Information Processing and Management 59 (2022) 103044

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/ipm

Time-series classification with SAFE: Simple and fast segmented word embedding-based neural time series classifier

Nuzhat Tabassum [a,b], Sujeendran Menon [a], Agnieszka Jastrzębska [a,*]

[a] Faculty of Mathematics and Information Science, Warsaw University of Technology, ul. Koszykowa 75, 00-662 Warsaw, Poland
[b] Department of Computer Science & Engineering, Military Institute of Science and Technology, Mirpur Cantonment, Dhaka 1216, Bangladesh

ARTICLE INFO

Keywords:
Time series
Classification
Word embedding
Neural network

ABSTRACT

Dictionary-based classifiers are an essential group of approaches in the field of time series classification. Their distinctive characteristic is that they transform time series into segments made of symbols (words) and then classify time series using these words. Dictionary-based approaches are suitable for datasets containing time series of unequal length. The prevalence of dictionary-based methods inspired the research in this paper. We propose a new dictionary-based classifier called SAFE. The new approach transforms the raw numeric data into a symbolic representation using the Simple Symbolic Aggregate approXimation (SAX) method. We then partition the symbolic time series into a sequence of words. Then we employ the word embedding neural model known in Natural Language Processing to train the classifying mechanism. The proposed scheme was applied to classify 30 benchmark datasets and compared with a range of state-of-the-art time series classifiers. The name SAFE comes from our observation that this method is safe to use. Empirical experiments have shown that SAFE gives excellent results: it is always in the top 5%–10% when we rank the classification accuracy of state-of-the-art algorithms for various datasets. Our method ranks third in the list of state-of-the-art dictionary-based approaches (after the WEASEL and BOSS methods).

1. Introduction

Time series classification has attracted much attention in the last decade (Bagnall, Lines, Bostrom, Large, & Keogh, 2017). In this area, dictionary-based classifiers have become increasingly important for various applications. Their advantage emanates from the fact that they are usually fused with advanced machine learning models such as deep neural networks (Ismail Fawaz et al., 2020).

Motivation. The research undertaken in this paper was inspired by recent studies related to neural networks in Natural Language Processing (NLP). In particular, we wanted to explore the potential of a neural network model enriched with an embedding layer in application to time series analysis. We emphasize the conceptual similarity between a document (a consequence of words) and a time series (a consequence of numeric values). This paper proposes a new time series classifier that operates on time series transformed into sequences of words. The rationale for using an embedding layer is that it is known for its ability to capture relationships between words. The similarity of words is inferred based on the context in which they occur. We envision that when a time series is converted to a sequence of words, we will be able to mine some significant patterns originating in the numeric data. The use of an embedding layer allows us to take advantage of the context in which this pattern occurs.

* Corresponding author.
E-mail addresses: nuzhat@cse.mist.ac.bd (N. Tabassum), A.Jastrzebska@mini.pw.edu.pl (A. Jastrzębska).

https://doi.org/10.1016/j.ipm.2022.103044
Received 26 November 2021; Received in revised form 14 July 2022; Accepted 20 July 2022
Available online 2 August 2022

Contribution in a nutshell. We defined a new time series classification scheme. It consists of two steps: (1) Conversion of time series into sequences of words and (2) Classification using a neural network with an embedding layer. We treat the time series as strings of words made of a predefined alphabet set. The step of transforming the numeric data into a symbolic representation is guarded by hyperparameters specifying the variability and length of words. The time series in the new form are used to train the neural network model. We present an extensive experimental evaluation that points to a recommended neural architecture. The new method is named SAFE, which stands for *Simple And Fast segmented word Embedding based neural time series classifier*. The flow of the designed procedure was tailored to achieve satisfying classification quality. As shown later in the paper, this algorithm ranks in the top 5%–10% of the best algorithms for each dataset tested. Thus, we believe it is SAFE to use. Our method ranks third in the list of state-of-the-art dictionary-based approaches, after the WEASEL and BOSS methods. Detailed research objectives and the novelty of the contribution are discussed in Section 2.

Practical advantages. The great advantage of our approach is that despite being a neural network, the models created are small in size. More specifically, the number of trainable parameters is around 20k even for very large datasets. Thus, the model can be both trained and used on devices with limited resources, such as wearable electronics. Unlike many existing dictionary-based approaches, the proposed method does not require dimensionality reduction before training the classifier. The method does not truncate any time series data points when converting numeric time series to symbolic time series. Our proposed method preserves information concerning each time point in a time series, contrasting previous time series classifiers such as BOSS, SAX-VSM, and others. This is an advantage making our scheme quite straightforward. All named features are present in the model, which is additionally versatile and achieves high classification accuracy rates for different types of datasets.

Organization. The developed scheme was used in a series of empirical experiments conducted using datasets from the https://timeseriesclassification.com/ web service. We compare our method's classification results to those achieved by state-of-the-art methods. Our comparisons show that the new scheme in several cases gives better results than state-of-the-art.

The remainder of the paper is structured as follows. Section 2 outlines our research objectives. Section 3 presents literature review. Section 4 presents the proposed approach. Section 5 discusses conducted experimental evaluation and a comparison with the state-of-the-art methods. Sections 6 and 7 concludes the paper.

2. Research objectives and contribution

Objectives. Time series are ubiquitous, thus, it does not come as a surprise that time series classification became a thriving area of study. Methods from this domain after appropriate generalizations and modifications find applications in other research areas, for instance, in event sequence classification (Rebane, Karlsson, Bornemann, & Papapetrou, 2021). We need new, cross-sectional approaches that have the potential to be applicable to various sub-domains of data processing. Such methods ease the integration of machine learning modules in practical domains. To meet this goal, we have developed a *novel time series classifier that uses word-based time series representation and performs the task of time series classification using a word embedding neural network model*.

Position in the landscape of other word-based time series classifiers. The concept of tokenizing time series into sequences of words is already recognized in the literature. Such group of classifiers is known as dictionary-based and it includes, for instance, SAX-VSM (Senin & Malinchik, 2013) and BOSS (Schäfer, 2015). Nevertheless, the existing dictionary-based classifiers do not use neural networks. Needless to say, they do not use an embedding layer to process data transformed into word sequences. Therefore, the successful application of neural networks with a word embedding layer is the crucial contribution of this work. It is worth noting that using neural networks in this role is a significant advance because it opens up the possibility of incremental training when processing data streams. We have tested and proposed a well-performing (safe to use) configuration of preprocessing algorithms that can be applied with our method and achieves satisfactory classification accuracy on multiple datasets.

Because our classifier has a low computing cost and a simple and compact model, it may be implemented on a mobile or small device with limited resources and perform real-time classification.

Table 1 compares the features of our proposed time series classifier (SAFE) with other dictionary-based classifiers. In the feature extraction phase, all dictionary-based classifiers (WEASEL, BOSS, SAX-VSM) truncate the time series, while our proposed method does not truncate the time series and retains the entire length. Authors of the other techniques employ a fixed-size alphabet made of four symbols (a, b, c, and d) to represent the time series and suggest maintaining this solution. In contrast, we believe that the alphabet size should be a tunable hyperparameter, and we propose to check the range between 7 and 20. In addition, our proposed solution used a word embedding neural network for the classification task. In contrast, the rest of the dictionary-based classifiers relied on logistic regression, closest neighbor, and tf-idf cosine distance approaches.

Position in the landscape of word-based approaches to time series analysis. We should mention that word-level-based approaches were delivered for time series processing tasks other than classification. A dominant group of such procedures aimed at extending the idea behind the word2vec method to numeric data. Word2vec is a technique that uses a neural network model to learn word associations from a large corpus of text (Mikolov, Chen, Corrado, & Dean, 2013). A very natural application area of such an approach is finances. In this domain, we can take the benefit of the method termed stock2vec (Bruyn, 2018; Wang, Wang, Weng, & Vinel, 2020). Principally, the stock2vec model is trained on time series coming from a single domain. In other words, the dataset is quite homogeneous.

An approach that turns a generic numeric time series into a sequence of symbols was termed signal2vec (Nalmpantis & Vrakas, 2019). The authors of this approach delivered it to work with big-data time series. Their corpus was composed of words obtained

Table 1
Comparison of dictionary-based time series classifiers.

Algorithm	Features						
	Time series to string method	Discretization method	Truncation of time series	Alphabet size	Word length	Remove duplicate words	Classifier model
WEASEL	Truncated SFA	Multiple Coefficient Binning (MCB)	Yes	4	[4, 6, 8]	No	Logistic regression
BOSS	Truncated DFT	Multiple Coefficient Binning	Yes	4	[8, 10, 12 14, 16]	Yes	BOSS distance function with KNN
SAX-VSM	SAX with PAA	Equiprobability	Yes	4	Proportional to noise level	No	Tf-idf weighting features frequency with cosine distance
SAFE	SAX without PAA	Equiprobability	No	tuned (up to 20)	[2, 4, 6, 8]	No	Word embedding neural network

from the same time series. This ensured the validity of the method. An essential contribution mentioned by the authors of the signal2vec was that it allowed efficient forecasting of big-data in an online manner.

An intuitive limitation of a corpus-based training scheme for word-level time series representations is that produced words should capture some characteristic patterns occurring in the time series. The same is required from a sequence of patterns. In other words, there is an intuitive need for repetitiveness for such a method to work, in an analogous way as we rely on this repetitiveness when we process data in natural language. This requirement was satisfied in the works on the stock2vec and signal2vec method. In the case of time series classification, the prerequisite is the same: to have a corpus made of time series from an area similar to the dataset that we wish to classify. Since the benchmark data used at the moment in time series classification research comes from a wide variety of sources (such as electric signals, audio signals, sensors, and many more), we believe that building a model using a corpus other than made of one time series dataset could raise doubts. Furthermore, one of our design goals was to deliver an approach that can be both trained and used on low-resource hardware. We did not want to consider options counter-intuitive to this assumption.

3. Literature review

3.1. Literature review on time series classification

3.1.1. Dictionary-based approaches

There are several groups of algorithms dedicated to time series classification. From the perspective of this study the most relevant ones are dictionary-based approaches. In this domain, a dictionary is a collection of time series segments made of symbols. Each such segment is interpreted a word. We envision that patterns allowing to categorize time series into a correct class emerge both on the level of a single word and on the level of a sequence of words (Pan, Pan, Su, & Chen, 2018).

Many dictionary-based algorithms use Symbolic Aggregate ApproXimation (SAX) as a first step. SAX creates a high-level symbolic representation of time series and simultaneously can, optionally, reduce the dimensionality of a given time series (Bai, Li, Wang, Wu, & Yan, 2021). SAX relies heavily on computing the average of subsequences for symbolization. Therefore, it is prone to fail when applied to noisy data.

A famous dictionary-based algorithm is Bag of Patterns (BOP) (Hatami, Gavet, & Debayle, 2019). It uses a histogram representation for time series data, similar to the bag of words approach known from NLP. The discovered patterns are then classified based on their frequency of occurrence.

An extension of the BOP approach combined with SAX is SAX-VSM (Senin & Malinchik, 2013). In SAX-VSM, patterns are discovered by using a shapelet extraction method. (Shapelets are subsequences of time series with different lengths and offsets that contain distinct patterns.) Shapelets in SAX-VSM are treated as words, weighted using tf-idf, and then classified. The elementary parameters of this procedure are the length of the sliding window, the number of segments in the window, and the size of the SAX alphabet.

Another popular dictionary-based method is BOSS, short for Bag of Symbolic Fourier Approximation Symbols (Schäfer, 2015). According to the authors, it is three times more robust to noisy data than classifiers based on Dynamic Time Warping (a baseline approach used in many areas of data processing) (Lopez-Otero, Parapar, & Barreiro, 2019). It uses Symbolic Fourier Approximation (SFA) (Rezvani, Barnaghi, & Enshaeifar, 2021), which is responsible for its relatively high tolerance to noise. SFA is applied to each window to generate words, which are then classified. It computes the mean values to approximate and quantize the given time series. In the classification stage, the order of the words is ignored, making it a bag of words method (same as BOP). Spatial Bag of SFA Symbol (sBOSS) (Large, Bagnall, Malinowski, & Tavenard, 2019) is another dictionary-based approach that uses Spatial Pyramids combined with BOSS and HI (Histogram Interaction) to incorporate temporal and dictionary features, yielding much more accurate

results than standard BOSS. It performs better when discriminatory shape frequency features are incorporated into the noisy data. Nevertheless, it requires more memory than the standard BOSS, and it is arguable whether this extra memory is worth the slight increase of accuracy. Both methods require a significant amount of time and space for large datasets. To mitigate these issues, a variant of this technique called RBOSS was proposed. It also known under the name contract BOSS (cBOSS). It is faster than both BOSS and sBOSS, due to the inclusion of a randomization stage (Middlehurst, Vickers, & Bagnall, 2019). The introduced modification concerns parameter selection which is inbuilt in the algorithm. The cBOSS tests only k random parameter sets, which speeds up the procedure. This comes at the cost of a slightly lower accuracy, but the difference is small.

Another successful dictionary-based classifier is WEASEL (Word ExtrAction for time SEries cLassification) (Schäfer & Leser, 2017). It converts time series into feature vectors. Similar to BOSS, a sliding window method is used to generate words. Then, the words are processed using a standard machine learning classifier. It calculates the Fourier coefficients of the words in the training set and uses the Chi-Squared test to identify the words that are most informative. And afterwards, these words are taken into account as the procedure's features. Words in WEASEL come in pairs, depending on the window's length and whether they were quantized as a single word or as two separate words. The key difference between BOSS and WEASEL (both created by the same team) is that WEASEL produces less distinct features, which reduces training time.

3.1.2. Prominent distance-based approaches to time series classification

Naturally, the field of time series classification is rich in non-dictionary approaches. First of all, we should mention distance-based methods based on the distances between warped time series. Behind the vast majority of methods that fall into this category is the Dynamic Time Warping (DTW) algorithm. It is an optimization procedure that, given two time series, X and Y, can align some neighboring points in X to corresponding points far apart in Y. In this way, DTW shrinks or expands parts of a time series.

Another variant is Feature-Based Dynamic Time Warping (abbreviated as FBDTW or DTW_F) (Franses & Wiemann, 2020). It aligns two sequences based on every point's local and global properties rather than its value or derivative. It outperforms both classical and derivative DTW in estimating pairwise distances of time series sequences. We shall also mention the Local Slope feature DTW (LSDTW) that focuses on time series subsequences when performing similarity calculations (Yuan, Lin, Zhang, & Wang, 2019). Another popular variant called Weighted DTW (WDTW) penalizes points with a larger phase difference between the reference and test point from the two compared series X and Y to prevent minimal distance distortion caused by outliers. Geler, Kurbalija, Ivanović, and Radovanović (2020) applied an analogous procedure but the DTW distance is computed in a constrained manner that aims at reducing its computational cost. DTW itself is only a flexible similarity measure. The final task of assigning a class label still needs to be performed by a separate classification algorithm. In this role, we often encounter the Nearest Neighbor classifier. However, other methods are also used. For example, Iwana, Frinken, and Uchida (2020) use a neural network (NN).

