A Framework to Evaluate Early Time-Series Classification Algorithms

Charilaos Akasiadis
Institute of Informatics &
Telecommunications, NCSR
'Demokritos'
Agia Paraskevi, Greece
cakasiadis@iit.demokritos.gr

Evgenios Kladis
Institute of Informatics &
Telecommunications, NCSR
'Demokritos'
Agia Paraskevi, Greece
eukladis@iit.demokritos.gr

Petro-Foti Kamberi
Institute of Informatics &
Telecommunications, NCSR
'Demokritos'
Agia Paraskevi, Greece
pkamberi@iit.demokritos.gr

Evangelos Michelioudakis
Institute of Informatics &
Telecommunications, NCSR
'Demokritos'
Agia Paraskevi, Greece
vagmcs@iit.demokritos.gr

Elias Alevizos
Institute of Informatics &
Telecommunications, NCSR
'Demokritos'
Agia Paraskevi, Greece
alevizos.elias@iit.demokritos.gr

Alexander Artikis
Department of Maritime Studies,
University of Piraeus
Piraeus, Greece
Institute of Informatics &
Telecommunications, NCSR
'Demokritos'
Agia Paraskevi, Greece
a.artikis@iit.demokritos.gr

ABSTRACT

Early Time-Series Classification (ETSC) is the task of predicting the class of incoming time-series by observing as few measurements as possible. Such methods can be employed to obtain classification forecasts in many time-critical applications. However, available techniques are not equally suitable for every problem, since differentiations in the data characteristics can impact performance in terms of earliness, accuracy, F_1-score, or training time. We evaluate five existing ETSC algorithms on publicly available data, as well as on two newly introduced datasets originating from the life sciences and maritime domains. Existing ETSC algorithms are also compared against a method that selectively truncates time-series and incorporates state-of-the-art algorithms for full time-series classification. Our main goal is to provide a framework for the evaluation and comparison of ETSC algorithms and to obtain intuition on how such approaches perform on real-life applications. The presented framework can serve as a benchmark for new related approaches.

1 INTRODUCTION

The integration of sensors and data transmitters on many physical objects has facilitated the production of time-series data in high volumes [45]. For instance, the integrated sensory and telecommunication devices on ships generate constant information streams, reporting trajectories in the form of time-series data [13]. Such information is utilized by machine learning techniques [11] to solve numerous problems. Full time-series classification (TSC) methods in particular, are applied on numerous fields, e.g. quality assurance via spectrograms [17], power consumption analysis [26], and medical applications [22]. Such methods train models using labeled time-series to classify unlabelled ones, usually of equal length.

A slightly different domain, that of Early Time-Series Classification (ETSC), aims at classifying time-series as soon as possible, i.e. before the full series is observed. Thus, the training instances consist of labeled time-series, while the testing data are incomplete instances with unknown labels. ETSC algorithms aim to maximize the trade-off between predictive accuracy and earliness. The objective is to find the earliest time-point of a time-series at which a reliable prediction can be made, rendering ETSC desirable in a wide range of application domains. For instance, in the life sciences, simulation frameworks analyze how cellular structures respond to treatments, e.g., in the face of new experimental drugs [3]. Such simulations require vast amounts of computational resources, and produce gigabytes of data in every run. Simulations of treatments that do not generate significant cell response could be detected at an early stage and terminated, thus freeing valuable computational resources that otherwise would have been spent in vain. In the maritime domain, popular naval routes around the globe require continuous monitoring in order to avoid undesirable events, such as vessel collisions, illegal actions, etc. [33]. By utilizing available maritime time-series data, such events can be predicted to assist with traffic regulation and respective decision-making. See, e,g., [5, 41] for a further discussion on the applicability of ETSC algorithms on real-world applications.

Although several ETSC methods have been proposed, there is a lack of an experimental evaluation and comparison framework tailored to this domain. Existing reviews for full time-series classification focus on comparing algorithms that do not generate early predictions. Representative approaches may be found in [1, 4, 8, 12], and [36]. Moreover, ETSC methods are mostly evaluated and compared against only a few alternative algorithms, and this is mainly performed using datasets from the UCR repository,[1] which was originally created for evaluating full time-series classification approaches. A recent review of existing ETSC approaches is presented in [14] featuring, however, a theoretical comparison and not an empirical one. In general, benchmarking frameworks are considered valuable for facilitating objective comparisons as they enable a better understanding of how different approaches work on a variety of application domains [21, 23, 25].

[1] http://www.cs.ucr.edu/~eamonn/time_series_data_2018/

In this paper, we present a framework for the empirical comparison of ETSC algorithms on a curated set of appropriate datasets from relevant applications. We use this framework to provide insights about algorithms' strengths and weaknesses. It includes five existing algorithms i.e., *ECEC* [27], *ECONOMY-K* [2], *ECTS* [43], *EDSC* [44], and *TEASER* [39], and offers them for use via a publicly available and extensible repository written in Python.[2] The algorithms are evaluated on 12 ETSC-relevant datasets, and the results are reported in terms of metrics that are widely used in the ETSC domain. Two of the datasets are new, originating from the drug treatment discovery and the maritime domains, while the remaining datasets are a subset of the well-known UEA & UCR repository.[3] They all constitute cases on which ETSC can be considered valuable.

Furthermore, we propose *STRUT*, a method for ETSC that performs selective truncation of the time-series and incorporates state-of-the-art classification algorithms for full time-series. The method exhaustively explores each training dataset to discover the earliest time-point to generate an accurate prediction. Different algorithms designed for full time-series classification are incorporated, in particular MiniROCKET [7], MLSTM [24], and WEASEL [37], which are iteratively trained on data streams with gradually truncated prefices. After a first pass, *STRUT* recognizes the time-step where the best predictive performance can be achieved and chooses that when generating early class label predictions during testing. Although incorporating algorithms that were originally designed to analyze time-series in their full length, *STRUT* is introduced as a baseline to compare ETSC performance. It puts full time-series classification algorithms to the test with a straightforward mechanism for deciding when to produce a classification.

Our contributions are summarized as follows:

- We present an open-source framework for evaluating ETSC algorithms, which contains a wide spectrum of methods and appropriate datasets. Two of the included datasets are novel, from the fields of large-scale simulations of drug treatments for cancer and maritime situational awareness.
- We propose a method for ETSC that relies on state-of-the-art algorithms for full time-series classification.
- We empirically compare the aforementioned ETSC algorithms on a diverse set of datasets.
- The framework is easily extensible in order to incorporate additional algorithms and datasets.

The rest of the paper is organized as follows. In Section 2 we present a running example that is used to explain algorithm functionality, and the metrics we used for algorithm evaluation. Section 3 includes descriptions of the existing algorithms that we consider in our framework. Section 4 describes the method that incorporates full time-series classification algorithms for ETSC. Section 5 presents an overview of the datasets used. In Section 6 we present our empirical evaluation, and in Section 7 we conclude.

2 EXAMPLE AND EVALUATION METRICS

2.1 Running Example

In order to aid the presentation of the algorithms throughout the paper, we use a running example from the life sciences domain. This multivariate time-series stem from simulations of tumor cells during the administration of drug treatments [3, 34]. The values of each time-point indicate the effect of drug treatment configurations over time and they include three different counts for the number of Alive, Necrotic and Apoptotic tumor cells. Intuitively, if a drug configuration is successful, then the number of Alive cells should decrease with time, while the number of Necrotic that indicates the cells destroyed by the drug effects should increase. Figure 1 illustrates examples from different classes. Although apoptotic cells are not considered when generating class labels, there is no indication that their number is irrelevant during the class label prediction process. For example, we can observe different evolutions of apoptotic cells between the two cases of Fig. 1. For this reason we chose to include this variable in the dataset.

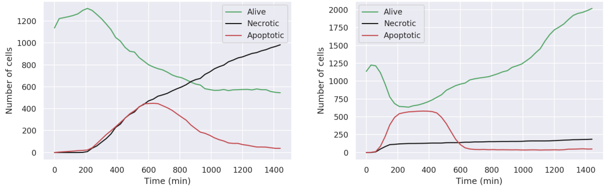

Figure 1: Examples of interesting (left) and not interesting (right) simulation outcomes.

Time-point	t_0	t_1	t_2	t_3	t_4	t_5	t_6
Alive cells	1137	1229	1213	1091	896	744	681
Necrotic cells	0	0	11	42	84	99	103
Apoptotic cells	0	1	17	118	282	432	509

Table 1: Prefix of a tumor drug treatment simulation.

Table 1 shows a prefix of such a simulation's outcome. In this example, the Alive tumor cells decrease after the third time-point, while the Necrotic cells are increasing, indicating that the drug is in effect. The Apoptotic cell count captures natural cell death, regardless of the drug effect. Each of the multivariate time-series is labeled as *interesting* or *non-interesting*, based on whether the drug treatment has been effective or not. The time-series in this example has been classified as interesting, since the tumor shrinks as a result of applying a drug treatment, i.e. the evolution of the number of the different cells indicates a successful treatment.

2.2 Evaluation Metrics

To benchmark ETSC algorithms, we rely on well-known metrics, i.e. accuracy [27, 28, 39, 42, 43] and F_1-score [44] for the predictive performance, earliness [27, 28, 39, 42, 43], the harmonic mean of accuracy and earliness [39], and training and testing times [42].

In particular the accuracy is defined as the ratio of all correct predictions (true positives (*TP*) and true negatives (*TN*)) divided by the total number of the predictions obtained for a particular dataset (adding also the false positives (*FP*) and false negatives (*FN*)), i.e.:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

The F_1-score is defined per class c, and we report the average value among the different classes:

$$F_1\text{-score} = \sum_c \frac{TP^c}{TP^c + \frac{1}{2} \cdot (FP^c + FN^c)} / |C|$$

[2] https://github.com/xarakas/ETSC

[3] https://www.timeseriesclassification.com

where $|C|$ is the total number of distinct class labels in the dataset.

