple algorithms and complex feature spaces to achieve that level of performance. However, an interesting future research direction would be to build a fast ensemble hybrid that is also interpretable, combining, for example, the feature space of BORF with other interpretable approaches such as shapelets or interval-based methods. Additionally, like all SAX-based methods, BORF faces the challenge of managing longer word lengths as the feature space expands rapidly. Our ablation study indicates that this does not seem to be a problem for this set of datasets; however, with very long time series, larger word lengths would probably be needed. To address this, we plan to hash the extracted patterns into a pre-fixed size, for example, using Bloom filters, to mitigate this issue.

Finally, given that the BORF transformation is unsupervised, we plan on tackling time series regression, forecasting, and clustering to achieve fast and deterministic explanations in different types of tasks.

APPENDIX A NOTATION

In Table 5, we summarize the main symbols used in the paper.

TABLE 5. Summary of notation.

Time Series Data	
$\mathcal{X}, X, \mathbf{x}, x$	time series dataset, instance, signal, entry
$\mathbf{y}, y$	labels vector, value
n	number of instances in a dataset
c	number of signals in a time series
m	number of observations in a signal
Convolution	
w	window/receptive field size
d	dilation
s	stride
v	number of windows/receptive fields
Discretization	
l	word length
q	segment size
$\alpha, \mathbf{b}$	alphabet size, Gaussian breakpoints
μ_i^{seg}, μ_i^{win}	moving average for segment, window i
σ_i^{win}	moving std for window i
$M, \mu_{i,j}$	wPAA matrix, entry
$M^*, \mu_{i,j}^*$	standardized wPAA matrix, entry
$\tilde{M}, \tilde{\mu}_{i,j}$	wSAX matrix, entry
$\mathbf{k}, \mathbf{k}^*, \mathbf{a}$	words, unique words, counts
Bag-Of-Receptive-Fields	
$\mathcal{Z}, Z, z$	3d tensor, 2d dataset, entry
p	number of features in BORF
XAI	
Φ, ϕ	feature importance matrix, entry
Ψ, ψ	saliency matrix, entry

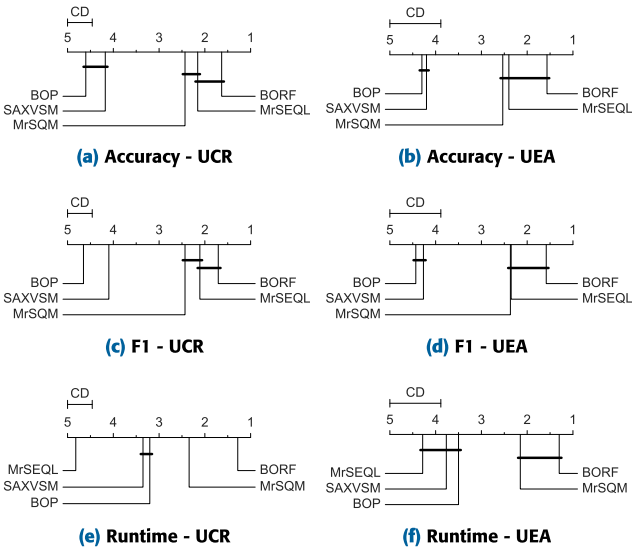


FIGURE 14. CD Plots for SAX-based methods. Best models to the right.

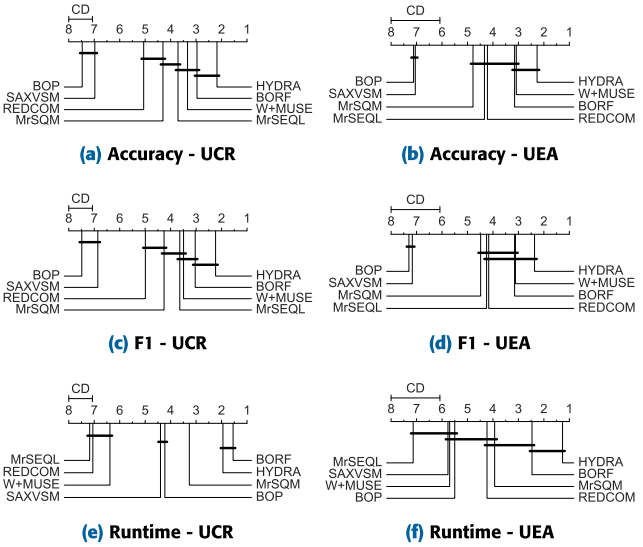


FIGURE 15. CD Plots for dictionary-based methods. Best models to the right.

APPENDIX B COMPLEXITY

Here, we show in greater detail that the worst-case complexity of the window-wise naive PAA is quadratic. For each window, for each segment, we need to average the values in each segment, therefore Equation (5) is repeated vl times, for a total of vlq iterations. Given that, by definition, $ql = w$, it follows that $vlq = vw$. The highest number of receptive fields is obtained when $d = 1$ and $s = 1$, i.e., when dilation and stride are both absent, given that they reduce the number of receptive fields by skipping some observations [37]. Thus, the total number of iterations simplifies to:

$$vw = \left(1 + \left\lfloor \frac{(m - w - (d-1)(w-1))}{s} \right\rfloor\right) w$$
$$= (m - w + 1)w = mw - w^2 + w,$$

TABLE 6. Accuracy of the top 13 best-performing models on all datasets. Missing values are due to exceeded runtime limits or out-of-memory errors.