3.1.3. Plain classifiers applied to time series classification

We should mention that plain classifiers prove to be very effective in time series classification tasks. In this context, a plain classifier is a method that was delivered to work with a regular data classification task.

An in-depth review of several state-of-the-art plain methods dedicated to time series data was presented by Bagnall et al. (2017). The authors report that the results produced by many of the algorithms tested were similar to the 1-Nearest Neighbor classifier with DTW and Rotation Forest. Rotation Forest (Juez-Gil, Arnaiz-González, Rodríguez, López-Nozal, & García-Osorio, 2021) is a decision tree-based ensemble using Principal Component Analysis.

Among new methods for time series classification that originated from a domain not typically associated with time series analysis, we may mention approaches based on Fuzzy Cognitive Maps, which are a soft computing tool for data processing (Homenda & Jastrzebska, 2020).

Bagnall et al. (2017) reported that COTE was found to be the best classifier. COTE, short for Collective of Transformation-Based Ensembles (Bagnall, Lines, Hills, & Bostrom, 2015), is an approach that uses an ensemble of classifiers and applies various transformations to the data. Thus, its effectiveness is enhanced as it is a meta-learning scheme rather than a new and distinct algorithm for class label assignment. An improved version of COTE was called Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) (Lines, Taylor, & Bagnall, 2016). HIVE-COTE can be described as a modular hierarchical structure with a probabilistic voting mechanism. Two modules representing dictionary-based and interval-based classifiers for each produced data transformation are created. A substantive disadvantage of the HIVE-COTE approach is that it is extremely slow.

3.1.4. Dedicated time series classifiers working with numeric time series representation

Recently, more scalable time series classification methods have been developed, such as TS-CHIEF (Time Series Combination of Heterogeneous and Integrated Embedding Forest) (Shifaz, Pelletier, Petitjean, & Webb, 2020) and Inception Time (Ismail Fawaz et al., 2020), which have achieved state-of-the-art accuracy. However, they are characterized by high computational complexity and require significant training time even for small datasets. TS-CHIEF uses the Proximity Forest classifier (Lucas et al., 2019) with a tree structure to create a classification model. The training complexity of TS-CHIEF is quasi-linear concerning the number of training examples but quadratic with respect to the length of the time series. On the other hand, InceptionTime is a collection of five deep Convolutional Neural Networks based on the Inception architecture that outperforms HIVE-COTE in terms of computational speed and accuracy. Compared to the recent success of time series classification using Conventional Neural Networks, a faster approach known as RandOm Convolutional KErnel Transform (ROCKET) (Dempster, Petitjean, & Webb, 2020) was developed. It achieves state-of-the-art accuracy by using simple linear classifiers with random convolutional kernels. This method is also better-suited to handle large datasets. The authors of ROCKET have also delivered its modified variant named MiniROCKET, which is a bit worse

than the original one in terms of accuracy, but it is much faster. Very recently, Tan, Dempster, Bergmeir, and Webb (2022) published a paper concerning a further extension of the ROCKET algorithm that adds multiple pooling operators to the feature extraction step of the original algorithm.

Last but not least, let us mention that in the time series classification task, like in the standard pattern classification task, we may distinguish two phases: feature extraction and classification. In the standard classification task, features typically come with a well-defined meaning. For example, if we classify patients as sick or healthy, we may have features such as age, Body Mass Index, sex, etc. In the case of time series classification, like for the case of image classification, or generally signal classification, we can extract features according to multiple schemes. The substantive difference between the aforementioned groups of approaches to time series classification is how features are computed (Middlehurst et al., 2019). Our SAFE and the other dictionary-based methods use word-based data representation. Sequences of words are assumed to be the features based on which we perform further processing, and finally, data classification. In contrast, methods such as ROCKET or HIVE-COTE use an intermediate numerical representation to extract features (Middlehurst, Large, Flynn, Bostrom, & Bagnall, 2021).

3.2. Literature review – word embedding models in NLP

The overwhelming success of classification strategies known from the domains not related to time series in the time series classification task inspired us to use a method known as word embedding, prevalent in NLP.

When dealing with text, the first step is to develop a technique for transforming strings into vectors (arrays of numbers) before feeding them into the model. This technique is commonly referred to as text vectorization. One of the simplest methods is to do a one-hot encoding of each word in the vocabulary. It is a vectorial representation of a vocabulary, in which each word is linked with one index in a vector. To encode a word, we place one in the position corresponding to this word. To represent a sequence of words, we concatenate several one-hot-encoded vectors. As one may imagine, this representation is vastly sparse. It also does not scale well, as the representation must be extended each time a new word appears. Named disadvantages of the one-hot-encoding technique make it quite inefficient.

We can encode each word in the vocabulary using a unique number to overcome the sparse vector problem. This can also cause issues because such encoding does not provide any information about the relationships between words. Also, the integer encoding can lead to problems with model training (Huang, Deng, Shen, & Chen, 2020). Furthermore, as Diallo et al. (2021) report, a model built using the integer encoding cannot be interpreted easily. Assume, for instance, that we build a linear regression classifier, which learns a single weight. The usage of the integer encoding would render learned feature-weight combinations meaningless because there is no relationship between the similarity of two words and the similarity of their encoding.

We can use word embedding to preserve the relationships between words and maintain a dense representation. A dense vector of floating-point values of a fixed length is called an embedding. They are not defined manually but instead trained as weights to understand the similarity of different words. The concept of embedding is similar to a lookup table. After learning these weights, you can encode each word by searching for the dense vector it refers to in the table (Lin, Fan, Chen, Chen, & Zhao, 2021). Recently, a combination of the continuous bag of words (CBOW) model with continuous skip-gram model has been shown to be very effective for learning syntactic and semantic word similarities (Mohasseb, Bader-El-Den, & Cocea, 2018; Roostaee, Sadreddini, & Fakhrahmad, 2020). The authors focused on undersampling frequent words and training the model with negative samples of words that do not occur in the same context. The CBOW model can predict the middle word based on the surrounding words, and the skip-gram model can predict the surrounding words given the word in the sentence.

A word embedding method called word2vec starts by creating a global vocabulary from a text data set. This global vocabulary, known also as dictionary will be represented with the use of vectors. The training procedure aims at filling in the values of these vectors in a way to represent word closeness. In other words, training procedure seeks for repetitive patterns in a corpus.Consequently, the word meaning in multiple contexts is neglected, and the word frequency takes precedence. Let us mention that word2vec in its original form was delivered with a two-layered neural network by Mikolov et al. (2013). Arora, May, Zhang, and Ré (2020) underline that with labeled data and basic language, non-contextual embedding works best. On the other hand, contextual embedding approaches are used to acquire sequence-level semantics by analyzing the order of the words in the texts. The embeddings are obtained by sending the complete sentence to the pre-trained model. Each word's embedding is determined by the other words in the text, referred to as context. This embedding works better for natural language data sets with complex text structures, yet these approaches are also more expensive and heavy in design (Arora et al., 2020).

Traditional word embedding models have been used in many contexts, including data classification tasks such as sentiment analysis (Behera, Jena, Rath, & Misra, 2021), Named Entity Recognition (Nozza, Manchanda, Fersini, Palmonari, & Messina, 2021) or topic analysis (Zhao et al., 2021) and other tasks such as business data parsing (Luo et al., 2021), reviews mining (Zhang, Fan, Zhang, Wang, & Fan, 2021) or fake news detection (Goldani, Safabakhsh, & Momtazi, 2021).

This paper uses the word embedding technique in a new role: for time series classification. We consider only the non-contextual approach.

3.3. Literature review – word embedding models in areas different than NLP

The literature suggests that employing a non-contextual word embedding scheme to data different from documents in natural language can result in positive outcomes. In light of the study addressed in this paper, it is worth recalling the relevant research on numeric time series modeling that employs NLP tools. We have already signaled these studies in the Introduction, where we mentioned that the word2vec model was applied to process standard time series.

A straightforward adaptation of the word2vec algorithm to time series data is called signal2vec (Nalmpantis & Vrakas, 2019). It consists of two steps: discretization and the skip-gram model. The latter one is responsible for the embedding model. The authors of this approach considered it mainly to process large-scale time series datasets efficiently. The paper focuses on the capability of dimensionality reduction achieved thanks to the embedding. The enclosed case study is oriented on very long time series. A similar concept was delivered by Lee, Kim, and Youn (2021). The authors applied embedding representation to large-scale time series in the task of online forecasting. The achieved advancement compared to standard approaches was that they managed to speed up the procedure without increasing the error too much because they randomly sampled embedding vectors in the model training phase.

We shall also recall a model termed stock2vec (Wang et al., 2020). It utilizes an embedding layer as a deep neural network component trained to forecast financial time series. Bruyn (2018) underlines that promising results were also achieved using the embedding model called GloVe.

Alternative applications of the word embedding model go beyond time series. Mohan and Pramod (2021) applied word2vec to forecast changes in graph topology, which is an advanced problem of temporal data forecasting. The embedding layer in this context is aimed at representing graph neighborhoods. Furthermore, we have seen successful word embedding applications to image processing tasks such as image feature extraction (Cheng, Tian, Yu, & You, 2020).

4. The method

We can divide the proposed approach into three steps.

Step 1 Transform time series into a symbolic representation.

Step 2 Produce sequences of words based on the symbolic representation. Assume that each time series converted into a sequence of words is a single document. Create a corpus made of such documents.

Step 3 Use the created corpus to train a neural network to classify time series.

The following sections outline the inner workings of this approach.

4.1. *Step 1* converting numeric time series into symbolic time series

4.1.1. Normalization

The introductory stage of the procedure preprocesses the data by applying z-normalization. This step allows the exploration algorithm to focus on structural similarities and contrasts while fitting the data to a Gaussian distribution. To calculate the z-score for a given time point, the mean value of the entire time series is first subtracted from the initial value of that time point. Then the outcome of the subtraction is divided by the standard deviation of the time series. A time series T_x of length n, $T_x = t_{x_1}, t_{x_2}, \ldots .., t_{x_n}$ can be normalized into time series T_z of length n $T_z = t_{z_1}, t_{z_2}, \ldots .., t_{z_n}$ by using z-score given in Eq. (1).

$$t_{z_i} = \frac{t_{x_i} - \mu_x}{\sigma_x}, \; i \in n, \tag{1}$$

where t_{z_i} is the ith element calculated for normalized time series, t_{x_i} is the observed time series value in ith time point, μ_x is and σ_x are the mean and standard deviation of the time series T_x respectively.

4.1.2. Converting time series to sequences of symbols using simple SAX

After normalizing the time series, we convert the numeric time series to a symbolic/alphabetic representation in the next step of the procedure. We considered the Simple Symbolic Aggregate approXimation (SAX) algorithm as a candidate method for this task (Bai et al., 2021). In a typical SAX implementation, the dimensionality of the time series is first reduced using PAA (Piecewise Aggregate Approximation). Then, the shortened time series is converted to symbols using a discretization method. For instance, in the paper (Senin & Malinchik, 2013), they applied SAX with PAA, reducing the time series' dimensionality with the PAA score and then convert the time series to SAX words. subsequently utilized the vector space model (VSM) for a further classification procedure. This PAA representation increases the chances of missing vital parts in the time series data. Therefore, we used a simple version of SAX conversion in our time series classification method that excludes the PAA step. We applied the SAX discretization method to convert a time series T of length n to a sequence of symbols (letter/alphabet) of the same length, where alphabet size is denoted as a, $a > 2$.

We applied the SAX discretization technique for each normalized time series, which converts it into a discrete symbolic representation corresponding to its equiprobability. In discretization process, a list of distinct breakpoints $Br = \{Br_1, Br_2, \ldots, Br_{a-1}\}$ ($Br_0 = -\infty$ and $Br_a = \infty$) such that $Br_j < Br_{j+1}$, where $j < a$, are created by evenly slicing the area under the Gaussian curve

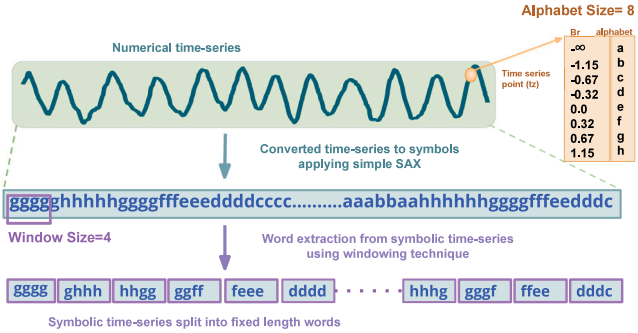


Fig. 1. Illustration of the idea behind converting numeric data to symbolic data and extracting words from it.

with the chosen alphabet size, a. A lookup table with local equiprobability is then generated, and the respective alphabet symbol (a letter) is allocated to each breakpoint interval $[Br_j, Br_{j+1}]$, where $j < a$. As normalized time series typically yield a normal distribution $N(0,1)$, each time point, t_{zi} of the normalized time series is then compared with the breakpoint list. If the time point is located between two break points then the numeric time point is replaced by the allocated alphabet symbol for that particular break point interval. Following this lookup process, the entire normalized time series (Tz) is converted into a symbolic time series $\hat{T}_s = \widehat{t_{s1}}, \widehat{t_{s2}}, \widehat{t_{s3}}, \ldots, \widehat{t_{sn}}$, is constructed according to Eq. (2).

$$\widehat{t_{si}} = alphabet_j, \ iff \ Br_j \leq t_{zi} < Br_{j+1}, \ where \ i \leq n \ , \ j < a, \tag{2}$$

where n is the length of time series (T_z), a is the alphabet size and $alphabet_j$ is the jth symbol in the alphabet. For example, if $a = 4$ then the time series is represented by $alphabet = \{alphabet_0 = a, alphabet_1 = b, alphabet_2 = c, alphabet_3 = d\}$.

All numeric time series in the training dataset are transformed into symbolic time series at this stage, resulting in a collection of strings. Following that, each symbolic time series is passed to a module that extracts words from it. The conversion of numeric time-series into symbolic time-series using an eight-letter alphabet is illustrated in Fig. 1.

