Universiteit Leiden
The Netherlands

Master Computer Science

MultiETSC: Automated Machine Learning for Early Time Series Classification

Gilles Ottervanger
s1309773

Supervisors:
Can Wang
Dr. Mitra Baratchi
Prof. dr. Holger H. Hoos

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

2021/01/29

Abstract

Early time series classification (EarlyTSC) involves the prediction of a class label based on partial observation of a given time series. In time-critical applications where data are observed over time, valuable time can often be saved at the cost of minor decreases in classification accuracy. Since accuracy and earliness are competing objectives, EarlyTSC algorithms must address this trade-off by deciding when enough data has been observed to produce a sufficiently reliable early classification, or simply: when to trigger the classification mechanism. Many algorithms have been proposed for this problem, using varying strategies for the classification and triggering mechanisms. Finding an optimal model or algorithm along with hyper-parameter settings for a given machine learning task has been the focus of a fast-moving research area known as automated machine learning (AutoML). In this thesis, we propose MultiETSC, an AutoML approach to EarlyTSC. This poses the challenge of optimising two conflicting objectives. We introduce an approach we dub multi-objective combined algorithm selection and hyper-parameter optimisation (MO-CASH). We compared our approach to hyper-parameter optimisation (HPO) on individual EarlyTSC algorithms based on their dominated hypervolume on 115 real-world and synthetic data sets from the UCR Time Series Classification Archive. We show MultiETSC achieves higher hypervolume than what achieved by HPO on each of the algorithms individually in 43% of all cases, with the best single algorithm only achieving highest hypervolume 29% of the time. Additionally, we demonstrate that MultiETSC outperforms a conceptually simpler single-objective optimisation approach, MO achieving higher hypervolume than SO 85% of the time. Finally, we find that only the most recently proposed algorithms are able to outperform a naïve fixed-time nearest neighbour approach.

Contents

Acknowledgement

I would like to thank Dr Mitra Baratchi and Can Wang for their time investment, dedication and input throughout this project and providing me with the regular feedback and inspiration that allowed me to not only bring this project to a successful conclusion but also to develop myself as a researcher. I would like to extend my gratitude to Prof. dr. Holger H. Hoos for sharing his bright insights in the field of automated machine learning and for being closely involved in every part of this project; To him, and all members of the ADA research group, for providing an excellent environment for sharing ideas, having meaningful discussions and having some fun along the way.

Finally, I would like to express my gratitude to all friends and family that have supported me during the time of this project and that have helped me to stay motivated and joined me during the rare occasions not spent working.

Chapter 1

Introduction

Classification is a long-standing, prominent problem in time series analysis. The goal of time series classification (TSC) is to assign a class label to a given time series: a sequence of observations that have been sampled over time. Practical applications include the diagnosis of heart conditions from ECGs, identification of patterns in financial markets, and detection of anomalies in seismic activity. Many time-critical applications can benefit from classification results being available as early as possible, preferably even before the full time series has been observed. This is clearly the case for detecting high-risk phenomena, such as earthquakes and market crashes, far in advance; further applications can be found in many other areas, including medicine, finance, traffic, engineering, and even sports. As an example, cardiac surgical patients in postoperative care are monitored for postoperative complications during an extended period of time. For some of these complications, indications of increased risk can be found far in advance of actual onset (Abdelghani et al., 2016). Being able to automatically detect these signals as soon as they occur, through a timely classification of the monitored time-series, can mean the difference between life and death.

Early time series classification (EarlyTSC) addresses the problem of classifying time series based on partial observations while maintaining a reasonable level of accuracy. The problem has been first described by Rodríguez Diez and Alonso González (2002) and has received an increasing amount of attention since. EarlyTSC introduces a second criterion to the classification problem: classifications do not only need to be accurate but also early. This results in a natural trade-off between accuracy and earliness (Mori et al., 2019). Most algorithms, proposed so far for EarlyTSC, provide some control over the earliness to accuracy trade-off by one or more hyper-parameters. Each hyper-parameter setting will occupy a different point in the earliness-accuracy space, even for a single algorithm, making it difficult to say how well any EarlyTSC algorithm performs. To make any progress in EarlyTSC, we must have a way of measuring an algorithm's performance and be able to objectively compare two algorithms without assuming relative preference for either earliness or accuracy. Thus far, there has been no such method of comparison, and no single algorithm can be designated as the best.

Recognising the diversity in both time series data (Dau et al., 2018) and early time series classification approaches (e.g., Parrish et al., 2013; Schäfer and Leser, 2020), a more reasonable goal would be to find what works best for any specific data set at hand, by automatically evaluating different algorithm configurations (i.e., algorithm choice and its hyper-parameters settings) aiming to optimise a clear objective function. Such an automated approach would be a form of automated machine learning (AutoML). Recent developments in the AutoML field of research has made collections of advanced machine learning tools easily accessible for non-experts (e.g., Thornton et al., 2013; Feurer et al., 2015; Koch et al., 2018), such that the user does not have to deal with difficult choices (e.g., the choice of algorithm, or hyper-parameter settings). However, so far, no such automated approach to EarlyTSC has been proposed. This is mainly due to the complex objective of this specific machine learning task.

Hutter et al. (2011) proposed the Sequential Model-based Algorithm Configuration (SMAC) method and showed its viability for the configuration of SAT and MIP solvers optimising for running time. Thornton et al. (2013) introduced Auto-WEKA, one of the first fully-fledged AutoML systems. Auto-WEKA leverages SMAC to address the problem of combined algorithm selection and hyper-parameter optimisation (CASH) for the configuration of machine learning algorithms optimising for accuracy. To extend this conceptual approach to the problem of EarlyTSC, the challenge is to take both earliness and accuracy into account as two competing objectives. However, generalising the CASH problem to a multi-objective setting is non-trivial. A way around this challenge would be to combine both objectives into a single-objective. Within the context of EarlyTSC, using the harmonic mean of earliness and accuracy as an objective function has been proposed (Schäfer and Leser, 2020). However, this approach fails to capture the complex trade-off between earliness and accuracy. In this thesis, we address this issue by considering a multi-objective optimisation approach. More specifically, our contributions in this thesis are as follows:

- We define multi-objective CASH, or MO-CASH, problem and propose MultiETSC, an approach addressing this problem within the context of EarlyTSC. This approach produces a set of algorithm configurations that are expected to produce classifiers that are non-dominated in earliness-accuracy space (i.e., no classifiers are both earlier and more accurate).
- We compiled a set of 9 EarlyTSC algorithms that in combination with their possible hyper-parameter settings form the search space of our approach and demonstrated the concept of AutoML for EarlyTSC on this search space using 115 data sets from the UCR archive (Dau et al., 2018). We evaluated our approach using the Hypervolume and Δ-spread which are well-known indicators for estimating the performance of multi-objective optimisation approaches.
- We compared our approach addressing MO-CASH for EarlyTSC with the hyper-parameter optimisation (HPO) of every individual EarlyTSC algorithm. This resulted in the MO-CASH-based approach achieving the best performance in 43% of all cases, whereas the best single algorithm approach only achieved the best performance in 29% of the cases in terms of hypervolume.
- We also compared our approach with a simplified single-objective approach based on the harmonic mean of earliness and accuracy as a baseline. MultiETSC achieves higher hypervolume in 84% of all cases. Note that all compared approaches constitute different forms of automated machine learning for EarlyTSC with differing levels of complexity.

