Data Mining and Knowledge Discovery (2024) 38:206–236
https://doi.org/10.1007/s10618-023-00969-x

Z-Time: efficient and effective interpretable multivariate time series classification

Zed Lee[1] · **Tony Lindgren**[1] · **Panagiotis Papapetrou**[1]

Received: 11 December 2022 / Accepted: 24 July 2023 / Published online: 5 September 2023

Abstract

Multivariate time series classification has become popular due to its prevalence in many real-world applications. However, most state-of-the-art focuses on improving classification performance, with the best-performing models typically opaque. Interpretable multivariate time series classifiers have been recently introduced, but none can maintain sufficient levels of efficiency and effectiveness together with interpretability. We introduce Z-Time, a novel algorithm for effective and efficient interpretable multivariate time series classification. Z-Time employs temporal abstraction and temporal relations of event intervals to create interpretable features across multiple time series dimensions. In our experimental evaluation on the UEA multivariate time series datasets, Z-Time achieves comparable effectiveness to state-of-the-art non-interpretable multivariate classifiers while being faster than all interpretable multivariate classifiers. We also demonstrate that Z-Time is more robust to missing values and inter-dimensional orders, compared to its interpretable competitors.

Keywords Multivariate time series · Temporal abstraction · Event interval sequences · Interpretable multivariate time series classification

Responsible editor: Charalampos Tsourakakis.

✉ Zed Lee
zed.lee@dsv.su.se

Tony Lindgren
tony@dsv.su.se

Panagiotis Papapetrou
panagiotis@dsv.su.se

[1] Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden

1 Introduction

The problem of time series classification has been drawing tremendous attention as time series can be found in several application areas, such as healthcare (Kaushik et al. 2020), predictive maintenance (Kiangala and Wang 2020), and human activity recognition (Xi et al. 2018). Time series datasets are typically generated by real-time sensors resulting in considerable amounts of collected instances of one (**univariate** time series) or oftentimes multiple (**multivariate** time series) dimensions per instance.

The complexity of multivariate time series classification can easily become prohibitive since interactions or relations between different dimensions may be detrimental to the classification outcome. As a result, multivariate time series classifiers are becoming more complex in terms of their underlying model structures or feature spaces (Ruiz et al. 2021). However, due to the inherent complexity of the problem and the existing opaque solutions, interpretability becomes both a challenge and a requirement.

There have been a few recent approaches towards interpretable time series classification. The first line of research uses symbolic discretization (Lin et al. 2007) to create the feature space together with a linear classifier. For univariate time series, multiple representation sequential learner (MR-SEQL) (Le Nguyen et al. 2019) combines a symbolic sequential learner with two discretization techniques, i.e., symbolic aggregate approximation (SAX) (Lin et al. 2007) and symbolic Fourier approximation (SFA) (Schäfer 2015) to create the feature space representation. Interpretability is then obtained from the class discriminatory subsequences. In a similar manner, pattern-based embedding for time series classification (PETSC) (Feremans et al. 2022) and its multiple representation version, MR-PETSC, employ standard frequent pattern mining with a relative duration constraint instead of a sequential learner to catch non-contiguous patterns as well as subsequences. While MR-PETSC supports multivariate time series classification, interpretability is studied only for univariate problems, so it is unclear how the pattern-based features can be used to interpret multivariate time series.

Furthermore, MR-PETSC is hampered by three limitations. Firstly, it does not exploit any inter-dimensional relations, which in many cases limits their classification performance. Secondly, it applies multiple sliding windows over the time series resulting in a quadratic (in time series length) number of discretization functions for each time point in the worst case. Finally, the obtained sequential features may be lengthy, containing repetitive event labels (e.g., *aaaabcdcaada*), hence impairing interpretability.

On the other hand, explainable-by-design ensemble (XEM) (Fauvel et al. 2022) provides explanations by considering multiple dimensions concurrently. Instead of symbolic discretization, it applies a sliding window across the time series dimensions concurrently and uses the values in the window together with their timestamps to create the feature space. For each instance, XEM provides a prediction probability for each stride of the sliding window and then uses the highest probability to determine the class label and obtain a discriminatory region. The

final classifier is based on extreme gradient boosting (Chen and Guestrin 2016) in combination with bagging. Due to this combination, XEM is considerably slow and memory-consuming, while the resulting model is opaque. Moreover, the explanation provided by XEM is a single time region across all dimensions having the highest prediction probability, which may be insufficient when multiple regions are responsible for classification.

In this paper, we claim that the quality of a multivariate time series classifier should be assessed in terms of efficiency, effectiveness, and interpretability. However, no single multivariate time series classifier manages to achieve competitive trade-offs between all three requirements. We, thus, propose `Z-Time`, an interpretable multivariate time series classifier that is also effective and efficient. `Z-Time` transforms the time series into *event sequences*, using various discretization techniques, and finally abstract them to *event intervals*, which maintain the start and end times of the continuous events with the same label. For example, time series $\langle 1, 2, 3, 4, 5 \rangle$ can be transformed into a sequence of events $\langle a, a, a, b, b \rangle$ and further into two event intervals (a, 1, 3) and (b, 4, 5). Finally, `Z-Time` exploits *temporal relations* of the generated event intervals and uses their frequencies as features for classification. Interpretability is induced by the event intervals and their inter-dimensional temporal relations.

Our main **contributions** and **novelty** of `Z-Time` include:

- *Interpretability* `Z-Time` employs *temporal abstraction* and builds temporal relations of event intervals to create interpretable features across multiple time series dimensions. Intepretability is demonstrated on two UEA multivariate time series (Bagnall et al. 2018) and three synthetic datasets.
- *Effectiveness* `Z-Time` achieves competitive classification performance compared to the state-of-the-art algorithms, using only an interpretable linear classifier on the UEA multivariate time series datasets.
- *Efficiency* `Z-Time` is faster than the two interpretable competitors, specially achieving speedups of over an order of magnitude against XEM on the UEA multivariate time series datasets.
- *Robustness* `Z-Time`'s structure naturally handles missing data without applying interpolation, which is also evident by our experiments where `Z-Time` is more robust to missing data than its two competitors.
- *Repeatability* Our code is available online as well as our synthetic dataset generator together with all the used datasets and experimental results.[1]

The rest of this paper is organized as follows: Sect. 2 presents the related work on multivariate time series classification; Sect. 3 covers the preliminaries; Sect. 4 describes our method `Z-Time`; Sect. 5 presents our experimental setup and results; and Sect. 6 presents our conclusions and directions for future work.

[1] https://github.com/zedshape/ztime.

2 Related work

Many attempts exist in the literature with the objective of optimizing classification performance for multivariate time series classification. The simplest baselines are distance-based methods, such as dynamic time warping (DTW) and its variants (Shokoohi-Yekta et al. 2017). Heterogeneous ensemble methods, such as hierarchical vote collective of transformation-based ensembles (HIVE-COTE) (Lines et al. 2016) and its improved version (Middlehurst et al. 2021) combine non-identical classifiers and achieve the best performance on the datasets in the UEA/UCR repository. However, they suffer from slow runtime and are not interpretable due to their complex ensemble structures.

Homogeneous ensembles use common feature types, such as discriminatory subsequences (Karlsson et al. 2016), or summary statistics over time intervals (Cabello et al. 2020; Middlehurst et al. 2020). Bag-of-SFA-symbols (BOSS) (Schäfer 2015) creates symbolic sequences using SFA, so the resulting sequences lose the original temporal order; thus, they are not interpretable. There have been many BOSS variants for univariate time series (Schäfer and Leser 2017a; Large et al. 2019; Middlehurst et al. 2019) and one for multivariate time series (Schäfer and Leser 2017b), extended from Schäfer and Leser (2017a). They also focus on classification performance by creating an ensemble of BOSS classifiers with a severe impact on interpretability.

While ensemble methods dominate the area, another line of research uses convolutional neural network architectures, such as residual networks (Wang et al. 2017) and InceptionTime (Ismail Fawaz et al. 2020) for univariate time series. For multivariate time series, Karim et al. (2019) uses both LSTM and CNN layers together. These methods also succeed in achieving competitive classification performance, but suffer from poor runtime and a complete lack of interpretability. Random convolutional kernel transform (ROCKET) (Dempster et al. 2020) and its variant (Dempster et al. 2021) generate features from the random convolutional kernels with a single linear classifier, outperforming other ensemble methods in terms of runtime by an order of magnitude, while not being significantly less accurate. However, their convolution-based feature space is inherently not interpretable.

On a final note, as mentioned earlier, `Z-Time` employs event intervals to generate temporal features for multivariate time series classification. Event intervals have been proven promising for representing other complex sequences to facilitate different learning tasks, such as classification of electronic health records (Sheetrit et al. 2019; Rebane et al. 2021) and disproportionality analysis of vehicle failure histograms (Lee et al. 2021), based on features in the form of frequent temporal patterns (Lee et al. 2020; Ho et al. 2022).

3 Preliminaries

Let $\mathbf{t} = \langle t_1, \ldots, t_m \rangle$ be a *time series variable* of *length* $|\mathbf{t}| = m$, i.e., spanning over m time points. One or more time series variables form a *time series instance* $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_d\}$ of $|\mathbf{T}| = d$ time series variables, which we refer to as the *dimensions* of the time series instance. If $d = 1$, $\mathbf{T}$ is a *univariate time series instance*,

while if $d > 1$, $\mathbf{T}$ is a *multivariate time series instance*. A *time series collection* $\mathcal{T} = \{\mathbf{T}_1, \ldots, \mathbf{T}_n\}$ is a set of $|\mathcal{T}| = n$ time series instances, with each $\mathbf{T}_k \in \mathcal{T}$ having its own number of dimensions. The number of dimensions of $\mathcal{T}$ is defined as $max(\{|\mathbf{T}_k| : \mathbf{T}_k \in \mathcal{T}\})$, while the length of $\mathcal{T}$ is the maximum length of all variables in $\mathcal{T}$, i.e., $max(\{|\mathbf{t}_{k,j}| : \mathbf{t}_{k,j} \in \mathbf{T}_k, \mathbf{T}_k \in \mathcal{T}\})$. Furthermore, each time series instance $\mathbf{T}_k \in \mathcal{T}$ is assigned with a class label $y_k \in \mathbf{y}$, where $\mathbf{y}$ is a list of class labels for each instance such that $|\mathcal{T}| = |\mathbf{y}|$. We also define a set of unique class labels $\mathcal{Y}$, such that $y_k \in \mathbf{y} \Rightarrow y_k \in \mathcal{Y}$.

Example 1 Consider two time series instances, a univariate instance $\mathbf{T}_1 = \{\langle 1, 2, 3, 3, 4\rangle\}$ of length five, and a multivariate instance $\mathbf{T}_2 = \{\langle 1, 2, 3\rangle, \langle 1, 2, 3, 4\rangle, \langle 1, 2, 3\rangle, \langle 1, 2\rangle\}$ with four dimensions of variable length. These two instances form a time series collection $\mathcal{T} = \{\mathbf{T}_1, \mathbf{T}_2\}$ of length five with four dimensions.

Temporal abstraction is a process of representing time series in a more abstract form, often using a symbolic representation, in the time (i.e., summarization) and value (i.e., discretization) dimensions. To avoid ambiguity, in this paper, we define temporal abstraction as the whole process of creating *event interval sequences* from time series instances, following Sheetrit et al. (2019).