4.2. *Step 2* extracting words from symbolic time series

In the proposed method, a sliding window technique is used to extract words from each time series. The window size is denoted as w, where $2 \geq w \geq 7$, which allows the extraction of words of length w from a time series. The range of word lengths is arbitrary. We want the number of words to be large enough to train a meaningful model, even for short time series. Therefore, we choose relatively short words. The secondary justification for choosing this range was that the average word length in English is 4–7 characters. The sliding window divides the symbolic time series $\hat{T}_s = \widehat{t_{s1}}, \widehat{t_{s2}}, \widehat{t_{s3}}, \ldots, \widehat{t_{sn}}$ into a word list. Words have a fixed length w. We focus on the variant in which word extraction is without overlap:

$$\begin{aligned} &WordList(\hat{T}_s, w) = \{W_1, W_2, W_3, \ldots, W_{n/w}\} \\ &\quad \text{where,} \\ &\quad W_1 = \{\widehat{t_{s_1}}, \ldots, \widehat{t_{s_w}}\}, W_2 = \{\widehat{t_{s_{w+1}}}, \ldots, \widehat{t_{s_{2w}}}\} \\ &\quad W_3 = \{\widehat{t_{s_{2w+1}}}, \ldots, \widehat{t_{s_{3w}}}\}, \ldots, \\ &\quad W_{n/w} = \{\widehat{t_{s_{(n/w-1)\cdot w+1}}}, \ldots, \widehat{t_{s_{(n/w)\cdot w}}}\}. \end{aligned} \tag{3}$$

Following the application of the windowing technique, each symbolic time series is transformed into a word list named $WordList$, which contains n/w words $W_i, I \leq n/w$, where n denotes the length of the time series and w denotes the fixed size. Word length ranges from 2 to 8. In the non-overlapping word extraction approach defined by Eq. (3), word extraction starts at position $p = 1$ of each time series. Assume that the window size is w. The w-element sequence of symbols contained in the window is treated as one word W_i. Then the window is moved to position $p = p + w$ in the time series to find the next word. In this way, a sequence of words each of length w is extracted from the string series. The word extraction approach using the sliding window technique is shown in Fig. 1. In this case, a window of length four is used.

In the SAX-VSM method presented in Senin and Malinchik (2013), the authors use a weighted word frequency method to perform the classification, which leads to the model losing out on information about the ordered sequence of events. This would mean that if two classes of time series data differ by order of their values and not the pattern of values in the word window, a word frequency method will not be able to differentiate between them. Similar can be said for methods such as BOSS (Schäfer, 2015) and WEASEL (Schäfer & Leser, 2017), which use Bag of SFA symbols and Bag of Patterns (BOP) and not the order information. We attempt to alleviate this issue by providing the data in its original order to the classifier. Although we do not rely on recurrent neural networks to capture the dependency of a word on the ones before or after it, for datasets that have a pattern occurring in specific time points of the series, the weights would be assigned accordingly. This can be a disadvantage in cases where we cannot guarantee a uniform window of time series data.

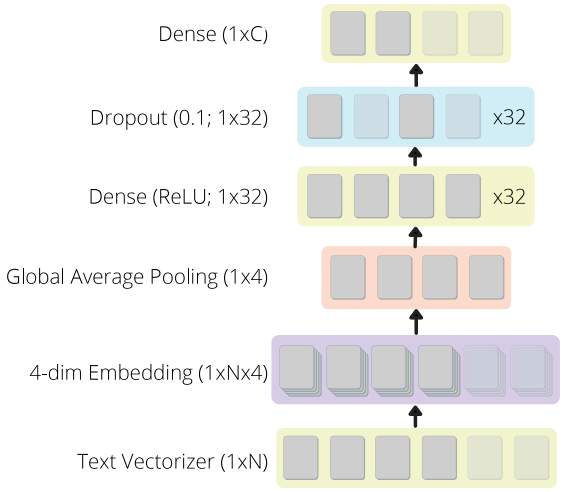


Fig. 2. Proposed neural model architecture. N here denotes the number of words generated for a single time series in a set, and C denotes the number of label classes, which can be 2 (binary) or above. N is different for each dataset, and it depends on the word length (which is a parameter denoted as w in later parts of the paper). The one at the start of the dimensions for each layer is to denote the one-dimensional array of words that constitute a sample of time series fed to the model.

A note on time complexity of the preprocessing step. When executing the SAX algorithm, we do not perform dimensionality reduction. That is, we do not perform the PAA algorithm. Since the Python library used (Senin, 2021) in our experiments for SAX operations has a dictionary of precomputed segments for alphabet in the range 3–20, the bin determination operation for each alphabet is performed in O(1) time. Thus, both SAX and z-normalization have O(n) complexity. The final step is to segment the strings into words, which is also performed in O(n) time.

4.3. *Step 3* neural model design and training

Let us reiterate that the essential task discussed is to convert time series data into sequences of fixed-length words. After obtaining the time series corpus, we construct a neural classifier model using word embedding. Our model employs text vectorization and word embedding to create a dense numeric representation of the input string.

Fig. 2 shows the model architecture used to train a multi-class classification model on text data. The text vectorizer is fitted first to the training set. It analyzes the vocabulary of this data and generates a numeric index for each word. Since all words in the input strings have the same length, padding is not necessary. This vectorized data is fed to the embedding layer to encode each number into a vector of four floating-point numbers.

From the experimental evaluation reported in Section 5.3, we have discovered that an embedding dimensionality higher than four does not improve model accuracy but increases train execution time. Since we aim to develop a faster classifier, we set the embedding dimension as 4 for our proposed model to save time. The embedding weights are learned during the training to match the word's context. Embeddings are then passed through a one-dimensional global average pooling (GAP) layer, which takes the average of each embedding feature map rather than putting fully connected layers on top of the feature maps according to Eq. (4). Since GAP does not have any parameters to optimize and it replaces the neural network's fully connected blocks, overfitting is prevented at this layer.

$$GAP = \sum E_{m_i}/m, \;\; i \leq m, \tag{4}$$

where E_{m_i} denotes each of the four embedding layers and m is the number of selected embedding layer, in our case that is 4. Overfitting is avoided since there is no parameter to optimize in global average pooling. This also helps aggregate the temporal information from the time series, making it more robust. Pre-training of the embedding layer is not something that can be meaningfully done for time series datasets because the patterns are not agnostic across datasets. Vocabulary is synthesized, and we cannot expect the same words to be generated across multiple datasets and still have the same meaning. Literature underlines that contextualized word embedding models are most beneficial in the case of complex natural language structure (Arora et al., 2020). Furthermore, our experiments and the results obtained by other researchers working on dictionary-based methods for time series classification (Middlehurst et al., 2019; Schäfer, 2015; Schäfer & Leser, 2017; Senin, 2021; Senin & Malinchik, 2013) have shown that word length should be tuned individually for each dataset. Since the word length is fixed, different words are generated for different word lengths.

A fully connected layer then processes the data with ReLU activation and dropout of 0.1. Computing the ReLU (Rectified Linear Unit) activation function, $f(x) = max(0, x)$ is straightforward because it returns 0 if the input x is negative and for positive data the function returns x. This keeps the computation required to run the neural network simplified as an exponential function does

not need to be computed. The size of the dense layer was set to 32 as we noticed overfitting when increasing it past this size. Additionally, a 10% dropout rate is applied to avoid overfitting to the training data.

L2 regularizer is also added to the dense layer to avoid overfitting. It modifies the cost function by adding L2 regularization element ($L2R$) with cross-entropy loss function (L_{CE}).

$$cost function = L_{CE} + L2R \tag{5}$$

The regularization term is defined as the sum of squares of all link weights ω_i in a neural network multiplied by the regularization parameter λ for a size n training set as stated shown in Eq. (6). In most cases, the λ value is between 0 and 0.1 . In our experiments, we started with λ value as 1e-1 and found that the value of 1e-4 helped in stabilizing our model (Thakkar & Lohiya, 2021).

$$L2R = \lambda/2n \sum_{i} (\omega_i)^2, \tag{6}$$

The L2 penalty promotes weight depreciation. By deploying small weights into the network, it is possible to penalize the complicated model, which reduces overfitting and improves the performance of the models for fresh inputs.

The last layer is a fully-connected layer with the number of nodes equal to the number of classes. Since we have mutually exclusive classes, the model is compiled using a sparse categorical cross-entropy loss function for each class,

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(P_{t_i}) \tag{7}$$

where P is the softmax probability of the class, t_i is the integer encoded truth label for the class, n is the number class, and the function computes the logarithm only once for each instance of a class. The output is computed using the $argmax(x)$ operation on the output of the final layer. The proposed procedure requires two hyperparameters to be set: the learning rate and the number of epochs. The model was trained for 100 epochs, which ensured a good balance between training time and classification accuracy, especially for bigger datasets. Lastly, to optimize the network, the model was trained using the Adam optimizer. On average, Adam optimizer performs the best (Zhang et al., 2019). It alters the weights of the hidden layer by combining root mean square propagation (RMSprop and adaptive Gradient Algorithm (AdaGrad). Adam computes the exponential moving average of the gradient $\widehat{G_t^{agg}}$ and the squared gradient $\widehat{G_t^{sos}}$. Aggregated gradient $\widehat{G_t^{agg}}$ in Eq. (8) employs an exponentially weighted average of the gradients, which speeds up the algorithm's convergence to the minima. With sparse gradients, it maintains a per-parameter learning rate that enhances neural network's performance. On the other hand, root mean square propagation (RMSprop), utilizes the exponential moving average rather than the accumulated sum of squared gradients in Eq. (9). Here, the weight gradients' average recent magnitudes are also taken into account while adjusting the per-parameter learning rates. Adam optimizer computes the bias-corrected version of G_t^{agg} in Eq. (10) and G_t^{sos} in Eq. (11) to regulate the weights when approaching the global minimum and avoid excessive oscillations while near it. Then, instead of using the usual weight parameter, both $\widehat{G_t^{agg}}$ and $\widehat{G_t^{sos}}$ are combined in Eq. (12).

$$G_t^{agg} = \gamma_1 G_{t-1}^{agg} + (1-\gamma_1)[\frac{\Delta L_{CE}}{\Delta \omega_t}] \tag{8}$$

$$G_t^{sos} = \gamma_2 G_{t-1}^{sos} + (1-\gamma_2)[\frac{\Delta L_{CE}}{\Delta \omega_t}]^2 \tag{9}$$

$$\widehat{G_t^{agg}} = \frac{G_t^{agg}}{1-\gamma_1^t} \tag{10}$$

$$\widehat{G_t^{sos}} = \frac{G_t^{sos}}{1-\gamma_2^t} \tag{11}$$

$$\omega_{t+1} = \omega_t - \eta(\frac{\widehat{G_t^{agg}}}{\sqrt{\widehat{G_t^{sos}}} + \epsilon}) \tag{12}$$

where G_t^{agg} is the sum of gradients and G_t^{sos} is sum of square of past gradients at time t. The weight of the model and derivative of the weight at time t are denoted by ω_t and $\Delta\omega_t$ respectively. ΔL_{CE} is the derivative of loss function, η is the learning rate, ϵ is a small positive constant. Parameters γ_1 and γ_2 adjust the degradation rate of moving average for the two methods.

We chose the Matthews Correlation Coefficient (MCC) and accuracy to compare the performance of the classifiers. MCC has the advantage of generating a high score if the predictor correctly classifies a high percentage of negative data instances and a high percentage of positive data instances, regardless of the class. Its behavior is constant in both binary and multiclass contexts. It assumes values in the interval $[-1, +1]$. The score -1 denotes a complete misclassification, $+1$ indicates a perfect classifier and 0 represents a random choice.

5. Experimental evaluation

5.1. Datasets

We selected 30 datasets collected in different domains for our experiment, including recordings related to devices, ECG signals, time series extracted from contours in images, data describing human motion, some simulated data, and some spectrum-measurement

Table 2
Elementary information about processed time series. Data comes from the http://timeseriesclassification.com web page.

Dataset name	Num. of class	Length	Train size	Test size	Statistics on train set		Statistics on test set	
					min	max	min	max
BeetleFly	2	512	20	20	−2.52	2.51	−2.51	2.41
BirdChicken	2	512	20	20	−2.82	2.12	−3.10	2.44
Coffee	2	286	28	28	−2.06	2.18	−2.12	2.10
Computers	2	720	250	250	−3.75	21.60	−1.61	26.39
DiatomSizeReduction	4	345	16	306	−1.77	1.98	−1.98	2.45
Dist.Phal.Out.AgeGr.	3	80	139	400	−1.91	2.03	−1.99	2.06
Dist.Phal.Out.Corr.	2	80	276	600	−2.18	2.46	−2.16	2.45
Dist.Phal.TW	6	80	139	400	−1.90	2.00	−1.99	2.06
Earthquakes	2	512	139	322	−0.73	7.73	−0.89	7.87
ECG5000	5	140	500	4500	−5.80	4.06	−7.09	7.40
GunPoint	2	150	50	150	−2.37	2.05	−2.50	2.32
Herring	2	512	64	64	−2.19	2.13	−2.21	2.07
Meat	3	448	60	60	−1.54	3.39	−1.49	3.40
Mid.Phal.Out.AgeGr.	3	80	400	154	−1.72	1.72	−1.64	1.92
Mid.Phal.Out.Corr.	2	80	600	291	−1.72	1.88	−1.66	2.07
Mid.Phal.TW	6	80	154	399	−1.58	1.71	−1.72	1.92
OliveOil	4	570	30	30	−1.00	3.72	−1.00	3.73
Plane	7	144	105	105	−2.11	2.91	−2.12	2.92
Prox.Phal.Out.AgeGr.	3	80	400	205	−1.48	1.90	−1.44	1.82
Prox.Phal.Out.Corr.	2	80	600	291	−1.48	1.90	−1.44	1.82
Prox.Phal.TW	6	80	205	400	−1.47	1.85	−1.48	1.90
ShapeletSim	2	500	20	180	−1.81	1.89	−1.87	1.87
SonyAIBORob.Sur.1	2	70	20	601	−2.73	3.63	−3.63	4.00
Strawberry	2	235	370	613	−2.13	3.72	−2.33	3.68
ToeSegmentation2	2	343	36	130	−2.64	3.93	−3.68	5.55
Trace	4	275	100	100	−2.22	3.97	−2.39	3.94
Wafer	2	152	1000	6164	−3.05	11.79	−2.98	12.13
Wine	2	234	57	54	−1.94	3.20	−1.92	3.19
Worms	5	900	181	77	−4.89	4.20	−4.31	4.86
WormTwoClass	2	900	181	77	−4.89	4.20	−4.31	4.86

data. All datasets are publicly available under http://www.timeseriesclassification.com/. Table 2 summarizes the elementary properties of used datasets.

The chosen dataset suite encompasses a wide variety of properties and characteristics of time series. Furthermore, we ensured that the classes present in the selected datasets exhibit various properties. One can inspect this briefly in ribbon plots in Fig. 3 and violin plots in Fig. 4.