An added benefit of our automated approach is that, instead of producing a single classifier, MultiETSC can map out the full trade-off between earliness and accuracy. This means the user can make an informed decision on what point along the trade-off to pick based on the relative cost between earliness and accuracy. This is a luxury most EarlyTSC algorithms cannot offer.

The remainder of this thesis is structured as follows: Chapter 2 covers several fundamental definitions and formalises the MO-CASH problem for early time series classification. Chapter 3 covers related work on TSC, EarlyTSC and AutoML. In Chapter 4, we describe MultiETSC and cover its implementation details. The experimental evaluation of our approach, including results, is described in Chapter 5. Finally, in Chapter 6, we draw some general conclusions and discuss directions for future work.

Chapter 2

Problem Statement

Chapter 2 introduces terminology and definitions to finally provide a formal definition of the problem that we address in this work. First we formalise EarlyTSC and introduce the notation used for the problem of algorithm configuration for a single optimization objective. Then, we extend the latter to multiple objectives and formulate this problem for the specific case of EarlyTSC.

2.1 Preliminaries

A *time series* is a series of discrete observations over time. Time series can have varying sampling rates, but for the sake of simplicity, we consider only real-valued time series with a constant sampling rate. Note that for practical applications, the data could be transformed to fit this assumption. We denote a time series of l observations as $\mathbf{x} = [x_1, \dots, x_l] \in \mathbb{R}^l$.

Time Series Classification (TSC) is the problem of determining a function $f : \mathbb{R}^l \rightarrow \mathcal{C}$ that maps a given time series $\mathbf{x} \in \mathbb{R}^l$ to a class label $f(\mathbf{x}) = y \in \mathcal{C}$, where $\mathcal{C}$ is a finite set of labels. Function f is obtained from a learning algorithm A based on a set of training examples $\{d_1, ..., d_n\}$, where each example is a pair of a time series and a class label $d_i = (\mathbf{x}_i, y_i) \in \mathbb{R}^l \times \mathcal{C}$. Note that we assume time series of uniform lengths. Additionally, most learning algorithms expose a set of hyper-parameters λ that control some aspects of their inner workings; settings for these need to be chosen from a space $\mathbf{\Lambda}$ and often have a substantial impact on classification accuracy.

Early Time Series Classification (EarlyTSC) has been first mentioned in the literature by Rodríguez Diez and Alonso González (2002). The main difference to ordinary TSC occurs when performing the actual classification task on a given time series. While ordinary TSC assumes receiving the time series $\mathbf{x}$ as a single object, EarlyTSC considers multiple prefixes $\mathbf{x}_p = [x_1, \dots, x_p] \in \mathbb{R}^p$ of $\mathbf{x}$ of increasing lengths ($p \leq l$). At each prefix length considered, the classifier either classifies or postpones classification to await more data. When the full time series has been observed, the classifier must produce a class output.

Hyper-parameter optimisation (HPO) is the problem of finding the set of hyper-parameter settings $\lambda^* \in \mathbf{\Lambda}$ with optimal generalisation performance for a given algorithm A_λ with hyper-parameters λ and a set of training data $\mathcal{D}$. Generalisation performance can be estimated by repeatedly splitting $\mathcal{D}$ into non-overlapping training and validation subsets of $\mathcal{D}$, $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$ respectively, training on $\mathcal{D}_{train}$ and evaluating performance of the resulting classifier on $\mathcal{D}_{valid}$. Formally, hyper-parameter optimisation involves determining

$$\lambda^* \in \underset{\lambda \in \mathbf{\Lambda}}{\operatorname{argmin}} \, \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid}), \tag{2.1}$$

where $\mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ is the loss of algorithm A with hyper-parameters λ when trained on $\mathcal{D}_{train}$ and evaluated on $\mathcal{D}_{valid}$. For a classification problem such as TSC, this loss usually is a measure of prediction accuracy, such as misclassification rate, but any performance metric could be chosen as an optimisation target. It is important to note that the combined hyper-parameter space is a subset of the product of the permissible set of values for each individual parameter $\mathbf{\Lambda} \subset \Lambda_1 \times \cdots \times \Lambda_m$. In most cases, this subset is strict, since some hyper-parameters depend on the value of others, resulting in a tree-structured space (Bengio, 2000).

When there are multiple learning algorithms to choose from, we might want to not only optimise hyper-parameter settings, but simultaneously select the best learning algorithm for a given data set. This problem is called *combined algorithm selection and hyper-parameter optimisation or CASH* (Thornton et al., 2013). Given a set of algorithms $\mathcal{A}$, for each algorithm $A^{(j)} \in \mathcal{A}$, and a hyper-parameter space $\mathbf{\Lambda}^{(j)}$, the goal is to optimise generalisation performance, i.e., to determine

$$A^*_{\lambda^*} \in \operatorname*{argmin}_{A^{(j)} \in \mathcal{A}, \lambda \in \mathbf{\Lambda}^{(j)}} \mathcal{L}(A^{(j)}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid}) \tag{2.2}$$

In their description of the CASH problem, Thornton et al. (2013) note that the choice of algorithm can be considered a top level hyper-parameter λ_r that selects an algorithm from $A^{(1)}, ..., A^{(k)}$. Thereby, the CASH problem can be reformulated as an HPO problem over the combined hyper-parameter space $\mathbf{\Lambda} = \mathbf{\Lambda}^{(1)} \cup \cdots \cup \mathbf{\Lambda}^{(k)} \cup \{\lambda_r\}$, where each algorithm $A^{(i)}$ has its own subspace $\mathbf{\Lambda}^{(i)}$ that is conditional on λ_r being set to $A^{(i)}$. This places λ_r at the root of the tree-structured search space.