Earliness is defined as the ratio of the time-points that are required for an algorithm to consume, in order to generate a prediction before an event occurs (i.e. a class label is assigned to the time-series). That is, given a time-series instance consisting of L time-points, and assuming that the algorithm requires only the l first time-points ($l \leq L$), then the earliness measure is given by $Earliness = l/L$. To evaluate ETSC algorithms over a dataset we calculate the average earliness for the test data time-series instances. Lower earliness values are better, since the maximum value of 1 means that the algorithm was not able to generate a reliable prediction earlier, before the last time-point of the instance was observed.

The harmonic mean between accuracy and earliness, indicates the balance between prediction quality and time (number of time-points) saved. Since these two metrics have reversed objectives, we use the value of 1 - earliness in the calculation of the harmonic mean in order to align them:

$$HarmonicMean = 2 \cdot \frac{Accuracy \cdot (1 - Earliness)}{Accuracy + (1 - Earliness)}$$

When the harmonic mean is zero, the algorithm in question either achieves the lowest accuracy score, or needs to observe the full time-series for generating a decision on the instance's class label. As the harmonic mean increases, the algorithm can correctly predict class labels with less observations in the input.

Finally, we measure the training times in minutes and test times in seconds. Training times are mainly affected by the size of the dataset and the complexity of the algorithm's calculation. We also include a theoretical time complexity analysis for each algorithm.

3 ALGORITHMS OF THE BENCHMARKING FRAMEWORK

Various ETSC algorithms have been proposed in the literature [15, 19, 28, 29, 40, 46]. Not every algorithm has a publicly available implementation and, moreover, most require a domain-specific customization. According to [14] algorithms can be categorized into four different groups, based on their internal functionality: *Prefix-based* algorithms seek for the minimum prefix length at which a classifier can generate accurate predictions. *Shapelet-based* approaches aim to extract the most characteristic subseries for each particular class, and then use them to match incoming and incomplete time-series. *Model-based* methods estimate conditional probabilities using mathematical models, i.e. given a time-series prefix, calculate the probability of it belonging to a particular class. For algorithms that do not fall into these three groups, a fourth one termed *Miscellaneous* is defined, including e.g. deep learning or reinforcement learning techniques.

We have selected five algorithms representative of each category in the taxonomy presented in [14]: *ECONOMY-K* [2] (model-based), *ECTS* [43] (prefix-based), *EDSC* [44] (shapelet-based), *ECEC* [27] (model-based), and *TEASER* [39] (prefix-based). This selection was based on the following criteria: (a) there was an open-source implementation of the algorithm, and (b) the algorithm did not depend on domain-specific customisation or a large number of parameters. The only exception is *EDSC*, which we chose to implement ourselves, since it is a widely used algorithm. We also include three full time-series classification algorithms, i.e. MiniROCKET [7], MLSTM [24], and WEASEL [37], which can be incorporated by our *STRUT* method that we present in Sec. 4. Table 2 summarizes the evaluated algorithms and their characteristics.

Algorithm	Model-based	Prefix-based	Shapelet-based	Miscellaneous	Univariate	Multivariate	Early TSC	Full TSC	Language
ECEC	✓				✓		✓		Java
ECONOMY-K	✓				✓		✓		Python
ECTS		✓			✓		✓		Python
EDSC			✓		✓		✓		C++
MiniROCKET				✓		✓		✓	Python
MLSTM				✓		✓		✓	Python
WEASEL			✓		✓	✓		✓	Python
TEASER		✓			✓		✓		Java

Table 2: Characteristics of evaluated algorithms.

3.1 ECONOMY-K Algorithm

Dachraoui et al. [2] introduce *ECONOMY-K* which is based on a decision function that searches for the future time-point index at which a reliable classification can be achieved. It supports univariate time-series. Note that for all algorithms supporting only univariate datasets, in case of multivariate input we train a number of classifiers respective to the number of variables in parallel, and apply a simple voting method to obtain the class label predictions. Thus, in the running example, apart from the 'Alive' cells that is depicted as input stream, we also have the two other variables ('Necrotic' and 'Apoptotic') that are fed to separate classifiers.

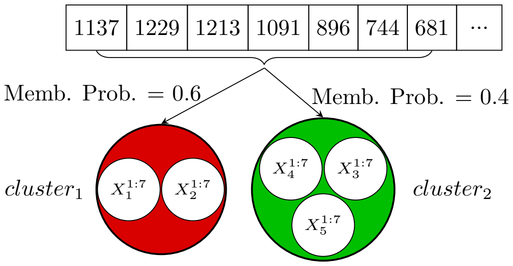

Figure 2: Illustration of *ECONOMY-K*'s cluster assignment.

According to *ECONOMY-K*, the full length training time-series are divided into k clusters using k-Means. For each time-point, a base classifier h_t is trained (e.g., XGBoost). For each cluster k and time-point t, the classifiers h_t are utilized to construct a confusion matrix that is used to compute the probability of a prediction being correct. When incomplete time-series $X^{1:t}$ are given to the decision function, a cluster membership probability is assigned. For example, Figure 2 shows that during the training step there are five time-series $X^{1:7}$ that share a similar prefix, i.e. $\{1137, 1229, 1213, 1091, 896, 744, 681\}$, which is passed in the input of *ECONOMY-K*. The highest membership probability belongs to $cluster_1$; therefore we select the classifier trained for the particular prefix length and cluster. The ultimate goal is to find the future time-point index τ out of the remaining ones that are expected to arrive as input, where the algorithm can generate a prediction with the best confidence. This is performed by considering a cost function $f_\tau(X^{1:t})$.

In our example the future time-point position τ at which a prediction could be safely generated would either be 0, or 1 if the most appropriate time-point was the one expected to arrive next. Suppose that the cost function $f_\tau(X^{1:t})$ for each τ takes values $\{0.5, 1.2\}$. The minimum value is that of f_0, therefore the best time-point to make a prediction would be the current one. If the lowest cost was for $\tau = 1$, then the algorithm would require more data. The efficient implementations of clustering and classification methods can enable *ECONOMY-K* to train very fast, even for very large datasets.

3.2 ECTS Algorithm

The Early Classification of Time-Series (*ECTS*) [43] algorithm is based on the 1-Nearest Neighbor (1-NN) method. *ECTS* stores all the nearest neighbor sets, for all time-series in the training set and for each prefix length. Next, judging by the structure of these sets, it computes the Reverse Nearest Neighbors (RNN).

For example, Figure 3 shows a directed graph on the left, with five time-series as nodes, which are prefixes of a given size. The "in-degree" of each node of the graph signifies how many RNNs it has. For instance, $X_5^{1:7}$ has two inward edges; therefore two nodes consider it as their nearest neighbor, and thus $X_5^{1:7}$ has the RNN set $\{X_4^{1:7}, X_3^{1:7}\}$. On the other hand, $X_4^{1:7}$ has zero inward edges, so its RNN set is empty. For each possible prefix, this procedure is repeated to come up with all RNN sets. The time-point from which the RNN set of a time-series remains the same until the end of the time-series, indicates that prefixes up to this time-point can be discerned from different class instances. This prefix length is called the Minimum Prediction Length (MPL). Intuitively, MPL signifies from which time-point onward the time-series belonging to the same class are quite similar. For example Figure 3 depicts the NN and RNN sets for 5 different time-series. The MPL (NN) of $X_2^{1:7}$ means that $X_2^{1:7}$ can act as a predictor for new time-series, using only the first 7 time-points of the time-series.

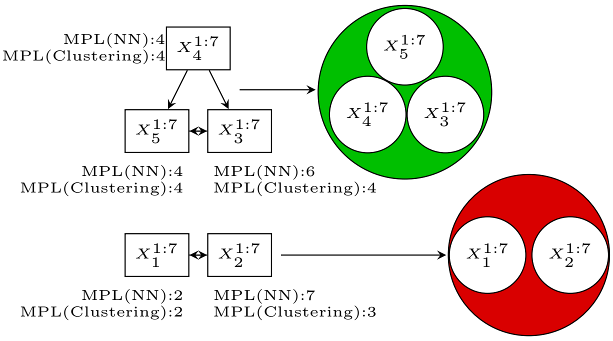

Figure 3: Reverse nearest neigbors (left) and clustering based on RNN and NN (right), as considered by *ECTS*.

In order to minimize the MPL and avoid needlessly late predictions, *ECTS* incorporates agglomerative hierarchical clustering. For every cluster, an MPL is assigned to it. The calculation of MPL is based on the RNN consistency, as well as on the 1-NN consistency. The RNN sets of the cluster for each prefix are calculated by applying relational division [10] on the union of the RNNs of the member time-series and the time-series of the cluster.

Now, the 1-NN consistency dictates that the nearest neighbor of each time-series in the cluster also belongs in that cluster for time-points up to the maximum length. The 1-NN and RNN are calculated for each prefix. The time-point from which both 1-NN and RNN sets are consistent up to the full time-series length, is the MPL of the cluster. During a cluster merging, time-series are assigned to the smallest MPL among the cluster they belong to, and their own MPL. The result of the clustering phase is shown in Figure 3, where the MPL (Clustering) values are much lower for $X_2^{1:7}$ and $X_3^{1:7}$, indicating the capability for earlier predictions. According to the testing phase, for each prefix, new incoming time-series are matched to their nearest neighbor. If the observed length of the time-series up to the current time-point is larger than the MPL of its nearest neighbor, a prediction can be generated. Similarly to the previous algorithms, *ECTS* operates on univariate time-series.

3.3 EDSC Algorithm

Early Distinctive Shapelet Classification (*EDSC*) [44] is one of the first methods proposed for ETSC. Shapelets are composed of subseries that are in some sense maximally representative of a class and are defined as triplets of the form (*subseries*, *threshold*, *class*). To compute the thresholds, *EDSC* finds all time-series that belong to different classes than *class*, and measures their minimum distance from the *subseries*. Then, the Chebyshev's Inequality utilizes the mean and variance of the distances and a user-defined tolerance k to calculate the *threshold*. That is, given a shapelet, each time-series excerpt with distance greater than the *threshold* signifies that the time-series belongs to a different class. The shapelets are finally ranked according to their "distinctive" capability and the best ones form the classification basis.