Ribbon plots in Fig. 3 depict properties of selected datasets used in our experiment. Each ribbon illustrates the values range in time series for the specific dataset (treated as a whole, without discerning particular classes). At the same time, the darker line represents a single time series randomly chosen from that specific dataset. We can observe from the figure that dataset Strawberry (spectral type) has a decreasing trend, whereas ShapeletSim (simulated type) and Worms (motion type) time series datasets have a constant trend. The dataset Plane (sensor type) and Worms contain seasonal regularities. Computers dataset (device type) contains time series with steps and pulse features, while BeetleFly dataset (image type) is made of primarily non-seasonal cyclic time series. Furthermore, structural break features can be found in the ECG datasets and in the Trace (sensor type) dataset. We can note that the Computers dataset contains time series with the highest difference between min and max values in the series.

Violin plots in Fig. 4 allow to inspect the number of classes present in the selected datasets and the average distribution of values in each class. Each time series class is represented by a violin, with the median of the class denoted by a horizontal line in the violin. Our chosen experimental time series data suit includes time series datasets ranging from two to seven classes. Datasets BeetleFly, Computers, ShapeletSim, and Strawberry contain two time series classes, whereas ECG500 and Worms have five classes. Datasets Plane and Trace contain seven and four classes respectively. A visual inspection of the violin plots leads us to the conclusion that some datasets appear to be easier to categorize (for example the Computers datasets), while others, such as ShapeletSim and Strawberry, appear to be more challenging to classify.

Please note that violin plots do not reflect temporal aspects of the datasets. A detailed inspection of time-related features of time series can be performed using ribbon plots prepared separately for each class present in the selected datasets. We refer the reader to Appendix, where we placed these plots.

5.2. Experimental setup

The following common setup guided conducted experiments:

- We have written the code in Python 3.7. We used the following key libraries: Tensorflow, scikit-learn, saxpy, matplotlib, seaborn, pandas, numpy.

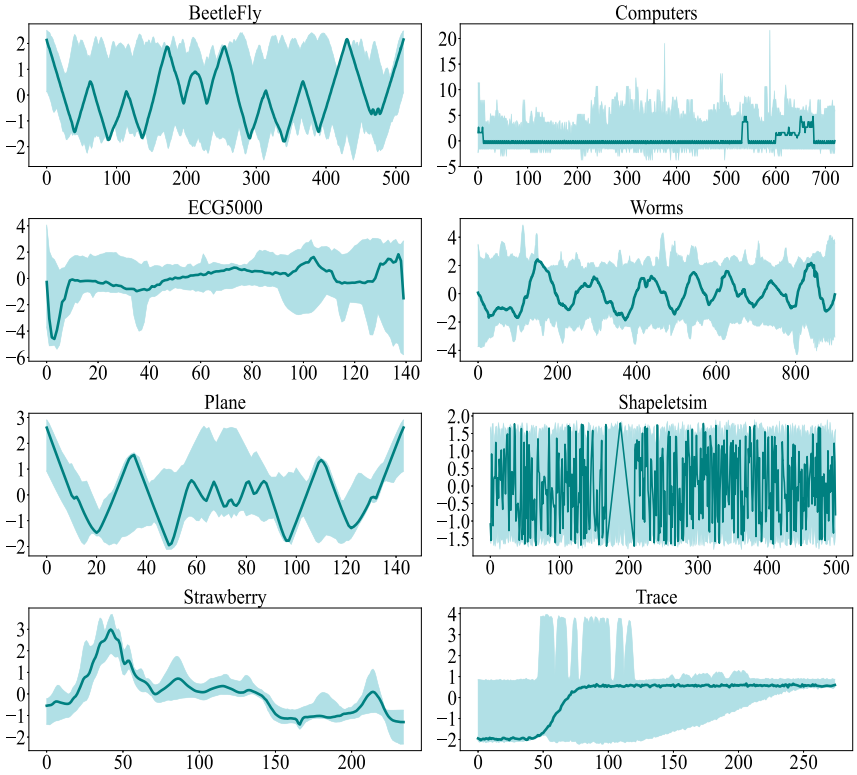


Fig. 3. Example time series from selected datasets showcasing different time-series properties (thin lines). Ribbons represent ranges of values in a given dataset.

- We executed the experiments on standard PCs with Windows 10.
- The dataset repository provides a ready split into training and test sets. For each dataset, the training set was used to create a model. A validation set with a fixed number of 64 samples was taken from the test set to tune the model hyperparameters, with duplicates allowed when the test set had less than 64 test samples. We decided to follow this split for validation as opposed to the usual method of splitting the training data to obtain validation samples to maximize the use of the training set. A similar approach for validation set construction is present in the experiments performed in Asif, Martiniano, Vicente, and Couto (2018). For example, if the train set contains 200 samples and the test set includes 200 samples, the validation set is prepared using 64 samples from the test set. We must emphasize that this set of 64 samples is used to evaluate the validation accuracy at the end of each training epoch and is *never* used for training weights.
- In the empirical study addressed in this paper, several models were generated for each dataset using different combinations of hyperparameters. We tested the models using alphabets with $a = 7$, 8, 9, 11, 12, 13, 15, 17, 18, 19, and 20 symbols as well as words of length $w = 2$, 4, 6, and 8. The models were further tested using learning rates $\eta = 0.1$, 0.05, 0.001, 0.0005, and 0.0001. In each case, we repeated the training procedure 10 times. The default number of epochs was 100, and we inspected the quality of the resulting models in the intermediate steps of training. In Table 7, we see that several datasets converged early to give the best results. Few showed improvements even in the final stages of the training, hinting at the possibility of better predictions if trained longer.
- For each constructed model, we computed the average and best accuracy (ACC) and *Matthews correlation coefficient* (MCC) on the test set. The averages concern ten training repetitions.
- Comparisons with state-of-the-art was prepared with the use of resources on http://www.timeseriesclassification.com/.

In the following subsections, we inspect achieved results. All reported values concern test sets.

5.3. Initial experiments concerning embedding layer dimensionality

The initial experiment aimed at establishing a suitable embedding layer dimensionality that will be common for all models.

The experiment was performed with model hyperparameters fixed for several datasets with different embedding layers. Hyperparameter values are reported in Table 7. We tested models with 4, 6, 8, 16, 32, and 64-dimensional embedding layers. The goal was to compare the accuracy and training time of the model for different dimensionalities. In the following two paragraphs, we report the average accuracy achieved for the test sets in 10 repetitions of the experiment for selected datasets. We also report the

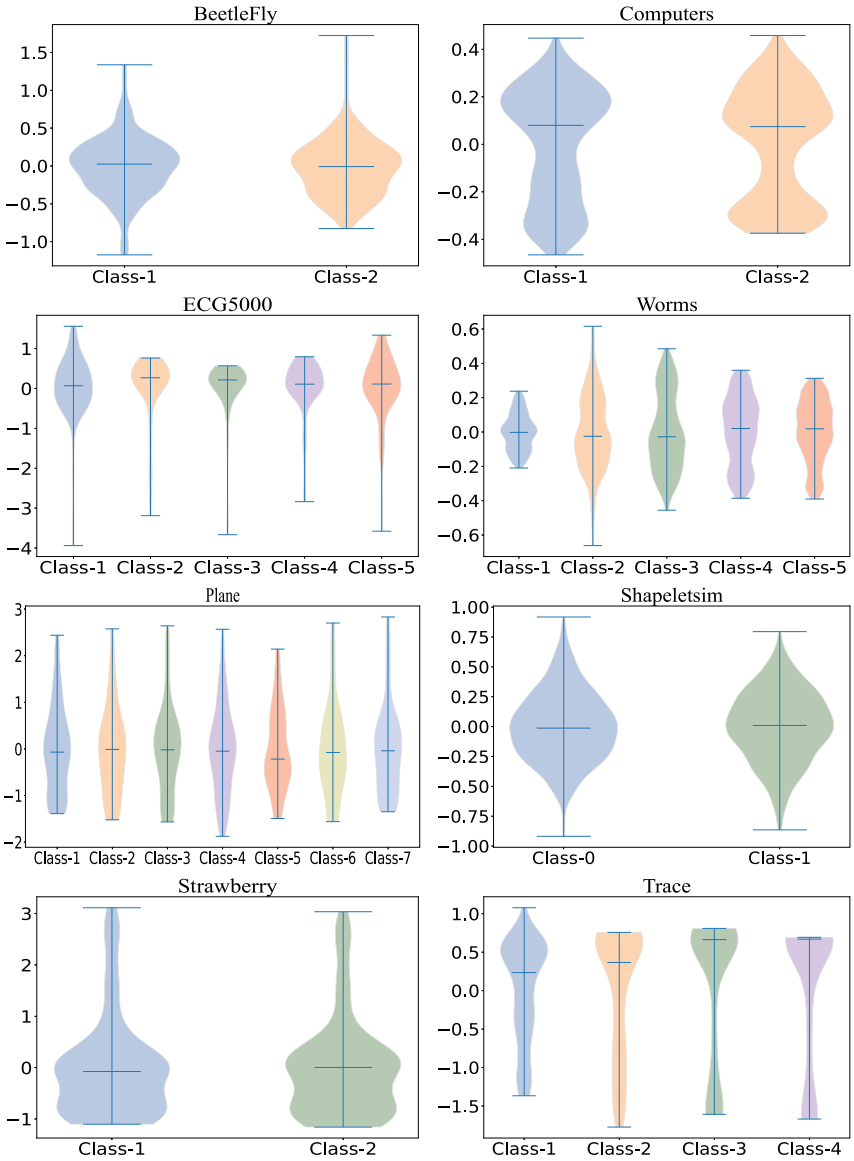


Fig. 4. Violin plots of selected datasets showing variability of values in time series classes.

average time required to train the models for embedding layers with different dimensionalities. The times concern only the neural network training step. Preprocessing time was omitted since it was constant for each dataset as the hyperparameters were fixed.

The achieved accuracy is shown in Table 3. We found that a 4-dimensional embedding delivers the best accuracy for a range of datasets. For a few datasets, a 4-dimensional embedding yields lower accuracy but is close to the best one. For instance, with a 4-dimensional embedding layer, we already achieved a 100% train accuracy for BeetleFly, BirdChicken, and Coffee datasets. It took on average 5.094, 5.954, and 5.646 s to train these models. In contrast, models with 32 and 64-dimensional embedding layers took longer to train. For instance, models for the BeetleFly dataset took around 5.603 and 5.665 s to train for 32 and 64-dimensional embeddings. With a 4-dimensional embedding, we also got the best accuracy of 85.71% and the shortest execution time of 6.304 s for WormsTwoClass. For this dataset, the execution time for the 32-dimensional case was 9.064 s, and 9.044 s for 64-dimensional embedding layer.

For ECG500, the highest test set accuracy was obtained with a 32-dimensional embedding, which yielded the test set accuracy of 90.24% with the train execution time of 10.453 s. The Strawberry dataset was also processed best with a 32-dimensional embedding layer that provided the accuracy of 95.05% and its train execution time was 9.226 s. For Wafer, an 8-dimensional embedding yielded the accuracy of 97.52% and the train execution time was 12.012 s. Nonetheless, the accuracy achieved for these three datasets using a 4-dimensional embedding was very close to the mentioned best results. In the case of a 4-dimensional embedding, ECG500

Table 3
Comparison of test set accuracy for different embedding layers.

Dataset	Embedding layer dimensionality					
	4	6	8	16	32	64
BeetleFly	100	100	85.00	95.00	90.00	95.00
BirdChicken	100	96.43	96.43	96.43	100	96.43
Coffee	100	96.43	96.43	96.43	96.43	96.43
Computer	76.00	74.80	75.20	75.20	75.20	75.20
DiatomSizeReduction	84.64	81.05	85.95	82.68	86.27	85.95
Dist.Phal.Out.AgeGr.	77.70	76.26	76.98	76.26	81.30	79.14
Dist.Phal.TW	69.78	68.35	69.06	68.35	69.78	69.78
Earthquakes	78.42	79.14	76.98	79.14	78.42	77.70
ECG5000	90.02	90.07	90.09	90.11	90.24	89.80
GunPoint	88.67	88.00	88.00	86.67	88.67	88.00
Meat	98.33	98.33	98.33	98.33	98.33	98.33
Mid.Phal.Out.AgeGr.	64.94	62.34	62.99	62.99	62.99	62.34
Mid.Phal.Outl.Corr.	76.98	75.60	75.26	74.57	77.32	74.91
OliveOil	93.33	96.67	93.33	93.33	93.33	93.33
Plane	97.14	99.05	97.14	98.10	100	99.05
Prox.Phal.Out.AgeGr.	88.29	88.29	86.83	87.32	87.80	86.83
Strawberry	94.05	94.05	93.24	93.78	95.05	93.32
ToeSegmentation2	88.46	86.15	88.26	82.31	87.15	87.96
Wafer	97.36	97.44	97.52	97.32	97.08	97.31
Worms	74.03	75.32	79.22	74.03	72.73	74.03
WormsTwoClass	85.71	84.42	84.42	83.12	81.82	83.12

recognition accuracy was 90.02%. It means that the difference between the accuracy of the 4-dimensional and the 32-dimensional model was just 0.22 percent point, while the execution time of the former was shorter and equal to 9.319 s. The difference between the 4-dimensional and the 32-dimensional model train time is 1.134 s and it constitutes a 12.17% increase in train time when moving from the 4-dimensional to the 32-dimensional model. In the case of a 4-dimensional embedding, the Strawberry dataset was recognized with the accuracy of 94.05%, the difference between the best result is 1 percent point, and the train time for the 4-dimensional embedding was much shorter, just 6.643 s. In the case of the Wafer dataset, accuracy provided by a 4-dimensional embedding was equal to 97.36%. The accuracy difference between the 4-dimensional and the 8-dimensional model was 0.16 percent point, and the execution time of the 4-dimensional case was 10.981 s, which is 1.031 s shorter than for the 8-dimensional model.

5.4. Achieved results

In this section, we address conduced experiments with the proposed model. We performed a grid search concerning several parameter configurations to obtain the best model for each dataset. The key parameters that can affect the classifier's performance are the size of the alphabet a, the length of words w received via segmentation, and the learning rate η in the training procedure.