To perform temporal abstraction, given a set $\Sigma = \{\epsilon_1, \ldots, \epsilon_\lambda\}$ of λ event labels, a discretization function maps a time series variable $\mathbf{t} = \langle t_1, \ldots, t_m\rangle$ to an *event sequence* $\mathbf{e} = \langle e_1, \ldots, e_m\rangle$, with $|\mathbf{t}| = |\mathbf{e}|$, such that each $t_i \in \mathbf{t}$ is mapped to an event label $\epsilon_a \in \Sigma$, creating an event $e_i = \epsilon_a$. The mapping is conducted using a discretization function $g(\cdot)$, such that $g(\mathbf{t}, \lambda) = \mathbf{e}$. Given the desired number of event labels λ, the discretization function defines Σ alongside a *discretization boundary* $[t_{min}^{\epsilon_a}, t_{max}^{\epsilon_a})$ for each event label $\epsilon_a \in \Sigma$. Each $t_i \in \mathbf{t}$ is then mapped to ϵ_a such that $t_i \in [t_{min}^{\epsilon_a}, t_{max}^{\epsilon_a})$.

In this paper, we employ three different discretization functions that use $\mathbf{t}$ to define the discretization boundaries by dividing the values of $\mathbf{t}$ into λ bins:

- *Equal width discretization (EWD)*: Assuming $\mathbf{t}$ follows a uniform distribution, discretization boundaries are defined so that all event labels have value ranges of equal length, i.e., $t_{max}^{\epsilon_a} - t_{min}^{\epsilon_a} = t_{max}^{\epsilon_b} - t_{min}^{\epsilon_b}$ (Yang and Webb 2002). This function is denoted as $g^{EWD}(\cdot)$.
- *Equal frequency discretization (EFD)*: Discretization boundaries are defined so that each event label occurs with the same frequency in $\mathbf{e}$, i.e., $|\{e_i \in \mathbf{e} : e_i = \epsilon_a\}| = |\{e_i \in \mathbf{e} : e_i = \epsilon_b\}|$ (Yang and Webb 2002). This function is denoted as $g^{EFD}(\cdot)$.
- *Symbolic aggregate approximation (SAX)*: SAX receives a window size w and an event label size λ to perform both discretization and summarization. First, it applies piecewise aggregate approximation (PAA) (Keogh et al. 2001) to summarize every w time points with their average value. Then discretization boundaries are defined assuming $\mathbf{t}$ follows a normal distribution (Lin et al. 2007), using the points that produce λ equi-sized areas under the normal distribution curve. In this paper, to align SAX with other discritization functions,

we denote $g^{SAX}(\cdot)$ as SAX with $w = 1$ (i.e., without PAA) and use PAA separately with any discretization function.

We apply PAA to reduce complexity and eliminate noise that can be present in the time series. However, PAA may potentially result in eliminating class discriminative trends or peaks. Thus, Z-Time uses both PAA and non-PAA representations, with all three discretization functions, to provide different views of the time series variables, with and without summarization, on the event interval space. This approach enables us to generate features based on the relations between these different representations.

Example 2 Given $\mathbf{t} = \langle 1, 3, 4, 4, 5, 9\rangle$ and $\lambda = 3$, we have $g^{EWD}(\mathbf{t}, \lambda) = \langle A, A, B, B, B, C\rangle$, $g^{EFD}(\mathbf{t}, \lambda) = \langle D, D, E, E, F, F\rangle$, and $g^{SAX}(\mathbf{t}, \lambda) = \langle G, G, H, H, H, I\rangle$. PAA of $\mathbf{t}$ with $w = 2$ is $\langle \frac{1+3}{2}, \frac{4+4}{2}, \frac{5+9}{2}\rangle = \langle 2, 4, 7\rangle$. To ensure compatibility between PAA and non-PAA representations in the time dimension, we maintain the original time granularity in the PAA representation by repeating each PAA value w times, resulting in $\langle 2, 2, 4, 4, 7, 7\rangle$.

Although PAA summarizes the time dimension, redundant events can still occur after discretization. To complete the temporal abstraction process, events with the same label that occur continuously over time are merged and form an event interval, and a set of event intervals defines an event interval sequence. Next, we provide more formal definitions for these two concepts.

Definition 1 (event interval) An event interval is a triplet $s = (\epsilon^s, \tau^s_{start}, \tau^s_{end})$, where $\epsilon^s \in \Sigma$ and τ^s_{start}, τ^s_{end} correspond to the start and end times at which an event with event label ϵ^s continuously occurs. It holds $\tau^s_{start} \leq \tau^s_{end}$, where an instantaneous event interval can also be defined when $\tau^s_{start} = \tau^s_{end}$.

There can be many event intervals with the same event label, since events with one label can occur at any time point. For example, if $\mathbf{e} = \langle A, B, A, A, C\rangle$, we define two event intervals for event label a: (A, 1, 1) and (A, 3, 4).

Definition 2 (event interval sequence) An event interval sequence, $\mathbf{s} = \langle s_1, \ldots, s_M\rangle$ is a sequence of M event intervals. We keep the temporal order of event intervals in $\mathbf{s}$ by their start times. In the case of ties, the end times are sorted in ascending order. If ties still exist, we sort these event intervals in event labels' lexicographic order.

According to Allen (1983), there are seven temporal relation types between two event intervals (Fig. 1): $\mathcal{I} = \{$*follows*, *meets*, *matches*, *overlaps*, *contains*, *left-contains*, *right-contains*$\}$. Given two event intervals s_i and s_j, we define their temporal relation $R(s_i, s_j)$ as a triplet $(r, \epsilon^{s_i}, \epsilon^{s_j})$, with $\epsilon^{s_i}, \epsilon^{s_j}$ being their event labels and $r \in \mathcal{I}$ denoting their temporal relation type. A temporal relation is only defined from the earlier interval to the later interval to avoid redundant information, following the order in Definition 2.

While temporal relations are defined by event interval pairs based on their relative temporal positions, we may be interested in how often a particular relation type occurs between two event labels throughout a given dataset.

Definition 3 (occurrence set) Given a pair of event labels (ϵ_a, ϵ_b), a temporal relation type $r \in \mathcal{I}$, and an event interval sequence $\mathbf{s}$, the *occurrence set* $\Omega(\cdot)$ of

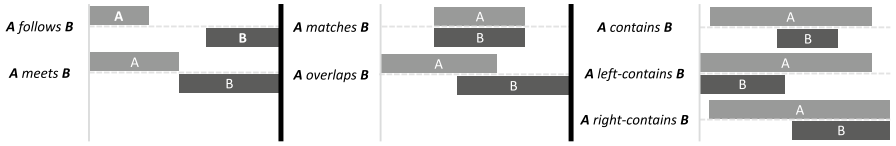

Fig. 1 Seven temporal relation types two event intervals can have, as defined in Allen's temporal logic (Allen 1983)

$((r, \epsilon_a, \epsilon_b), \mathbf{s})$ is the set of all possible event interval pairs (s_i, s_j) with event labels (ϵ_a, ϵ_b), respectively, forming temporal relation $R(s_i, s_j) = (r, \epsilon_a, \epsilon_b)$, i.e.:

$$\Omega((r, \epsilon_a, \epsilon_b), \mathbf{s}) = \{(s_i, s_j) : s_i \in \mathbf{s}, s_j \in \mathbf{s}, R(s_i, s_j) = (r, \epsilon_a, \epsilon_b)\}.$$

Definition 4 (horizontal support) Given an event interval sequence $\mathbf{s}$, an event label pair (ϵ_a, ϵ_b), and a temporal relation type $r \in \mathcal{I}$, *horizontal support* $H(\cdot)$ of $((r, \epsilon_a, \epsilon_b), \mathbf{s})$ is defined as the number of $s_i \in \mathbf{s}$ that yield a temporal relation $(r, \epsilon_a, \epsilon_b)$ with any event interval $s_j \in \mathbf{s}$ respectively, or simply

$$H((r, \epsilon_a, \epsilon_b), \mathbf{s}) = |\{s_i : (s_i, s_j) \in \Omega((r, \epsilon_a, \epsilon_b), \mathbf{s})\}|.$$

Given a single event interval s_i, with $\epsilon^{s_i} = \epsilon_a$, there can be multiple occurrences of the same temporal relation $(r, \epsilon^{s_i}, \epsilon_b)$ with more than one event intervals $s_j \in \mathbf{s}$ having $\epsilon^{s_j} = \epsilon_b$. Horizontal support only considers them once to prevent redundant counts of similar information.

Based on the above, our problem can be formulated as follows:

Problem 1 (time series classification) Given a time series collection $\mathcal{T}$, its corresponding labels $\mathbf{y}$, a set of unique class labels $\mathcal{Y}$, we would like to train a classification function $f : \mathcal{T} \rightarrow \mathcal{Y}$, with $f(\mathbf{T}_k) = \hat{y}_k \in \mathcal{Y}$, such that the expected loss on $(\mathcal{T}, \mathbf{y})$, denoted as $E_{k \leq |\mathcal{T}|} l(y_k, f(\mathbf{T}_k))$, is minimized. We use a 0/1 loss function $l(\cdot)$, i.e., $(y \neq y' \Rightarrow l(y, y') = 1) \wedge (y = y' \Rightarrow l(y, y') = 0)$.

Our goal in this paper is to construct a multivariate time series classifier that solves Problem 1 using temporal abstraction and temporal relations between event intervals.

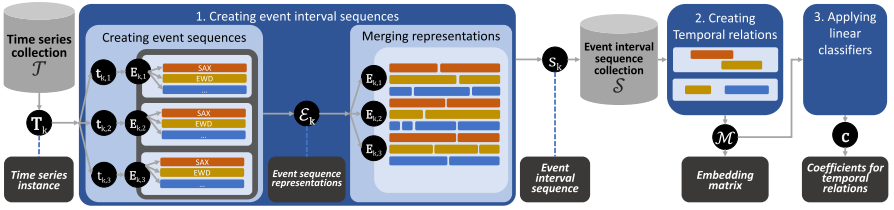

Fig. 2 An example of the three steps of `Z-Time`, where those steps are marked as blue boxes

4 Z-Time

Algorithm 1: Z-Time

Input : $\mathcal{T}$, λ_1: event label size for time series variables PAA forms, λ_2: event label size for finite differences, w: window size, σ: step size, G: discretization functions
Result: $\mathcal{M}$: embedding matrix, $\mathbf{c}$: coefficients from a linear model

$\mathcal{S} \leftarrow \{\}$
for $\mathbf{T}_k \in \mathcal{T}$ **do**
  for $\mathbf{t}_{k,j} \in \mathbf{T}_k$ **do**
    $\mathbf{t}^{D1}_{k,j} \leftarrow \langle t_{k,j,i} - t_{k,j,i-1} : t_{k,j,i} \in \mathbf{t}_{k,j}, t_{k,j,i-1} \in \mathbf{t}_{k,j} \rangle$
    $\mathbf{t}^{D2}_{k,j} \leftarrow \langle t^{D1}_{k,j,i} - t^{D1}_{k,j,i-1} : t^{D1}_{k,j,i} \in \mathbf{t}^{D1}_{k,j}, t^{D1}_{k,j,i-1} \in \mathbf{t}^{D1}_{k,j} \rangle$
    $\mathbf{t}^{PAA}_{k,j} \leftarrow \mathrm{PAA}(\mathbf{t}_{k,j}, w)$
    $\mathbf{E}_{k,j} \leftarrow \{\bigcup_{\gamma \in \{\emptyset, PAA\}} \bigcup_{g \in \mathcal{G}} g(\mathbf{t}^{\gamma}_{k,j}, \lambda_1)\} \cup \{\bigcup_{\gamma \in \{D1,D2\}} \bigcup_{g \in \mathcal{G} - \{g^{SAX}\}} g(\mathbf{t}^{\gamma}_{k,j}, \lambda_2)\}$
  $\mathcal{E}_k \leftarrow \cup_{j=1}^{|\mathbf{T}_k|} \mathbf{E}_{k,j}$
  $\mathbf{s}_k \leftarrow \mathrm{sort}(\bigoplus_{\mathbf{E}_{k,j} \in \mathcal{E}_k} \bigoplus_{\mathbf{e}^p_{k,j} \in \mathbf{E}_{k,j}} \mathrm{Merge}(\mathbf{e}^p_{k,j}))$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{s}_k\}$
$\mathcal{M} \leftarrow \mathrm{createEmbeddingMatrix}(\mathcal{S}, \sigma)$
$\mathbf{c} \leftarrow \mathrm{trainLinearClassifier}(\mathcal{M})$
return $\mathcal{M}, \mathbf{c}$