Consider the time-series of the Alive cells of the running example, as presented in Table 1. Let the shapelets length be 3. Figure 4 visualizes this example with three of the possible subseries. Let one subseries be $sb_1 = \{1137, 1229, 1213\}$, originating from a time-series of class *one*. The minimum distance of a shapelet to a time-series is computed by aligning the shapelet against all subseries, and finding the minimum among their distances. Suppose that the minimum distance to sb_1 is stored in a list, the mean of which is 330 and the variance 109. Given $k = 3$, we calculate the threshold $\delta = max\{mean - k \cdot var, 0\} = 3$ according to the Chebyshev's Inequality. This indicates that time-series with distance less than δ from sb_1 belong to the same class.

After this step, the shapelet $s_1 = (sb_1, \delta, class)$ is created. The same procedure is repeated for the remaining subseries. Then, for each shapelet in the shapelet's list, a utility score is calculated and the top-k are selected for classifying the whole training dataset. Assume that for the three shapelets s_1, s_2, s_3 of Figure 4, the list of utilities is $\{1.3, 3.67, 0.83\}$. The subseries of each shapelet are marked with ovals of different color. We first try to classify the whole training dataset using only s_2, since it has the highest utility. If we cannot correctly classify all instances, then we add the second most "valuable" shapelet, s_1, to the set. Supposing that s_1, s_2 are informative enough, then we can claim that we have

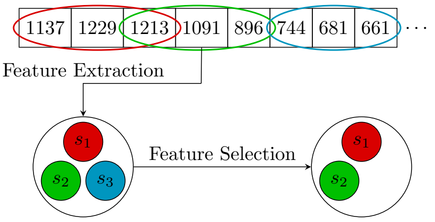

Figure 4: Illustration of *EDSC*.

succeeded in classifying the rest of the dataset, so the shapelet selection process is complete and the remaining shapelets are rejected, in this example s_3. When the minimum distance of a new, incoming time-series from a shapelet is less than δ, then the shapelet's class is returned. This procedure is carried out for all possible prefixes of the incoming data, until the algorithm generates a prediction.

EDSC supports only univariate time-series classification. Moreover, as the size of the dataset increases, so does the required time to extract and calculate shapelets. *EDSC* is one of the most widely used baselines for the comparison of ETSC algorithms.

3.4 WEASEL Algorithm

The full time-series classification algorithm[4] that we based on word extraction (WEASEL [37]) is incorporated by two existing ETSC approaches that we include in our analysis, as well as by our proposed method that we describe in Section 4. It is a sliding window approach, that transforms time-series into feature vectors, which are subsequently analyzed by a machine learning classifier. First, WEASEL extracts windows of varying lengths from the data at the input. The Fourier transform is then used to approximate each window, and the coefficients that best distinguish time-series from different classes are retained. The goal dimensionality, i.e. the number of Fourier values that the algorithm should operate upon, is learned via cross-validation. Next, WEASEL discretizes the remaining Fourier coefficients into a word through information gain binning, which also selects discretization boundaries to optimally discriminate the time-series classes. Ultimately, the method constructs a bag of patterns by utilizing the words (unigrams) and adjacent words (bigrams). Once WEASEL constructs this feature vector that can be considered to be highly discriminative, a fast linear-time logistic regression classifier is applied to obtain class label predictions. There is also a similar multivariate alternative, i.e. WEASEL+MUSE [38], which considers more features, e.g. the derivatives of the neighbouring time-points in each dimension, and also applies a weighing mechanism on the features to focus on the ones that are more important for each case.

3.5 ECEC Algorithm

The Effective Confidence-based Early Classification (*ECEC*) [27] algorithm supports univariate time-series. It truncates the input into N overlapping prefixes, starting from size equal to the length of the time-series divided by N, up to its full length, and trains N WEASEL classifiers h_t. Figure 5 illustrates this process for a univariate time-series. In the example, the time-series is passed to WEASEL, which subsequently extracts subseries (e.g. {1137, 1229, 1213}), transforms them to symbols forming words (w_1, w_2, . . .) and counts their instantiations (e.g., suppose {2,1,7}).

On the set of N base classifiers, *ECEC* conducts a cross validation to obtain the probabilistic predictions for each fold. Based on these probabilities, for each classifier h_t, *ECEC* measures the performance according to the probability of a true label being y, with the predicted label being $\hat{y}$. A core component of this algorithm is the confidence $C_t(h_t(X))$, which indicates the anticipated reliability of a prediction for each prefix of size k. For each time-series and each prefix, *ECEC* calculates the confidence of the prediction made from the corresponding classifier h_t during cross validation and populates a list. Then, this list is sorted, and the mean of adjacent values constitute threshold candidates (θ_i). For each θ_i and each time-series, the algorithm compares the confidence of each classifier's predictions at each time-point to θ_i. If a prediction is confident enough, *ECEC* stores it along with the time-point and the confidence value. Once all time-series for all prefixes are evaluated for θ_i, *ECEC* checks the performance of the given threshold. The time-points and the predictions stored during the previous step are then used to calculate the accuracy and earliness for each threshold, according to the evaluation cost function $CF(\theta)$ value: $CF(\theta) = \alpha(1 - Accuracy) + (1 - \alpha)Earliness$, a is a parameter that allows users to tune the trade-off between accuracy and earliness. The θ_i that minimizes this cost, is marked as the global best θ.

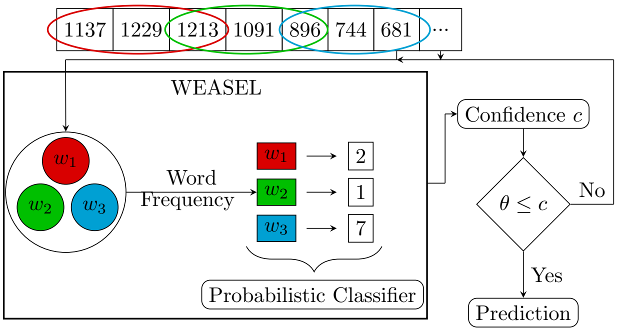

Figure 5: Illustration of *ECEC*.

Figure 5 shows an example of the WEASEL classifier used by *ECEC*. Assuming that the number of prefixes $N = 4$ and the length of the time-series is 10, the minimum prefix during training is $\lceil \frac{10}{4} \rceil$, i.e. $\{1137, 1229, 1213\}$. The prefixes are passed to WEASEL, which in turn outputs the corresponding words and frequencies. Subsequently, the confidence of a prediction is calculated as $c = C_t(h_t(X_{test}^{1:t}))$. Suppose that the confidence c is 0.45 and the confidence threshold θ is 0.5. In this case, the prediction is rejected and more time-points must be observed in order to form the next prefix. After the next 3 time-points are observed, a new prediction is generated and the confidence is recalculated and compared to θ. If, $c \geq \theta$ the prediction is accepted, otherwise more data is required.

3.6 TEASER Algorithm

The Two-tier Early and Accurate Series classifiER (*TEASER*) [39] method is also based on WEASEL. After the transformation of input time-series to words, the frequencies of the words are forwarded to a logistic regression classifier. The training dataset is z-normalized and truncated into S overlapping prefixes. The first prefix is of size equal to the length of the time-series divided by S, and the last one is the full time-series. For each prefix, a WEASEL-logistic regression pipeline is trained and then used to obtain probabilistic predictions. These predictions are passed on to a One-Class SVM, uniquely trained for each prefix length using only the correctly classified data by the pipeline. If the One-Class SVM accepts the prediction, i.e. it is marked as belonging to the correct class, then the last criterion to generate the final output is the consistency of the particular decision for v consecutive prefixes. The parameter v is selected during the training phase by performing a grid search over the set of values $\{1, \ldots, 5\}$. For each candidate value, the method tries to classify all the training time-series, and the one that leads to the highest harmonic mean of earliness and accuracy is finally selected.

[4] Although a full TSC algorithm, we chose to present *WEASEL* here, as it is incorporated by the two ETSC algorithms that we describe later in this section.

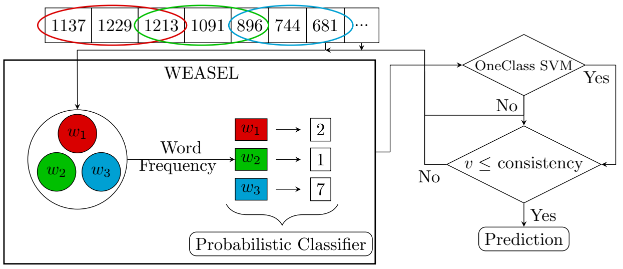

Figure 6: Illustration of *TEASER*.

Figure 6 shows a schematic view of the procedure followed by *TEASER*. Again, assume that the first examined prefix is of size 3 and that the prediction made is accepted by the One-Class SVM. Given that $v = 2$, the consistency check can only accept equal predictions that had been made for two consecutive prefixes. Thus, the current prediction will not be accepted, since in our case it was obtained by using only one prefix. *TEASER* will wait for the next 2 time-points. If *TEASER* does not manage to find an acceptable prediction by the time the final prefix arrives, then the prediction using the whole time-series is generated without passing through the One-Class SVM or any other consistency check.

TEASER uses a pre-defined number of overlapping prefixes that reduces the number of possible subseries that need to be examined, thus boosting the method's performance. On the other hand, bad choices of S can lead to suboptimal results. Also, *TEASER* supports only univariate data and relies on z-normalization. Z-normalization is performed by calculating the mean and standard deviation using all time-points of a time-series. This is unrealistic for online operation, since algorithms can only operate on data seen so far.

4 SELECTIVE TRUNCATION OF TIME-SERIES

We propose the 'Selective Truncation of Time-Series' (*STRUT*) as a baseline method to compare the predictive performance of full time-series classification algorithms in ETSC settings. *STRUT* determines the best time-point where time-series classification performance is superior to the performance of the classification at any other time-point of the time-series, following a repeated truncation approach and applying full TSC algorithms. We incorporate three existing ones, MiniROCKET [7], WEASEL [38], and MLSTM [24], all supporting multivariate datasets.