In Fig. 5, we show heatmaps for a few selected datasets where we may inspect the impact of the learning rate and epochs number on test set classification accuracy assuming a fixed size of the alphabet and a fixed word length. The plots allow us to see how the accuracy changes as we change the learning rate eta and the number of epochs. Fixed values of a and w were: Earthquakes [$a = 11$, $w = 6$], Coffee [$a = 15$, $w = 4$], Meat [$a = 17$, $w = 6$], OliveOil [$a = 15$, $w = 8$], ProximalPhalanxOutlineCorrect [$a = 17$, $w = 6$], WormsTwoClass [$a = 11$, $w = 6$], BeetleFly [$a = 7$, $w = 6$], MiddlePhalanxOutlineCorrect [$a = 15$, $w = 4$], BirdChicken [$a = 9$, $w = 6$], ShapeletSim [$a = 18$, $w = 4$], Plane [$a = 15$, $w = 4$], Strawberry [$a = 15$, $w = 8$], DistalPhalanxOutlineCorrect [$a = 13$, $w = 6$], DiatomSizeReduction [$a = 20$, $w = 8$], ECG5000 [$a = 13$, $w = 6$].

The same configuration does not work for every dataset, and each dataset has its own sweet spot for the model to start performing accurately. Notwithstanding, tests concerning not only the datasets presented in Fig. 5 have proven that optimal results can be obtained using many different parameter configurations. Some datasets, including OliveOil, Meat, Plane, DiatomSizeReduction, and others, show that classification accuracy is strongly correlated with learning rate. In these cases, the accuracy is very low for small learning rates. In contrast, some (but not that many) datasets, including BirdChicken and Coffee, are not as sensitive to changes of the learning rate. For these datasets, if the learning rate is greater than 0.0001, they achieve high classification accuracy. Increasing the learning rate further does not affect the result. Note that this behavior can be expected from very small datasets. For example, BirdChicken includes only 20 samples in both the training set, while Coffee contains 28 samples.

During training, validation is performed every five epochs. At the end of all the epochs, the weights from the epoch that gave the highest validation accuracy are restored as the final weights for testing. This step ensures that the model does not lose its best performance due to overfitting or unstable training.

Fig. 6 shows the effect of alphabet size a and word length w on the classification accuracy of the test set for the selected datasets. The results are for fixed learning rates for each dataset after 100 epochs. Here, the values of learning rate $\eta = 0.05$ is used for datasets: Meat, MiddlePhalanxOutlineCorrect, WormsTwoClass, OliveOil, MiddlePhalanxOutlineCorrect, Plane, Strawberry, and ECG5000. ProximalPhalanxOutlineCorrect, Earthquakes, ShapeletSim, DistalPhalanxOutlineCorrect, DiatomSizeReduction used learning rate $\eta = 0.1$. And for Coffee $\eta = 0.01$, BirdChicken $\eta = 0.001$ BeetleFly $\eta = 0.0001$. To get the best results, datasets like Meat, Strawberry,

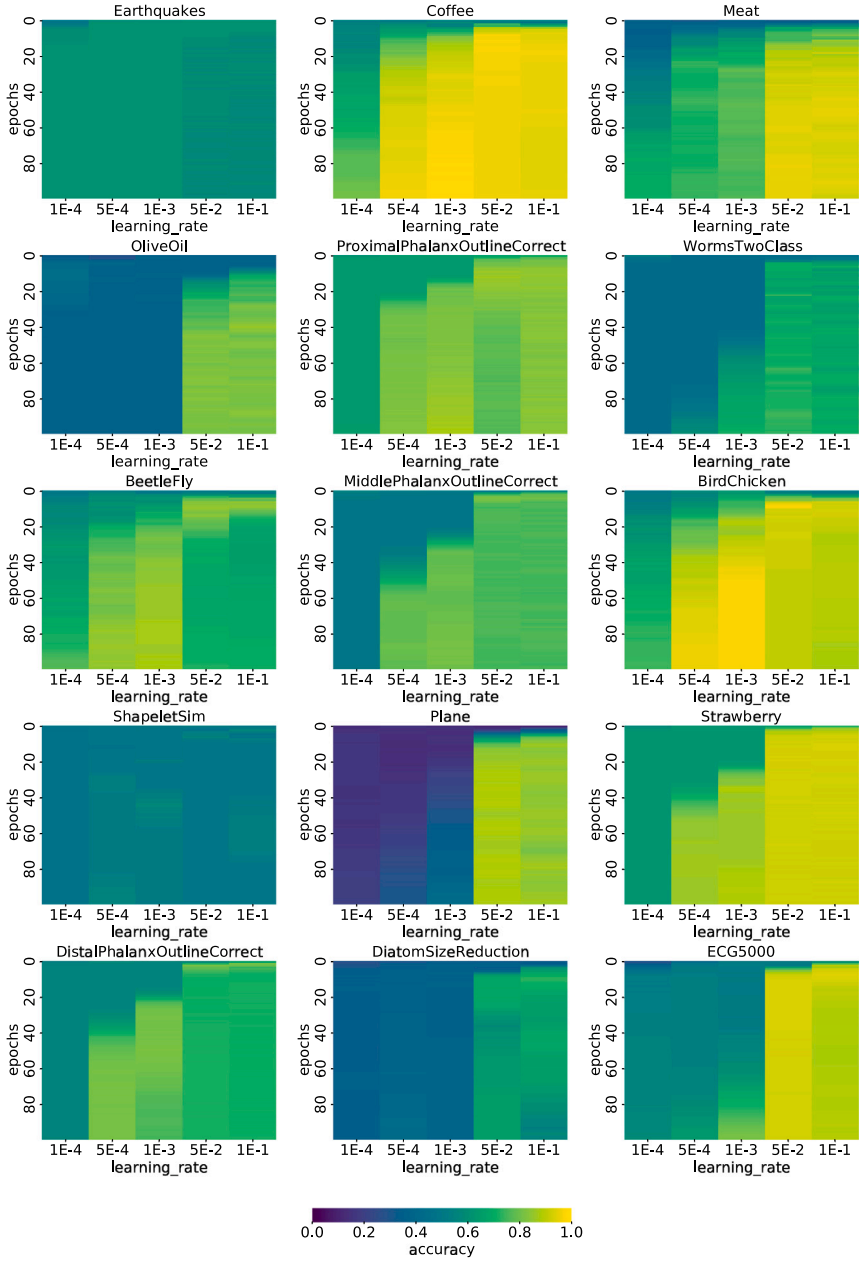


Fig. 5. Test accuracy fixed alphabet size a & word length w.

and ECG500 need a larger word length or a greater number of symbols in the alphabet. At the same time, BeetleFly, OliveOil, and WormsTwoClass prefer lower values of these parameters. Coffee, BirdChicken, DistalPhalanxOutlineCorrect, and Plane require average word length and alphabet size. On the other hand, it is tough to achieve the best accuracy utilizing different parameter combinations for Earthquakes and ShapeletSim. Those parameter values determine the size of the neural model created, as higher values will result in a more extensive vocabulary and the need for more parameters to learn text embeddings.

We calculated the model size and the time required to preprocess the data and train the model for datasets with different time series lengths and training observations. Table 4 contains three datasets that are representative of small, medium, and large dataset sizes. The number of parameters is directly proportional to the time series length because the text embedding and vectorization layers are created separately for each dataset. We see that the number of parameters for even the largest of the tested datasets is significantly low, to the point that the model can be trained and implemented even on a low-resource hardware configuration. The training time in Table 4 is the time required to train for a model for 100 epochs averaged over 10 runs for a given set of model parameters. In scenarios where parameter tuning is needed, we must consider that this needs to be repeated on several

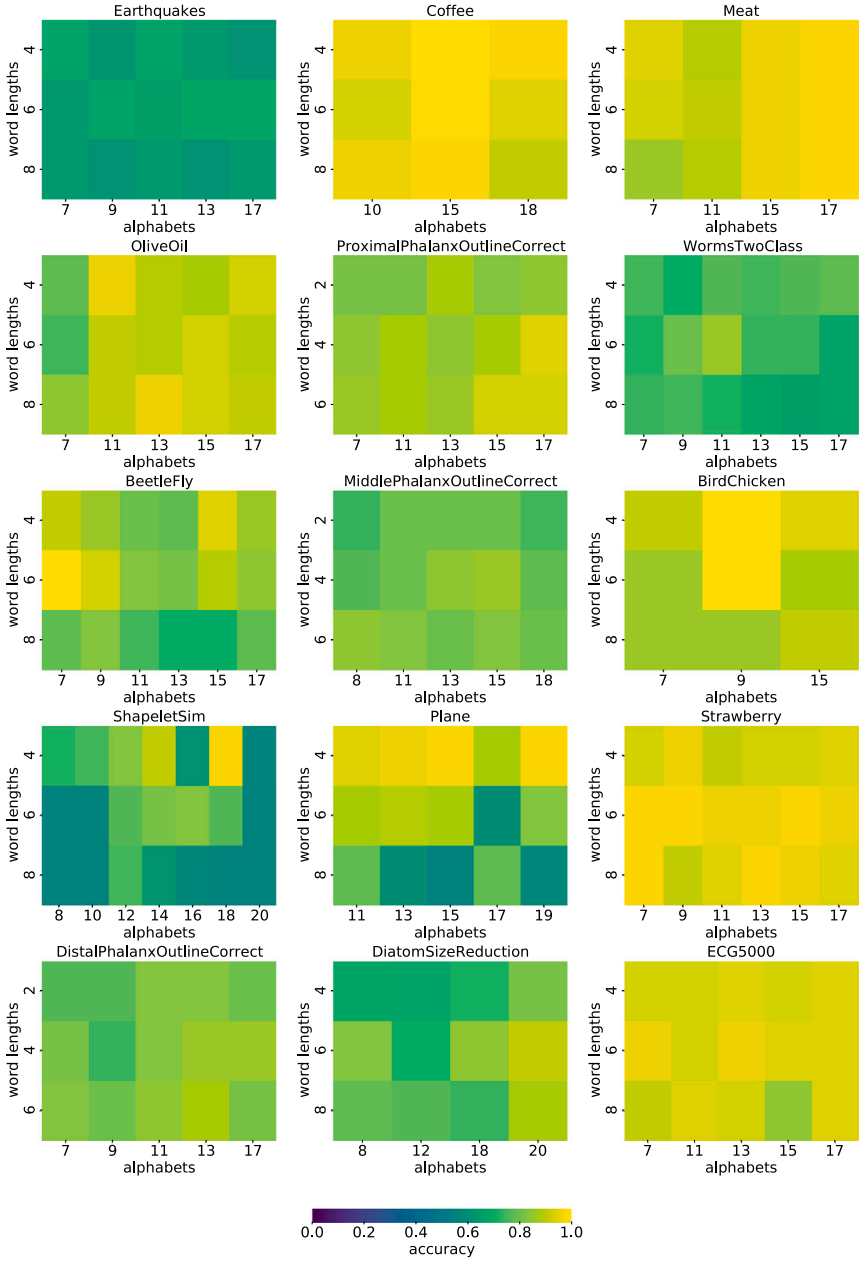


Fig. 6. Test accuracy for selected datasets and fixed values of the learning rate η.

configurations. Our tests ran training for 108 parameter combinations with an average time of 1 h 25 min per dataset. Fig. 7 demonstrates a line graph of train accuracy versus the number of train epochs for a few sample datasets. It can be observed from the graph that a satisfying accuracy rate is achieved early on in the training procedure and that the models are relatively stable.

5.5. Comparison with the state-of-the-art

Table 5 compares the best accuracy obtained using SAFE classifier, i.e., our dictionary-based method, with the results obtained by other existing dictionary-based methods, non-dictionary methods, and the baseline algorithm. The leading method name for both the dictionary-based and non-dictionary-based methods is also given. For the comparison, we considered:

- 1-Nearest Neighbor classifier based on Euclidean distance playing the role of our baseline classifier (denoted in the following text as Euclidean_1NN),

Table 4
Time consumption for training and model parameter size. Average of 10 repetitions, each with 100 epochs.

	Dataset name		
	SonyAIBORobotSurface1	Earthquakes	Worms
Time series length	70	512	900
Training samples	20	139	181
Avg. Train time (s)	64.0	1684.5	1370.5
Number of parameters	946	10 934	22 073
Accuracy (%)	86.19	78.47	75.32
MCC	0.72	0.33	0.65

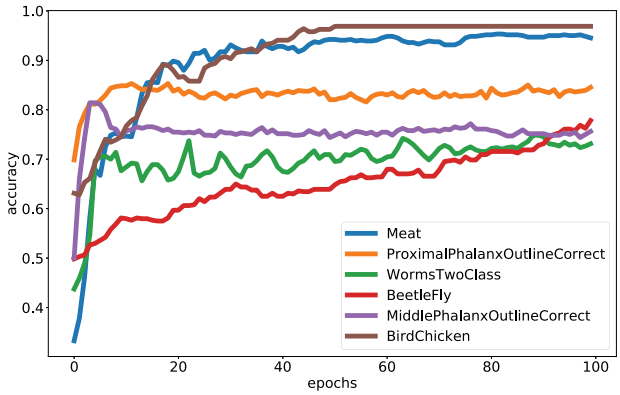


Fig. 7. Train accuracy against the number of training epochs for selected datasets.

as well as seven dictionary-based classifiers:

- Bag of Symbolic Fourier Approximation Symbols, (BOSS) (Schäfer, 2015),
- Contract Bag of SFA Symbols (cBOSS, also called RBOSS) (Middlehurst et al., 2019),
- Spatial Bag of SFA Symbol (sBOSS) (Large et al., 2019),
- Word ExtrAction for time SEries cLassification (WEASEL) (Schäfer & Leser, 2017),
- Symbolic Aggregate approXimation and Vector Space Model (SAX-VSM) (Senin & Malinchik, 2013),
- Dynamic Time Warping Features (DTW_F, also called FBDTW) (Franses & Wiemann, 2020),
- Bag of Patterns (BoP) (Hatami et al., 2019),

and ten highly competitive non-dictionary-based classifiers:

- Time Series Forest (TSF) (Tan, Bergmeir, Petitjean, & Webb, 2021),
- Random Interval Spectral Ensemble (RISE) (Flynn, Large, & Bagnall, 2019),
- Shapelet Transform Classifier (STC) (Shu, Yao, Lyu, Li, & Chen, 2021),
- ProximityForest (Lucas et al., 2019),
- ResNet (Wang, Yan, & Oates, 2017),
- InceptionTime (Ismail Fawaz et al., 2020),
- CAnonical Time-series CHaracteristics (Catch22) (Lubba et al., 2019),
- Time Series Combination of Heterogeneous and Integrated Embeddings Forest (TS_CHIEF) (Shifaz et al., 2020),
- Hierarchical Vote Collective of Transformation-based Ensembles (HIVE COTE) (Lines et al., 2016),
- RandOm Convolutional KErnel Transform (ROCKET) (Dempster et al., 2020).

Let us recall that SAFE (the method introduced in this paper) belongs to the family of dictionary-based classifiers. Hence, we pay special attention to the comparison with this group.