2.2 Problem Definition

So far, we have presented the single-objective CASH problem. To formulate the CASH problem for EarlyTSC, both accuracy and earliness objectives need to be considered. To accommodate that, we will introduce MO-CASH, the multi-objective extension to the CASH problem, and formulate this problem for the specific case of EarlyTSC. In Equations 2.1 and 2.2, we assumed a one-dimensional loss function defining a total order on the configuration space. When generalising to multi-objective optimisation, we can no longer speak of a total order. Let $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(2)}$ be vectors in objective space $\mathbb{R}^m$ with m objectives. We say $\mathbf{y}^{(1)}$ is dominated by $\mathbf{y}^{(2)}$ if the following two conditions are met: 1) $y^{(2)}_i \leq y^{(1)}_i$ for all $i \in 1, \ldots, m$ and 2) $y^{(2)}_i < y^{(1)}_i$ for at least one $i \in 1, \ldots, m$. We denote this relation with $\mathbf{y}^{(2)} \prec \mathbf{y}^{(1)}$. The domination relation is a partial order, meaning that it leaves some vectors incomparable. In MO-CASH, we are interested in the *efficient set* of configurations, i.e., a set that consists solely of non-dominated, or Pareto-optimal, configurations. Since CASH can be formalised as a special case of HPO (as discussed in Section 2.1), we will be using the simpler notation of HPO. Let $\mathcal{L}$ be a vector-valued loss function on $\mathbb{R}^m$; then the efficient set $\mathbf{\Lambda}^*$ and the Pareto front $\mathcal{P}$ can be formalised as follows:

$$\mathbf{\Lambda}^* = \{\lambda^* \in \Lambda \mid \nexists \lambda \in \Lambda : \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid}) \prec \mathcal{L}(A_{\lambda^*}, \mathcal{D}_{train}, \mathcal{D}_{valid})\} \tag{2.3}$$

$$\mathcal{P} = \{\mathcal{L}(A_{\lambda^*}, \mathcal{D}_{train}, \mathcal{D}_{valid}) \in \mathbb{R}^m \mid \lambda^* \in \mathbf{\Lambda}^*\} \tag{2.4}$$

Note that the single-objective formulation from Equation 2.1 can be considered as a special case of a more general multi-objective problem. In the single-objective case, the efficient set is guaranteed to contain only a single configuration, since the ordering in a 1-dimensional solution space is guaranteed to be a total order. In the multi-objective case, on the other hand, there can be an arbitrarily large number of non-dominated configurations, and an optimiser needs to find improvement over a range of trade-off points in the objective space.

The problem we address in this work is the MO-CASH problem for the specific case of EarlyTSC, which can be defined as follows. Given the set $\mathcal{D} = \{d_1, ..., d_n\}$ of pairs of time series and class label

$d_i = (\mathbf{x}_i, y_i) \in \mathbb{R}^l \times \mathcal{C}$, and given the combined space of EarlyTSC algorithms and their hyper-parameter settings, in the form of a configuration space $\mathbf{\Lambda}$, find the best set of non-dominated configurations $\mathbf{\Lambda}^*$ in terms of earliness and accuracy.

Chapter 3

Related Work

In this chapter, we discuss work from the literature that has been fundamental to the algorithms we used as the basis of MultiETSC. Since all EarlyTSC methods build on concepts developed for non-early time series classification, we start with an overview of TSC methods. This is followed by a section dedicated to EarlyTSC methods. Finally, we cover the foundations of AutoML.

3.1 TSC Methods

In the past two decades, many TSC algorithms have been proposed. Bagnall et al. (2016) have provided an overview of influential methods for the ordinary TSC problem. There are several possible approaches to TSC. These can roughly be split up into four categories: Nearest Neighbour, Shapelets, Dictionary-based, and Feature-Based. We will discuss these in more detail.

Nearest Neighbour One of the most common methods is the 1-nearest neighbour (1NN) approach, where a time series is classified based on the class of the nearest neighbour in the training set. The distance between time series can simply be the *Euclidean Distance* (ED) in the $\mathbb{R}^n$. Alternative distance measures have been proposed to exploit the temporal relation of the observations. A common method is the *Dynamic Time Warping* distance (DTW), which allows for local shifts in the time domain (Bagnall et al., 2016). This way the DTW puts time series with shifts of phase or with temporal scaling closer together than they would be using ED. Chen et al. (2005) proposed Edit Distance on Real sequences, or EDR, a distance metric based on the number of editing operations (i.e., insert, delete, replace) that need to be executed to transform one time series into the other. Finally, there are methods that are completely agnostic about the exact time dimension and completely focus on the frequency domain. For example, the Fourier distance metric, proposed by Agrawal et al. (1993), computes the Euclidean distance between the first n Fourier coefficients.

Shapelets are subsequences of time series. They commonly represent some “shape” or temporal feature of a time series. Shapelets can be used in different ways. By setting a threshold value for the minimal distance, a shapelet can be used as a binary feature which can be present or absent (Ye and Keogh, 2011). Presence of a certain shapelet can be an indication of class membership. This can be used to construct a decision tree as done for example by Rakthanmanon and Keogh (2013). In shapelet transform (Lines et al., 2012; Hills et al., 2013), a set of k shapelets is extracted and the minimum distance between a time series and a shapelet is used as a feature. This results in a k dimensional feature vector for each time series which can be used as input for a more conventional classification algorithm. A challenge in shapelet based methods is to quickly identify interesting shapelets since the space of all possible shapelets is large and difficult to search efficiently. Therefore, most recent developments on shapelet methods have been on shapelet discovery (Grabocka et al., 2014; Renard et al., 2015; Grabocka et al., 2016; Zhao et al., 2019).

Dictionary-Based Methods build a dictionary from a time series using a sliding window extracting a word at every window position. A word is build up by discretising the time series at the window. Once the dictionary is built, the histogram of all words is used as a feature vector input for any classifier of choice. The main difference between different dictionary-based methods is the method of discretization (Bagnall et al., 2016). The Bag of Patterns (BOP) classifier (Lin et al., 2012) was one of the first dictionary-based methods proposed, using Symbolic Aggregate Approximation, or SAX (Lin et al., 2007), to turn time series windows into words. The Bag of SFA Symbols, or BOSS (Schäfer, 2015), is similar to BOP, but using a different discretization method based on the truncated Discrete Fourier Transform. A more recent development in dictionary-based methods came in the form of the WEASEL algorithm developed by Schäfer and Leser (2017) which can do accurate classification while at the same time being very fast in terms of training time.