MiniROCKET transforms time-series using convolutional kernels and utilizes the transformed features to train a linear classifier [7]. Each input time-series is convolved using 10,000 convolutional kernels and only one feature, called Proportion of Positive Values, which is equivalent to computing the empirical cumulative distribution function. Subsequently, dilation is applied in order to "spread" a kernel over the input, as well as zero padding. The time complexity of the method is linear to the number of kernels/features, the dataset size, and time-series length.

The Multivariate Long Short Term Memory Fully Convolutional Network (*MLSTM*) consists of two sub-models that are assigned the same input [24]. The first sub-model comprises three Convolutional Neural Network layers. The outputs of each of the first two layers are batch normalized [20] and are then passed on to an activation function, i.e., a Rectified Linear Unit (ReLU). In order to maximize the efficiency of the model for multivariate time-series, the activated output is also passed into a Squeeze-and-Excite block [18]. The second sub-model consists of a masking layer whose output is passed on to an attention-based LSTM. The output of the two sub-models is concatenated and passed through a dense layer with as many neurons as the classes, and via a softmax activation function it generates probabilistic predictions. The complexity is linear to the dataset's size and to the number of epochs during training.

The key idea of the *STRUT* method is to determine the time-point at which the performance of the early classification reaches its optimal level, in other words, is superior to the performance observed when consuming different prefix lengths of the time-series. The term performance corresponds to a user-defined metric (i.e accuracy, F_1-score, or the harmonic mean of earliness and accuracy). Figure 7 displays an overview of our approach.

The full-time series examples in the training dataset are iteratively truncated to obtain prefices of gradually increasing lengths. In each iteration, the truncated examples along with the corresponding class labels are provided to a full time-series classification algorithm, which fits on the training data, and then is tested on a validation dataset, which is respectively truncated to the particular prefix length. Finally, the prefix length for which the algorithm achieved the best score, indicates the time-point at which the algorithm will predict the class label during the testing phase.

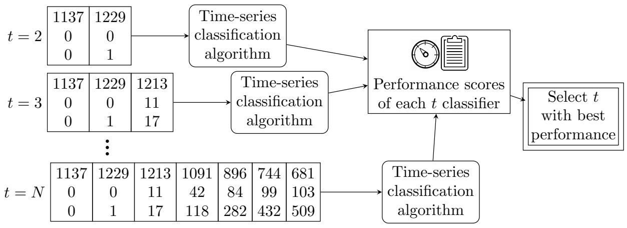
$t = 2$
1137 1229
0 0
0 1
Time-series classification algorithm
$t = 3$
1137 1229 1213
0 0 11
0 1 17
Time-series classification algorithm
⋮
$t = N$
1137 1229 1213 1091 896 744 681
0 0 11 42 84 99 103
0 1 17 118 282 432 509
Time-series classification algorithm
Performance scores of each t classifier
Select t with best performance

Figure 7: *STRUT* method overview.

The end-user specifies the desired full TSC algorithm to be used, as well as the performance metric to be optimized. The time-point at which the truncated time-series are shown produce the best results is when the final prediction is generated. WEASEL and WEASEL+MUSE, which we use in univariate and multivariate cases respectively, are modified to remove a default normalization step. This step would transform the input data so that its distribution would have a mean of 0 and standard deviation 1. In a streaming setting, however, it cannot always be assumed that we know in advance the range of input values.

Aiming to lower the total execution time required to perform the exhaustive approach that *STRUT* follows, especially in the case of time-series of considerably large length, we follow an iterative binary search process to determine the minimum t, skipping this way a substantial number of iterations.

5 DATASETS & FRAMEWORK EXTENSIBILITY

We have employed a subset of the publicly available UEA & UCR Time-series Classification Repository [6], as well as two new datasets that we introduce, from the life sciences and maritime domains. The selected datasets include both univariate and multivariate cases, and are all available via our source-code repository.[2] Recall that, for the algorithms that cannot operate on multivariate cases—i.e., *ECEC*, *TEASER*, *ECTS*, *ECONOMY-K*, and *EDSC*—different algorithm instances were trained for each variable, and a simple voting over the individual class label predictions was applied to obtain the final one. Detailed descriptions of each dataset may be found online.[5]

5.1 UEA & UCR repository

We selected 10 out of 178 UEA & UCR datasets for our evaluation, according to the following criteria: *(a)* data should have a temporal dimension (e.g., image shapes are not acceptable), *(b)* data should not be normalized, and *(c)* the time horizon should be more than a few seconds. Note that several datasets from the UEA & UCR repository have missing values or time-series of varying length, whereas *ECTS* and *ECONOMY-K* implementations do not support such cases. To address this, we fill in the missing values with the mean of the last value before the data gap and the first one after it.

5.2 Biological dataset: Cancer cell simulations

This dataset originates from the life sciences domain, in particular drug discovery. As we explained earlier in Section 2.1, to explore potentially helpful cancer treatments, researchers conduct large-scale model exploration with simulations of how tumor cells respond to drug administration [3, 34]. Each simulation is configured with a particular drug treatment configuration, and its course can be summarized by three time-evolving variables, for alive, necrotic and apoptotic cells. Each experiment differs from the others based on a set of configurable parameters related to the treatment, i.e. the frequency of drug administration, its duration, and the drug concentration. These values remain fixed during each simulation. The time-series are labeled as *interesting* or *non-interesting*, based on whether the treatment was found to be effective or not, i.e. managing to constrain tumor cell growth, according to a classification rule that was defined by domain experts. The dataset consists of 644 time-series, each having 48 time-points. The measurements were obtained using a parallel version of the PhysiBoSSv2.0 simulator.[6]

Performing large numbers of simulations to sufficiently explore the space of drug treatment configurations requires significant computational resources. However, when particular simulations turn up to be non-interesting, the respective resources spent may be considered as wasted. Thus, obtaining a prediction for the outcome of the simulation at its early stages can be quite helpful, by utilizing it to decide which simulations to terminate before they are completed, and subsequently free up resources that could otherwise be used to explore more interesting areas of the drug treatment space.

In this dataset, classes are imbalanced. The *interesting* time-series constitute 20% of the dataset, while the remaining 80% accounts for *non-interesting* cases. Also, many interesting and non-interesting instances tend to be very similar during the early stages of the simulation, until the drug treatment takes effect, which is usually after the first 30% of the time-points of each experiment. Consequently, it is difficult to obtain accurate predictions earlier. For these reasons, this is a challenging benchmark for ETSC.

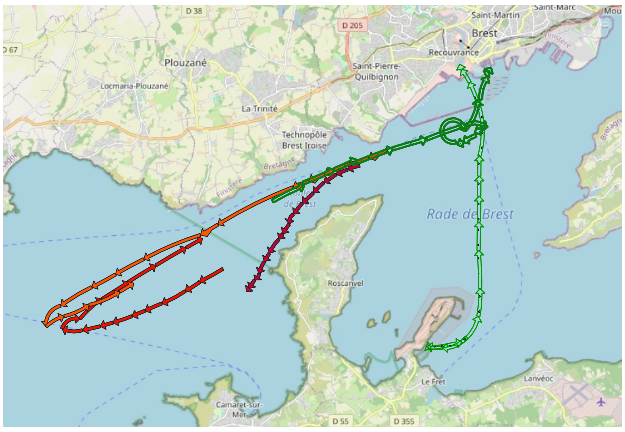

Figure 8: Maritime dataset: Green instances indicate that the vessel is located inside the port of Brest at the end of the 30 minute time interval, while the red ones indicate that the vessel is cruising outside the port.

5.3 Maritime dataset: Vessel position signals

The maritime dataset contains position signals from nine vessels that cruised around the port of Brest, France. In this domain, it is important to detect events of interest ahead of time, so that any reaction can be performed in a proactive manner. Examples of such events include congestions, vessel collisions, illegal activity, etc.

The dataset that we incorporate originates from real world information and is derived from [32, 35]. Each point in a time-series includes the following attributes: the timestamp, ship's ID, longitude, latitude, speed, heading, and course over ground of a vessel at a given time-point. Originally, the dataset was unlabelled and divided into nine time-series, one per vessel, expressing the trajectories of the vessel, with over 12,000 time-points, and a frequency of one measurement per minute. We labelled the time-series depending on whether the vessel in question ended inside the Brest port. To achieve this, we divided the time-series to 30 minute overlapping intervals, and checked if during the last time-points of these intervals the vessel's position lied inside the Brest port polygon. Examples of positive (i.e. the vessel reached the port) and negative (not located inside the port) trajectories are given in Figure 8.

In total, 80,591 time-series instances were formed, each one having 30 time-points, corresponding to 30 minutes. Apart from being multivariate (7 variables) and imbalanced (65,124 negative and 15,467 positive examples), the dataset includes the largest number of examples in our dataset list, making it a challenging application.

5.4 Dataset Categorization

We group the selected datasets according to characteristics that might impact algorithm performance. The eight groups that we consider are shown in Table 3 as columns. We measured the dataset size in terms of length and height, where the length refers to the maximum time-series horizon in the dataset (number of time-points per time-series) and the height corresponds to the number of time-series instances (number of time-series examples). Datasets with length > 1,300 are considered as 'Wide' and with height > 1,000 as 'Large'. We also computed the coefficient of variation (CoV) for each dataset (standard deviation

[5] https://github.com/xarakas/ETSC/blob/master/supplementary.pdf
[6] https://github.com/xarakas/spheroid-tnf-v2-emews

over time-points and instances divided by their average) and the class imbalance ratio (CIR), in order to detect 'Unstable' (CoV > 1.08) and 'Imbalanced' (CIR > 1.73) datasets respectively. The CIR is calculated by dividing the number of instances of the most populated class with that of the least populated one. The number of classes indicates another group of datasets that include more than two ('Multiclass'). The thresholds for height and length were set empirically. For the class imbalance ratio and the coefficient of variance we used the median of the dataset values as the threshold. Finally, datasets not belonging to any of the above groups are marked as 'Common', and we also add two groups for 'Univariate' and 'Multivariate' cases, to capture possible effects of the voting method which was not foreseen by the original algorithms. Note that some categories are not mutually exclusive, e.g. the HouseTwenty dataset belongs to the 'Wide', the 'Unstable', and the 'Univariate' categories.