We present Z-Time, an efficient and effective three-step algorithm for interpretable multivariate time series classification. Given a time series collection $\mathcal{T}$, the feature space is created by converting $\mathcal{T}$ into a set of temporal relations across multiple dimensions, keeping their horizontal support. The first step converts each time series instance in $\mathcal{T}$ to an event interval sequence by creating multiple event sequence representations for each variable and merging them into a single event interval sequence. The second step defines the *embedding matrix* of $\mathcal{T}$ that represents each time series instance by a set of horizontal support values of temporal relations. The third and last step trains a linear model which provides interpretability by classifying multivariate time series instances based on the presence of temporal relations. These steps are outlined in Fig. 2 and Algorithm 1 and are described in more detail below.

4.1 Creating event interval sequences

The first step of Z-Time receives a time series collection $\mathcal{T}$, a PAA window size w, and two event label sizes λ_1 and λ_2 and creates event interval sequences. This is achieved by iterating over $\mathcal{T}$ and converting each time series instance $\mathbf{T}_k \in \mathcal{T}$ into an event interval sequence $\mathbf{s}_k$, and finally returning a collection of event interval sequences $\mathcal{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$. More concretely, the following two substeps are applied to each $\mathbf{T}_k$: (1) generating a set of different event sequence representations for each variable $\mathcal{E}_k = \{\mathbf{E}_{k,1}, \ldots, \mathbf{E}_{k,|\mathbf{T}_k|}\}$, (2) merging all event sequences in $\mathcal{E}_k$ to an event interval sequence $\mathbf{s}_k$, hence creating the collection of event interval sequences $\mathcal{S}$.

4.1.1 Creating event sequences

This substep receives a time series instance $\mathbf{T}_k \in \mathcal{T}$, a PAA window size w, and two event label sizes λ_1, λ_2 as input. It then iterates through each variable $\mathbf{t}_{k,j} \in \mathbf{T}_k$ and converts it into a set of event sequences $\mathbf{E}_{k,j}$, where each event sequence represents a different view of $\mathbf{t}_{k,j}$. This substep finishes when all time series variables in $\mathbf{T}_k$ are converted, resulting in $\mathcal{E}_k$. This is done by applying three discretization functions (i.e., g^{EWD}, g^{EFD}, and g^{SAX} defined in Sect. 3) to four different representations of $\mathbf{t}_{k,j}$. More concretely, apart from the original representation $\mathbf{t}_{k,j}$, we also consider the first- and second-order finite differences ($\mathbf{t}_{k,j}^{D1}$ and $\mathbf{t}_{k,j}^{D2}$) that capture trend information in time series (Górecki and Łuczak 2013), as well as the PAA representation of $\mathbf{t}_{k,j}$, denoted as $\mathbf{t}_{k,j}^{PAA}$, over the window size w.

The application of various discretization functions is very helpful for the classification process as it provides concurrent views of the time series instances used in the next step when temporal relations are defined. For the time series variable $\mathbf{t}_{k,j}$ and its PAA form $\mathbf{t}_{k,j}^{PAA}$, all three discretization functions are applied. However, we do not apply SAX to the finite difference forms ($\mathbf{t}_{k,j}^{D1}$ and $\mathbf{t}_{k,j}^{D2}$), as the values do not have enough diversity to match the normal distribution assumption of SAX. We define two event label sizes, one for the original time series variable and its PAA form (λ_1) and one for the two finite difference forms (λ_2), as the finite difference representations may have different value ranges. Hence, for each $\mathbf{t}_{k,j} \in \mathbf{T}_k$, we define $\mathbf{E}_{k,j}$, a set of ten event sequence representations where each $\mathbf{t}_{k,j}$ and its corresponding PAA form $\mathbf{t}_{k,j}^{PAA}$ generate three event sequences, while the two finite differences $\mathbf{t}_{k,j}^{D1}$ and $\mathbf{t}_{k,j}^{D2}$ produce two event sequences each, resulting in a total of six and four event sequence representations, as follows:

$$\mathbf{E}_{k,j} = \{ \bigcup_{\gamma \in \{\emptyset, PAA\}} \bigcup_{g \in \mathcal{G}} g(\mathbf{t}_{k,j}^{\gamma}, \lambda_1) \} \cup \{ \bigcup_{\gamma \in \{D1, D2\}} \bigcup_{g \in \mathcal{G} - \{g^{SAX}\}} g(\mathbf{t}_{k,j}^{\gamma}, \lambda_2) \},$$

where $\mathcal{G} = \{g^{EWD}, g^{EFD}, g^{SAX}\}$ and $\mathbf{t}_{k,j}^{\emptyset}$ refers to the original time series representation.

Example 3 In Fig. 3, a time series instance $\mathbf{T}_k$ of length six with three dimensions is transformed to event sequences with $\{w : 2, \lambda_1 : 3, \lambda_2 : 3\}$. The result of the discretization process is depicted for each discretization function. While there are three time series variables, we only show the first one, i.e., $\mathbf{t}_{k,1}$, but the same process is applied to all variables. For illustration purposes, for the event labels used in the figure we use the convention that increasing alphabetical order also corresponds to increasing value order. For example, letter *H* corresponds to a lower value than letter *I*. We observe that each event sequence represents a different view of variable $\mathbf{t}_{k,1}$. For the first four time points, $\mathbf{e}_{k,1}^{PAA,EWD}$ and $\mathbf{e}_{k,1}^{PAA,EFD}$ convey the same increasing value pattern (*J* to *K* versus *M* to *N*), while $\mathbf{e}_{k,1}^{PAA,SAX}$ indicates a steady value pattern as it contains four repetitions of *P*.

Finally, $\mathbf{T}_k \in \mathcal{T}$ is converted to a list of event sequence sets by applying the same process for all $\mathbf{t}_{k,j} \in \mathbf{T}_k$, creating $\mathcal{E}_k = \cup_{j=1}^{|\mathbf{T}_k|} \mathbf{E}_{k,j}$, which is the output of this

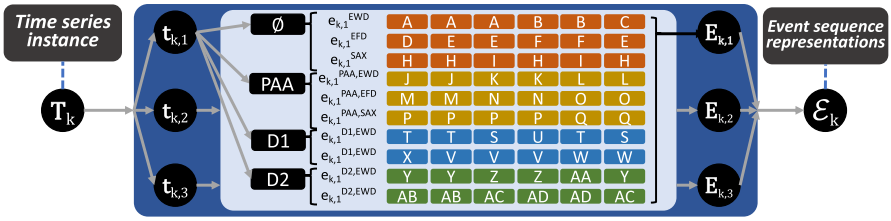

Fig. 3 An example of the substep of creating event sequences. The letters that come later in the alphabet correspond to higher values (e.g., $t^B_{max} = t^C_{min}$)

substep. An example is shown in Fig. 3, where the three event sequence sets ($\mathbf{E}_{k,1}, \mathbf{E}_{k,2}$, and $\mathbf{E}_{k,3}$) are generated and form $\mathcal{E}_k$.

4.1.2 Merging event sequence representations

The objective of this substep is to define an event interval sequence $\mathbf{s}_k$ using set $\mathcal{E}_k$ from the previous substep. An event interval sequence $\mathbf{s}_k$ for a time series instance $\mathbf{T}_k$ is formed by merging all event sequence representations in $\mathcal{E}_k$ defined for $\mathbf{T}_k$ in the form of event intervals and sorting the collected event intervals following Definition 2. This means we no longer have multiple representations but a single event interval sequence $\mathbf{s}_k$ for $\mathbf{T}_k$.

For each event sequence representation $\mathbf{e}^p_{k,j} \in \mathbf{E}_{k,j}$ for a time series variable $\mathbf{t}_{k,j}$, where p is a pair of discretization functions and time series representations used for $\mathcal{E}_k$ (i.e., $p \in (\mathcal{G} \times \{\emptyset, PAA\}) \cup ((\mathcal{G} - g^{SAX}) \times \{D1, D2\})$, we transform each event sequence into a set of event intervals. To do this, we define a function $Merge(\cdot)$ that takes an event sequence $\mathbf{e}$ as input and with the help of a recursive function $merge_i(\cdot)$ over the time points of $\mathbf{e}$, it creates the set of event intervals for $\mathbf{e}$ as follows:

$$Merge(\mathbf{e}) = merge_1(e_1, 1),$$

$$merge_i(\epsilon_{prev}, \tau_{prev}) = \begin{cases} merge_{i+1}(e_i, i) & \text{if } i = 1 \\ merge_{i+1}(\epsilon_{prev}, \tau_{prev}) & \text{if } \epsilon_{prev} = e_i \\ \langle(\epsilon_{prev}, \tau_{prev}, i-1)\rangle \oplus merge_{i+1}(e_i, i) & \text{if } \epsilon_{prev} \neq e_i \end{cases},$$

where $\oplus$ denotes concatenation of two sequences, and i iterates over the time points of $\mathbf{e}$. Then we create $\mathbf{s}_k$ by collecting event intervals from all event sequences in $\mathcal{E}_k$ created by *Merge* and sorting them following Definition 2:

$$\mathbf{s}_k = sort(\bigoplus_{\mathbf{E}_{k,j} \in \mathcal{E}_k} \bigoplus_{\mathbf{e}^p_{k,j} \in \mathbf{E}_{k,j}} Merge(\mathbf{e}^p_{k,j})).$$

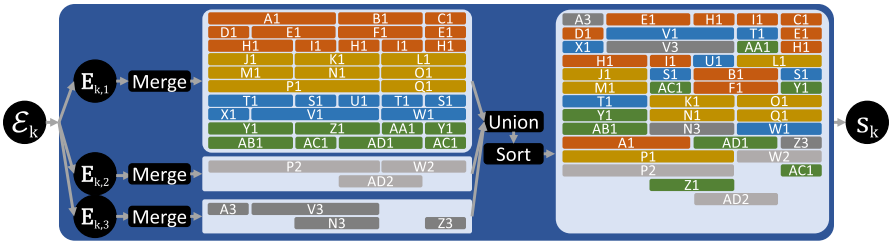

Fig. 4 An example of the substep of merging event sequence representations

Each event sequence has its own set of event labels, as we would like to distinguish between events belonging to different dimensions. As a result, different event sequences do not share any common event labels.