	BORF	DR CIF	HYDRA	IT	MINI ROCKET	MR HYDRA	MR SEQL	MR SQM	RDST	ROCKET	RS TSF	ST	W+ MUSE
ACSF1	0.610	0.880	0.850	0.920	0.910	0.880	0.710	0.770	0.920	0.880	0.890	0.890	0.920
Adiac	0.652	0.829	0.818	0.813	0.816	0.844	0.739	0.509	0.742	0.785	0.803	0.801	0.824
AGW-X	0.656	0.647	0.686	0.789	0.703	0.736	0.601	0.590	0.693	0.699	0.643	0.590	0.487
AGW-Y	0.706	0.661	0.750	0.817	0.769	0.791	0.680	0.624	0.754	0.774	0.711	0.626	0.563
AGW-Z	0.633	0.661	0.671	0.784	0.679	0.724	0.579	0.546	0.653	0.716	0.674	0.589	0.516
ArrowHead	0.829	0.823	0.834	0.834	0.863	0.863	0.754	0.709	0.857	0.823	0.749	0.806	0.874
AWRecog	0.993	0.983	0.990	0.983	0.990	0.993	0.967	0.950	0.993	0.993	0.980	0.967	0.993
AFib	0.267	0.400	0.133	0.200	0.133	0.067	0.400	0.267	0.200	0.067	0.333	0.267	0.200
BME	0.980	1.000	1.000	0.993	1.000	1.000	0.900	0.840	0.993	1.000	1.000	0.893	0.887
BasicMotions	0.975	1.000	1.000	1.000	1.000	1.000	1.000	0.900	1.000	1.000	1.000	1.000	1.000
Beef	0.733	0.833	0.833	0.667	0.833	0.767	0.600	0.767	0.867	0.800	0.733	0.733	0.833
BeetleFly	0.900	0.900	0.900	0.800	0.850	0.900	0.850	0.900	0.950	0.900	0.900	0.800	0.950
BirdChick	0.900	0.950	0.900	0.900	0.900	0.900	0.800	0.900	0.900	0.900	0.900	0.900	0.900
CBF	1.000	0.999	0.993	0.999	0.999	0.996	0.999	1.000	0.992	1.000	0.992	0.970	0.992
Car	0.917	0.867	0.933	0.883	0.917	0.933	0.850	0.750	0.917	0.900	0.833	0.833	0.900
CharTraj	0.969	0.981	0.986	0.995	0.993	0.992	0.978	0.953	0.992	0.991	0.985	0.974	0.984
Chinatown	0.962	0.985	0.983	0.985	0.983	0.980	0.950	0.968	0.977	0.980	0.983	0.971	0.939
ChlConc	0.721	0.738	0.760	0.870	0.755	0.789	0.718	0.682	0.754	0.812	0.743	0.729	0.772
CinCECGT	0.751	1.000	0.995	0.857	0.857	0.969	0.846	0.960	0.993	0.827	0.982	0.938	0.999
Coffee	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.964	0.964	1.000	1.000	1.000	1.000
Computers	0.772	0.732	0.716	0.816	0.732	0.772	0.728	0.736	0.760	0.772	0.752	0.720	0.732
Cricket	1.000	0.986	0.986	0.986	0.986	0.986	0.986	0.958	1.000	1.000	0.986	0.972	1.000
CricketX	0.821	0.746	0.800	0.800	0.821	0.803	0.769	0.705	0.821	0.826	0.746	0.800	0.718
CricketY	0.821	0.790	0.841	0.836	0.828	0.851	0.792	0.710	0.841	0.854	0.800	0.785	0.826
CricketZ	0.838	0.797	0.805	0.810	0.828	0.838	0.764	0.697	0.864	0.859	0.792	0.808	0.764
Crop	0.678	0.780	0.734	0.777	0.766	0.778	0.642	0.641	0.766	0.757	0.777	-	0.754
DiatomSR	0.980	0.928	0.948	0.944	0.928	0.964	0.941	0.889	0.922	0.977	0.928	0.938	0.889
DP-OAG	0.770	0.755	0.763	0.727	0.734	0.770	0.748	0.727	0.770	0.777	0.734	0.741	0.806
DP-OC	0.743	0.786	0.786	0.761	0.790	0.808	0.775	0.754	0.768	0.764	0.783	0.812	0.757
DP-TW	0.712	0.669	0.712	0.683	0.655	0.698	0.676	0.705	0.719	0.705	0.655	0.669	0.662
DLDay	0.537	0.637	0.512	0.512	0.662	0.600	0.613	0.487	0.625	0.588	0.625	0.550	0.350
DLGame	0.855	0.899	0.855	0.877	0.870	0.884	0.746	0.696	0.833	0.877	0.891	0.848	0.630
DLWeekend	0.978	0.986	0.978	0.978	0.986	0.978	0.949	0.957	0.978	0.971	0.986	0.964	0.906
DuckDG	0.440	0.480	0.680	0.620	0.700	0.640	-	-	0.420	0.460	-	0.560	0.460
ECG200	0.900	0.870	0.870	0.940	0.910	0.910	0.890	0.830	0.900	0.920	0.910	0.900	0.850
ECG5000	0.944	0.942	0.949	0.939	0.945	0.946	0.943	0.938	0.945	0.946	0.944	0.945	0.944
ECG5D	1.000	0.999	1.000	1.000	1.000	1.000	0.995	0.994	0.999	1.000	0.997	0.997	1.000
EOG-HS	0.610	0.599	0.572	0.633	0.613	0.633	0.630	0.398	0.677	0.616	0.599	0.619	0.500