Dataset Name	Wide	Large	Unstable	Imbalanced	Multiclass	Common	Univariate	Multivariate
BasicMotions			✓		✓			✓
Biological				✓				✓
DodgerLoopDay					✓		✓	
DodgerLoopGame						✓	✓	
DodgerLoopWeekend				✓			✓	
HouseTwenty	✓		✓				✓	
LSST		✓	✓	✓	✓			✓
Maritime		✓	✓	✓				✓
PickupGestureWiimoteZ					✓		✓	
PLAID	✓	✓	✓	✓	✓		✓	
PowerCons						✓	✓	
SharePriceIncrease		✓	✓	✓			✓	

Table 3: Dataset characteristics.

5.5 Framework Extensibility

The proposed benchmarking framework can be extended to include new algorithms and datasets. To add a new algorithm, one needs to create a Python interface that implements the abstract class `EarlyClassifier`, and provide the algorithm functionality for `train` and `predict` methods in a separate .py script placed inside the `etsc/algorithms` folder. Then, the file `cli.py` must be extended *(a)* to import the new algorithm implementation and *(b)* to define input parameters and other execution options. Note that algorithms can be implemented in any language, as long as a Python wrapper is provided. For enriching the datasets list, measurements must be in `.csv` file format, where each row constitutes a time-series example of a single variable, and the first value of each row, the class label. Files of type `.arff` are also supported.

6 EMPIRICAL COMPARISON

6.1 Experimental setup

In our empirical evaluation we use the metrics defined in Sec. 2.2. In particular, we employ accuracy and F_1-score to measure the quality of the predictions, the earliness score, and the harmonic mean between earliness and accuracy. Moreover, we present the training times for each algorithm and an assessment regarding online performance, which is dictated by testing times.

We compare *ECEC*, *ECONOMY-K*, *ECTS*, *EDSC*, *TEASER*, and the three variants of the *STRUT* faster approximation variant, i.e. *S-MINI* (MiniROCKET), *S-MLSTM*, and *S-WEASEL*. Most implementations were readily available in Python or Java, except for *EDSC* and *STRUT* that we had to implement ourselves in C++ and Python respectively. We must note that different programming languages and algorithm implementations might affect training and testing times, however we do not assess such implementation aspects. The experiments were run on a computer operating in Linux, equipped with an Intel Xeon E5-2630 2.60GHz (24-cores) and 256 GB RAM. For all datasets we performed a stratified random sampling 5-fold cross-validation.

Since most of the algorithms we incorporate don't support multivariate input and not all our dataset are univariate, a voting method is applied, similar to the one employed in [36]. According to it, each univariate classifier is trained and tested separately for each variable of the input time-series. Upon collecting the output predictions (one per variable), the most popular one among the voters is chosen, nevertheless assigned with the worst earliness among them. In the case of equal votes, we select the first class label. In cases where the different variables are not independent, the employed voting scheme may be suboptimal. However, the voting scheme did not forbid univariate algorithms achieve higher accuracy than multivariate algorithms on multivariate datasets - the details are presented in the section that follows.

Algorithm	**Parameter values**
ECEC	$N = 20$, a= 0.8
ECONOMY-K	$k = \{1, 2, 3\}$, λ=100, $cost = 0.001$
ECTS	$support = 0$
EDSC	CHE, $k = 3$, $minLen = 5$, $maxLen = L/2$
TEASER	S : 20 for UCR S:10 for the Biological and Maritime

Table 4: Parameter values of ETSC algorithms.

Regarding the configuration parameters that each algorithm requires, the best performing values were selected after exploratory smaller scale experiments. In particular, for *TEASER* the number of prefixes/classifiers S was set to 10 for the biological and maritime datasets, whereas for the UCR & UEA datasets it was set to 20. For *ECEC*, the number of prefixes N was set to 20. The original *TEASER* approach applies z-normalization internally according to the original algorithm design, however, as already discussed, this might not always be suitable for an online setting in the ETSC domain. Thus, we decided to test a variant of this algorithm without the normalization step. The v parameter for *TEASER*'s consistency check is optimized on a per-dataset basis. For *ECONOMY-K* (abbreviated as *ECO-K* for the rest of this paper) we experimented on $\{1, 2, 3\}$ clusters for each dataset. Table 4 summarizes all the parameter values used in our empirical comparison. Finally, for *S-MLSTM*, due to the increased time required for training, the designated truncation time-point was determined by evaluating at $\{0.05, 0.2, 0.4, 0.6, 0.8, 1\}$ times each dataset's length, fixing this way the number of iterations regardless of the dataset's length. Also, the number of LSTM cells was calculated by using a grid search among $\{8, 64, 128\}$, choosing the one yielding the best score.

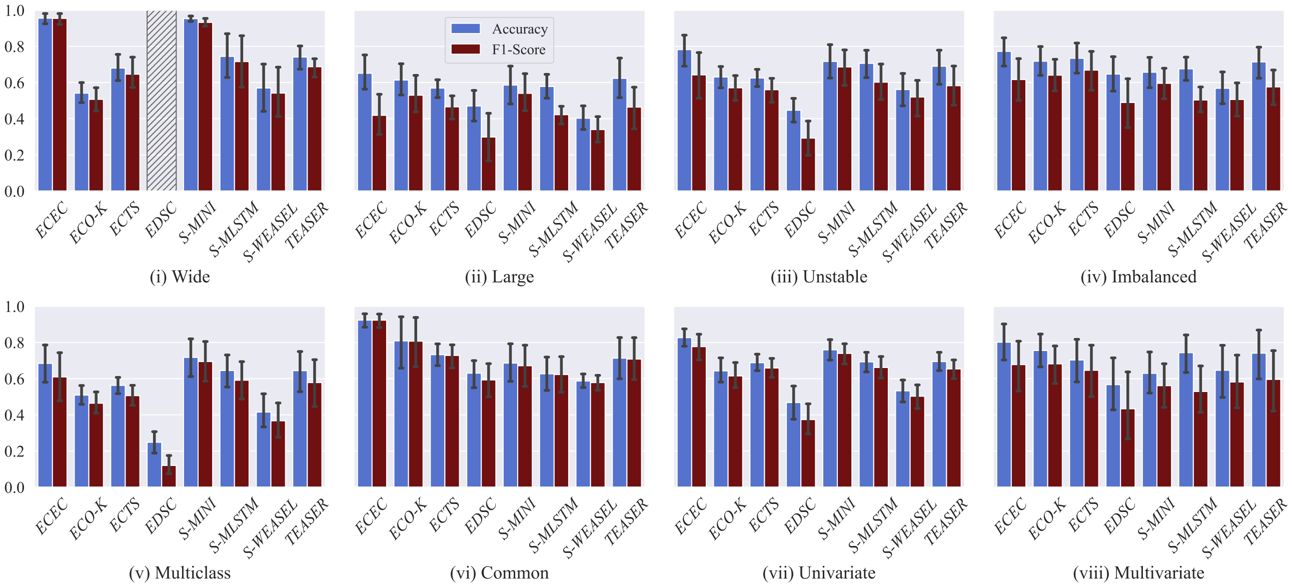

Figure 9: Accuracy and F_1-Score. *EDSC* did not produce results for 'Wide' datasets within 48 hours.

When an algorithm required more than 48 hours for training we terminated the experiment. The source code, datasets and execution instructions are publicly available.[2] Thus, our empirical comparison is *reproducible*. In the following section, we report the experimental results per dataset type, i.e. the average scores of the algorithms for all datasets in the respective category, while the results per dataset can be found in the supplementary material.[5]

6.2 Experimental results

6.2.1 Predictive Accuracy. Figure 9 presents the accuracy and F_1-score values for each dataset category. In terms of accuracy, *ECEC* is shown to be the best for all categories, apart from 'Multiclass' for which it ranks second. *S-MINI* is very competitive, ranking first for 'Multiclass' datasets, and second for 'Wide', 'Univariate' and 'Unstable' cases (after *ECEC*). This is mainly the result of the confidence thresholds that *ECEC* employs, which effectively favor its predictive performance against other algorithms, and, for the case of *S-MINI*, the MiniROCKET full time-series classification algorithm that has been shown to be very accurate in a variety of datasets and settings. *TEASER* has a stable performance and its accuracy results are not shown to be significantly impacted by dataset characteristics, apart from 'Large' cases for which it achieves slightly better than 0.6 on average, nevertheless ranking second, followed by *ECO-K*, *S-MINI* and *S-MLSTM*. This higher ranking is the result of *TEASER*'s consistency check. *EDSC* and *S-WEASEL* do not perform well, with *EDSC* outperforming *S-WEASEL* in 'Common', 'Imbalanced', and 'Large' datasets. *ECTS* lies in the middle ranks.

For some dataset categories the algorithms' F_1-score is greatly impacted. For the 'Common', and 'Wide', accuracy is pretty close to the F_1-score achieved values, whereas for the rest the differences are noticeable. The main reason for this is class imbalance, which can render accuracy inappropriate as an evaluation metric [16]. Also, since imbalanced datasets are also categorized as 'Large', 'Multiclass' and 'Unstable', performance drop is observed for these cases as well. *ECEC*, *TEASER* and *S-MLSTM* are shown to be the most impacted algorithms in terms of F_1-score as a result of class imbalance. This is expected, since the absence of adequate samples of the minority class results to looser thresholds for *ECEC*, poorly fitted classifiers for *TEASER*, and demands more complex and sophisticated cost functions for training the *S-MLSTM*. Moreover, the univariate algorithms *ECEC*, *ECO-K*, *ECTS*, and *TEASER* that utilize the voting scheme achieve higher accuracy on multivariate cases than the multivariate *S-MINI* algorithm.