Example 4 Fig. 4 shows the process of merging event sequence representations for a time series instance $\mathbf{T}_k$ into an event interval sequence $\mathbf{s}_k$. First, consecutive events are merged into event intervals. For instance, three *A1* events are merged into a single interval $(A1, 1, 3)$. The events created by PAA can also be further summarized, as illustrated by event interval $(P1, 1, 4)$, which is created by merging four *P1* events. These event intervals then form a single event interval sequence $\mathbf{s}_k = \langle (A3, 1, 1), \ldots, (V3, 2, 4), \ldots, (AC1, 6, 6) \rangle$, where the suffix of each event label indicates the dimension to distinguish event intervals from different variables.

In summary, the first step of `Z-Time` iterates through each time series instance $\mathbf{T}_k \in \mathcal{T}$ and creates the corresponding event interval sequence $\mathbf{s}_k$ through the two substeps described earlier. Each $\mathbf{s}_k$ is then stored in $\mathcal{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, which is the final output of this step (Algorithm 1, lines 9-10), and is later used to create the hash table structure in the next step (Algorithm 1, line 11).

4.2 Creating temporal relations

The second step of `Z-Time` receives a collection of event interval sequences $\mathcal{S}$ and creates the embedding matrix $\mathcal{M}$, which represents each time series instance by the set of horizontal support values of the temporal relations that describe it. We naturally consider inter-dimensional relations, since an event interval sequence contains the merger of all event intervals formed in each dimension of the time series instance.

We employ a special hash table structure, referred to as *RS*-Hashtable, which serves two main purposes: (1) storing the horizontal support values for each temporal relation and (2) keeping track of the locations of the event interval pairs forming each temporal relation (i.e., occurrences). *RS*-Hashtable has two components: *R*-Hashtable and *S*-Hashtable. *R*-Hashtable is defined only once, and it uses all possible temporal relations as keys, while the values are the *S*-Hashtable instances corresponding to each temporal relation. *S*-Hashtable is defined for each temporal relation and stores the event interval sequences containing a particular temporal relation from *R*-Hashtable as keys along with its occurrence set and horizontal support in each sequence as values.

More formally, let $\mathcal{H} = f : K \rightarrow V$ denote a hash table $\mathcal{H}$ using function $f(\cdot)$ that maps a set of keys K to a set of values V. S-Hashtable, denoted as $\mathcal{H}^S_{(r,\epsilon_a,\epsilon_b)} = f : K^S_{(r,\epsilon_a,\epsilon_b)} \rightarrow V^S_{(r,\epsilon_a,\epsilon_b)}$, is defined for each temporal relation $(r, \epsilon_a, \epsilon_b)$ and is accessed by R-Hashtable, denoted as $\mathcal{H}^R = f : K^R \rightarrow V^R$, which holds temporal relations as its keys, i.e., $\mathcal{H}^R((r, \epsilon_a, \epsilon_b)) = \mathcal{H}^S_{(r,\epsilon_a,\epsilon_b)}$. The keys and values of RS-Hashtable are formally defined as follows:

$$
\begin{aligned}
K^R &= \{(r, \epsilon_a, \epsilon_b) : \epsilon_a \in \Sigma, \epsilon_b \in \Sigma\} \\
V^R &= \{\mathcal{H}^S_{(r,\epsilon_a,\epsilon_b)} : \epsilon_a \in \Sigma, \epsilon_b \in \Sigma\} \\
K^S_{(r,\epsilon_a,\epsilon_b)} &= \{k : \mathbf{s}_k \in \mathcal{S}, (\exists (s_i, s_j) \in \mathbf{s}_k \times \mathbf{s}_k : R(s_i, s_j) = (r, \epsilon_a, \epsilon_b))\} \\
V^S_{(r,\epsilon_a,\epsilon_b)} &= \{(\Omega((r, \epsilon_a, \epsilon_b), \mathbf{s}_k), H((r, \epsilon_a, \epsilon_b), \mathbf{s}_k)) : \mathbf{s}_k \in \mathcal{S}\}.
\end{aligned}
$$

In the interest of computation time but also for simplicity and interpretability, we restrict the temporal relations space in two ways. First, we consider a smaller set of possible temporal relations by merging {*overlaps, left-contains, right-contains, meets, contains*} to a single temporal relation, i.e., *overlaps*. We apply this merging since our empirical observation suggests that, depending on the PAA window, event intervals tend to share either their start time or end time, and create left- or right-matching boundaries that are not necessarily meaningful beyond indicating time overlaps. The above simplification, which has also been applied in earlier research (Thiel et al. 2010; Moskovitch and Shahar 2015), additionally provides flexibility to our temporal relations when irregular outlier values occur at the time boundaries of the event intervals, in which case they are considered under the same relation type. However, we regard *matches* as a separate relation, as we consider it important to capture concurrent similarities across dimensions. Thus, we only leave three relations {*follows, matches, overlaps*}.

Second, we introduce a *step size* σ to control the maximum number of event intervals our target is paired with. This acts as a limit on the time horizon that the target can reach. We use this limit because event intervals that are too far from the target may not have a strong relation with it, and this can negatively affect the embedding matrix. Note that this is not solely related to time series length. Event intervals are created based on consecutive event labels, which means that if there is no information change for a long time, a small step size can cover a long time period.

S-Hashtable saves both horizontal support $H((r, \epsilon_a, \epsilon_b), \mathbf{s}_k)$ and the occurrence set $\Omega((r, \epsilon_a, \epsilon_b), \mathbf{s}_k)$, as a set of values for each S-Hashtable. Thus, the values of $\mathcal{H}^R((r, \epsilon_a, \epsilon_b))(k)$ include the pointers to all event interval pairs with event labels ϵ_a, ϵ_b forming r in $\mathbf{s}_k$. For our classification task at hand, we only use horizontal support, but occurrences can still be useful for interpretability purposes since the structure keeps all the locations of pairs forming a specific temporal relation. After constructing a complete hash table structure, we create the embedding matrix $\mathcal{M}$ of size $|\mathcal{S}| \times |K^R|$ and use the keys as the indices of $\mathcal{M}$ and save the horizontal support, i.e., $\mathcal{M}_{k,(r,\epsilon_a,\epsilon_b)} = H((r, \epsilon_a, \epsilon_b), \mathbf{s}_k)$. $\mathcal{M}$ is then used as a feature set with $\mathcal{M}_{k,*}$ denoting the set of features for $\mathbf{T}_k \in \mathcal{T}$.

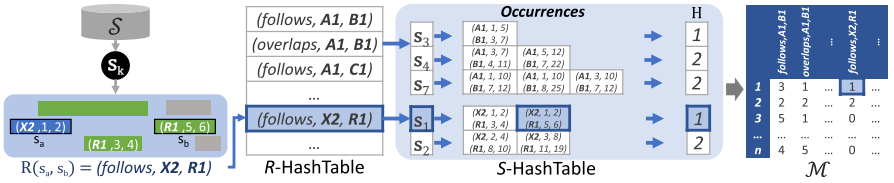

Fig. 5 An example of Z-Time's *RS*-Hashtable structure

Example 5 In Fig. 5, consider event interval $s_a = (X2, 1, 2)$ as our current target with $\sigma = 3$. Z-Time checks three steps ahead based on the sorted order of event interval sequence $\mathbf{s}_k$ (e.g., three event intervals in green). Then, s_a is paired with each event interval, and their temporal relation is calculated. When temporal relation $(follows, X2, R1)$ is found between s_a and s_b, *RS*-Hashtable is accessed by the temporal relation (for $\mathcal{H}^R$) and the event interval sequence index (for $\mathcal{H}^S$) (marked with blue outlines), and the pair of event intervals $((X2, 1, 2), (R1, 5, 6))$ is added to the occurrence set. However, the horizontal support remains 1 as $(X2, 1, 2)$ already has another event interval $(R1, 3, 4)$ forming the same *follows* relation. Once all $\mathbf{s}_k \in \mathcal{S}$ are traversed, *RS*-Hashtable is used to construct the embedding matrix $\mathcal{M}$ and the horizontal support of $((follows, X2, R1), \mathbf{s}_k)$ is added to $\mathcal{M}_{k,(follows,X2,R1)}$.

4.3 Segmentation of time series

We introduce an optional preliminary step, where a time series collection $\mathcal{T}$ is divided into ρ equal segments $\{\mathcal{T}^1, \ldots, \mathcal{T}^\rho\}$, and then the first two steps of Z-Time are applied to each segment individually. This results in ρ embedding matrices $\{\mathcal{M}^1, \ldots, \mathcal{M}^\rho\}$, which are concatenated to create a single feature set $\mathcal{M} = \{\mathcal{M}^1 \ldots \mathcal{M}^\rho\}$ that serves as input to the linear classifier.

This optional step is particularly relevant when the time series consist of multiple unrelated parts or follow different distributions, thus temporal relations between different segments are misleading. In these cases, applying temporal abstraction to the entire time series may not produce appropriate event labels. For instance, in the *CricketX* dataset in the UEA repository (Bagnall et al. 2018), the movements of the right hand are stored in the first half of the time series and the movements of the left hand are stored in the second half. In this case, temporal abstraction should be applied separately for each hand's movement.

Parameter ρ can be set using domain knowledge or can be explored with parameter search. Unlike the sliding window approaches which apply a quadratic (in time series length) number of discretization functions, our segmentation approach maintains a compact representation, having the same number of event sequence representations after segmentation, since the abstraction is applied individually to each segment.

4.4 Applying linear classifiers

The third step of Z-Time takes the embedding matrix $\mathcal{M}$ and applies a linear classifier to it, returning coefficients for the temporal relations. The coefficients are trained based on the horizontal support values of the temporal relations found in the training set, so any temporal relations that only appear in the test set are not considered. We follow the recommendation from Dempster et al. (2020), which suggests that a logistic regression classifier should be used when the number of time series instances is larger than the number of features, and the ridge classifier for the opposite case. Our experiments in Sect. 5 are conducted with the ridge classifier using the same regularization parameter optimization process as Dempster et al. (2020), since in the time series datasets in the UEA repository the number of features (in our case, the number of temporal relations) is generally larger than the number of training instances. The ridge classifier then creates class discriminatory temporal relations, which can be used for interpretability purposes (see Appendix D).

4.5 Time complexity

Given a time series collection $\mathcal{T}$ of size n, length m, with d dimensions, we first, in the first step, create event sequences, which involve discretization that requires $\mathcal{O}(dnmlog(m))$. We then create event intervals checking every time point in the time series collection, yielding a linear complexity of $\mathcal{O}(dmn)$. In the worst case, we assume that the number of created event intervals is the same as the number of time points in all dimensions, i.e., dm, for each event interval sequence. Then, in the second step, we assess the temporal relations, which requires quadratic time in the number of event intervals $\mathcal{O}(d^2m^2)$ for all n event interval sequences, leading to $\mathcal{O}(d^2m^2n)$, which is our final complexity. This means that the dimension and the length of $\mathcal{T}$ are deciding factors of Z-Time's scalability. However, in reality, the average number of event intervals in an event interval sequence is expected to be much smaller than dm, since Z-Time merges all consecutive points into one event interval.

5 Experiments

In this section, we benchmark Z-Time in terms of efficiency, effectiveness, and interpretability for solving Problem 1.

5.1 Experimental setup

Our experiments are performed on a Linux machine with an AMD 2950X processor and 128 GB of memory. We compare Z-Time to the state-of-the-art interpretable multivariate time series classifiers, i.e., MR-PETSC (Feremans et al. 2022) and XEM (Fauvel et al. 2022). For completeness, we also benchmark it against the state-of-the-art non-interpretable classifiers reported in Middlehurst et al. (2021).