EOG-VS	0.506	0.594	0.492	0.530	0.522	0.514	0.517	0.398	0.550	0.552	0.552	0.522	0.326
ERing	0.952	0.993	0.985	0.870	0.978	0.989	0.837	0.896	0.985	0.981	0.978	0.952	0.959
Earthquakes	0.748	0.748	0.734	0.727	0.727	0.741	0.683	0.705	0.734	0.741	0.748	0.748	0.748
EigenWorms	0.870	0.931	0.985	0.725	0.939	0.962	-	0.809	0.908	0.916	0.954	0.779	0.947
ElecDev	0.757	0.736	0.752	0.705	0.742	0.743	0.696	0.727	0.743	0.729	0.742	-	0.765
Epilepsy	0.993	1.000	1.000	0.993	1.000	1.000	0.986	0.993	0.993	0.986	1.000	1.000	1.000
EthanolConc	0.369	0.631	0.548	0.232	0.483	0.589	0.456	0.354	0.620	0.433	0.620	0.760	0.540
EthanolLvl	0.508	0.596	0.588	0.830	0.578	0.638	0.648	0.502	0.664	0.598	0.542	0.696	0.558
FaceAll	0.802	0.764	0.898	0.764	0.807	0.808	0.780	0.783	0.798	0.947	0.864	0.750	0.778
FaceDet	0.589	0.591	0.523	0.650	0.582	0.619	-	0.491	0.629	0.606	-	-	-
FaceFour	1.000	0.977	0.886	0.966	0.989	0.955	1.000	0.989	0.989	0.977	1.000	0.886	0.841
FacesUCR	0.942	0.897	0.956	0.961	0.960	0.960	0.929	0.893	0.967	0.961	0.903	0.889	0.906
FiftyWords	0.730	0.804	0.842	0.815	0.837	0.862	0.716	0.593	0.840	0.833	0.763	0.730	0.769
FingMvmnt	0.560	0.470	0.550	0.500	0.570	0.560	0.480	0.550	0.540	0.550	0.570	0.570	0.580
Fish	0.966	0.954	0.989	0.971	0.977	0.983	0.966	0.949	0.989	0.983	0.909	0.960	0.994
FordA	0.937	0.971	0.961	0.950	0.948	0.955	0.934	0.911	0.950	0.935	0.975	0.939	-
FordB	0.801	0.806	0.825	0.847	0.820	0.831	0.790	0.735	0.833	0.810	0.828	0.826	-
FRTrain	0.998	0.999	0.998	0.997	1.000	1.000	0.998	0.995	0.999	0.997	0.999	0.996	0.995
FSTrain	0.991	0.999	0.926	0.845	0.967	0.995	0.996	0.960	0.996	0.951	0.980	0.959	0.949
Fungi	1.000	0.930	1.000	0.995	1.000	1.000	0.930	0.806	1.000	1.000	0.919	0.839	1.000
GMD1	0.746	0.723	0.754	0.692	0.715	0.769	0.762	0.646	0.785	0.785	0.731	0.769	0.615
GMD2	0.662	0.685	0.638	0.562	0.631	0.669	0.677	0.577	0.715	0.723	0.615	0.646	0.462
GMD3	0.469	0.554	0.500	0.377	0.392	0.492	0.500	0.377	0.554	0.485	0.531	0.577	0.323
GPZ1	0.872	0.878	0.826	0.907	0.890	0.890	0.849	0.878	0.924	0.878	0.866	0.808	0.651
GPZ2	0.873	0.810	0.810	0.835	0.835	0.823	0.854	0.810	0.861	0.842	0.804	0.816	0.620
GunPoint	1.000	0.993	1.000	1.000	0.993	1.000	0.993	0.987	1.000	1.000	0.973	1.000	1.000
GP-AS	0.987	0.991	1.000	0.975	0.997	1.000	0.991	0.994	0.997	0.997	0.991	0.984	1.000
GP-MVF	1.000	1.000	1.000	0.997	1.000	1.000	0.987	0.987	1.000	0.997	1.000	0.968	1.000
GP-OY	0.997	1.000	1.000	1.000	1.000	1.000	0.968	0.965	1.000	0.994	1.000	0.975	1.000
Ham	0.743	0.705	0.743	0.714	0.714	0.743	0.676	0.762	0.705	0.714	0.752	0.762	0.686
HandMovDir	0.284	0.514	0.216	0.405	0.392	0.365	0.351	0.351	0.324	0.514	0.459	0.338	0.311
HandOutl	0.930	0.922	0.941	0.962	0.935	0.946	0.943	0.911	0.946	0.943	0.914	0.922	0.924
Handwrit	0.520	0.362	0.466	0.648	0.518	0.514	0.461	0.322	0.600	0.587	0.365	0.449	0.366
Haptics	0.487	0.519	0.513	0.539	0.529	0.529	0.510	0.513	0.575	0.526	0.500	0.529	0.390
Heartbeat	0.741	0.790	0.761	0.678	0.741	0.732	0.702	0.727	0.756	0.756	0.780	0.732	0.746
Herring	0.609	0.578	0.750	0.656	0.688	0.734	0.609	0.578	0.688	0.594	0.578	0.672	0.641
Hse20	0.992	0.916	0.958	0.941	0.958	0.992	0.899	0.941	0.966	0.966	0.916	0.899	0.975
InlSkate	0.405	0.531	0.491	0.489	0.476	0.496	0.445	0.329	0.447	0.469	0.615	0.425	0.578
IEPG-RT	0.988	1.000	1.000	1.000	1.000	1.000	0.976	0.948	1.000	1.000	1.000	0.996	1.000