6.2.2 Earliness. In terms of earliness, we show algorithm performance in Figure 10 (lower values are better). Here we can see that *S-MLSTM* generates earlier predictions for most dataset categories apart from the 'Wide' case, where *ECO-K* and *TEASER* are shown to be better. We can also observe that although *S-MLSTM* mainly captures the class differences, the time-series truncation process may 'hide' important features during training making this more prominent in datasets with more time-points per example. On the contrary, the earliness cost function of *ECO-K* and the One-Class SVM of *TEASER* are shown to be valuable components for such cases. *S-WEASEL* ranks second for 'Common' and 'Univariate' cases, and is also better than *TEASER* for 'Multiclass' and 'Multivariate', illustrating however that the One-Class SVM and validation component that *TEASER* incorporates do not always help.

6.2.3 Harmonic Mean. We now discuss the algorithms' performance in terms of harmonic mean between accuracy and earliness (cf. Sec. 2.2). Figure 11 shows a plot of the results. *S-MLSTM* achieves the highest scores for most dataset categories, apart from the 'Wide' case, for which ranks fourth, and for 'Unstable' and 'Common' that comes second. This is mainly due to lower and highly varying accuracy scores for the PLAID dataset and the worse Earliness score in the case of HouseTwenty. Other than that, *S-MLSTM* results demonstrate the strength of neural networks in performing early and accurately. In the 'Wide' category, *ECEC* is shown to be the most competitive, followed by *TEASER*, *S-MINI*, and *S-MLSTM*. Considering that *S-WEASEL*, a full time-series classification algorithm that *TEASER* incorporates as an underlying module, performs consistently worse than *TEASER*, makes apparent that the One-Class SVM of *TEASER* helps to

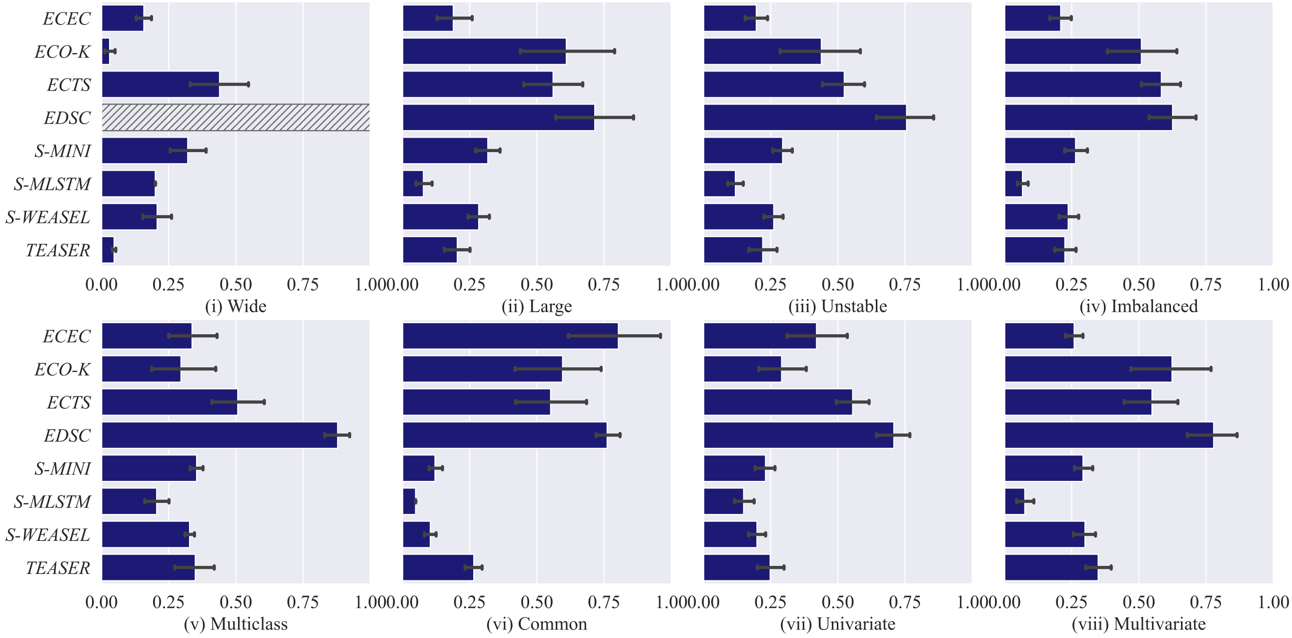

Figure 10: Earliness (lower values are better, *EDSC* did not produce results for 'Wide' datasets within 48 hours).

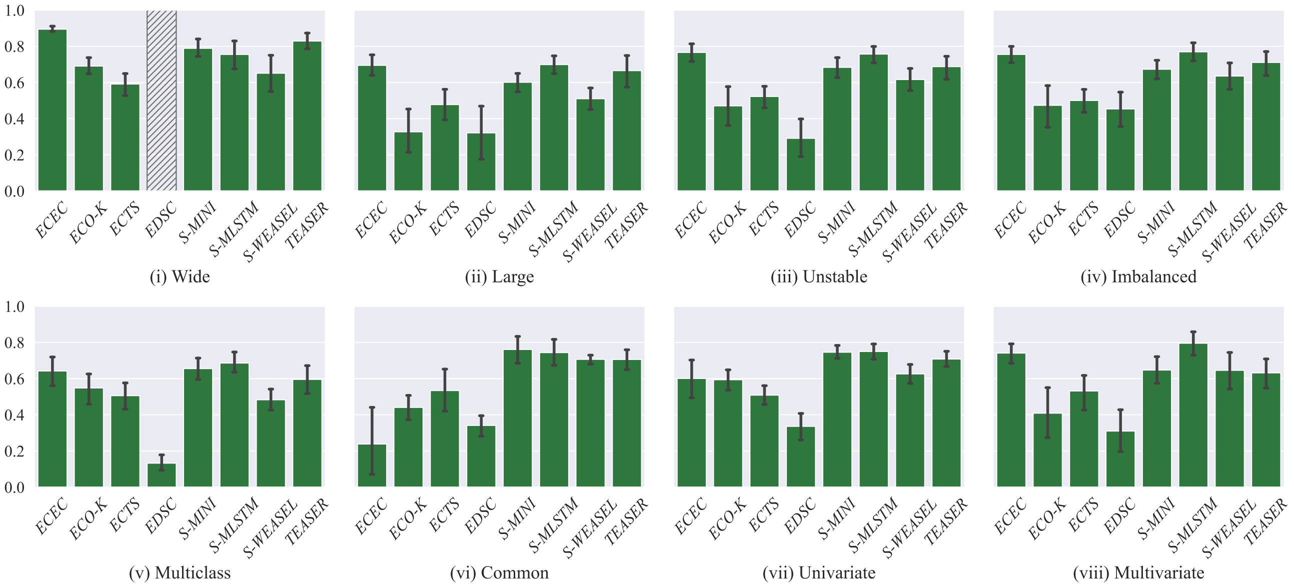

Figure 11: Harmonic Mean of earliness and accuracy. *EDSC* did not produce results for 'Wide' datasets within 48 hours.

boost the overall performance. There is though an exception for the 'Common' datasets where *TEASER* and *S-WEASEL* are pretty close, while *TEASER* is more accurate, *S-WEASEL* performs better in terms of earliness. *S-MINI* seems to typically be more performing than *S-WEASEL*, for all datasets.

ECEC is shown to be the method that is mostly impacted by dataset characteristics, achieving the best score in the 'Unstable', the second best in 'Imbalanced' and 'Multivariate' cases, reaching top-3 in 'Large', and 'Multiclass', but ending up to be the worst for the 'Common' datasets. The significant harmonic mean drop is not due to accuracy, since *ECEC* ranks on top positions for most datasets for this metric, but it is due to bad earliness scores, especially for the DodgerLoop{Game, Day} and PowerCons cases where it surpasses a value of 0.5. The reason for this is the high similarity between the instances of the different classes in the early time points. Consequently, this fact leads *ECEC* to train strict thresholds, thus not allowing their surpassing early enough. On the other hand, *EDSC*, *ECTS* and *ECO-K* rank in the last positions in most categories, excluding 'Common' for *ECO-K*, where *ECEC* ranks last.

6.2.4 Training Times.

We now compare the training times of the evaluated algorithms, i.e. the total time in minutes that an algorithm requires to fit a model, summing up individual iterations e.g. for the case of *STRUT* variants. Figure 12 shows a plot of our

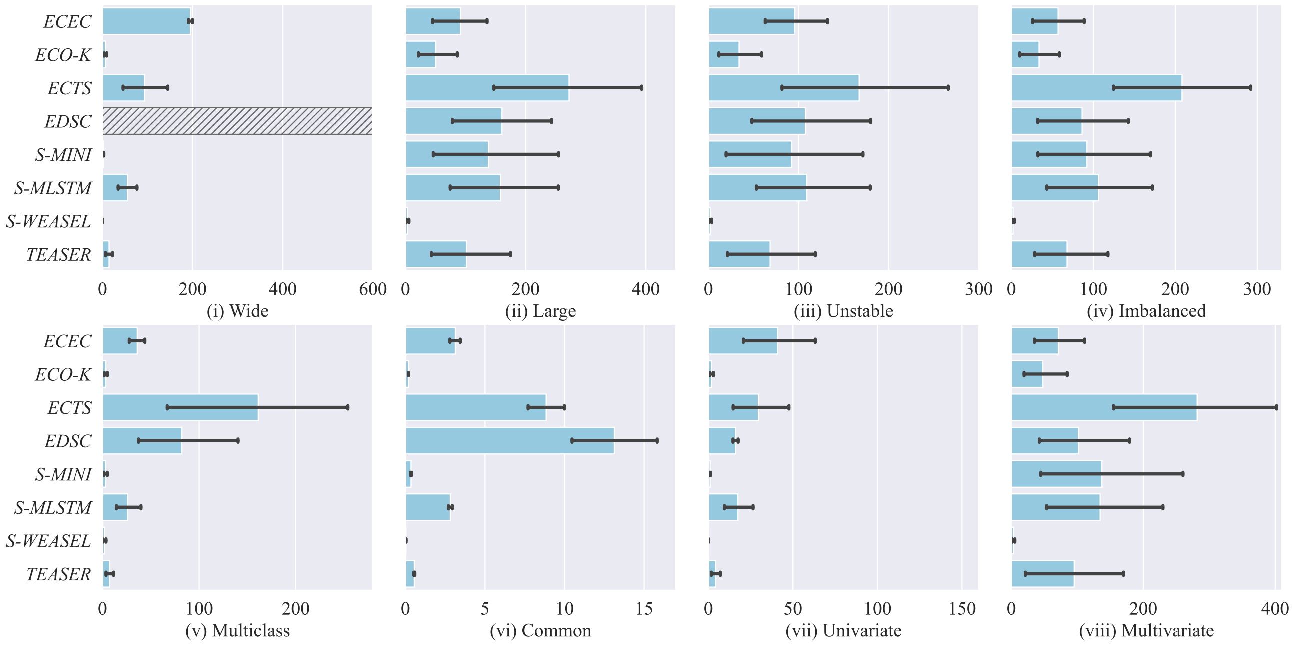

Figure 12: Training times in minutes (lower values are better, *EDSC* did not produce results for 'Wide' datasets within 48 hours).