Datasets We evaluate Z-Time and its main competitors on the multivariate time series datasets in the UEA repository (Bagnall et al. 2018). Since our focus is on classification performance, runtime, and interpretability rather than pre-processing, out of the 30 datasets we use the 26 that contain time series of equal-length. While we aim to employ all 26 datasets, we only present the results on the datasets for which all algorithms in comparison complete, since some algorithms fail to complete on some datasets due to high memory requirements (over 128 GB) or excessive runtime (over seven days). We also use our three synthetic datasets to investigate the multivariate interpretability cases in more detail. We report the full results in our repository.

While Z-Time aims to be an interpretable multivariate time series classifier exploiting inter-dimensional temporal relations, we also test its efficiency and effectiveness on the 112 UCR univariate datasets (Bagnall et al. 2018).

Implementation For the competitors, we use the authors' implementations and follow the experimental settings in the original articles. As the algorithms involve parameter search, we report the average accuracy over ten trials for effectiveness of all interpretable classifiers. For efficiency, we report the average runtime over 20 runs with randomly selected parameters. Z-Time is implemented in Python and includes randomized search over 100 iterations for $w \in \{3, 5, 7, 10\}, \lambda_1 \in \{3, 5, 7, 10\}, \lambda_2 \in \{3, 5, 7, 10\}, \sigma \in \{10, 20, 30, 50, 100\}$, and $\rho \in \{1, 2, 4, 8\}$. An ablation study of Z-Time is provided in Appendix C.

5.2 Effectiveness: classification performance

We compare Z-Time to the state-of-the-art multivariate time series classifiers in terms of accuracy on the 20 UEA multivariate datasets, as our competitors fail to run on six datasets, i.e., six for MR-PETSC and five for XEM, due to high memory requirements. Figure 6 shows the critical difference diagram of Z-Time against ten multivariate time series classifiers, using a two-sided Wilcoxon signed-rank test with $\alpha = 0.05$. Both Z-Time and XEM are not significantly less accurate than all state-of-the-art non-interpretable algorithms, while Z-Time is still ranked higher than XEM. However, MR-PETSC fails to achieve the same level of performance, being significantly less accurate than Z-Time but still comparable to XEM. Both XEM and Z-Time show lower scores on *Crickets* and *ArticularyWordRecognition* in ranking compared to the state-of-the-art non-interpretable classifiers. However, these two datasets are the only ones where the difference in accuracy of all reported algorithms is less than 2%, since the classification task is easy for these two datasets.

Figure 7 shows the pairwise comparisons between Z-Time and its two interpretable competitors. We include all datasets for which both paired algorithms complete, i.e., 21 for XEM and 20 for MR-PETSC. In Fig. 7a, Z-Time outperforms XEM on 12/21 cases, achieving $\geq 10\%$ accuracy on four datasets. In Fig. 7b, Z-Time beats MR-PETSC on 18/20 datasets with $\geq 10\%$ accuracy on six datasets. Z-Time achieves the highest performance difference against XEM on *Libras* and against MR-PETSC on *HandMovement*. On the other hand, Z-Time is less than 10% accurate than XEM only on *HandMovement*. Compared to MR-PETSC, Z-Time only

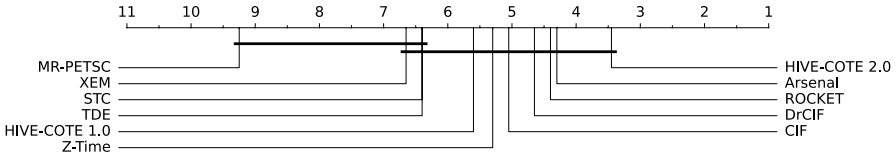

Fig. 6 Average rank of Z-Time against ten multivariate time series classifiers in terms of accuracy on the 20 UEA multivariate datasets

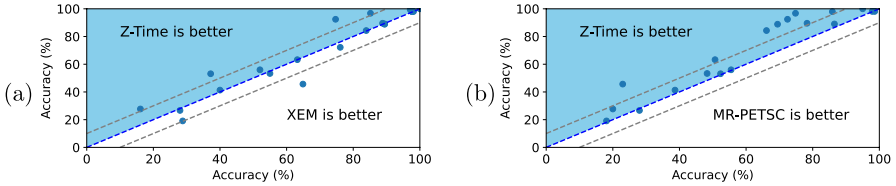

Fig. 7 Pairwise accuracy scatter plots of **a** Z-Time against XEM and **b** Z-Time against MR-PETSC. The dotted lines indicate ±10% intervals on accuracy

loses on two datasets, but not significantly, being only 1.3% less accurate on *AtrialFibrilation*, and 0.5% less accurate on *Crickets*.

We also test Z-Time on the 112 UCR univariate datasets to prove Z-Time is competitive in effectiveness compared to the state-of-the-art interpretable univariate classifiers, i.e., MR-SEQL and MR-PETSC (See Appendix A).

5.3 Efficiency: runtime performance

We evaluate the runtime efficiency of the three interpretable multivariate time series classifiers. Runtime is averaged over 20 runs with randomly selected parameters, and measured on a single core for a fair comparison. XEM fails to complete on six additional datasets, i.e., 12 in total, and MR-PETSC is unsuccessful on six datasets due to memory requirements (See Table 1). We compare the runtime on the datasets for which both paired algorithms complete.

While Z-Time and MR-PETSC employ a linear classifier for interpretability, XEM uses multiple boosting-bagging structures, making itself considerably slower than all the other algorithms. Considering total runtime, Z-Time is, on average, 2,274.1 times faster than XEM, ranging from 51.5 times (on *RacketSports*) to 12,624.6 times (on *UWaveGestures*). There is no single case where XEM is faster than Z-Time. Furthermore, the result is in favor of XEM since we exclude the 12 datasets where it fails, and these datasets can increase the average runtime of XEM. MR-PETSC is faster than Z-Time on 9/20 datasets, up to 5.6 times (on *LSST*). However, Z-Time is still 2.1 times faster than MR-PETSC on the average of 20 datasets and up to 20.4 times (on *EthanolConcentration*). With this experiment, we confirm that Z-Time is more efficient than its competitors, and specifically by an order of magnitude more efficient than XEM. Comparing Z-Time to the non-interpretable classifiers, Z-Time is faster than tree-based

Table 1 Pairwise runtime comparisons between `Z-Time` and competitors. A larger *Lose* value is in favor of `Z-Time`. A negative value implies a competitor is faster and vice versa

Algorithm	Dataset			Runtime proportion		
	Complete	Win	Lose	Min.	Max.	Avg.
XEM	14	0	14	51.5	12,624.6	2274.1
MR-PETSC	20	9	11	−5.6	20.4	2.1

classifiers, 7.6 times faster than DrCIF and 4.2 times faster than CIF, while 13.5 times slower than ROCKET and 3.8 times slower than ARSNAL, which is an ensemble of ROCKET classifiers.

Since `Z-Time` constructs the embedding matrix for both the training and test sets, the proportions of the training and test phases are directly proportional to the size of each dataset. However, other algorithms may have shorter testing times. Specifically, XEM spends 98.8% of its runtime on training, while MR-PETSC spends 66.7% on average. Even when considering only test time, `Z-Time` is still 38.4 times faster than XEM and 1.2 times faster than MR-PETSC. For non-interpretable classifiers, `Z-Time` is 4.9 times slower than DrCIF and 3.9 times slower than CIF, as both of these algorithms also require significant training time.

For completeness, we test the classifiers on the 112 UCR univariate datasets. Since `Z-Time`'s quadratic time complexity in dimensions has no effect for the univariate cases, `Z-Time` can be even faster, i.e., 2.8 and 5 times than MR-SEQL and MR-PETSC on average, respectively. `Z-Time` is also 1.8 times faster than ROCKET on average, since `Z-Time` is quadratic in time series length, which is, in general, less time consuming than passing 10,000 random convolutional kernels. This confirms `Z-Time`'s competitive runtime in the univariate cases even compared to non-interpretable classifiers.

5.4 Interpretability: four use-cases

Our experiments confirm that `Z-Time` is the first multivariate time series classifier that maintains high levels of effectiveness and efficiency. More concretely, `Z-Time` is (1) significantly more accurate than MR-PETSC and (2) an order of magnitude faster than XEM, thus leaving no direct competitor in terms of both effectiveness and efficiency. Nevertheless, we compare the interpretability of `Z-Time` against MR-PETSC and XEM on two UEA multivariate datasets (*Libras* and *SelfRegulationSCP1*) and three synthetic datasets for completeness. The main objective is to validate how well each algorithm can detect and exploit interpretable inter-dimensional relations, which is essential for multivariate time series classification. While MR-PETSC can be directly excluded since it returns a set of features from each dimension separately, we include it for completeness. The top five classification features for `Z-Time` and MR-PETSC can be found in Appendix B.

We validate two common cases in multivariate time series classification: (1) a single location is not enough to classify and explain and (2) inter-dimensional orders at different time points can be detrimental for the classification outcome

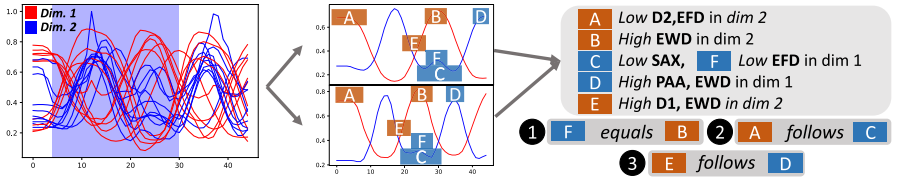

Fig. 8 Ten 2-dimensional time series instances of class 1 from *Libras* with an example region detected by XEM with a window size of 60% marked as blue (left), and two instances of class 1 with Z-Time's event intervals forming top three temporal relations (right)

and for providing explanations. In Figs. 8, 9 and 10, we display the features with the simplest parameter setup $\{\lambda_1 : 3, \lambda_2 : 3, \sigma : 10, w : 10, \rho : 1\}$ using three

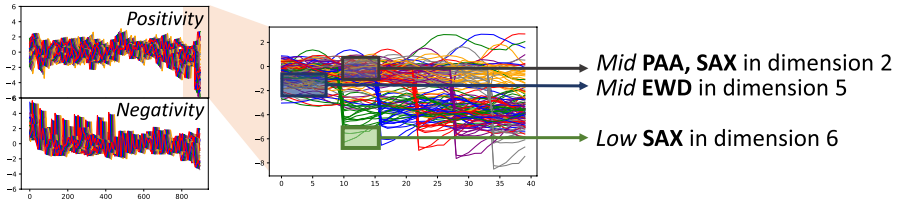

Fig. 9 Ten time series instances of two classes in *SelfRegulationSCP1* and the last 30 time points of *positivity* with Z-Time's event intervals forming the top two temporal relations

event label types {*high*, *mid*, *low*}, for each dimension and representation.

5.4.1 Use case 1: Libras

Libras contains 15 classes of 24 instances, each of which refers to a hand movement type for the Brazilian sign language. The hand movement is represented as a two-dimensional curve performed over a period of time. XEM and Z-Time achieve the highest performance difference in accuracy on this dataset since XEM cannot successfully distinguish the relations between the two sinusoidal patterns in *Libras*. Z-Time achieves an accuracy of 88.9% with the simplest parameter setup, outperforming all interpretable competitors.