Feature-Based Methods extract a limited set of features from each time series as a form of dimensionality reduction (Wang et al., 2006). These features can for example be the mean, standard deviation, maximum and minimum. The feature vectors are then used as the input of conventional classifiers. The time series forest (TSF) algorithm (Deng et al., 2013) for example builds a random forest, based on a set of predefined features. Initial forms of feature-based methods used a limited set of theorised features. With the number of features increasing or even automatically generated features, an additional feature selection step was introduced to select only relevant and informative features and keep the dimensionality low (Fulcher and Jones, 2014). Most feature-based methods compute features based on intervals of the full time series, resulting in a more efficient feature extraction but increasing the space of possible features. For example, the Time Series Bag of Features algorithm is based on relatively simple features like slope, mean and variance on a large set of random intervals from which only the most informative are used (Baydogan et al., 2013). Recently, software packages have been developed specifically for the extraction and selection of features from time series data, e.g., FRESH (Christ et al., 2017) and tsfresh (Christ et al., 2018)

It shows from the number and variety in TSC methods that time series classification is not a trivial problem to solve. Different aspects of the training data like the number of training examples, number of classes, length of the time series and even the nature of the data itself can make one method more appropriate than the other. Even for the non-early TSC problem, it would seem reasonable to try different methods, preferably in a systematic way, to see what works best in any specific instance. We will show that when considering early classification, the space of methods is similarly large, indicating the need for automatic algorithm selection.

3.2 EarlyTSC Methods

Many methods for solving the EarlyTSC problem have been proposed over the past two decades. Generally, these are adaptions of classification methods for full time series. Most methods split the problem into two parts: one that addresses the classification of the partial data, aiming to maximise the classification accuracy; and a separate part that manages the trade-off between earliness and accuracy, by deciding whether enough data has been evaluated to base a reliable classification on. We will call this decision *triggering* and the function that controls it the *trigger function.* Next, we briefly discuss the existing literature on EarlyTSC algorithms.

Rodríguez Diez and Alonso González (2002) were the first to address the classification of time series based on partially observed data. They used a set of simple base classifiers based on interval-based binary features which they call literals. The following is an example for such a literal: "the difference between point t_i and t_j is more than v". The given set of base classifiers is then combined by boosting using ADABoost (Freund and Schapire, 1999). Each literal gets a weight value for each class. To classify, the weights of the literals that evaluate to 'true' are summed, and the class with the highest weight sum is the predicted class. To do early classification, the literals based on data that has not been observed are simply ignored. Although this is the first study to mention early time series classification, this contribution was only a serendipitous side effect, since the method by Rodríguez Diez and Alonso González (2002) does not actually make an explicit decision when to classify, and consequently does not solve the EarlyTSC problem as we have defined it.

Xing et al. (2011b) were the first to explicitly address the problem of EarlyTSC by looking for a favourable trade-off between earliness and accuracy. The authors proposed **ECTS**, a method based on 1NN ED classification using the observed prefix of the time series. In a training phase, the *minimum prediction length* (MPL) of each time series is learned – the length at which the prediction based on the time series prefix is likely to be equal to the prediction on the full time series. When a partial time series is classified, and the MPL of the nearest neighbour is less than or equal to the observed prefix length, the classification is triggered.

EDSC (Xing et al., 2011a) is a shapelet-based method where the shapelets learned in a first training phase. In the second phase of training, shapelets are selected based on the earliness of appearance in most time series, ensuring early classification. In this selection phase, the trade-off between accuracy and earliness is explicitly controlled by a single parameter. A benefit of classification with shapelets is the interpretability and more efficient, possibly even more accurate, classification compared to distance metrics over full time series. However, this comes at the cost of a computationally expensive shapelet extraction phase.

RelClass (Parrish et al., 2013) explicitly estimates classification reliability (i.e., the probability of the early class prediction being equal to the classification of the complete time series). It computes the posterior distribution conditional on the observed and training data and applies linear or quadratic discriminant analysis (LDA or QDA, respectively). This approach leads to constant classification and reliability output, which gives this method its name. The trigger mechanism is simply a minimum reliability threshold that needs to be met. This enables the trade-off between earliness and accuracy to be made at classification time rather than during training.

Hatami and Chira (2013) proposed a method that based the triggering on the “agreement” among different classifiers in an ensemble. When the individual classifiers do not agree, the classification is rejected, and the method waits for more data. Antonucci et al. (2015) proposed a method based on “imprecise hidden Markov models”, where a Markov model is fitted to the incoming data with some uncertainty. The classification is done if only a single model of a time series in the training set remains within the uncertainty bounds. Dachraoui et al. (2015) suggest a meta-algorithm that considers both the cost of classification quality and the cost of delaying the classification decision. Additionally, this method predicts in advance how much data will be needed to make a decision, and only triggers when the observed amount reaches or exceeds the required amount.

ECDIRE, proposed by Mori et al. (2016), is based on a set of prefix classifiers, each prefix classifier being a fully-fledged time series classifier trained on a specific prefix length. Figure 3.1 illustrates this idea. In ECDIRE, each prefix classifier is a probabilistic time series classifier providing a probability distribution over the class labels. A learned threshold value of class probability is used as a triggering mechanism.

SR-CF (Mori et al., 2018) extends the ideas of ECDIRE by making the trade-off between earliness and accuracy more explicit. It uses a stopping rule (SR) based on posterior class probabilities; this rule is trained to minimise a cost function (CF) based on accuracy and earliness. Recently, Mori et al. (2019) suggested a multi-objective variant of SR-CF, by directly optimising the stopping rule using multi-objective optimisation. However, this method is difficult to compare to earlier EarlyTSC algorithms, because rather than constructing a single classifier, it produces a set of classifiers in a single run.

ECEC (Lv et al., 2019) combines the ideas of RelClass and ECDIRE. A set of prefix classifiers is trained where the reliability score (earlier defined in RelClass) of these classifiers is learned from the training data. The reliability of the subsequently evaluated classifiers can be combined to get

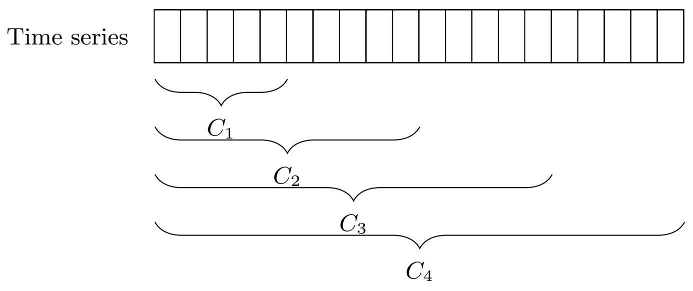


Figure 3.1: Four prefix classifiers and the prefix they are trained on visualised for a time series of length 20.

an increasingly improving reliability estimate. As in RelClass, triggering is done using a reliability threshold, but Lv et al. (2019) provide a mechanism to learn a threshold that favourably trades off accuracy for earliness, based on a single parameter.

TEASER (Schäfer and Leser, 2020) is a two-tier method that explicitly separates the task of classifying and triggering. A set of 'slave' prefix classifiers is trained, and their output is sent to a set of 'master' classifiers (one for each prefix classifier) that decide whether to trigger or not. TEASER can be used with any TS classifier that provides class probabilities, while the master classifier is a one-class SVM.