Algorithm	Complexity
ECEC	$O(N \cdot L^3 \cdot$ # of classifiers· # of classes· # of variables)
ECO-K	$O(L \cdot logN + 2 \cdot N \cdot L +$ # of classes· # of typical groups $\cdot N \cdot$ # of variables)
ECTS	$O(N^3 \cdot L \cdot$ # of variables)
EDSC	$O(N^2 \cdot L^3 \cdot$ # of variables)
S-MINI	$O(N \cdot L \cdot log(L) \cdot$ # of kernels)
S-MLSTM	$O(N \cdot$ # of epochs $\cdot L)$
S-WEASEL	$O(N \cdot L^2 \cdot log(L) \cdot$ # of variables)
TEASER	$O(L/S \cdot L^2 \cdot$ # of variables)

Table 5: Worst-case complexity. N denotes dataset height and L, time-series length. For the univariate algorithms that incorporate voting, we multiply with the no. of variables.

experimental results and Table 5 describes the algorithm complexities. *S-WEASEL* has the lowest training times for all dataset categories, whereas *ECO-K* comes second apart from 'Multiclass', 'Wide' and 'Univariate', where the second best is *S-MINI*. Note that *S-WEASEL*'s worst-case complexity is worse than, e.g., that of *S-MINI* (see Table 5), but in our experiments the worst case was not encountered. The remaining algorithms behave as expected, i.e. *ECEC* suffers in the 'Wide' cases, though in terms of harmonic mean was the best for this dataset category, as a result of the cubic superscript on L; *ECO-K* is quite time-effective; the dataset's size (N) indeed impacts *ECTS* and *EDSC* training times; *TEASER* is time-efficient for datasets with large N and considering also its harmonic mean performance would be a good choice for 'Wide', 'Multiclass', 'Common' and 'Univariate' datasets; and the *STRUT* variants can scale for all dataset categories. Specifically, *S-MLSTM*, although being the most competitive in terms of harmonic mean, it ranks second or third worse with respect to training times, nevertheless managing to train on all 12 datasets. *TEASER* lies in the fourth position in most datasets, except for 'Unstable', for which it is the third fastest.

6.2.5 Testing Times. In order for an algorithm to operate in real-time, the generation of class label predictions must take place earlier than the arrival of new observations. To that end, Figure 13 presents the algorithms' testing times in seconds, divided by the frequency of observations in each dataset. Hatches indicate cases for which algorithms did not train. For *ECEC* and *TEASER* that analyze prefixes with length of more than one time-point, we divide by the frequency multiplied by the user defined prefix length. Values lower than one (blue) indicate that the algorithm is able to generate a prediction before the next time-point (or batch of time-points) arrives in the input. We can see that for datasets with higher frequency of observations (less than 1 second, such as BasicMotions, PickupGestureWiimoteZ, and PLAID), ETSC algorithms are not able to generate predictions in a timely manner, apart from *EDSC* that can generate predictions very fast, with an average of 0.003 seconds across all datasets and *ECEC* and *S-WEASEL* for the PickupGestureWiimoteZ dataset. Recall however, that in the case of PLAID, *EDSC* did not manage to train within a reasonable timeframe. In the HouseTwenty case, where observations arrive with a frequency of 8 seconds, all evaluated algorithms that managed to train apart from *ECTS* are suitable for online operation. Finally, *ECEC* cannot handle the Maritime dataset, as the algorithm's testing time is negatively impacted by the increased dataset height.

6.3 Results Overview

Towards general recommendations for the use ETSC algorithms, we summarize here the results of our evaluation.

For datasets with lengthy instances, the most early and accurate predictions are achieved by *ECEC*, though paying the price of increased training times. For applications that time is an issue,

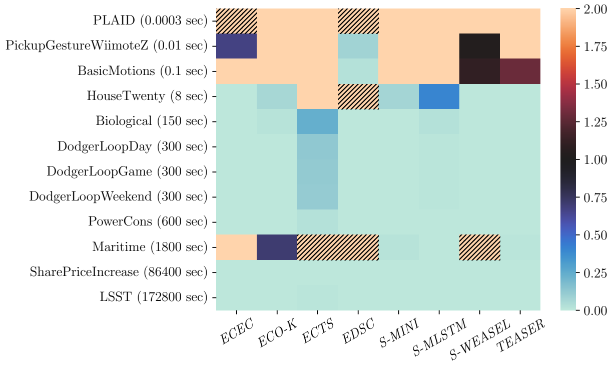

Figure 13: Heatmap of online algorithm performance. Values lower than 1 indicate feasible cases. Hatches indicate inability to train. Parentheses next to dataset names include the frequency of measurements for each case.

TEASER would be the best alternative. If the number of examples in the dataset increases, *S-MLSTM* competes with *ECEC* and *TEASER* in terms of harmonic mean, but is slower in training than both.

For cases where the measurements have high variance, *ECEC* and *S-MLSTM* achieve the best harmonic mean scores, with *ECEC* having higher accuracy, but *S-MLSTM* lower earliness scores. These two algorithms achieve the best harmonic mean scores also for cases of high class imbalance, though paying attention to their F$_1$-score for this category we can see that it is quite impacted.

In cases multiple classes exist, the best choice would be *S-MLSTM* with the lowest earliness score, followed this time by *S-MINI*, which achieves high accuracy with quite reduced training times.

For common datasets, *S-MINI* would be the most appropriate algorithm in terms of harmonic mean, with very low earliness scores and training times, though worse accuracy than *ECEC*, *ECO-K*, and *ECTS*. These three, however, suffer from high earliness scores.

For univariate datasets, *S-MLSTM* and *S-MINI* are shown to balance best the trade-off between accuracy and earliness. For applications with many variables *S-MLSTM* is the most competitive in terms of earliness, though with worse predictive accuracy than *ECEC*, which has the best accuracy for both univariate and multivariate categories. Also, *S-MLSTM* trains faster than *ECEC*, but slower than *S-MINI* for univariate cases. *ECEC* is the fastest among the three for multivariate datasets.

We now summarize the lessons learned from our study. First, not every algorithm is directly available for benchmarking, i.e. open-source and with sufficient configuration descriptions that allow for the reproduction of published empirical analyses. Second, we can confirm the published results, i.e. that *ECEC*, *ECO-K* and *TEASER* outperform *EDSC* and *ECTS*. *TEASER*'s performance, is quite close to the original paper with only 5% difference in mean accuracy and earliness, which is due to the normalization step that we skip.

Third, we have observed that implementation heterogeneity can induce compatibility issues and may require the development of additional modules for conversions; e.g., the Python libraries `sklearn` (used by *ECTS*) and `sktime` (used by *S-MINI*) require different input formats than `pyts` (used by *WEASEL*). Java implementations seem to exhibit better task parallelization, but some algorithms (e.g. *STRUT* and its variants) are more easily implemented in Python due to the existence of specialized libraries.

Overall, the use of ETSC algorithms can introduce tangible benefits in several applications: In resource-hungry biological simulations for drug treatment exploration, there is a need to determine as early as possible whether a simulation exhibits a case in which drug treatment is effective [3]. Our results have shown that ETSC allows the early identification of 65% of simulations that are not deemed interesting from a biological perspective, and thus helps save significant amounts of computational resources. In the maritime domain, port authorities need to determine as early as possible whether a vessel will reach a port in order to manage port traffic and operations [30]. Our evaluation has illustrated that the early classification of vessel trajectories is a challenging problem for ETSC algorithms and there is room for improvement in terms of scalability.

7 SUMMARY AND FUTURE WORK

In this work we introduced an open-source framework for evaluating ETSC algorithms. We empirically evaluated five state-of-the-art early-time-series classification algorithms on benchmark datasets, as well as three variants of a new ETSC algorithm. Moreover, we employed two datasets from the domains of cancer cell simulations and maritime situational awareness. We summarized the functionality of each algorithm using a simple running example in order to illustrate their operation. We divided the incorporated datasets to eight categories and observed that the shape of the dataset and the presence of multiple variables per time-series induce fluctuations in performance. The repository that we developed for the presented empirical comparison includes implementations of all algorithms, as well as all the datasets, and is publicly available allowing for experiment reproducibility. The repository may be extended with new implementations and datasets, thus facilitating further research. In the near future, more ETSC algorithms will be added, such as T-SMOTE [47], SDRE [9], and MOO-ETSC [29]. Moreover, we plan to incorporate to our framework hyper parameter tuning techniques as in [31], for optimizing the configurations and analyze the performance of alternative voting schemes for applying univariate algorithms on multivariate datasets. Moreover, we aim to re-implement all algorithms in the same language, as e.g. in [25].

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Horizon Europe Programme under the CREXDATA Project, grant agreement n° 101092749. We would also like to acknowledge the effort of making the UEA & UCR datasets openly available.