Figure 8 depicts ten 2-dimensional instances of class 1 taken from *Libras*, with the first dimension indicated with red and the second one with blue. In addition, we present an example of a region provided by XEM for one instance under its best parameter setting (left), and two instances of the same class with Z-Time's top three temporal relations (right). While representing the same class, hand movements in different time series instances are not timely aligned since gestures are performed at different speeds. As shown on the right side of Fig. 8, the event intervals with the same event label in each example have different start and end time points. On the other hand, the explanatory region provided by XEM covers 60% of the time series length but is less informative than Z-Time, since

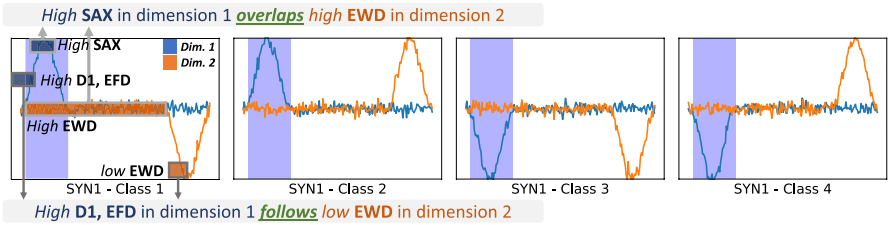

Fig. 10 An example of the four classes of *SYN1* that cannot be distinguished by a single region. The area detected by XEM with a window size of 20% is marked as blue. `Z-Time` 's event intervals forming the top two temporal relations for class 1 are also presented

the latter provides more concrete and explicit temporal patterns that characterize class 1.

Moreover, in Fig. 8, we observe that `Z-Time` picks similar *low* values in dimension 1 with two different representations (*C* and *F*), and creates two different temporal relations with *B* and *A*, respectively. Even for the same area, there can be different representations that are more effective in catching temporal relations. For example, `Z-Time` finds the *equals* relation between *F* and *B*, which cannot be formed between *C* and *B* even though *F* and *C* cover similar area since *equals* requires strict time point matching.

On the other hand, MR-PETSC is robust to the location factors since it also uses the horizontal support of its patterns as features. However, it still shows similar accuracy as XEM, because the inter-dimensional relations cannot be detected. By observing Fig 8, we can deduce that MR-PETSC cannot figure out if *B* should happen after *A* or whether *F* and *B* should occur simultaneously. Detecting these two inter-dimensional relations is necessary for understanding the properties of class 1 in the dataset.

5.4.2 Use case 2: SelfRegulationSCP1

This dataset contains electroencephalograms for identifying slow cortical potentials. Each subject is asked to move a mouse pointer on the screen up or down. Detailed instructions are given for the first 5.5 s, and the last 0.5 s only involve free movement. The dataset has two classes: *positivity* and *negativity*. It only contains the last 3.5 s in 896 time points of which each second comprises 256 points.

Both XEM and `Z-Time` perform equally well on this dataset achieving an accuracy of approximately 84%. `Z-Time` shows 77.1% accuracy with the simplest parameter setup. However, XEM chooses the 100% window size as the optimal parameter for this performance, failing to deliver any interpretability. Figure 9 shows ten examples of two classes and the last 30 time points (about one second) of the *positivity* class with three event intervals forming `Z-Time`'s top two temporal relations. Even though the discriminative features can easily be localized, it is hard for XEM to identify them, as the features can only be found by the fact that it is globally low, and this can only be detected with the 100% window size. On the other hand, the most discriminative feature found by `Z-Time` is *equals* between

'*mid* **PAA, EWD** in dimension 2' and '*low* **SAX** in dimension 6' (Note that the dimension information is also part of an event label), followed by *follows* between '*mid* **EWD** in dimension 5' and '*low* **SAX** in dimension 6'. The '*low* **SAX**' event intervals are prevalent at the end of the *positivity* instances in every dimension (i.e., for the last 0.5 s of free movement) and also common in few dimensions of *negativity*. However, in the **negativity** instances, '*low* **SAX** in dimension 6' rarely occurs at the same time with '*mid* **PAA, EWD** in dimension 2', making the relation distinctive. This is why Z-Time picks the one in dimension 6 while '*low* **SAX**' occurs in every dimension at the end of the *positivity* time series instances.

MR-PETSC is not successful on this dataset, returning close to random accuracy (66.1%) for this binary problem, since multiple overlapping sliding windows cannot distinguish the global importance of a specific region.

5.4.3 Synthetic case 1: Robustness to conflicting regions

We test the case where using a window on the same time region across all dimensions cannot fully explain the dataset, due to the presence of *conflicting regions*. *SYN1* contains 100 training instances and 20 test instances for each class with two dimensions and four classes, as depicted in Fig. 10. In order to create conflicting regions, we introduce specific concave and convex curves in the first and last 25% of the time series, so that when one time region is selected, there are always two classes having the same pattern on that region (e.g., classes 1 and 2 for the first 25%) but they are distinguished by looking at another time region concurrently (e.g., the last 25% for both classes).

XEM's average accuracy over 20 runs under random parameter settings is 35.3%, even returning a minimum of 25% with a window size of 20%. The only way for XEM to succeed in this classification task it is to use a window size of 100%, which is equivalent to running a non-interpretable classifier. On the other hand, Z-Time achieves perfect classification accuracy (i.e., 100%) on this dataset over 20 runs under the same setting and with the simplest parameter setup, by exploiting inter-dimensional relations. MR-PETSC achieves an average accuracy of 90.2% as it is also not affected by conflicting regions. However, it still cannot perfectly classify the dataset as its sliding windows even detect the discriminative features in the noisy areas.

Figure 10 shows Z-Time's top two features formed by four event intervals in two dimensions. These two temporal relations give sufficient reasoning for understanding what affects the classification performance. Also, Z-Time creates a compact representation by merging consecutive events with the same label into one event interval, such as '*high* **EWD** in dimension 2'.

5.4.4 Synthetic case 2: Robustness to inter-dimensional orders

We benchmark the ability of the three interpretable multivariate time series classifiers to catch inter-dimensional orders that are not related to a specific temporal location. Since an event can occur at different time points in each dimension of different

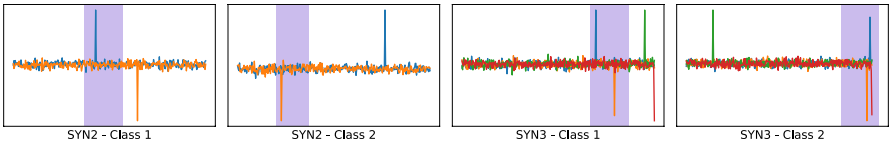

Fig. 11 Examples of two synthetic datasets (*SYN2* and *SYN3*) forming inter-dimensional orders. The blue regions indicate the areas XEM finds with a window size of 20%

time series instances, discriminative features cannot be obtained by a region of specific length across all dimensions (as we see on *Libras*) since we cannot confirm if all discriminative events occur in the chosen region.

Our two synthetic datasets (*SYN2* and *SYN3*) comprise 100 training instances and 20 test instances for each class. Figure 11 depicts *SYN2* and *SYN3* with inter-dimensional orders represented by peaks: (1) two dimensions and two classes (*SYN2*) and (2) four dimensions and three classes (*SYN3*). In *SYN2*, class 1 is defined in *blue-yellow* order and class 2 is defined in the opposite order, i.e., *yellow-blue*. *SYN3* has the same order-based class definition, i.e., *blue-yellow-green-red* for class 1, *green-yellow-blue-red* for class 2, and *blue-red-green-yellow* for class 3. The colored regions are examples of explanation areas provided by XEM with a window size of 20%.

We run the three algorithms 20 times on the two synthetic datasets under random parameter settings. `Z-Time` predicts both *SYN2* and *SYN3* with 100% accuracy, regardless of the chosen parameters and also with the simplest parameter setup, if segmentation is not enabled, since `Z-Time` can easily detect the order with the *follows* relations between the peaks.

XEM fails to achieve a higher than 80% accuracy on both datasets (62.5% on average on *SYN2* and 59.5% on *SYN3*) over the 20 runs. This indicates that XEM cannot detect any pattern unrelated to specific time points but is rather based on temporal orders. MR-PETSC shows 65.1% average accuracy on *SYN2* and 81.7% on *SYN3*, and finds the features independently without considering the order. Thus, in terms of interprerability it is not as clear as `Z-Time` since their sliding windows can even find similar features in the noisy area. so they do not help interpret the inter-dimensional orders.

5.5 Effect of missing data

We compare `Z-Time` against its two competitors on four UEA multivariate datasets (*RacketSport*, *StandWalkJump*, *LSST*, and *PenDigits*) from different domains in terms of robustness to missing data. As `Z-Time` is based on event intervals, it does not need to perform interpolation for handling missing data, but rather skip them and generate event intervals using only the time points where with data values. Since missing data is regarded as a time gap, `Z-Time` may have additional *follows*.

We remove a fraction of randomly chosen time points from each time series, increasing the proportion of missingness from 5 to 25% in increments of 5%. Figure 12 shows the average accuracy loss of the three algorithms for each dataset per

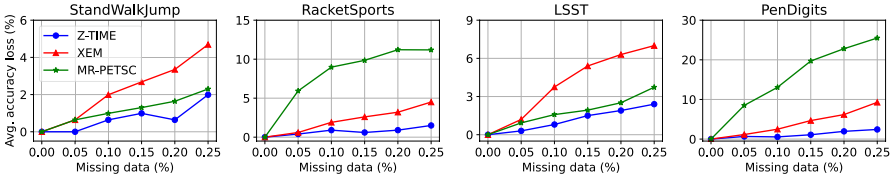

Fig. 12 Average accuracy loss plots for Z-Time, XEM, and MR-PETSC over 20 runs on four UAE datasets with randomly inserted missingness from 5% to 25% in increments of 5%

missingness increment, i.e., how much accuracy drops for each missingness level compared to the accuracy of the original dataset. As missingness increases, accuracy is expected to decrease.

On all four datasets, Z-Time shows less than 3% of accuracy loss, while XEM reaches up to 9.8% loss and MR-PETSC even has up to 25.5% loss. XEM shows $\geq 5\%$ loss on *PenDigits* and *LSST* with 25% missing data. As seen in Fig. 12, the gap in the accuracy loss between Z-Time and XEM increases as missingness also increases, on all datasets. MR-PETSC behaves inconsistently and is unstable since on *RacketSports* and *PenDigits* it is even worse than XEM having larger gaps as the missingness increases showing $\geq 10\%$ losses, but on *LSST* and *SandWalkJump* it shows similar accuracy losses as Z-Time. The largest gap of Z-Time to both algorithms is found on *PenDigits* where Z-Time only has 2.2% loss, while XEM has 9.8% and MR-PETSC has 25.5%, i.e., 7.6% and 23.3% gaps respectively. This experiment confirms that Z-Time is generally more robust and stable to missing data than both XEM and MR-PETSC.

5.6 Effect of dimensionality

As discussed in Sect. 4.5, the computational complexity of Z-Time is quadratic in time series length and dimensions. While Z-Time, in practice, reduces the time series length by creating event intervals, it maintains the original number of dimensions, which can pose a problem to the size of the embedding matrix. The size of the embedding matrix is also quadratic in the number of dimensions, as event labels are defined for each dimension, and pair-wise combinations of these labels are created to capture temporal relations. Thus, reducing the dimensionality may improve the efficiency of Z-Time and reduce the feature space size. While it is already confirmed that Z-Time is more efficient than all of its interpretable competitors, we explore the trade-off between efficiency and effectivenes of Z-Time and its competitors on the UEA multivariate datasets.