TABLE 6. *(Continued.)* **Accuracy of the top 13 best-performing models on all datasets. Missing values are due to exceeded runtime limits or out-of-memory errors.**

	BORF	DR CIF	HYDRA	IT	MINI ROCKET	MR HYDRA	MR SEQL	MR SQM	RDST	ROCKET	RS TSF	ST	W+ MUSE
IEPG-ST	0.928	1.000	1.000	1.000	0.992	1.000	0.896	0.795	0.992	0.980	1.000	0.976	1.000
IWingImp	0.519	0.664	0.679	0.692	0.673	0.667	-	0.303	0.482	0.399	-	-	-
IWingSound	0.633	0.682	0.655	0.627	0.672	0.667	0.599	0.552	0.665	0.666	0.668	0.648	0.627
ItaPwrDem	0.954	0.968	0.966	0.971	0.963	0.968	0.907	0.926	0.856	0.970	0.972	0.962	0.948
JapVow	0.795	0.973	0.978	0.989	0.992	0.978	0.965	0.930	0.965	0.854	0.965	0.762	0.954
LSST	0.603	0.565	0.607	0.506	0.651	0.636	0.681	0.576	0.647	0.636	0.628	0.639	0.603
LrgKitchAppl	0.923	0.821	0.883	0.888	0.875	0.896	0.872	0.915	0.824	0.896	0.811	0.851	0.672
Libras	0.844	0.894	0.933	0.878	0.917	0.928	0.817	0.800	0.944	0.906	0.917	0.872	0.906
Lightn2	0.803	0.754	0.721	0.770	0.754	0.672	0.672	0.623	0.738	0.787	0.787	0.787	0.672
Lightn7	0.795	0.753	0.808	0.822	0.795	0.822	0.795	0.630	0.781	0.822	0.753	0.753	0.644
Mallat	0.939	0.925	0.941	0.944	0.946	0.929	0.938	0.897	0.954	0.956	0.972	0.982	0.942
Meat	0.850	0.933	0.917	0.933	0.967	0.933	0.917	0.883	0.950	0.950	0.967	0.933	0.833
MedImgs	0.696	0.786	0.778	0.780	0.791	0.797	0.701	0.617	0.763	0.807	0.826	0.742	0.734
MelbPed	0.836	0.964	0.926	0.972	0.966	0.967	0.812	0.845	0.965	0.908	0.971	0.932	0.887
MP-OAG	0.623	0.610	0.597	0.487	0.552	0.584	0.519	0.552	0.571	0.552	0.623	0.571	0.571
MP-OC	0.759	0.832	0.835	0.797	0.849	0.845	0.729	0.691	0.808	0.838	0.845	0.838	0.790
MP-TW	0.591	0.591	0.539	0.506	0.526	0.513	0.532	0.487	0.558	0.539	0.591	0.526	0.539
MxShpRT	0.952	0.956	0.981	0.967	0.971	0.982	0.963	0.962	0.983	0.973	0.952	0.962	0.986
MxShpST	0.920	0.922	0.965	0.904	0.950	0.967	0.903	0.930	0.967	0.936	0.898	0.938	0.960
MoteStrain	0.946	0.932	0.929	0.888	0.927	0.946	0.894	0.923	0.918	0.917	0.951	0.903	0.933
MotImag	0.460	0.500	0.500	0.570	0.500	0.480	-	-	0.540	0.540	0.500	0.510	0.570
NATOPS	0.917	0.828	0.872	0.967	0.944	0.917	0.911	0.867	0.878	0.872	0.861	0.878	0.922
NIFECGT1	0.902	0.931	0.943	0.910	0.949	0.964	0.925	0.780	0.924	0.958	0.926	0.927	0.913
NIFECGT2	0.916	0.935	0.958	0.959	0.964	0.968	0.946	0.838	0.938	0.966	0.942	0.947	0.926
OSULeaf	0.950	0.847	0.988	0.897	0.959	0.963	0.942	0.967	0.988	0.942	0.872	0.880	0.988
OliveOil	0.900	0.933	0.933	0.833	0.933	0.967	0.867	0.933	0.900	0.900	0.933	0.800	0.867
PEMS-SF	0.908	1.000	0.803	0.809	0.827	0.775	-	-	0.919	0.844	1.000	0.971	-
PLAID	0.885	0.907	0.819	0.534	0.877	0.909	0.883	0.890	0.905	0.858	0.888	0.784	0.736
PenDigits	0.920	0.973	0.968	0.987	-	0.968	0.872	0.941	0.985	0.979	-	-	0.953
POC	0.748	0.844	0.823	0.831	0.840	0.844	0.742	0.703	0.819	0.837	0.844	0.837	0.807
Phoneme	0.317	0.400	0.324	0.334	0.294	0.354	0.276	0.341	0.341	0.281	0.369	0.346	0.339
PhonSpect	0.251	0.296	0.329	0.306	0.317	0.332	0.232	0.220	0.313	0.279	0.287	0.307	-
PGWZ	0.700	0.720	0.700	0.640	0.780	0.780	0.680	0.680	0.780	0.740	0.720	0.580	0.520
PAP	0.659	0.245	0.769	0.933	0.865	0.712	0.587	0.957	0.822	0.394	0.245	0.668	0.851
PAPress	0.986	0.885	0.986	1.000	0.981	0.957	0.904	0.971	0.942	0.962	0.750	0.962	1.000
PCVP	0.952	0.591	0.966	0.947	0.947	0.894	0.707	0.952	0.938	0.933	0.644	0.846	0.976
Plane	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
PwrCons	0.889	1.000	0.978	1.000	0.994	0.983	0.878	0.872	0.994	0.933	1.000	0.983	0.933
PP-OAG	0.844	0.849	0.868	0.829	0.834	0.868	0.820	0.844	0.844	0.859	0.854	0.834	0.844