EARLIEST (Hartvigsen et al., 2019) is based on a recurrent neural network (RNN) with LSTM cells. The base RNN produces a vector representation at each time step. This vector representation is used by two classifiers, one for classification and one binary classifier for triggering. The system is trained as a whole, minimising a loss function combining earliness and accuracy.

As is apparent from this overview, a diverse set of EarlyTSC algorithms can be found in the literature, each with its strengths and weaknesses. To the best of our knowledge, we are the first to attempt to combine the strengths of all these algorithms into a single, integrated system for early time series classification. Additionally, while all EarlyTSC algorithms manage the trade-off between earliness and accuracy in some way, they do not provide insight into this trade-off. Our automated approach, described in the following, can produce this insight, without the need to understand the details of the underlying EarlyTSC algorithms, making EarlyTSC more accessible for non-experts.

3.3 Automated Machine Learning

The application of machine learning to a specific problem often encompasses decisions about data pre-processing, choice of algorithm and hyper-parameter settings. Automated machine learning attempts to automate these decisions. Hutter et al. (2009) addressed the hyper-parameter optimisation problem using sequential model-based optimisation (SMBO). Hutter et al. (2011) used SMBO as the basis for a general-purpose algorithm configuration procedure, SMAC (Hutter et al., 2011), which enables the efficient search of large and complex configuration spaces with categorical and numerical parameters.

Bergstra et al. (2011) applied two forms of SMBO, Gaussian process (GP) regression and the so-called Tree-structured Parzen Estimator (TPE), to HPO in deep belief networks. For this 32-dimensional configuration space, they achieved better results within 24 hours of computing time than had been achieved by manual configuration in earlier work. Snoek et al. (2012) built further on SMBO for AutoML by proposing Spearmint, an algorithm that takes the variable cost, in terms of training time, into account. Thornton et al. (2013) introduced Auto-WEKA, a software package based on SMAC that makes AutoML available for end-users familiar with the WEKA interface, being the first AutoML system addressing the full CASH problem. Feurer et al. (2015) introduced AUTO-SKLEARN, a SMAC-based AutoML system for Python. Additionally, AUTO-SKLEARN supports a meta-learning step before the SMBO phase and an ensemble building phase after optimisation, improving the efficiency of the configuration process and the quality of the results thus obtained.

Olson et al. (2016) introduced a Tree-based Pipeline Optimisation Tool, or TPOT, which optimises classification pipelines using decision trees and random forests. TPOT uses a tree representation for classification pipelines including feature selection, transformation and construction operators, as well as model selection and parameter optimisation elements. These pipelines are optimised using a Genetic Algorithm (GA). Olson et al. (2016) introduced an extension of TPOT, called TPOT-Pareto, which not only considers classification accuracy but also pipeline complexity (i.e., number of pipeline operators). During optimisation, not just the best k performing pipelines are kept as the population for the GA, but a Pareto front of non-dominated pipelines (in terms of accuracy and complexity) is used. However, the selection of the final pipeline is still based solely on accuracy. Therefore, TPOT does not address the more general MO-CASH problem.

Koch et al. (2018) developed the Autotune framework for the proprietary statistical software package SAS. Autotune uses a hybrid search strategy consisting of random search, Latin Hypercube Sampling (LHS), global and local search, GA and Bayesian optimisation using a GP surrogate. Autotune is implemented to maximally exploit parallel computation. The authors show competitive performance compared to only Bayesian optimisation and the Spearmint package. Gardner et al. (2019) extended the work on Autotune to address multi-objective optimisation. They reduced their hybrid search to only employ LHS, GA and Generating Set Search, a local search strategy. In their study, the authors address common conflicting objectives for binary classification, e.g., false-negative rate *vs* misclassification rate.

Jin et al. (2019) addressed the problem of Neural Architecture Search (NAS) by developing the open-source Keras-based system: Auto-Keras. NAS is an interesting special case of AutoML since the search space of possible architectures is complex and highly hierarchical. Auto-Keras employs a custom GP kernel for SMBO, based on the edit-distance of the neural network architecture. The downside of Auto-Keras is that it only takes into account a single loss metric, without penalising architecture complexity.

All these AutoML systems are built upon existing machine learning packages and aim to provide easier access to advanced machine learning pipelines and algorithms to end-users. For EarlyTSC, there does not yet exist a software package that allows for such a direct extension. This introduced the additional challenge of integrating all algorithms into a common framework with all required hyper-parameters exposed. As with existing AutoML implementations, MultiETSC aims to benefit end-users by making machine learning more accessible. Additionally, MultiETSC could also help in the development of new EarlyTSC algorithms since it provides a framework for a fair comparison of performance.

Chapter 4

MultiETSC

In this chapter, we describe the approach developed for automatically configuring EarlyTSC algorithms and the system that implements our approach. At the core of our approach, we make use of a general-purpose automated algorithm configurator. It is the task of the configurator to efficiently search for the best performing configurations by searching a vast space of EarlyTSC algorithms and their hyper-parameters. The inner loop of the search process consists of three steps: 1) selecting a candidate configuration (i.e., a combination of algorithm and its hyper-parameters); 2) training the configuration on training data; 3) evaluating the configuration on validation data. The evaluation is fed back into the configurator enabling informed decisions for selecting new configurations. Thus the final output obtained is a set of configurations that are mutually non-dominated, based on evaluation on the given validation data. The overall framework of our proposed approach is illustrated in Figure 4.1. In the remainder of this section, we will first describe in detail the space of EarlyTSC algorithms and hyper-parameter settings that we search over. Second, we describe how configurations are evaluated. Third, we describe the algorithm configurator.

4.1 Configuration Space

As discussed in Section 2.1, the configuration space is a tree-structured space defined by the choices of algorithm and hyper-parameter settings. MultiETSC includes 9 EarlyTSC algorithms: ECTS, EDSC, RelClass, ECDIRE, SR-CF, TEASER, ECEC, EARLIEST (all described in Section 3.2) and a naïve fixed-time Euclidean 1NN algorithm, which we will refer to as 'Fixed' and is described in more detail below. Table 4.1 shows a concise overview of the included algorithms and their hyper-parameters.