We do not aim to compare dimensionality reduction techniques, so we choose the elbow class pairwise (ECP) method as it shows the best performance in Ruiz and Bagnall (2022). We apply ECP to 14 multivariate time series datasets, for which all three algorithms complete on a single core, out of which ECP succeeds

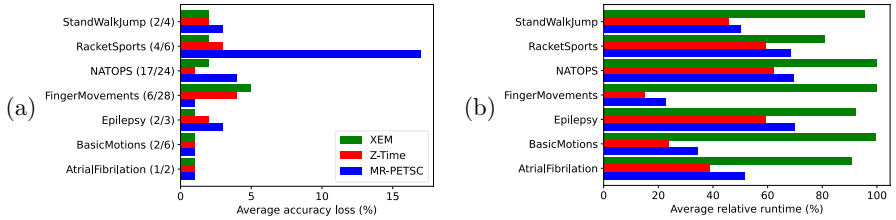

Fig. 13 **a** Average accuracy loss and **b** average relative runtime over 20 runs for `Z-Time`, XEM, and MR-PETSC on seven UEA datasets after dimensionality reduction

in reducing dimensions of seven datasets. Figure 13 shows the accuracy loss (Fig. 13a) and the relative runtime (Fig. 13b) for each dataset averaged over 20 runs with random parameter settings. XEM and `Z-Time` maintain the accuracy scores well, with no more than a 5% loss on all seven datasets, while MR-PETSC reaches an accuracy loss of 17.2% on *RacketSports*.

While `Z-Time` is already on average 2,274.1 times faster than XEM without dimensionality reduction, when the dimensions are reduced it finishes in 43.6% of the original runtime, maintaining 2% accuracy loss on average. However, XEM's time complexity depends on its concatenated representation and this does not seem to be greatly improved by dimensionality reduction. XEM still takes 94.2% of the original runtime, failing to reduce the runtime more than 10% on average. It only finishes in 80.9% of the original runtime on *RocketSports*, but on three datasets (*NATOPS*, *FingerMovements*, and *BasicMotions*), it shows negligible runtime difference ($< 1\%$) from the original runtimes. `Z-Time` is 3,932 times faster than XEM on average, ranging from 64.7 times (on *RacketSports*) to 20,702 times (on *StandWalksJump*). MR-PETSC also shows improvements by completing the tasks in 52.5% of original time and also maintaining accuracy except for *RacketSports*. `Z-Time` is still 1.7 times faster than MR-PETSC on average, ranging from -2.9 times (on *NATOPS*) to 7.1 times (on *StandWalksJump*).

6 Conclusion

We proposed `Z-Time`, a fast and interpretable multivariate time series classifier which employs temporal abstraction and temporal relations of event intervals to create its interpretable feature space. We benchmaked `Z-Time` on the UEA multivariate time series datasets where it achieved a higher average rank than state-of-the-art interpretable multivariate time series classifiers, being also not significantly less accurate than any state-of-the-art non-interpretable classifier. We also showed that `Z-Time` is the fastest both on univariate and multivariate datasets, even having comparable runtime to ROCKET for the univariate case. We also investigated

interpretability related cases both on real-world and synthetic datasets showing that Z-Time can identify inter-dimensional features from conflicting regions or inter-dimensional orders regardless of their specific locations, while XEM and MR-PETSC could not.

Future work involves integrating different discretization functions, increasing the interpretability of Z-Time by considering temporal relations formed by more than two event intervals. A more robust way of segmentation than equally dividing the area can be also explored to define better local abstraction. Z-Time may also be applied to other interpretability related tasks, such as counterfactual analysis, as event intervals can provide more understandable counterfactuals than when using the raw time series space.

Appendix A: Performance on the UCR univariate datasets

While Z-Time focuses on multivariate time series classification, we additionally test Z-Time on 112 UCR univariate datasets to show that Z-Time maintains its competitive classification power even on a single dimension compared to two state-of-the-art interpretable univariate classifiers (i.e., MR-PETSC and MR-SEQL). We exclude XEM as it is infeasible to run it on all 112 datasets within a reasonable time frame with its parameter search (see Sect. 5.3).

Figure 14 shows the average rank of Z-Time against the state-of-the-art univariate time series classifiers on 112 UCR univariate datasets using a critical difference diagram. While none of the interpretable classifiers outperforms the non-interpretable ones, this is expected as the former only exploit a linear classifier with human-interpretable features (Fig. 14a). MR-SEQL is the only interpretable classifier being not significantly less accurate than STC (Hills et al. 2014), but it is also not significantly more accurate than Z-Time. MR-SEQL and Z-Time are both significantly better than MR-PETSC.

Unlike MR-SEQL and MR-PETSC, which pass multiple windows with different event labels and window sizes, Z-Time only uses fixed event labels and PAA window sizes to perform abstraction with parameter search to keep a compact and robust representation. This means that Z-Time can be more sensitive to the parameter

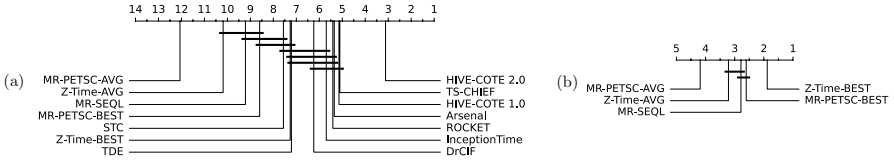

Fig. 14 Critical difference diagram for rankings of **a** Z-Time against 11 univariate time seires classifiers and **b** Z-Time and three interpretable univariate time series classifiers, by accuracy on the 112 UCR univariate datasets

settings. Similar to the two easy UEA multivariate problems (*Crickets* and *ArticularyWordRecognition*), the UCR repository also involves several easy problems (i.e., `Z-Time` achieves more than 90% accuracy on 41 out of 112 datasets), making parameter search unstable. For example, under different parameter settings, `Z-Time` achieves the same validation accuracy, while averaging multiple trials lead to some deduction on test accuracy (e.g., on Trace `Z-Time` is ranked on 13^{th} with an accuracy of 99.8%, as it fails to achieve perfect accuracy in only one trial). This is why we also include the best cases of ten parameter search trials (denoted by the suffix '-BEST') for `Z-Time` and MR-PETSC as references, while still evaluating the classifiers based on their average values. `Z-Time` can potentially be significantly comparable to most state-of-the-art algorithms except HIVE-COTE 2.0 and TS-CHIEF.

Figure 14b shows the critical difference diagram of three interpretable time series classifiers for a clear comparison. While `Z-Time` and MR-SEQL have comparable performance, MR-PETSC is significantly less accurate than the others. `Z-Time` beats MR-SEQL on around half of the datasets (i.e., 55/112) and loses on the rest (i.e., 57/112). With this experiment, we can confirm that `Z-Time` is also competitive on the univariate classification problem.

Appendix B: Top five classification features of `Z-Time` and MR-PETSC

Tables 2-8 summarize the list of the most discriminative features of `Z-Time` (Tables 4, 5, 6, 7 and 8) and MR-PETSC (Tables 2, 3) detected in terms of classification coefficients for each dataset we use in our experiments. Note that the subsequences are generated from each dimension and it is unclear how we can obtain inter-dimensional interpretability from those patterns, so we list them here as a reference. XEM's most discriminative regions differ for each test instance so we are not able to include them here.

Table 2 The top five features found by MR-PETSC on *Libras* and *SelfRegulationSCP1*

Rank	Libras		SelfRegulationSCP1	
	Feature	Dim	Feature	Dim
1	dddddccbb	1	dbccc	3
2	cbbaa	2	aaaabcc	5
3	aaaabc	2	aacba	1
4	baaaaaa	1	baabb	2
5	ddcbaaa	1	cbabb	3

Table 3 The top five features found by MR-PETSC on the three synthetic datasets

Rank	SYN1		SYN2		SYN3	
	Feature	Dim	Feature	Dim	Feature	Dim
1	cdddd	2	bdcbc	1	acccccccc	1
2	cccbb	1	ccacb	1	bbcac	1
3	cbcbccc	2	bbbbbb	2	dcabc	1
4	bcbcbb	2	cbbbbbbbb	1	ddbbb	1
5	ccbbbc	2	acbcc	2	bbccb	1

Table 4 The top five temporal relations found by Z-Time on *Libras*

Rank	Temporal relation
1	(*equals*, *low* **SAX** in dim. 1, *high* **EWD** in dim. 2)
2	(*follows*, *low* **D2, EFD** in dim. 2, *low* **SAX** in dim. 1)
3	(*follows*, *high* **D1, EWD** in dim. 2, *high* **PAA, EWD** in dim. 1)
4	(*follows*, *mid* **D1, EWD** in dim. 2, *mid* **PAA, SAX** in dim. 2)
5	(*follows*, *low* **D2, EWD** in dim. 2, *high* **PAA, SAX** in dim. 1)

Table 5 The top five temporal relations found by Z-Time on *SelfRegulationSCP1*

Rank	Temporal relation
1	(*equals*, *mid* **PAA, SAX** in dim. 2, *low* **SAX** in dim. 6)
2	(*follows*, *mid* **EWD** in dim. 5, *low* **SAX** in dim. 6)
3	(*follows*, *high* **D1, EFD** in dim. 1, *mid* **EWD** in dim. 5)
4	(*follows*, *mid* **EWD** in dim. 5, *low* **SAX** in dim. 5)
5	(*follows*, *low* **SAX** in dim. 5, *low* **EFD** in dim. 6)

Table 6 The top five temporal relations found by Z-Time on *SYN1*

Rank	Temporal relation
1	(*overlaps*, *high* **SAX** in dim. 1, *high* **EWD** in dim. 2)
2	(*follows*, *high* **D1, EFD** in dim. 1, *low* **EWD** in dim. 2)
3	(*follows*, *high* **SAX** in dim. 1, *mid* **EFD** in dim. 1)
4	(*follows*, *low* **D2, EWD** in dim. 2, *high* **PAA, EFD** in dim. 2)
5	(*follows*, *low* **D1, EWD** in dim. 2, *high* **PAA, SAX** in dim. 1)

Table 7 The top five temporal relations found by `Z-Time` on *SYN2*

Rank	Temporal relation
1	(*follows*, *high* **EWD** in dim. 2, *low* **EWD** in dim. 1)
2	(*overlaps*, *low* **D1, EWD** in dim. 1, *mid* **D1, EFD** in dim. 1)
3	(*follows*, *high* **D1, EFD** in dim. 1, *low* **PAA, EWD** in dim. 1)
4	(*overlaps*, *low* **D2, EWD** in dim. 1, *mid* **D1, EFD** in dim. 1)
5	(*equals*, *mid* **PAA, EFD** in dim. 1, *mid* **D2,EWD** in dim. 1)

Table 8 The top five temporal relations found by `Z-Time` on *SYN3*

Rank	Temporal relation
1	(*equals*, *low* **SAX** in dim. 1, *high* **EWD** in dim. 2)
2	(*follows*, *low* **D2, EFD** in dim. 2, *low* **SAX** in dim. 1)
3	(*follows*, *high* **D1, EWD** in dim. 2, *high* **PAA, EWD** in dim. 1)
4	(*follows*, *mid* **D1, EWD** in dim. 2, *mid* **PAA, SAX** in dim. 2)
5	(*follows*, *low* **D2, EWD** in dim. 2, *high* **PAA, SAX** in dim. 1)

Appendix C: Ablation study

While we employ parameter search in our experiments, we also investigate how each parameter works on 20 UEA multivariate datasets. The best parameter setting always depends on each dataset, but this experiment can help us understand how robust `Z-Time` in general is to variations of each parameter. We set the default parameter values to $\{\lambda_1 : 5, \lambda_2 : 5, w : 5, \sigma : 20, \rho : 1\}$ and change one parameter to another value from its default value, while fixing the others.