PP-OC	0.859	0.897	0.911	0.918	0.904	0.924	0.804	0.811	0.883	0.900	0.911	0.893	0.900
PP-TW	0.800	0.795	0.810	0.776	0.800	0.805	0.815	0.761	0.820	0.805	0.810	0.746	0.815
RacketS	0.908	0.888	0.914	0.882	0.855	0.908	0.816	0.829	0.882	0.908	0.901	0.822	0.862
RefDev	0.608	0.613	0.528	0.533	0.480	0.517	0.616	0.635	0.568	0.512	0.600	0.536	0.552
Rock	0.880	0.700	0.920	0.640	0.820	0.860	0.820	0.880	0.880	0.900	0.800	0.940	0.800
ScrnType	0.515	0.555	0.493	0.603	0.461	0.565	0.552	0.509	0.509	0.467	0.541	0.533	0.472
SRSCP1	0.758	0.870	0.867	0.829	0.918	0.952	0.635	0.652	0.898	0.853	0.867	0.836	0.799
SRSCP2	0.472	0.511	0.589	0.550	0.544	0.567	0.533	0.533	0.511	0.528	0.567	0.467	0.600
SHG-Ch2	0.892	0.932	0.847	0.865	0.888	0.950	0.840	0.805	0.923	0.928	0.973	0.957	0.797
SHM-Ch2	0.638	0.849	0.496	0.456	0.676	0.773	0.638	0.367	0.702	0.629	0.856	0.876	0.284
SHS-Ch2	0.867	0.913	0.771	0.711	0.867	0.902	0.882	0.758	0.853	0.887	0.922	0.949	0.611
SWGZ	0.820	0.860	0.840	0.780	0.860	0.900	0.760	0.880	0.880	0.900	0.800	0.860	0.680
ShapeSim	1.000	0.972	0.989	0.956	1.000	1.000	0.672	1.000	1.000	1.000	0.972	0.978	1.000
ShapesAll	0.897	0.867	0.928	0.917	0.923	0.928	0.928	0.822	0.927	0.905	0.868	0.838	0.945
SmKitchAppl	0.757	0.821	0.808	0.763	0.827	0.805	0.837	0.797	0.827	0.808	0.832	0.811	0.773
SmthSubsp	0.880	0.980	0.927	0.993	0.953	0.980	0.800	0.913	0.987	0.973	0.980	0.953	0.800
SonySurf1	0.759	0.917	0.920	0.907	0.887	0.887	0.747	0.790	0.892	0.925	0.899	0.937	0.517
SonySurf2	0.935	0.918	0.948	0.955	0.921	0.942	0.879	0.937	0.901	0.919	0.881	0.925	0.940
SpArabDig	0.986	0.978	0.989	0.993	0.993	0.993	0.961	0.933	0.997	0.996	0.983	0.945	0.986
SWJ	0.400	0.400	0.400	0.467	0.400	0.600	0.667	0.667	0.467	0.467	0.467	0.467	0.400
StarLC	0.976	0.981	0.982	0.979	0.982	0.981	0.980	0.969	0.983	0.981	0.980	0.978	0.979
Strawberry	0.968	0.965	0.973	0.849	0.984	0.976	0.962	0.954	0.976	0.981	0.968	0.973	0.973
SweLeaf	0.926	0.974	0.978	0.970	0.962	0.976	0.942	0.918	0.965	0.962	0.958	0.917	0.971
Symbols	0.956	0.967	0.980	0.983	0.983	0.981	0.963	0.972	0.980	0.973	0.968	0.942	0.974
SynthCtrl	0.977	1.000	0.957	0.997	0.987	0.997	0.957	0.967	0.993	1.000	0.993	0.997	0.980
ToeSeg1	0.969	0.925	0.961	0.956	0.961	0.956	0.965	0.947	0.969	0.952	0.899	0.956	0.961
ToeSeg2	0.923	0.885	0.908	0.954	0.923	0.915	0.869	0.908	0.908	0.931	0.885	0.815	0.938
Trace	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.990	1.000
2LeadECG	0.999	0.996	0.999	0.995	0.998	0.998	0.993	0.998	0.996	0.999	0.986	0.996	1.000
TwoPatt	0.995	1.000	0.987	1.000	0.997	1.000	0.998	0.921	1.000	1.000	0.997	0.988	0.996
UMD	0.993	0.972	0.993	0.993	0.993	0.993	0.938	0.972	1.000	0.993	0.986	0.972	0.986
UWGL	0.944	0.938	0.909	0.891	0.931	0.941	0.778	0.591	0.931	0.941	0.906	0.891	0.909
UWGL-All	0.940	0.972	0.973	0.941	0.971	0.980	0.919	0.734	0.980	0.976	0.961	0.960	0.889
UWGL-X	0.811	0.843	0.854	0.827	0.848	0.875	0.807	0.691	0.854	0.853	0.829	0.811	0.826
UWGL-Y	0.731	0.769	0.792	0.753	0.776	0.810	0.711	0.619	0.764	0.779	0.758	0.731	0.732
UWGL-Z	0.748	0.781	0.803	0.765	0.801	0.821	0.759	0.642	0.800	0.797	0.776	0.780	0.768
Wafer	1.000	0.999	1.000	0.999	0.999	1.000	0.995	1.000	1.000	0.998	1.000	1.000	0.999
Wine	0.648	0.815	0.907	0.667	0.852	0.926	0.741	0.815	0.889	0.778	0.870	0.944	0.833
WordSyn	0.654	0.680	0.743	0.730	0.760	0.777	0.629	0.487	0.745	0.752	0.636	0.654	0.657
Worms	0.688	0.753	0.753	0.805	0.740	0.779	0.766	0.753	0.688	0.727	0.753	0.766	0.805
Worms2C	0.727	0.805	0.766	0.766	0.779	0.779	0.805	0.740	0.740	0.805	0.792	0.844	0.792
Yoga	0.857	0.873	0.927	0.904	0.910	0.929	0.869	0.761	0.932	0.910	0.877	0.890	0.934