REFERENCES

[1] A. Abanda, U. Mori, and J. A. Lozano. 2019. A review on distance based time series classification. *Data Mining and Knowledge Discovery* 33, 2 (2019), 378–412.

[2] Y. Achenchabe, A. Bondu, A. Cornuéjols, and A. Dachraoui. 2021. Early classification of time series: Cost-based optimization criterion and algorithms. *Machine Learning* 110, 6 (2021), 1481–1504.

[3] C. Akasiadis, M. Ponce-de Leon, A. Montagud, E. Michelioudakis, A. Atsidakou, E. Alevizos, A. Artikis, A. Valencia, and G. Paliouras. 2022. Parallel model exploration for tumor treatment simulations. *Computational Intelligence* 38, 4 (2022), 1379–1401. https://doi.org/10.1111/coin.12515 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/coin.12515

[4] A. J. Bagnall, J. Lines, A. Bostrom, J. Large, and E. J. Keogh. 2017. The great time series classification bake off: a review and experimental evaluation of

recent algorithmic advances. *Data Mining and Knowledge Discovery* 31, 3 (2017), 606–660.

[5] A. Bondu, Y. Achenchabe, A. Bifet, F. Clérot, A. Cornuéjols, J. Gama, G. Hébrail, V. Lemaire, and P.-F. Marteau. 2022. Open challenges for machine learning based early decision-making research. *ACM SIGKDD Explorations Newsletter* 24, 2 (2022), 12–31.

[6] H. A. Dau, A. J. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. J. Keogh. 2019. The UCR time series archive. *IEEE CAA J. Autom. Sinica* 6, 6 (2019), 1293–1305.

[7] A. Dempster, D. F. Schmidt, and G. I. Webb. 2021. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 248–257.

[8] B. Dhariyal, T. L. Nguyen, S. Gsponer, and G. Ifrim. 2020. An Examination of the State-of-the-Art for Multivariate Time Series Classification. In *20th International Conference on Data Mining Workshops*. IEEE, New York, 243–250.

[9] A. F. Ebihara, T. Miyagawa, K. Sakurai, and H. Imaoka. 2023. Toward Asymptotic Optimality: Sequential Unsupervised Regression of Density Ratio for Early Classification. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1–5.

[10] R. Elmasri and S. B. Navathe. 2015. *Fundamentals of Database Systems* (7th ed.). Pearson, New Jersey.

[11] P. Esling and C. Agon. 2012. Time-series data mining. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 1–34.

[12] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. 2019. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery* 33, 4 (2019), 917–963.

[13] G. Fikioris, K. Patroumpas, A. Artikis, G. Paliouras, and M. Pitsikalis. 2020. Fine-tuned compressed representations of vessel trajectories. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2429–2436.

[14] A. Gupta, H. P. Gupta, B. Biswas, and T. Dutta. 2020. Approaches and Applications of Early Classification of Time Series: A Review. *IEEE Transactions on Artificial Intelligence* 1, 1 (2020), 47–61.

[15] G. He, Y. Duan, R. Peng, X. Jing, T. Qian, and L. Wang. 2015. Early classification on multivariate time series. *Neurocomputing* 149 (2015), 777–787.

[16] H. He and E. A. Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering* 21, 9 (2009), 1263–1284.

[17] J. K. Holand, E. K. Kemsley, and R. H. Wilson. 1998. Use of Fourier transform infrared spectroscopy and partial least squares regression for the detection of adulteration of strawberry purées. *Journal of the Science of Food and Agriculture* 76, 2 (1998), 263–269.

[18] J. Hu, L. Shen, and G. Sun. 2018. Squeeze-and-Excitation Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Washington DC, 7132–7141.

[19] H.-S. Huang, C.-L. Liu, and V. S. Tseng. 2018. Multivariate Time Series Early Classification Using Multi-Domain Deep Neural Network. In *5th IEEE International Conference on Data Science and Advanced Analytics*. IEEE, New York, 90–98.

[20] S. Ioffe and C. Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. of the 32nd Int. Conf. on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 448–456.

[21] A. A. Ismail, M. Gunady, Corrada B. H., and S. Feizi. 2020. Benchmarking deep learning interpretability in time series predictions. *Advances in neural information processing systems* 33 (2020), 6441–6452.

[22] S. H. Jambukia, V. K. Dabhi, and H. B. Prajapati. 2015. Classification of ECG signals using machine learning techniques: A survey. In *2015 International Conference on Advances in Computer Engineering and Applications*. 714–721.

[23] A. Javed, B. S. Lee, and D. M. Rizzo. 2020. A benchmark study on time series clustering. *Machine Learning with Applications* 1 (2020), 100001.

[24] F. Karim, S. Majumdar, H. Darabi, and S. Harford. 2019. Multivariate LSTM-FCNs for time series classification. *Neural Networks* 116 (2019), 237–245.

[25] M. Khayati, A. Lerner, Z. Tymchenko, and P. Cudré-Mauroux. 2020. Mind the gap: an experimental evaluation of imputation of missing values techniques in time series. In *Proceedings of the VLDB Endowment*, Vol. 13. 768–782.

[26] J. Lines, A. J. Bagnall, P. Caiger-Smith, and S. Anderson. 2011. Classification of Household Devices by Electricity Usage Profiles. In *12th International Conference on Intelligent Data Engineering and Automated Learning (LNCS, Vol. 6936)*. Springer, Switzerland, 403–412.

[27] J. Lv, X. Hu, L. Li, and P.-P. Li. 2019. An Effective Confidence-Based Early Classification of Time Series. *IEEE Access* 7 (2019), 96113–96124.

[28] U. Mori, A. Mendiburu, E. J. Keogh, and J. A. Lozano. 2017. Reliable early classification of time series based on discriminating the classes over time. *Data Mining and Knowledge Discovery* 31, 1 (2017), 233–263.

[29] U. Mori, A. Mendiburu, I.M. Miranda, and J.A. Lozano. 2019. Early classification of time series using multi-objective optimization techniques. *Information Sciences* 492 (2019), 204–218. https://doi.org/10.1016/j.ins.2019.04.024

[30] E. Ntoulias, E. Alevizos, A. Artikis, C. Akasiadis, and A. Koumparos. 2022. Online fleet monitoring with scalable event recognition and forecasting. *GeoInformatica* 26, 4 (2022), 613–644.

[31] G. Ottervanger, M. Baratchi, and H. H. Hoos. 2021. MultiETSC: automated machine learning for early time series classification. *Data Mining and Knowledge Discovery* (2021), 1–53.

[32] K. Patroumpas, D. Spirelis, E. Chondrodima, H. Georgiou, P. Petrou, P. Tampakis, S. Sideridis, N. Pelekis, and Y. Theodoridis. 2018. *Final dataset of Trajectory Synopses over AIS kinematic messages in Brest area (ver. 0.8)*. https://doi.org/10.5281/zenodo.2563256

[33] M. Pitsikalis, A. Artikis, R. Dreo, C. Ray, E. Camossi, and A. Jousselme. 2019. Composite Event Recognition for Maritime Monitoring. In *the 13th ACM International Conference on Distributed and Event-based Systems, DEBS 2019, Darmstadt, Germany, June 24-28*. ACM, 163–174.

[34] M. Ponce-de Leon, A. Montagud, C. Akasiadis, J. Schreiber, T. Ntiniakou, and A. Valencia. 2022. Optimizing dosage-specific treatments in a multi-scale model of a tumor growth. *Frontiers in Molecular Biosciences* 9 (2022). https://doi.org/10.3389/fmolb.2022.836794

[35] C. Ray, R. Dreo, E. Camossi, and A. L. Jousselme. 2018. Heterogeneous Integrated Dataset for Maritime Intelligence, Surveillance, and Reconnaissance, 10.5281/zenodo.1167595.

[36] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. J. Bagnall. 2021. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 35, 2 (2021), 401–449.

[37] P. Schäfer and U. Leser. 2017. Fast and accurate time series classification with WEASEL. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 637–646.

[38] P. Schäfer and U. Leser. 2017. Multivariate time series classification with WEASEL+ MUSE. *arXiv preprint arXiv:1711.11343* (2017).

[39] P. Schäfer and U. Leser. 2020. TEASER: early and accurate time series classification. *Data mining and knowledge discovery* 34, 5 (2020), 1336–1362.

[40] S. Shekhar, D. Eswaran, B. Hooi, J. Elmer, C. Faloutsos, and L. Akoglu. 2023. Benefit-aware early prediction of health outcomes on multivariate EEG time series. *Journal of Biomedical Informatics* 139 (2023), 104296. https://doi.org/10.1016/j.jbi.2023.104296

[41] R. Wu, A. Der, and E. Keogh. 2021. When is Early Classification of Time Series Meaningful. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[42] Z. Xing, J. Pei, and P. S. Yu. 2009. Early prediction on time series: A nearest neighbor approach. In *21^{st} Joint Conference on Artificial Intelligence*. Citeseer.

[43] Z. Xing, J. Pei, and P. S. Yu. 2012. Early classification on time series. *Knowledge and information systems* 31, 1 (2012), 105–127.

[44] Z. Xing, J. Pei, P. S. Yu, and K. Wang. 2011. Extracting interpretable features for early classification on time series. In *Proceedings of the 2011 SIAM international conference on data mining*. SIAM, 247–258.

[45] X. Xu, S. Huang, Y. Chen, K. Browny, I. Halilovicy, and W. Lu. 2014. TSAaaS: Time series analytics as a service on IoT. In *2014 IEEE International Conference on Web Services*. IEEE, 249–256.

[46] L. Yao, Y. Li, Y. Li, H. Zhang, M. Huai, J. Gao, and A. Zhang. 2019. DTEC: Distance transformation based early time series classification. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 486–494.

[47] P. Zhao, C. Luo, B. Qiao, L. Wang, S. Rajmohan, Q. Lin, and D. Zhang. 2022. T-SMOTE: temporal-oriented synthetic minority oversampling technique for imbalanced time series classification. In *Proceedings of International Joint Conference on Artificial Intelligence*.

Received 3 October 2023; revised 8 January 2024; accepted 1 February 2024