In general, the proposed classifier is quite well-performing in comparison to the other classifiers. This is partly because it is a parameterized model. Thus, we can test it with different combinations of parameters before deciding on the best model for a given dataset. In terms of accuracy, it outperforms several dictionary-based methods and is as good or even better than many non-dictionary-based methods. The best classification accuracy obtained for each dataset is shown in bold in Table 5. From this comparison, we determined our toughest state-of-the-art competitor, which appears several times in Table 5. For ten of the 30 datasets, our SAFE obtained the best results. In the category of non-dictionary-based methods, ROCKET obtained the best results

Table 5
Comparison of time series classification accuracy with state-of-the-arts, including non-dictionary, dictionary-based, and baseline (Euclidean distance with 1NN) classifiers, and the proposed method (SAFE). Results concern test sets. ACC — accuracy (in %).

Dataset id	Euclidean	SOTA(non-dictionary)		SOTA(dictionary)		SAFE
	ACC	ACC	Algorithm	ACC	Algorithm	ACC
BeetleFly	75.00	96.33	HIVE_COTE	97.50	RBOSS	**100**
BirdChicken	55.00	96.33	TS_CHIEF	**100**	SAX-VSM	**100**
Coffee	**100**	**100**	ROCKET	99.05	RBOSS	**100**
Computers	57.60	**86.56**	InceptionTime	82.00	sBOSS	76.00
DiatomSizeReduction	93.46	**95.79**	ROCKET	94.54	sBOSS	84.64
Dist.Phal.Out.AgeGr.	71.73	**82.81**	TS_CHIEF	82.134	sBOSS	77.69
Dist.Phal.Out.Corr.	62.59	82.73	STC	**84.17**	SAX-VSM	79.35
Dist.Phal.TW	63.31	**70.12**	ROCKET	69.78	BoP	69.78
Earthquakes	71.22	74.96	ProximityForest	74.82	SAX-VSM FBDTW	**78.42**
ECG5000	92.49	**94.85**	TS_CHIEF	94.59	WEASEL	90.02
GunPoint	91.33	**100**	TS_CHIEF	99.98	RBOSS	88.00
Herring	51.56	63.28	STC	62.50	SAX-VSM	**70.31**
Meat	93.33	**99.39**	ResNet	98.39	sBOSS	98.33
Mid.Phal.Out.AgeGr.	76.63	71.08	ROCKET	**78.69**	FBDTW	64.93
Mid.Phal.Out.Corr.	51.95	**83.45**	ROCKET	82.83	WEASEL	76.98
Mid.Phal.TW	51.29	58.98	ROCKET	56.73	RBOSS	**63.64**
OliveOil	86.67	91.67	TS_CHIEF	91.33	WEASEL	**93.33**
Plane	96.19	**100**	ROCKET and more	**100**	RBOSS	97.14
Prox.Phal.Out.AgeGr.	80.76	85.84	Catch22	84.98	RBOSS	**88.29**
Prox.Phal.Out.Corr.	78.54	**90.63**	InceptionTime	87.63	WEASEL	85.22
Prox.Phal.TW	70.73	81.61	HIVE_COTE	81.46	FBDTW	**84.39**
ShapeletSim	53.89	**100**	HIVE_COTE TS_CHIEF	**100**	sBOSS BOSS	93.89
SonyAIBORob.Sur.1	69.55	**96.04**	ResNet	90.93	WEASEL	86.19
Strawberry	94.59	**97.87**	ROCKET	97.86	WEASEL	94.05
ToeSegmentation2	80.77	**96.82**	HIVE_COTE	96.49	RBOSS	88.46
Trace	76.00	**100**	ROCKET and more	**100**	BOSS	97.00
Wafer	99.55	**99.99**	STC	99.99	WEASEL	97.36
Wine	61.11	91.42	ROCKET	**96.29**	SAX-VSM	90.74
Worms	45.46	**77.96**	InceptionTime	77.83	WEASEL	74.03
WormTwoClass	61.04	80.35	InceptionTime	80.78	BOSS	**85.71**

for ten datasets, TS-CHIEF obtained the best results for eight datasets, and HIVE-COTE obtained the best results for six datasets. In the group of dictionary-based approaches, the best algorithm is WEASEL. It obtained nine best results for 30 datasets. The other competing algorithms are RBOSS, sBOSS, and SAX-VSM, which acquired the best results for seven, five, and five datasets, respectively.

Table 6 compares other time series classifiers with our proposed SAFE approach, based on the MCC score. We collected available MCC results for different algorithms from http://www.timeseriesclassification.com/results.php. Among the 30 datasets, our technique obtained the highest MCC score for 11. WEASEL and ROCKET won for 13 and 11 datasets, respectively. RBOSS and sBOSS both achieved best results for 7 datasets. We observed that the average MCC score value for SAFE classifier is higher than 0.73 which makes this method better than most of the SOTA classifiers while giving the lowest standard deviation compared to the rest. In Fig. 8, we can see a considerable difference in the MCC standard deviation between our method and the next best, which strengthens our belief that the results we achieve with this method are more stable across datasets.

We tuned the parameters to obtain good models that give the best results for each dataset. The details of the winning model, such as alphabet size a, word length w, learning rate η, and the number of epochs, are listed in Table 7. The learning rate and the number of epochs for the best model vary significantly across the experimental set. It is also worth noting that the best results for different datasets are more often achieved with alphabet sizes from 10 to 17. The only exceptions are BeetleFly and BirdChicken datasets, for which alphabet sizes 7 and 9 were the best choice. Furthermore, for most of the datasets, the best model was based on words of length 6.

Table 8 shows the average time taken to construct a model for different datasets using SAFE that includes transformation to symbols, words, and then neural network training. We prepared the table based on 10 repetitions of the procedure. It also covers a comparison with analogous times for the best-performing state-of-the-art algorithms. The data provided in Table 2 (datasets description) and Table 8 clearly show that our proposed method is much faster than the state-of-the-art dictionary and non-dictionary-based methods in training by a factor of up to 3–4.5x in some cases, such as Earthquakes, Worms, and others. Other methods may struggle with an increase in the dataset sample size. Due to its linear time complexity in the preprocessing step and the simple neural network architecture, SAFE is expected to perform fast and scale well with the increase in dataset size.

Table 6
Comparison of time series classification MCC with state-of-the-arts, including non-dictionary, dictionary-based classifiers, and the proposed method (SAFE). Results concern test sets. MCC — Matthews correlation coefficient.

Dataset id	SOTA(non-dictionary)		SOTA(dictionary)		SAFE
	MCC	Algorithm	MCC	Algorithm	MCC
BeetleFly	0.931	HIVE_COTE	0.953	RBOSS	**1**
BirdChicken	0.932	TS_CHIEF	0.969	BOSS	**1**
Coffee	**1**	Rocket	0.981	RBOSS	**1**
Computers	**0.733**	InceptionTime	0.644	sBOSS	0.521
DiatomSizeReduction	**0.944**	Rocket	0.927	sBOSS	0.789
Dist.Phal.Out.AgeGr.	**0.708**	TS_CHIEF	0.697	sBOSS	0.637
Dist.Phal.Out.Corr.	**0.644**	STC	0.626	WEASEL	0.581
Dist.Phal.TW	**0.623**	Rocket	0.592	WEASEL	**0.629**
Earthquakes	0.090	ResNet	−0.005	sBOSS WEASEL	**0.329**
ECG5000	**0.902**	TS_CHIEF	0.897	WEASEL	0.806
GunPoint	**1**	TS_CHIEF	0.999	RBOSS	0.774
Herring	0.202	Rocket	0.158	sBOSS	**0.398**
Meat	**0.991**	ResNet	0.976	sBOSS	0.975
Mid.Phal.Out.AgeGr.	**0.483**	Rocket	0.435	RBOSS	0.357
Mid.Phal.Out.Corr.	**0.666**	Rocket	0.651	WEASEL	0.526
Mid.Phal.TW	0.469	Rocket	0.439	RBOSS	**0.533**
OliveOil	0.888	TS_CHIEF	0.882	WEASEL	**0.909**
Plane	**1**	Rocket and more	**1**	RBOSS	0.967
Prox.Phal.Out.AgeGr.	0.755	RISE	0.739	RBOSS	**0.795**
Prox.Phal.Out.Corr.	**0.780**	InceptionTime	0.708	WEASEL	0.654
Prox.Phal.TW	0.743	HIVE_COTE	0.722	WEASEL	**0.785**
ShapeletSim	**1**	HIVE_COTE TS_CHIEF	**1**	sBOSS BOSS	0.880
SonyAIBORob.Sur.1	**0.923**	ResNet	0.834	WEASEL	0.725
Strawberry	**0.954**	Rocket	**0.954**	WEASEL	0.869
ToeSegmentation2	**0.905**	HIVE_COTE	0.889	RBOSS	0.579
Trace	**1**	Rocket and more	**1**	BOSS	0.947
Wafer	**0.999**	STC	**0.999**	WEASEL	0.879
Wine	0.834	Rocket	**0.866**	WEASEL	0.290
Worms	**0.701**	InceptionTime	0.695	WEASEL	0.659
WormTwoClass	0.698	InceptionTime	0.611	BOSS	**0.714**

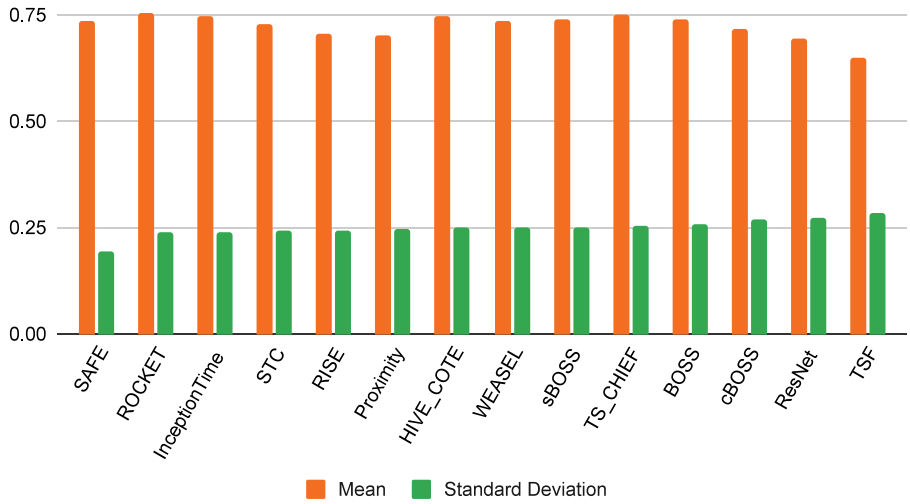


Fig. 8. MCC — means and standard deviations obtained using SAFE compared against the state-of-the-art methods for 30 datasets (computed on test sets). The values displayed from left–right are sorted in the increasing order of their standard deviation.

6. Implications

We believe that the described experiments demonstrated that the proposed classifier performs admirably across all data types. We selected seven types of datasets: device-related, ECG, image-related, motion-related, sensor-based, simulated, and spectrum-analysis-related. Our classifier achieves very high accuracy for both image-related datasets (BeetleFly, BirdChicken, Herring, etc.)

Table 7
Accuracy (in %) obtained by SAFE and parameters associated with the winning model. a denotes the size of the alphabet, w stands for the word's length, η is the learning rate. Boldface indicates cases when SAFE was achieving the highest score.

Dataset id	Accuracy	Associated parameters			
		a	w	η	Epochs
BeetleFly	**100**	7	6	0.0001	90
BirdChicken	**100**	9	6	0.001	20
Coffee	**100**	15	4	0.001	30
Computers	76.00	16	6	0.0005	20
DiatomSizeReduction	84.64	20	8	0.1	40
Dist.Phal.OutlineAgeGr.	77.69	13	2	0.05	40
Dist.Phal.OutlineCorr.	79.35	13	6	0.1	5
Dist.Phal.TW	69.78	11	2	0.1	40
Earthquakes	**78.42**	11	6	0.1	60
ECG5000	90.02	13	6	0.05	30
GunPoint	88.67	17	4	0.05	40
Herring	**70.31**	18	8	0.0005	5
Meat	98.33	17	6	0.05	50
Mid.Phal.OutlineAgeGr.	64.93	13	2	0.1	20
Mid.Phal.OutlineCorr.	76.98	15	4	0.05	70
Mid.Phal.TW	**63.64**	13	6	0.05	10
OliveOil	**93.33**	15	8	0.05	50
Plane	97.14	15	4	0.05	30
Prox.Phal.xOutlineAgeG.	**88.29**	15	6	0.05	10
Prox.Phal.OutlineCorr.	85.22	17	6	0.1	40
Prox.Phal.TW	**84.39**	11	6	0.1	10
ShapeletSim	93.89	18	4	0.1	5
SonyAIBORobotSurface1	86.19	7	4	0.1	20
Strawberry	94.05	15	8	0.05	10
ToeSegmentation2	88.46	15	4	0.1	5
Trace	97.00	12	4	0.1	70
Wafer	97.36	17	6	0.05	80
Wine	90.74	7	6	0.1	80
Worms	74.03	11	8	0.1	90
WormTwoClass	**85.71**	11	6	0.05	30

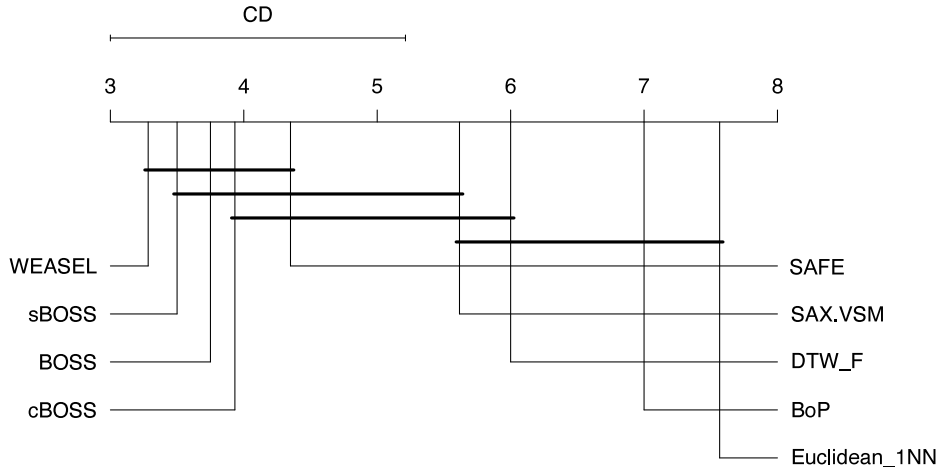


Fig. 9. Critical difference diagram for the proposed approach (called SAFE) and state-of-the-art dictionary-based time series classifiers and the baseline approach (Euclidean distance with 1-Nearest Neighbor). The experiment assumed the significance level $\alpha = 0.05$.

and spectrum-analysis-related (Coffee, OliveOil). SAFE also achieves higher accuracy for sensor-based datasets (Earthquakes) and motion-related datasets (WormsTwoClass). This result allows us to be optimistic that the method is versatile, its applicability is not narrowed to one particular data domain.