with $1 \leq w \leq m$. Given $f(w) = mw - w^2 + w$, we find the critical point $\frac{df}{dw} = m - 2w + 1 = 0$, which indicates that $w = \frac{m+1}{2}$ is a local maximum, given that the second derivative is negative. The maximum number of iterations is:

$$f\left(\frac{m+1}{2}\right) = m\left(\frac{m+1}{2}\right) - \left(\frac{m+1}{2}\right)^2 + \left(\frac{m+1}{2}\right) = \frac{(m+1)^2}{4}.$$

Thus the worst-case complexity is $O(\frac{(m+1)^2}{4}) = O(m^2)$.

APPENDIX C MORE EXPERIMENTS

In Figure 14 to 17, we report the CD plots from different classifier families for the UEA and UCR repositories separately, also showing performance in terms of F1 score. Figure 18 shows a comparison between median runtime in seconds and median F1 score against competitor models. Figure 19 shows the boxplot of runtime performance (in seconds). Finally, Table 6 reports the results in terms of accuracy of the top-13 models.

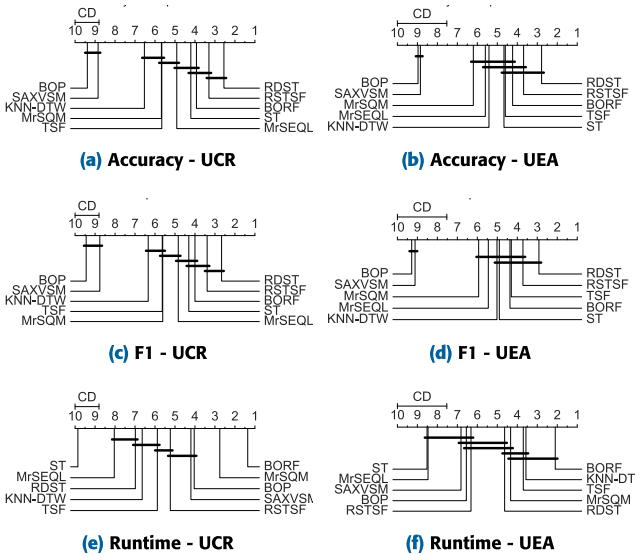


FIGURE 16. CD Plots for interpretable methods. Best models to the right.

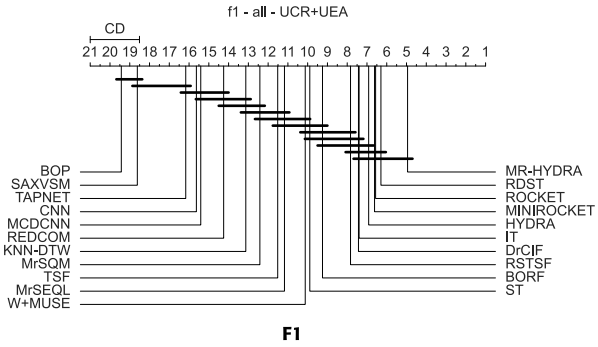


FIGURE 17. CD Plots for all benchmarked methods. Best models to the right.

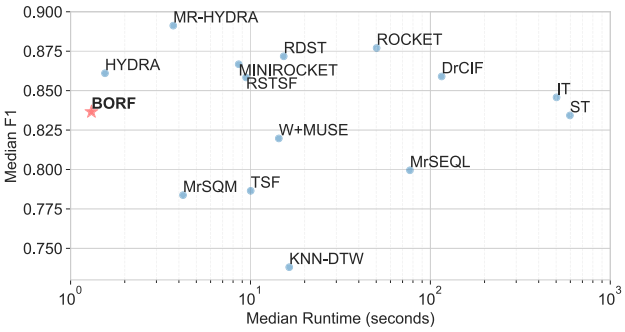


FIGURE 18. Comparison of median F1 performance against median runtime (seconds). Best models are on the top left.

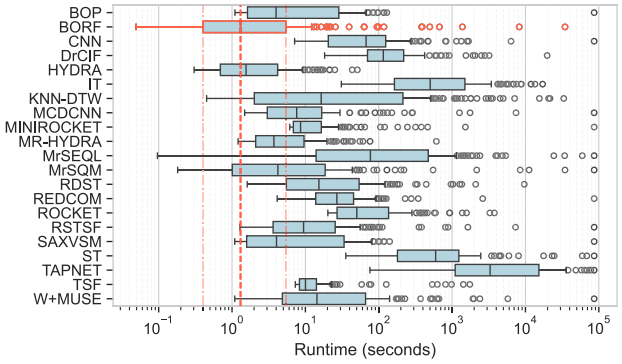


FIGURE 19. Runtime boxplots (seconds), Lower is better. BORF median and interquartile range are highlighted in red.

REFERENCES

[1] M. Middlehurst, P. Schäfer, and A. Bagnall, "Bake off redux: A review and experimental evaluation of recent time series classification algorithms," *Data Mining Knowl. Discovery*, vol. 38, no. 4, pp. 1958–2031, Apr. 2024.

[2] A. Dempster, D. F. Schmidt, and G. I. Webb, "Hydra: Competing convolutional kernels for fast and accurate time series classification," *Data Mining Knowl. Discovery*, vol. 37, no. 5, pp. 1779–1805, Sep. 2023.