Figures 15, 16, 17, 18 and 19 show the average ranks of `Z-Time` under different parameter settings on the 20 UEA multivariate datasets. `Z-Time` is generally robust enough not showing extreme differences. Most of the parameters do not show statistically significant differences in their average ranks. This is expected since every dataset has different properties and the best parameter should differ. Nevertheless we can observe meaningful differences from event label size for original time series representation λ_1 (Fig. 15) and segmentation parameter ρ (Fig. 19).

We observe that a higher λ_1 value yields better performance. The average rank of `Z-Time` with $\lambda_1 = 10$ is 1.1 higher than the average rank with $\lambda_1 = 3$. However, more than this gap is needed to conclude any significant differences, which means all values in the option are comparable and the optimal value differs for each dataset. We cannot get any statistically significant differences or any linear trend for λ_2 and w. The biggest gap in average ranks with these two parameters is at most 0.7, and since parameters are mixed on the diagram, we cannot see any trend such as higher value achieves better performance.

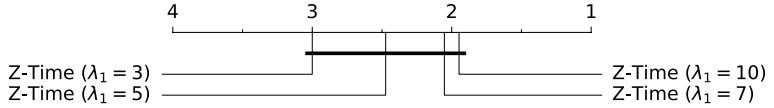

Fig. 15 Critical difference diagram showing the effect of the event label size for the original time series representation (λ_1)

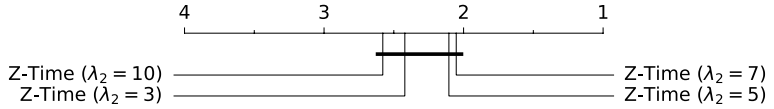

Fig. 16 Critical difference diagram showing the effect of the event label size for the finite difference forms (λ_2)

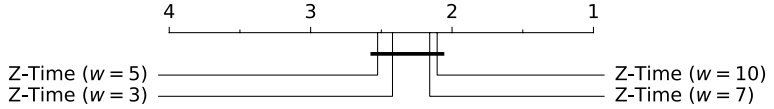

Fig. 17 Critical difference diagram showing the effect of the window size (w)

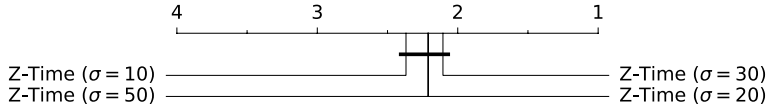

Fig. 18 Critical difference diagram showing the effect of the step size (σ)

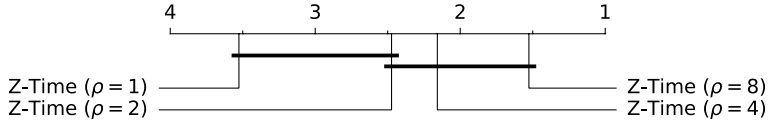

Fig. 19 Critical difference diagram showing the effect of the number of splits (ρ)

Segmentation plays a vital role in the UEA datasets, and many splits lead to better performance. Z-Time with $\rho = 8$ is even significantly more accurate than Z-Time with $\rho = 1$. This means that when temporal relations are created independently in eight different segments, Z-Time gets better classification features. However, we still do not always restrict Z-Time to segment datasets as it all depends on each dataset since it can lead to overfitting the UEA datasets. For example, even on the UEA datasets, *SelfRegulationSCP2* and *StandWalkJump* show higher accuracy scores

with smaller ρ values, being ranked first with $\rho = 1$ and second with $\rho = 2$. Also, the parameters are not entirely independent. We may understand segmentation works well when the dataset needs to be split, and since segmentation can yield clear performance differences, this can also be found by parameter search without any problem.

Appendix D Investigation on feature weights with the ridge classifier

Figure 20 illustrates the weight distributions with the the ridge classifier on the two UEA datasets (*Libras*, *SelfRegulationSCP1*) and the largest synthetic dataset (*SYN3*) used in the interpretability experiments with the same simplest parameter setup. The figure displays the cumulative weight distributions on the left side and the sorted weight distributions on the right side.

It is essential to retain the most significant temporal relations to ensure the interpretability of the features. For instance, when applied to the *Libras* dataset, if we consider the top features in terms of weights, we observe that 20% of the features weighted by the ridge classifier account for 50.4% of the weights. On *SelfRegulationSCP1*, 20% of the features with the ridge classifier account for 47.5% of the total weights. Finally, on the *SYN3* dataset, 49.7% of the total weights are explained by the top 20% of the features with the ridge classifier.

This experiment confirms that our temporal relation features can yield class distinctive weight distributions and that these distributions remain similar across different datasets.

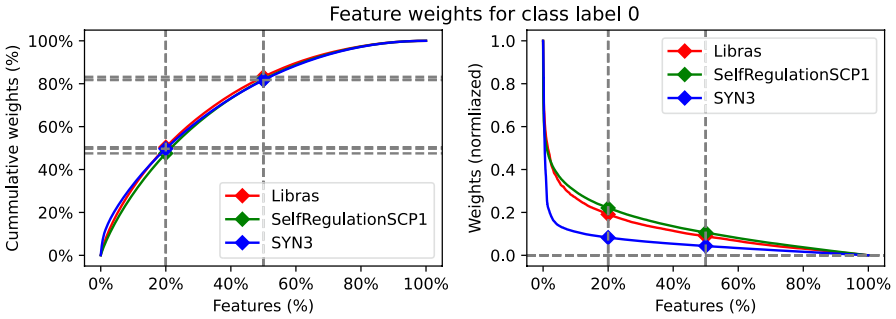

Fig. 20 Feature weight distributions from the ridge classifier on the two UEA datasets as well as the largest synthetic dataset (SYN3) used in our interpretabilty experiments for class label 0. On the left side, we present the cumulative weight distributions, while on the right side, we show the sorted weight distributions

Funding Open access funding provided by Stockholm University.

Declarations

Conflict of interest We have no conflicts of interest to disclosure.

Human or animal participants This research does not involve any human or animal participants.

References

Allen JF (1983) Maintaining knowledge about temporal intervals. Commun ACM 26(11):832–843

Bagnall A, Lines J, Vickers W, et al (2018) The uea & ucr time series classification repository. http://www.timeseriesclassification.com

Cabello N, Naghizade E, Qi J, et al (2020) Fast and accurate time series classification through supervised interval search. In: ICDM, IEEE, pp 948–953

Chen T, Guestrin C (2016) Xgboost: a scalable tree boosting system. In: KDD, pp 785–794

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

Dempster A, Schmidt DF, Webb GI (2021) Minirocket: A very fast (almost) deterministic transform for time series classification. In: KDD, pp 248–257

Fauvel K, Fromont E, Masson V et al (2022) Xem: an explainable-by-design ensemble method for multivariate time series classification. Data Min Knowl Discov 36(3):917–957

Feremans L, Cule B, Goethals B (2022) Petsc: pattern-based embedding for time series classification. Data Min Knowl Dicov 36(3):1015–1061

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

Hills J, Lines J, Baranauskas E et al (2014) Classification of time series by shapelet transformation. Data Min Knowl Dicov 28(4):851–881

Ho NTT, Pedersen TB, et al (2022) Efficient temporal pattern mining in big time series using mutual information. In: VLDB, VLDB Endowment, pp 673–685

Ismail Fawaz H, Lucas B, Forestier G et al (2020) Inceptiontime: finding alexnet for time series classification. Data Min Knowl Dicov 34(6):1936–1962

Karim F, Majumdar S, Darabi H et al (2019) Multivariate lstm-fcns for time series classification. Neural Netw 116:237–245

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

Kaushik S, Choudhury A, Sheron PK et al (2020) Ai in healthcare: time-series forecasting using statistical, neural, and ensemble architectures. Front Big Data 3:4

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

Kiangala KS, Wang Z (2020) An effective predictive maintenance framework for conveyor motors using dual time-series imaging and convolutional neural network in an industry 4.0 environment. IEEE Access 8:121033–121049

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

Le Nguyen T, Gsponer S, Ilie I et al (2019) Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. Data Min Knowl Dicov 33(4):1183–1222
Lee Z, Lindgren T, Papapetrou P (2020) Z-miner: an efficient method for mining frequent arrangements of event intervals. In: KDD, pp 524–534
Lee Z, Anton N, Papapetrou P, et al (2021) Z-hist: A temporal abstraction of multivariate histogram snapshots. In: International Symposium on Intelligent Data Analysis, Springer, pp 376–388
Lin J, Keogh E, Wei L et al (2007) Experiencing sax: a novel symbolic representation of time series. Data Min Knowl Dicov 15(2):107–144
Lines J, Taylor S, Bagnall A (2016) Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In: ICDM, IEEE, pp 1041–1046
Middlehurst M, Vickers W, Bagnall A (2019) Scalable dictionary classifiers for time series classification. In: IDEAL, Springer, pp 11–19
Middlehurst M, Large J, Bagnall A (2020) The canonical interval forest (cif) classifier for time series classification. In: BigData, IEEE, pp 188–195
Middlehurst M, Large J, Flynn M et al (2021) Hive-cote 2.0: a new meta ensemble for time series classification. Mach Learn 110(11):3211–3243
Moskovitch R, Shahar Y (2015) Classification of multivariate time series via temporal abstraction and time intervals mining. Knowl Info Syst 45(1):35–74
Rebane J, Karlsson I, Bornemann L et al (2021) Smile: a feature-based temporal abstraction framework for event-interval sequence classification. Data Min Knowl Dicov 35(1):372–399
Ruiz AP, Bagnall A (2022) Dimension selection strategies for multivariate time series classification with hive-cotev2.0. In: ECML-PKDD Workshop on AALTD
Ruiz AP, Flynn M, Large J et al (2021) The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Discov 35(2):401–449
Schäfer P (2015) The boss is concerned with time series classification in the presence of noise. Data Min Knowl Discov 29(6):1505–1530
Schäfer P, Leser U (2017a) Fast and accurate time series classification with weasel. In: CIKM, pp 637–646
Schäfer P, Leser U (2017b) Multivariate time series classification with weasel+ muse. ECML-PKDD Workshop on AALTD
Sheetrit E, Nissim N, Klimov D, et al (2019) Temporal probabilistic profiles for sepsis prediction in the icu. In: KDD, pp 2961–2969
Shokoohi-Yekta M, Hu B, Jin H et al (2017) Generalizing dtw to the multi-dimensional case requires an adaptive approach. Data Min Knowl Discov 31(1):1–31
Thiel SW, Rosini JM, Shannon W et al (2010) Early prediction of septic shock in hospitalized patients. J Hosp Med Off Publ Soc Hosp Med 5(1):19–25
Wang Z, Yan W, Oates T (2017) Time series classification from scratch with deep neural networks: A strong baseline. In: IJCNN, IEEE, pp 1578–1585
Xi R, Li M, Hou M et al (2018) Deep dilation on multimodality time series for human activity recognition. IEEE Access 6:53381–53396
Yang Y, Webb GI (2002) A comparative study of discretization methods for naive-bayes classifiers. In: PKAW