These algorithms were chosen based on the fact that their implementations were made available by the

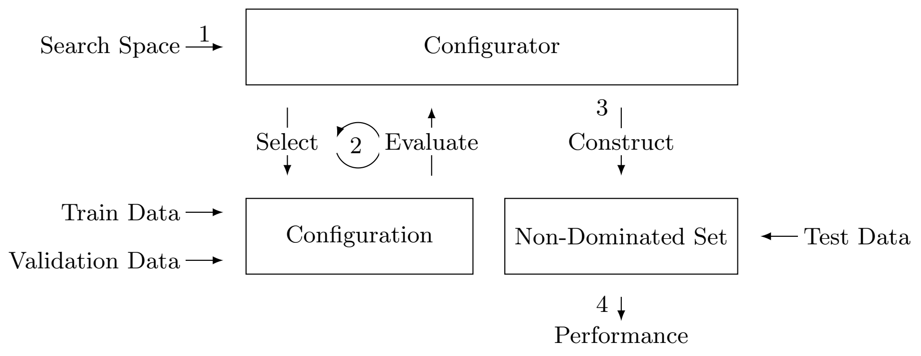


Figure 4.1: Design of our Automated Machine Learning system. Numbers indicate the order of steps with step 2 being the inner loop of the system.

original authors (except for Fixed, which we implemented), which helped us to ensure correctness and efficiency. Algorithm performance was not used as an inclusion criterion, meaning that we included all available algorithms, regardless of expected performance. The naïve Fixed method was added to counterbalance the quite complex EarlyTSC algorithms. However, in some cases, this decision limited the flexibility of the individual algorithms. Therefore, all algorithms were modified and wrapped to provide a common command-line interface, where test data, training data, hyper-parameters and, in case of a stochastic algorithm, a random seed can be passed to the algorithm for reproducibility. To achieve this, the original implementations required a varying degree of modifications. Next, we describe each included EaryTSC algorithm in more detail. In particular, we cover their hyper-parameters and implementation details.

- **ECTS** (Xing et al., 2011b) (`C++`): one of the first implementations of an EarlyTSC algorithm and still often used as a baseline in experimental evaluations of new algorithms. ECTS is a 1NN ED based algorithm that uses the observed partial time series, or prefix, for classification. In a training phase, the *minimum prediction length* (MPL) for clusters of time series is learned. The MPL is the length at which the prediction based on the time series prefix is likely to be equal to the 1NN prediction on the full time series. When a partial time series is classified, and the MPL of the nearest neighbour is less than or equal to the observed prefix length, the prediction is accepted. The *minimal support* hyper-parameter sets a lower bound to the number of time series required for a cluster to be considered and an MPL to be computed. This is done using a minimal fraction of time series per cluster which is real valued between zero and one, with zero (no lower bound) as default. This hyper-parameter is called minimal support. Time series that cannot be accurately classified using a 1NN ED classifier can have a considerable impact on the MPL as computed using this method. Therefore, the authors suggest a 'Relaxed' version that ignores the time series that cannot be accurately classified using the full length. Switching between these versions is achieved using the hyper-parameter "version" with the strict version as default.
- **EDSC** (Xing et al., 2011a) (`C++`): EDSC stands for Early Distinctive Shapelet Classification. As the name suggests, it is a shapelet based EarlyTSC algorithm introducing the idea of a local shapelet: A triplet of a time sub-series, a distance threshold and a target class. A local shapelet matches when the shortest Euclidean distance between the sub-series is below the threshold. Training starts with a shapelet extraction phase where for each shapelet a robust threshold is learned. In the second phase of training, shapelets are selected based on earliness, distinctiveness and frequency of matches. New time series can be classified by assigning the target class of the first matching shapelet. To limit the search for shapelets, a maximum and minimum shapelet length ($maxL$ and $minL$) are passed as hyper-parameters. A minimum recall threshold is set as hyper-parameter in order to filter out shapelets that are less distinctive, i.e., they show low specificity of matching series. A hyper-parameter α is used to trade off earliness versus accuracy. An additional threshold is learned to filter out shapelets that show poor earliness versus accuracy performance. The author's implementation allows for two methods for shapelet threshold learning which will be switched between using the binary hyper-parameter "method".
- **RelClass** (Parrish et al., 2013) (`MATLAB` modified for `GNU Octave`): this algorithm is based on an explicit estimation of the classification reliability, i.e., the probability of the early class prediction being equal to the full data classification. RelClass computes a neighbourhood set of a partially observed time series based on a minimum reliability τ and the posterior distribution conditioned on the observed and training data. When this set is completely on a single side of a decision boundary, a classification is made. Otherwise, classification is postponed until more data becomes available which will reduce the uncertainty in the posterior of the time series to be classified. This approach is able to give statistical guarantees on the reliability of classification, which gives this method its name: RelClass. The trade-off between earliness and accuracy can be made by changing the τ hyper-parameter. For the construction of the neighbourhood sets, the authors propose three different variants: 1) based on Chebyshev's inequality, 2) Naive Bayes 3) Naive

Bayes box. The selection of these variants is implemented as a categorical hyper-parameter. The authors additionally suggest using local discriminative Gaussian (LDG) dimensionality reduction, to reduce training complexity without losing much performance. This is implemented as a binary hyper-parameter. RelClass is the only included EarlyTSC algorithm that is not based on any time series specific method. However, it still is a very competitive EarlyTSC algorithm.

- **ECDIRE** (Mori et al., 2016) (`R`): ECDIRE is based on a set of probabilistic prefix classifiers. During the training phase, timestamps are learned where the accuracy reaches a predefined (as hyper-parameter) proportion of full-length accuracy. Using these timestamps, a threshold value is learned for the minimal difference between the highest and second-highest class probability. This threshold is then used for the actual classification of time series. ECDIRE uses a set of Gaussian Process (GP) classifiers. This makes it theoretically better suited for situations where little training data is available.

  Since we chose to stick to the original implementation published by the authors, this algorithm uses 20 equally spaced GP classifiers. While, in theory, any number of classifiers with any temporal distribution could be used, in this particular case, it is fixed. Since these GP classifiers are based on kernels applied to distances, it is possible to replace the conventionally used Euclidean distance by time series specific distance functions such as DTW, EDR and Fourier distance. This is exactly what the original authors did and we can switch between those distance functions using the categorical hyper-parameter “distance”. Additionally, we can switch between different kernel functions to use in the GP classifiers. To control the trade-off between earliness and accuracy, we can control the desired proportion of full-length accuracy with the acc_perc hyper-parameter.

  For the training of the GP classifiers, there is the binary hyper-parameter “doHPO” controlling whether to do optimization of the kernel parameters. Note that, although this optimization step could only improve performance of the overall method, our experimental setup included an algorithm running time limit, making it important to be able to tune the computational complexity of the algorithms.

- **SR-CF** (Mori et al., 2018) (`R`): SR-CF stands for Stopping Rule - Cost Function. SR-CF is a continuation of the work on ECDIRE, where the threshold rule is replaced with a more sophisticated, parameterised stopping rule, which is trained by optimising a cost function. The stopping rule is a function taking the output of a probabilistic classifier (i.e., the probability distribution over the classes) and returns a binary output: 1 to accept the classification, 0 to reject the classification and wait for more data. The stopping rule is optimised based on the weighted average of the earliness and the error rate, the relative weight being the α hyper-parameter, controlling the trade-off between earliness and accuracy. Additionally, the authors propose two ways of regularization: the $L0$-“norm” and the $L1$-norm. These options are controlled using the “reg” hyper-parameter. The author’s implementation enables the choice of four different optimization methods (using the “optimiser” hyper-parameter), all permitted a budget of 100 evaluations of the cost function. Due to the implementation, the prefix classifiers are fixed to 20 equally spaced GP classifiers, similar to ECDIRE.