[3] C. W. Tan, A. Dempster, C. Bergmeir, and G. I. Webb, "MultiRocket: Multiple pooling operators and transformations for fast and effective time series classification," *Data Mining Knowl. Discovery*, vol. 36, no. 5, pp. 1623–1646, Sep. 2022.

[4] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall, "HIVE-COTE 2.0: A new meta ensemble for time series classification," *Mach. Learn.*, vol. 110, nos. 11–12, pp. 3211–3243, Dec. 2021.

[5] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining Knowl. Discovery*, vol. 34, no. 5, pp. 1454–1495, Sep. 2020.

[6] A. Dempster, D. F. Schmidt, and G. I. Webb, "Minirocket: A very fast (almost) deterministic transform for time series classification," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2021, pp. 248–257.

[7] M. Middlehurst, J. Large, and A. Bagnall, "The canonical interval forest (CIF) classifier for time series classification," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2020, pp. 188–195.

[8] P. Schäfer and U. Leser, "WEASEL 2.0: A random dilated dictionary transform for fast, accurate and memory constrained time series classification," *Mach. Learn.*, vol. 112, no. 12, pp. 4763–4788, Dec. 2023.

[9] M. Middlehurst and A. Bagnall, "Extracting features from random subseries: A hybrid pipeline for time series classification and extrinsic regression," in *Proc. Int. Workshop Adv. Anal. Learn. Temporal Data*. Cham, Switzerland: Springer, Sep. 2023, pp. 113–126.

[10] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Müller, and F. Petitjean, "InceptionTime: Finding AlexNet for time series classification," *Data Mining Knowl. Discovery*, vol. 34, no. 6, pp. 1936–1962, Nov. 2020.

[11] A. Guillaume, C. Vrain, and W. Elloumi, "Random dilated shapelet transform: A new approach for time series shapelets," in *Proc. Int. Conf. Pattern Recognit. Artif. Intell.* Cham, Switzerland: Springer, Jun. 2022, pp. 653–664.
[12] N. Cabello, E. Naghizade, J. Qi, and L. Kulik, "Fast, accurate and explainable time series classification through randomization," *Data Mining Knowl. Discovery*, vol. 38, no. 2, pp. 748–811, Mar. 2024.
[13] A. Theissler, F. Spinnato, U. Schlegel, and R. Guidotti, "Explainable AI for time series classification: A review, taxonomy and research directions," *IEEE Access*, vol. 10, pp. 100700–100724, 2022.
[14] M. G. Baydogan, G. Runger, and E. Tuv, "A bag-of-features framework to classify time series," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2796–2802, Nov. 2013.
[15] Z. S. Harris, "Distributional structure," *Word*, vol. 10, nos. 2–3, pp. 146–162, 1954.
[16] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: A novel symbolic representation of time series," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 107–144, Aug. 2007.
[17] P. Senin and S. Malinchik, "SAX-VSM: Interpretable time series classification using SAX and vector space model," in *Proc. IEEE 13th Int. Conf. Data Mining*, Dec. 2013, pp. 1175–1180.
[18] T. Le Nguyen, S. Gsponer, I. Ilie, M. O'Reilly, and G. Ifrim, "Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations," *Data Mining Knowl. Discovery*, vol. 33, no. 4, pp. 1183–1222, Jul. 2019.
[19] T. L. Nguyen and G. Ifrim, "Fast time series classification with random symbolic subsequences," in *Proc. Int. Workshop Adv. Anal. Learn. Temporal Data*. Cham, Switzerland: Springer, Sep. 2022, pp. 50–65.
[20] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining Knowl. Discovery*, vol. 31, no. 3, pp. 606–660, May 2017.
[21] N. Q. V. Hung and D. T. Anh, "Combining SAX and piecewise linear approximation to improve similarity search on financial time series," in *Proc. Int. Symp. Inf. Technol. Converg. (ISITC)*, Nov. 2007, pp. 58–62.
[22] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, "The great multivariate time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining Knowl. Discovery*, vol. 35, no. 2, pp. 401–449, Mar. 2021.
[23] P. Schäfer and M. Högqvist, "SFA: A symbolic Fourier approximation and index for similarity search in high dimensional datasets," in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 516–527.
[24] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *Knowl. Inf. Syst.*, vol. 3, no. 3, pp. 263–286, Aug. 2001.
[25] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *J. Intell. Inf. Syst.*, vol. 39, no. 2, pp. 287–315, Oct. 2012.
[26] P. Schäfer and U. Leser, "Fast and accurate time series classification with weasel," in *Proc. ACM Conf. Inf. Knowl. Manag.*, 2017, pp. 637–646.
[27] P. Schäfer and U. Leser, "Multivariate time series classification with WEASEL+MUSE," 2017, *arXiv:1711.11343*.
[28] L. Ye and E. Keogh, "Time series shapelets: A novel technique that allows accurate, interpretable and fast classification," *Data Mining Knowl. Discovery*, vol. 22, nos. 1–2, pp. 149–182, Jan. 2011.
[29] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 289–297.
[30] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 392–401.
[31] I. Karlsson, P. Papapetrou, and H. Boström, "Generalized random shapelet forests," *Data Mining Knowl. Discovery*, vol. 30, no. 5, pp. 1053–1085, Sep. 2016.
[32] A. Bostrom and A. Bagnall, "Binary shapelet transform for multiclass time series classification," in *Big Data Analytics and Knowledge Discovery: 17th International Conference, DaWaK 2015, Valencia, Spain, September 1-4, 2015, Proceedings 17*. Springer, 2015, pp. 257–269.
[33] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Inf. Sci.*, vol. 239, pp. 142–153, Aug. 2013.
[34] C. H. Lubba, S. S. Sethi, P. Knaute, S. R. Schultz, B. D. Fulcher, and N. S. Jones, "Catch22: Canonical time-series characteristics: Selected through highly comparative time-series analysis," *Data Mining Knowl. Discovery*, vol. 33, no. 6, pp. 1821–1852, 2019.
[35] A. Ismail-Fawaz, M. Devanne, J. Weber, and G. Forestier, "Deep learning for time series classification using new hand-crafted convolution filters," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2022, pp. 972–981.
[36] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, "TS-CHIEF: A scalable and accurate forest algorithm for time series classification," *Data Mining Knowl. Discovery*, vol. 34, no. 3, pp. 742–775, May 2020.
[37] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016, *arXiv:1603.07285*.
[38] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2. Reading, MA, USA: Addison-Wesley, 2014.
[39] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.
[40] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–10.
[41] N. Saito, *Local Feature Extraction and Its Applications Using a Library of Bases*. New Haven, CT, USA: Yale University, 1994.
[42] L. A. Bennett and Z. S. Abdallah, "RED CoMETS: An ensemble classifier for symbolically represented multivariate time series," in *Proc. Int. Workshop Adv. Anal. Learn. Temporal Data*. Cham, Switzerland: Springer, Sep. 2023, pp. 76–91.
[43] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *Proc. Int. Conf. Web-Age Inf. Manag.* Cham, Switzerland: Springer, Jun. 2014, pp. 298–310.
[44] X. Zhang, Y. Gao, J. Lin, and C. T. Lu, "TapNet: Multivariate time series classification with attentional prototypical network," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 6845–6852.
[45] H. A. Dau, A. Bagnall, K. Kamgar, C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The UCR time series archive," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 6, pp. 1293–1305, Nov. 2019.
[46] A. Bagnall, H. Anh Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh, "The UEA multivariate time series classification archive, 2018," 2018, *arXiv:1811.00075*.
[47] J. Faouzi and H. Janati, "pyts: A Python package for time series classification," *J. Mach. Learn. Res.*, vol. 21, no. 46, pp. 1–6, 2020.
[48] M. Middlehurst, A. Ismail-Fawaz, A. Guillaume, C. Holder, D. Guijo Rubio, G. Bulatova, L. Tsaprounis, L. Mentel, M. Walter, P. Schäfer, and A. Bagnall, "Aeon: A Python toolkit for learning from time series," 2024, *arXiv:2406.14231*.
[49] M. Löning, A. Bagnall, S. Ganesh, V. Kazakov, J. Lines, and F. J. Király, "Sktime: A unified interface for machine learning with time series," 2019, *arXiv:1909.07872*.
[50] A. Ismail-Fawaz, A. Dempster, C. Wei Tan, M. Herrmann, L. Miller, D. F. Schmidt, S. Berretti, J. Weber, M. Devanne, G. Forestier, and G. I. Webb, "An approach to multiple comparison benchmark evaluations that is stable under manipulation of the comparate set," 2023, *arXiv:2305.11921*.
[51] B. Blankertz, G. Curio, and K. R. Müller, "Classifying single trial EEG: Towards brain computer interfacing," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 14, 2001, pp. 1–8.