Nonetheless, we would like to emphasize that the state-of-the-art classifiers were challenging to beat. Because we used several algorithms, we performed non-parametric tests to compare the classifiers following the methodology proposed by Demsar statistically (Wainer & Cawley, 2017). We conducted the Nemenyi test, which compares multiple classifiers over multiple datasets. This test is based on the absolute difference of the mean rankings of the classifiers. Thus, the results always depend on the choice of the datasets. We assumed a significance level of $\alpha = 0.05$. SAFE shows some similarity to the results achieved by BOSS and WEASEL.

Table 8
Comparison of training time in seconds of state-of-the-arts methods, including non-dictionary and dictionary-based classifiers, against the proposed method (SAFE).

Dataset set	Best SOTA time	SOTA algorithm	SAFE time
BeetleFly	264.5	ROCKET	**237.0**
BirdChicken	266.0	ROCKET	**254.2**
Coffee	**197.0**	ROCKET	215.2
Computers	1828.5	ResNet	**330.5**
DiatomSizeReduction	**106.0**	ROCKET	162.7
Dist.Phal.OutlineAgeGr.	737.5	ROCKET	**638.6**
Dist.Phal.OutlineCorr.	**1146.0**	ROCKET	1154.6
Dist.Phal.TW	733.0	ROCKET	**600.4**
Earthquakes	1684.5	ResNet	**527.4**
ECG5000	1362.0	ROCKET	**598.9**
GunPoint	**162.5**	ROCKET	193.6
Herring	319.5	ResNet	**151.8**
Meat	287.0	ResNet	**186.8**
Mid.Phal.OutlineAgeGr.	**591.0**	ResNet	596.8
Mid.Phal.OutlineCorr.	**906.0**	ResNet	972.6
Mid.Phal.TW	**593.0**	ResNet	659.9
OliveOil	278.0	ResNet	**224.6**
Plane	298.5	ROCKET	**259.0**
Prox.Phal.xOutlineAgeG.	745.0	ROCKET	**568.8**
Prox.Phal.OutlineCorr.	1154.5	ROCKET	**1145.4**
Prox.Phal.TW	742.0	ROCKET	**635.9**
ShapeletSim	263.0	ROCKET	**205.0**
SonyAIBORobotSurface1	**64.0**	ROCKET	190.0
Strawberry	1059.0	ResNet	**802.8**
ToeSegmentation2	267.0	ROCKET	**196.9**
Trace	496.5	ROCKET	**286.8**
Wafer	3046.5	Catch22	**1073.8**
Wine	242.5	ROCKET	**177.8**
Worms	1370.5	ResNet	**307.9**
WormsTwoClass	1368.5	ResNet	**302.6**

We present the critical difference diagram in Fig. 9 to take a closer look at the problematic cases. The test concerned dictionary-based methods to which SAFE belongs. In the critical difference diagram convention, the bars connect groups of algorithms that are not significantly different. The observed similarity in SAFE and BOSS and WEASEL behavior is due to the similarity of these approaches. They all transform the time series to a symbolic representation and then work on words.

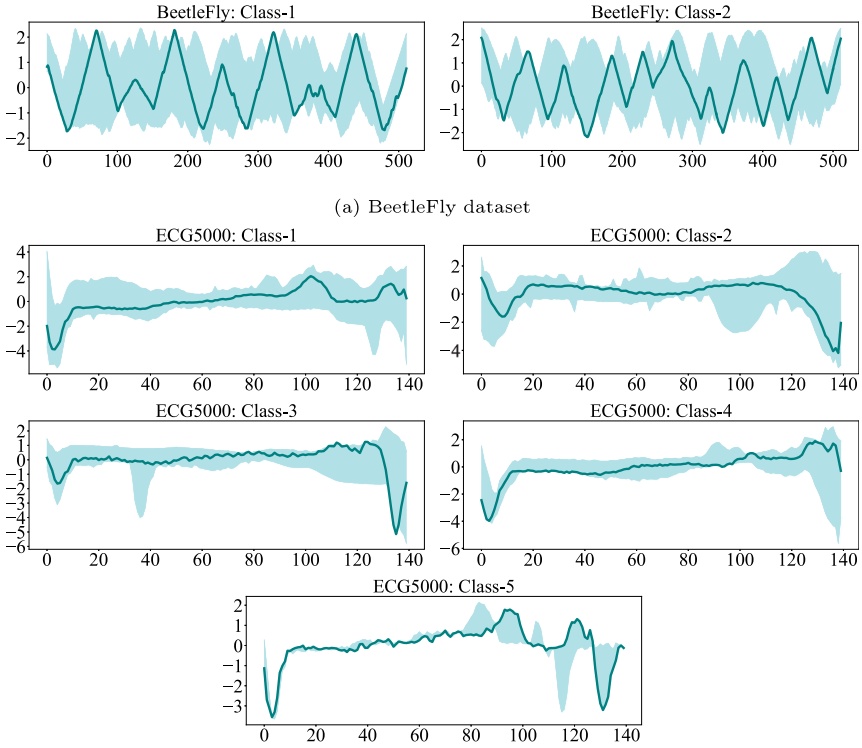
7. Conclusion

The SAFE method described in this paper for time series classification shows that a simple step of preprocessing numeric time series into text data based on the SAX algorithm and an NLP-based neural model for text classification can achieve reasonably good results for a range of various datasets. The comparisons performed with state-of-the-art dictionary-based and neural classifiers show the increase in accuracy achieved when applying the proposed procedure. In the group of dictionary-based classifiers, our SAFE is preceded by WEASEL and BOSSes, but it outperforms all other methods in this family.

The essential advantages of SAFE are its low computational complexity and a small size of the model. These two properties make it a suitable choice for deployment in mobile and low-resource environments that perform real-time classification.

The main limitation of SAFE is the need to tune the hyperparameters to obtain the best results. Hyperparameters such as the number of letters in the alphabet and the length of words considerably impact accuracy. They need to be adjusted for each dataset individually.

A tempting area for future investigations on dictionary-based approaches to time series analysis would be to take advantage of models pre-trained on large corpora, as it is done with massive success in the NLP field (Briskilal & Subalalitha, 2022; Lin, Kung, & Leu, 2022). We were hesitant to work with such a processing scheme because the data we process does not consist of words in natural language. In our case, a letter corresponds to a value that falls into a specific interval. Each interval is assigned a unique letter. However, the discretization is not done via a simple split of the time series values range into equal-sized intervals. Instead, we apply SAX, which makes sure that produced symbols correspond to time series features with equal probability. In other words, if there will be a large number of time series data points, say, in the range $[0, 0.5]$, there will be more symbols assigned to represent intervals in this range than to represent the intervals in the range $(0, 5, 1.0]$. This entails that a particular symbol will mean something different for datasets of different characteristics. Furthermore, each word has the same fixed length, and the length is a tunable hyperparameter of the procedure. A consequence of this fact is that the “meaning” of a sequence of symbols will be dataset-dependent, and we want to take advantage of this fact.



(a) BeetleFly dataset

(b) ECG5000 dataset

Fig. A.10. Ribbon plots for selected samples from different classes in BeetleFly and ECG5000 datasets.

In closer future, we propose to explore methods to automate and speed up the computationally intensive tuning step according to the characteristics of the time series curve. The algorithm works very well for short and average-length datasets but not for long datasets. We plan to improve this in the future and work out a better training procedure to enhance the results for datasets with many training and testing samples.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported by the National Science Centre, grant No. 2019/35/D/HS4/01594, decision no. DEC-2019/35/D/HS4/01594.

Appendix. Plots displaying the properties of processed datasets

In Figs. A.10, A.11, and A.12, we present ribbon plots of selected time series datasets. For each dataset, we prepared one plot for each class. Ribbons inform about the span of values for all instances in the training set. Darker line plots show one randomly selected time series from a given class. The plots allow inspecting the variability of instances between the present classes. For example, we see that both classes in the Strawberry dataset have a similar shape and variability (ribbons are thin). In the case of the Plane dataset, we have classes in which time series instances have low variance (classes 3, 4, 6, and 7), but some have high (classes 1, 2, and 5). High variability of instances can be the reason for an increased difficulty of class recognition. We present these plots to demonstrate that we considered datasets of various properties for the empirical study presented in the paper.

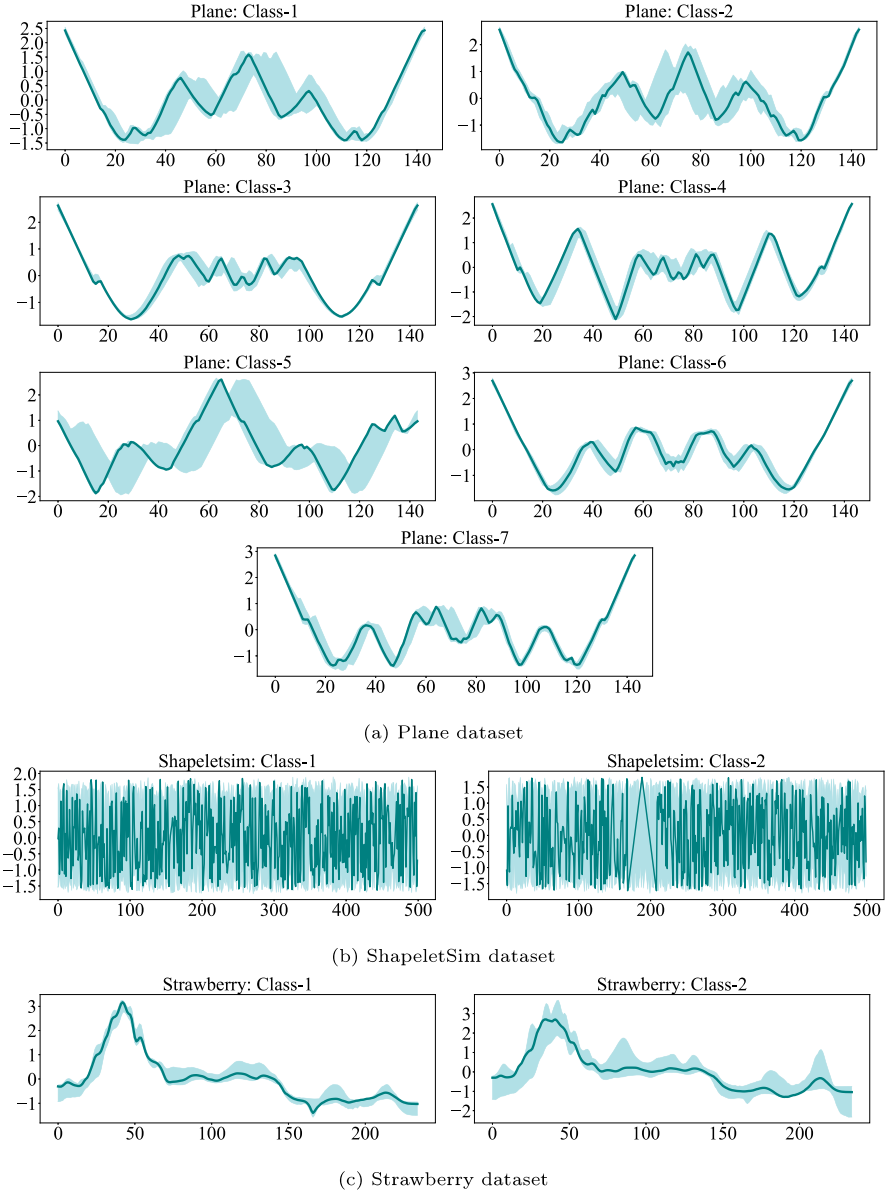


Fig. A.11. Ribbon plots for selected samples from different classes in Plane, ShapeletSim, and Strawberry datasets.

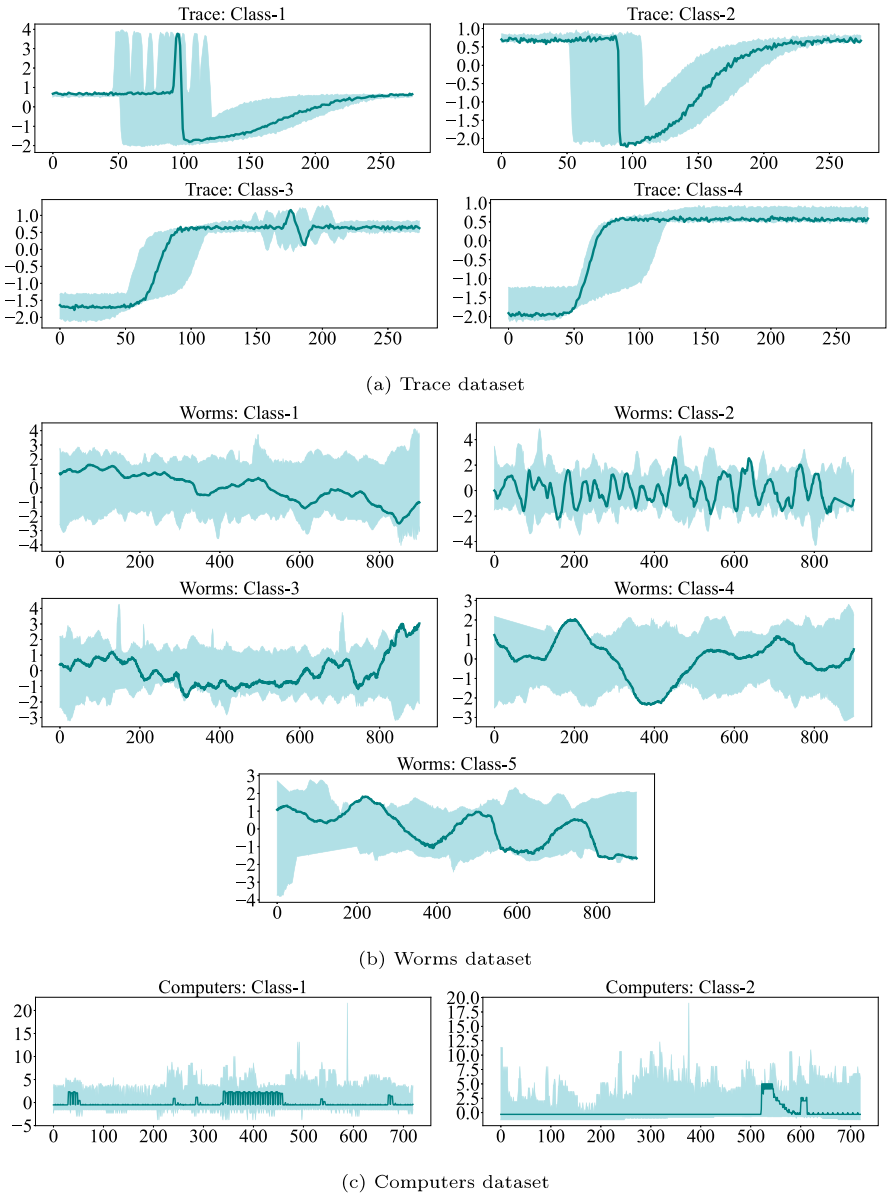


Fig. A.12. Ribbon plots for selected samples from different classes in Trace, Worms, and Computers datasets.