- **TEASER** (Schäfer and Leser, 2020) (`Java`): TEASER is a recent incarnation of the set-of-prefix-classifiers approach, introducing a new approach to the triggering mechanism. The authors proposed the use of a separate classifier deciding whether to accept the classified label at the current time or wait for more data. For each prefix classifier, a one-class support vector machine (oc-SVM) is trained separately to detect when the prefix classifier correctly classifies the time series. We can control the oc-SVM kernel type and its ν hyper-parameter. In their paper, the authors show the best results are found using the WEASEL time series classifier (Schäfer and Leser, 2017) as a basis. The number and placing of the prefix classifiers are more flexible in the implementation of TEASER compared to SR-CF and ECDIRE. A freely chosen number of classifiers (“nClassifiers”) is distributed evenly between a start index and an end index.

This allows focusing more computational resources on more interesting parts of the time series. During the classification phase, the actual classification is done only when a specified number of subsequent oc-SVMs predict a reliable classification. This number can be controlled with the v hyper-parameter, which directly controls the earliness accuracy trade-off. In the original implementation, this parameter was optimised using a grid-search maximising harmonic mean of earliness and accuracy. However, since we want to use our system to find both very early as well as very accurate configurations, we removed this internal optimization step.

- **ECEC** (Lv et al., 2019) (`Java`): takes yet another approach to the trigger mechanism, using empirical reliability of each prefix classifier, computed using cross-validation. ECEC uses a method for smoothly merging the classifier outputs of multiple prefixes to get a single confidence estimate. A single confidence threshold is learned by minimising a cost function based on the weighted average of the earliness and the error rate (similar to SR-CF). If during classification, the confidence surpasses this threshold, the classification is accepted. Since the implementation is largely based on the implementation of TEASER, the same base TS classifier is used and the same hyper-parameters are available to choose the number and placement of prefix classifiers. To get more control over the computational cost of the algorithm, the number of cross-validation folds is exposed as the "nFolds" hyper-parameter. Similar to SR-CF, the relative weight α of earliness and accuracy in the cost function enables control of the earliness versus accuracy trade-off.

- **EARLIEST** (Hartvigsen et al., 2019) (`Python`): EARLIEST is a recurrent neural network (RNN) based approach to early time series classification. EARLIEST consists of three parts: 1) a base RNN, producing a low dimensional vector representation of the time series at each time step; 2) a Discriminator, mapping the vector representations to class predictions; and 3) a Controller, also taking the representation as input and outputting whether to stop and let the Discriminator classify, or whether to continue processing more data. The discriminator is trained with respect to the classification task and the controller is trained using reinforcement learning, receiving reward for successful classification by the discriminator and punished for each time step processed. The trade-off between earliness and accuracy can be controlled by the relative amount of reward and punishment using the tunable hyper-parameter λ. Additionally, hyper-parameters controlling the neural architecture can be tuned, including the number of layers (nLayers), the number of hidden nodes per layer (hiddenDim), and the RNN type (cellType) (e.g., LSTM, GRU, Elman). Furthermore, the learning rate (lr), learning rate decay factor (lrf) and the number of epochs can be tuned. The latter is the primary factor in the computational cost of the algorithm.

- **Fixed** (us) (`Python`): We complemented the set of relatively advanced EarlyTSC algorithms with a more naive method. This method extends the naive Euclidean 1-Nearest Neighbour classifier to the problem of EarlyTSC, by training the classifier on a fixed-length prefix of all time series instead of the full length. This results in a classifier that is equally early for any time series to be classified and only requires a single evaluation once the prefix has been fully observed. A single hyper-parameter "percLen" controls the prefix length at which classification is done as a proportion of the full time series length. A prefix length of zero is treated as a special case where the most frequent class in the training set is used as the prediction. We expect that any EarlyTSC algorithm with more control over "when to classify" should perform at least as well as this naïve algorithm, either by being more accurate with the same average earliness or by being earlier with the same level of accuracy or both. We implemented this algorithm specifically for the purpose of this thesis.

4.2 Algorithm Performance

As described earlier, the EarlyTSC algorithms are evaluated on both earliness and accuracy. For our setup, we need to define two metrics representing the loss in both of these objectives. For the loss relating to accuracy, we used the *error rate* C_a defined as follows:

$$C_a = \frac{|\{\mathbf{x} \in \mathcal{D}_{test} | f(\mathbf{x}_{l^*_{\mathbf{x}}}) \neq Class(\mathbf{x})\}|}{|\mathcal{D}_{test}|} \tag{4.1}$$

Where $l^*_{\mathbf{x}}$ is the length at which the classification is triggered for time series $\mathbf{x}$, $f(\mathbf{x}_{l^*_{\mathbf{x}}})$ is the early class prediction and $Class(\mathbf{x})$ is the true class of $\mathbf{x}$.

The earliness C_e is quantified by the proportion of the time series needed to produce a classification averaged over the number of samples classified. It can be written as follows:

$$C_e = \frac{1}{|\mathcal{D}_{test}|} \sum_{\mathbf{x} \in \mathcal{D}_{test}} \frac{l^*_{\mathbf{x}}}{l_{\mathbf{x}}} \tag{4.2}$$

Where $l_{\mathbf{x}}$ is the length of time series $\mathbf{x}$.

4.3 Algorithm Configurator

Sequential Model-Based Optimisation (SMBO) has shown to be a promising approach to the single-objective CASH problem (Thornton et al., 2013). However, the problem of optimising for multiple objectives is substantially more complex than the single-objective case. While there have been methods proposed for model-based optimisation for multi-objective problems (e.g., Emmerich et al., 2015), these methods are not able to handle the tree-structured search space that is typical for CASH problems.

We chose to use MO-ParamILS developed by Blot et al. (2016) for its availability and its ability to do multi-objective optimisation in a tree-structured search space, in particular algorithm configuration space. MO-ParamILS uses iterated local search (ILS), a stochastic local search method, to find promising configurations. MO-ParamILS maintains a set of non-dominated configurations referred to as the *archive*. The *one-exchange neighbourhood* of a configuration λ is the set of configurations that is obtainable by changing a single parameter. This one-exchange neighbourhood of the configurations in the archive is used for local search steps. This strategy is complemented with random search steps to increase exploration of the search space, enabling to escape local optima.