FRANCESCO SPINNATO received the bachelor's degree in economics and management from the University of Padua, in 2017, the master's degree in data science from the University of Pisa, in 2020, and the Ph.D. degree in data science from Scuola Normale Superiore, in 2024.

In 2024, he collaborated with the Visualization and Data Analytics (VIDA) Research Center, New York University. He is currently a Researcher with the University of Pisa. His research interests include explainable AI (XAI) applied to sequential data, particularly in interpreting black-box models for univariate and multivariate time series.

Dr. Spinnato was a recipient of the 2023–2024 Socint G-Research Italian National Prize for his Ph.D. thesis on "Explanation Methods for Sequential Data Models."

RICCARDO GUIDOTTI received the B.S. and M.S. (cum laude) and Ph.D. degrees in computer science from the University of Pisa, Pisa, Italy, in 2010 and 2013, respectively. His Ph.D. thesis focused on personal data analytics with the University of Pisa.

He was an Intern with IBM Research, Dublin, Ireland, in 2015. He is currently an Assistant Professor (RTD-B) with the Department of Computer Science, University of Pisa, and a member with the Knowledge Discovery and Data Mining Laboratory (KDD Lab), a joint research group with the Information Science and Technology Institute of the National Research Council, Pisa. His research interests include personal data mining, clustering, explainable models, and the analysis of transactional data.

Dr. Guidotti received the IBM Fellowship Program.

ANNA MONREALE received the Ph.D. degree in computer science from the University of Pisa, Pisa, Italy, in June 2011, with a dissertation on privacy by design in data mining.

She is currently an Associate Professor with the Department of Computer Science, University of Pisa, and a member with the Knowledge Discovery and Data Mining Laboratory (KDD Lab), a joint research group with the Information Science and Technology Institute, National Research Council, Pisa. Her research interests include big data analytics, social networks, and the privacy issues arising in mining these types of social and human-sensitive data. She focuses particularly on evaluating privacy risks during analytical processes and designing privacy-by-design technologies in the era of big data.

MIRCO NANNI received the degree and the Ph.D. degree in computer science from the University of Pisa, in 1997 and 2002, respectively. He is currently a Senior Researcher with ISTI-CNR, Pisa, and the Head of the KDD Laboratory. His research interests include several aspects of human mobility, such as data mining on spatio-temporal data, applications for transportation and smart cities, mobile phone data analysis, and big data for official statistics.

Open Access funding provided by 'Università di Pisa' within the CRUI CARE Agreement