References

Arora, S., May, A., Zhang, J., & Ré, C. (2020). Contextual embeddings: When are they worth it? http://dx.doi.org/10.48550/ARXIV.2005.09117, arXiv.

Asif, M., Martiniano, H. F. M. C. M., Vicente, A. M., & Couto, F. M. (2018). Identifying disease genes using machine learning and gene functional similarities, assessed through gene ontology. *PLoS One*, *13*(12), Article e0208626. http://dx.doi.org/10.1371/journal.pone.0208626.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *31*(3), 606–660. http://dx.doi.org/10.1007/s10618-016-0483-9.

Bagnall, A., Lines, J., Hills, J., & Bostrom, A. (2015). Time-series classification with COTE: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, *27*(9), 2522–2535. http://dx.doi.org/10.1109/TKDE.2015.2416723.

Bai, B., Li, G., Wang, S., Wu, Z., & Yan, W. (2021). Time series classification based on multi-feature dictionary representation and ensemble learning. *Expert Systems with Applications*, *169*, Article 114162. http://dx.doi.org/10.1016/j.eswa.2020.114162.

Behera, R. K., Jena, M., Rath, S. K., & Misra, S. (2021). Co-LSTM: Convolutional LSTM model for sentiment analysis in social big data. *Information Processing & Management*, *58*(1), Article 102435. http://dx.doi.org/10.1016/j.ipm.2020.102435.

Briskilal, J., & Subalalitha, C. N. (2022). An ensemble model for classifying idioms and literal texts using BERT and RoBERTa. *Information Processing & Management*, *59*(1), Article 102756. http://dx.doi.org/10.1016/j.ipm.2021.102756.

Bruyn, M. D. (2018). From word to financial time series embedding. http://dx.doi.org/10.2139/ssrn.3184513, SSRN.

Cheng, J., Tian, S., Yu, L., & You, H. (2020). Multi-attention mechanism medical image segmentation combined with word embedding technology. *Automatic Control and Computer Sciences*, *54*, 560–571. http://dx.doi.org/10.3103/S0146411620060024.

Dempster, A., Petitjean, F., & Webb, G. I. (2020). ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, *34*(5), 1454–1495. http://dx.doi.org/10.1007/s10618-020-00701-z.

Diallo, B., Hu, J., Li, T., Khan, G. A., Liang, X., & Zhao, Y. (2021). Deep embedding clustering based on contractive autoencoder. *Neurocomputing*, *433*, 96–107. http://dx.doi.org/10.1016/j.neucom.2020.12.094.

Flynn, M., Large, J., & Bagnall, T. (2019). The contract random interval spectral ensemble (c-RISE): The effect of contracting a classifier on accuracy. In H. Pérez García, L. Sánchez González, M. Castejón Limas, H. Quintián Pardo, & E. Corchado Rodríguez (Eds.), *Hybrid artificial intelligent systems* (pp. 381–392). Cham: Springer International Publishing.

Franses, P. H., & Wiemann, T. (2020). Intertemporal similarity of economic time series: An application of dynamic time warping. *Computational Economics*, *56*, 59–75. http://dx.doi.org/10.1007/s10614-020-09986-0.

Geler, Z., Kurbalija, V., Ivanović, M., & Radovanović, M. (2020). Time-series classification with constrained DTW distance and inverse-square weighted k-NN. In *2020 international conference on innovations in intelligent systems and applications (INISTA)* (pp. 1–7). http://dx.doi.org/10.1109/INISTA49547.2020.9194639.

Goldani, M. H., Safabakhsh, R., & Momtazi, S. (2021). Convolutional neural network with margin loss for fake news detection. *Information Processing & Management*, *58*(1), Article 102418. http://dx.doi.org/10.1016/j.ipm.2020.102418.

Hatami, N., Gavet, Y., & Debayle, J. (2019). Bag of recurrence patterns representation for time-series classification. *Pattern Analysis and Applications*, *22*, 877–887. http://dx.doi.org/10.1007/s10044-018-0703-6.

Homenda, W., & Jastrzebska, A. (2020). Time-series classification using fuzzy cognitive maps. *IEEE Transactions on Fuzzy Systems*, *28*(7), 1383–1394. http://dx.doi.org/10.1109/TFUZZ.2019.2917126.

Huang, T., Deng, Z.-H., Shen, G., & Chen, X. (2020). A window-based self-attention approach for sentence encoding. *Neurocomputing*, *375*, 25–31. http://dx.doi.org/10.1016/j.neucom.2019.09.024.

Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., et al. (2020). InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, *34*(6), 1936–1962. http://dx.doi.org/10.1007/s10618-020-00710-y.

Iwana, B. K., Frinken, V., & Uchida, S. (2020). DTW-NN: A novel neural network for time series recognition using dynamic alignment between inputs and weights. *Knowledge-Based Systems*, *188*, Article 104971. http://dx.doi.org/10.1016/j.knosys.2019.104971.

Juez-Gil, M., Arnaiz-González, A., Rodríguez, J. J., López-Nozal, C., & García-Osorio, C. (2021). Rotation forest for big data. *Information Fusion*, *74*, 39–49. http://dx.doi.org/10.1016/j.inffus.2021.03.007.

Large, J., Bagnall, A., Malinowski, S., & Tavenard, R. (2019). On time series classification with dictionary-based classifiers. *Intelligent Data Analysis*, *23*, 1073–1089. http://dx.doi.org/10.3233/IDA-184333.

Lee, C., Kim, S.-H., & Youn, C.-H. (2021). Pattern-wise embedding system for scalable time-series database. In *2021 IEEE international conference on big data and smart computing (BigComp)* (pp. 358–361). http://dx.doi.org/10.1109/BigComp51126.2021.00078.

Lin, G., Fan, C., Chen, W., Chen, Y., & Zhao, F. (2021). Class label autoencoder with structure refinement for zero-shot learning. *Neurocomputing*, *428*, 54–64. http://dx.doi.org/10.1016/j.neucom.2020.11.061.

Lin, S.-Y., Kung, Y.-C., & Leu, F.-Y. (2022). Predictive intelligence in harmful news identification by BERT-based ensemble learning model with text sentiment analysis. *Information Processing & Management*, *59*(2), Article 102872. http://dx.doi.org/10.1016/j.ipm.2022.102872.

Lines, J., Taylor, S., & Bagnall, A. (2016). HIVE-COTE: The hierarchical vote collective of transformation-based ensembles for time series classification. In *2016 IEEE 16th international conference on data mining (ICDM)* (pp. 1041–1046). http://dx.doi.org/10.1109/ICDM.2016.0133.

Lopez-Otero, P., Parapar, J., & Barreiro, A. (2019). Efficient query-by-example spoken document retrieval combining phone multigram representation and dynamic time warping. *Information Processing & Management*, *56*(1), 43–60. http://dx.doi.org/10.1016/j.ipm.2018.09.002.

Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., & Jones, N. S. (2019). catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery*, *33*(6), 1821–1852. http://dx.doi.org/10.1007/s10618-019-00647-x.

Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., et al. (2019). Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, *33*, http://dx.doi.org/10.1007/s10618-019-00617-3.

Luo, Y., Yao, C., Mo, Y., Xie, B., Yang, G., & Gui, H. (2021). A creative approach to understanding the hidden information within the business data using deep learning. *Information Processing & Management*, *58*(5), Article 102615, URL https://www.sciencedirect.com/science/article/pii/S0306457321001114.

Middlehurst, M., Large, J., Flynn, M., Bostrom, A., & Bagnall, A. (2021). HIVE-COTE 2.0: a new meta ensemble for time series classification. *Machine Learning*, *110*, 3211–3243. http://dx.doi.org/10.1007/s10994-021-06057-9.

Middlehurst, M., Vickers, W., & Bagnall, A. (2019). Scalable dictionary classifiers for time series classification. In H. Yin, D. Camacho, P. Tino, A. J. Tallón-Ballesteros, R. Menezes, & R. Allmendinger (Eds.), *Intelligent data engineering and automated learning – IDEAL 2019* (pp. 11–19). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-33607-3_2.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781. URL http://arxiv.org/abs/1301.3781.

Mohan, A., & Pramod, K. V. (2021). Link prediction in dynamic networks using time-aware network embedding and time series forecasting. *Journal of Ambient Intelligence and Humanized Computing*, *12*, 1981–1993. http://dx.doi.org/10.1007/s12652-020-02289-0.

Mohasseb, A., Bader-El-Den, M., & Cocea, M. (2018). Question categorization and classification using grammar based approach. *Information Processing & Management*, *54*(6), 1228–1243. http://dx.doi.org/10.1016/j.ipm.2018.05.001.

Nalmpantis, C., & Vrakas, D. (2019). Signal2Vec: Time series embedding representation. In J. Macintyre, L. Iliadis, I. Maglogiannis, & C. Jayne (Eds.), *Engineering applications of neural networks* (pp. 80–90). Cham: Springer International Publishing.

Nozza, D., Manchanda, P., Fersini, E., Palmonari, M., & Messina, E. (2021). LearningToAdapt with word embeddings: Domain adaptation of named entity recognition systems. *Information Processing & Management*, *58*(3), Article 102537. http://dx.doi.org/10.1016/j.ipm.2021.102537.

Pan, W., Pan, L., Su, T., & Chen, Z. (2018). Time series classification based on dictionary learning and sparse representation. In *2018 eighth international conference on instrumentation measurement, computer, communication and control (IMCCC)* (pp. 1139–1144). http://dx.doi.org/10.1109/IMCCC.2018.00237.

Rebane, J., Karlsson, I., Bornemann, L., & Papapetrou, P. (2021). SMILE: a feature-based temporal abstraction framework for event-interval sequence classification. *Data Mining and Knowledge Discovery*, *35*, 372–399. http://dx.doi.org/10.1007/s10618-020-00719-3.

Rezvani, R., Barnaghi, P., & Enshaeifar, S. (2021). A new pattern representation method for time-series data. *IEEE Transactions on Knowledge and Data Engineering*, *33*(7), 2818–2832. http://dx.doi.org/10.1109/TKDE.2019.2961097.

Roostaee, M., Sadreddini, M. H., & Fakhrahmad, S. M. (2020). An effective approach to candidate retrieval for cross-language plagiarism detection: A fusion of conceptual and keyword-based schemes. *Information Processing & Management*, *57*(2), Article 102150. http://dx.doi.org/10.1016/j.ipm.2019.102150.

Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, *29*(6), 1505–1530. http://dx.doi.org/10.1007/s10618-014-0377-7.

Schäfer, P., & Leser, U. (2017). Fast and accurate time series classification with WEASEL. In *CIKM '17, Proceedings of the 2017 ACM on conference on information and knowledge management* (pp. 637–646). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3132847.3132980.

Senin, P. (2021). Seninp/saxpy. original-date: 2017-10-25T16:38:16Z. URL https://github.com/seninp/saxpy.

Senin, P., & Malinchik, S. (2013). SAX-VSM: Interpretable time series classification using SAX and vector space model. In *2013 IEEE 13th international conference on data mining* (pp. 1175–1180). http://dx.doi.org/10.1109/ICDM.2013.52.

Shifaz, A., Pelletier, C., Petitjean, F., & Webb, G. (2020). TS-CHIEF: a scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, *34*, http://dx.doi.org/10.1007/s10618-020-00679-8.

Shu, W., Yao, Y., Lyu, S., Li, J., & Chen, H. (2021). Short isometric shapelet transform for binary time series classification. *Knowledge and Information Systems*, *63*, 2023–2051. http://dx.doi.org/10.1007/s10115-021-01583-3.

Tan, C. W., Bergmeir, C., Petitjean, F., & Webb, G. I. (2021). Time series extrinsic regression. *Data Mining and Knowledge Discovery*, *35*, 1032–1060. http://dx.doi.org/10.1007/s10618-021-00745-9.

Tan, C. W., Dempster, A., Bergmeir, C., & Webb, G. I. (2022). MultiRocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, http://dx.doi.org/10.1007/s10618-022-00844-1.

Thakkar, A., & Lohiya, R. (2021). Analyzing fusion of regularization techniques in the deep learning-based intrusion detection system. *International Journal of Intelligent Systems*, *n/a*(n/a), 1–49. http://dx.doi.org/10.1002/int.22590.

Wainer, J., & Cawley, G. (2017). Empirical evaluation of resampling procedures for optimising SVM hyperparameters. *Journal of Machine Learning Research*, *18*(15), 1–35, URL http://jmlr.org/papers/v18/16-174.html.

Wang, X., Wang, Y., Weng, B., & Vinel, A. (2020). Stock2Vec: A hybrid deep learning framework for stock market prediction with representation learning and temporal convolutional network. arXiv abs/2010.01197.

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 international joint conference on neural networks (IJCNN)* (pp. 1578–1585). http://dx.doi.org/10.1109/IJCNN.2017.7966039.

Yuan, J., Lin, Q., Zhang, W., & Wang, Z. (2019). Locally slope-based dynamic time warping for time series classification. In *CIKM '19, Proceedings of the 28th ACM international conference on information and knowledge management* (pp. 1713–1722). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3357384.3357917.

Zhang, M., Fan, B., Zhang, N., Wang, W., & Fan, W. (2021). Mining product innovation ideas from online reviews. *Information Processing & Management*, *58*(1), Article 102389. http://dx.doi.org/10.1016/j.ipm.2020.102389.

Zhang, J., Karimireddy, S. P., Veit, A., Kim, S., Reddi, S. J., Kumar, S., et al. (2019). Why ADAM beats SGD for attention models. arXiv abs/1912.03194.

Zhao, X., Wang, D., Zhao, Z., Liu, W., Lu, C., & Zhuang, F. (2021). A neural topic model with word vectors and entity vectors for short texts. *Information Processing & Management*, *58*(2), Article 102455. http://dx.doi.org/10.1016/j.ipm.2020.102455.