One caveat of using MO-ParamILS is the requirement of a discrete search space requiring discretisation of continuous variables. This means that the optimal hyper-parameter values can only roughly be approximated at best. On the other hand, it reduces the search space facilitating the optimisation.

In the context of reproducible research and to share our efforts with the community, we have made the source code for MultiETSC available at `https://github.com/Ottervanger/MultiETSC`.

Algorithm	hyper-parameter		Description
ECTS (Xing et al., 2011b)	min. support	int	Controls over-fitting
	version	cat	Strict or Loose version
EDSC (Xing et al., 2011a)	$minL$	int	Min. shapelet length
	$maxL$	int	Max. shapelet length
	k	real	Precision-recall trade off of shapelets
	α	real	Accuracy-earliness trade off
	method	cat	Threshold learning method
	recall thres.	real	Min. threshold for shapelet distinctiveness
RelClass (Parrish et al., 2013)	τ	real	Min. reliability threshold
	variant	cat	Chebyshev, Quadratic, Box
	LDG	bin	Use LDG dimensionality reduction or not
ECDIRE (Mori et al., 2016)	*acc_perc*	int	Desired level of accuracy as percentage of full data classification
	doHPO	bin	Contolling optimisation of kernel parameters of GPCs
	kernel	cat	Gaussian process regression kernel
Shared with SR-CF	distance	cat	Distance metric to use
	sigma	real	Theshold value for the EDR metric
	N	int	No. components for the Fourier distance metric
SR-CF (Mori et al., 2018)	α	real	Accuracy-earliness trade off
	optimiser	cat	Method used for CF optimisation
	reg	cat	Method used for CF regularisation
	lambda	real	Regularisation factor
TEASER (Schäfer and Leser, 2020)	v	int	No. required consistent classifications
	kernel	cat	Kernel type to use for oc-SVM
	ν	real	Training parameter for oc-SVM
	nClassifiers	int	
Shared with ECEC	start index	int	Position of the first prefix classifier
	stop index	int	Position of the last prefix classifier
ECEC (Lv et al., 2019)	α	real	Accuracy-earliness trade off
	nFolds	int	Number of cross-validation folds
EARLIEST (Hartvigsen et al., 2019)	λ	real	Accuracy-earliness trade off
	lr	real	Learning rate
	lrf	real	Learning rate decay factor
	epochs	int	
	nLayers	int	No. hidden layers
	hiddenDim	int	No. nodes per hidden layer
	cellType	cat	RNN cell type
1NN-Fixed	percLen	real	Prefix length as prop. of total TS length

Table 4.1: EarlyTSC algorithms and their hyper-parameters. 'int', 'cat', 'real' and 'bin' stand for integer, categorical, real and binary valued parameters, respectively.

Chapter 5

Experimental Evaluation

We have designed our experiments to answer the following two questions:

- What improvement can be achieved by solving the MO-CASH problem for EarlyTSC compared to multi-objective HPO of any single competitive EarlyTSC algorithm?
- What improvement can be achieved by solving the MO-CASH problem compared to solving the single-objective CASH problem optimising for the harmonic mean of accuracy and earliness?

In the rest of this section, we will introduce our baselines, data sources, and evaluation protocol. Next, we will present the experimental results that will answer these questions.

5.1 Baselines

To answer the first question we compared algorithm selection, using MO-ParamILS on the previously defined search space, with hyper-parameter optimisation of each individual algorithm in the search space by fixing the algorithm choice. This results in 9 baseline methods, one for each included algorithm: ECTS, EDSC, RelClass, ECDIRE, SR-CF, TEASER, ECEC, EARLIEST and Fixed.

To address the second question, we compared our multi-objective approach with a method that optimises a single-objective (we refer to this baseline as SO-all). For this objective, we chose the harmonic mean of earliness and accuracy as suggested by Schäfer and Leser (2020):

$$HM = 1 - \frac{2 \cdot (1 - C_e) \cdot (1 - C_a)}{(1 - C_e) + (1 - C_a)} \tag{5.1}$$

Resulting in a value on the closed interval $[0, 1]$ which is to be minimised. This metric has the property that it will be low when both C_e and C_a are low and high when either is high. We will refer to this as the *HM* metric.

Although MO-ParamILS is capable of single-objective algorithm configuration, more advanced systems are available for this task. To make a fair comparison, we chose the state-of-the-art SMAC (Lindauer et al., 2017) algorithm configurator for its ability to efficiently search tree-structured search spaces. We will refer to the baseline method searching the full algorithm space using SMAC optimising the *HM* metric as SO-All.

5.2 Data

As the main source of data, we will use the University of California, Riverside (UCR) Time Series Archive (Chen et al., 2015; Dau et al., 2018), last updated in 2018. As of 2018, the archive consists of

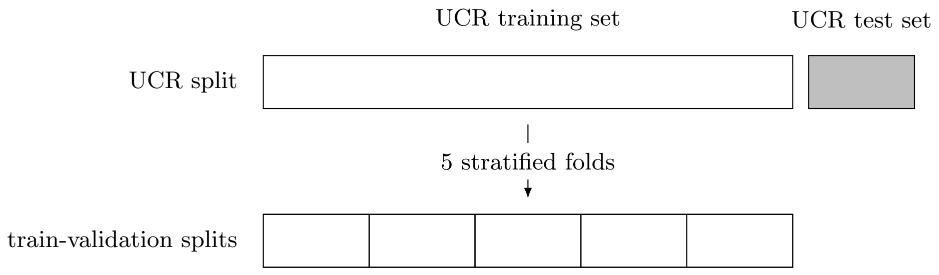


Figure 5.1: Train, validate and test sets.

128 time series data sets for time series classification. Since its introduction in 2015, it has become the de facto standard for the evaluation of time series classification methods. The composers of the UCR archive recommend evaluating on all 128 data sets, and to motivate excluding any. In our experiments we used 115 of these data sets due to the reason explained in Section 5.3. The data sets in the UCR archive contain real-world data and simulated data originating from various sources with varying degrees of complexity. Sources include ECG, EEG, spectrographs, image outlines, three-axis accelerometers and gyroscopes, and audio samples. In addition to real-world data, the archive contains nine synthetic data sets specifically designed to evaluate time series classification methods. For data set details and descriptions, we refer to the Time Series Classification Website (Bagnall and Lines, 2020).

Although some data sources are intrinsically multi-variate, the data is split up into uni-variate time series. Time series lengths vary between a few dozen samples to several thousand samples. For most data sets, all time series within one set are of equal length. 15 of the 128 data sets contain time series of varying lengths or with missing values. For these data sets the archive also includes same-length versions that are imputed using linear interpolation and padded up to the length of the longest time series with low amplitude noise. For this thesis, we only used the same-length versions of these data sets.

5.3 Evaluation Protocol

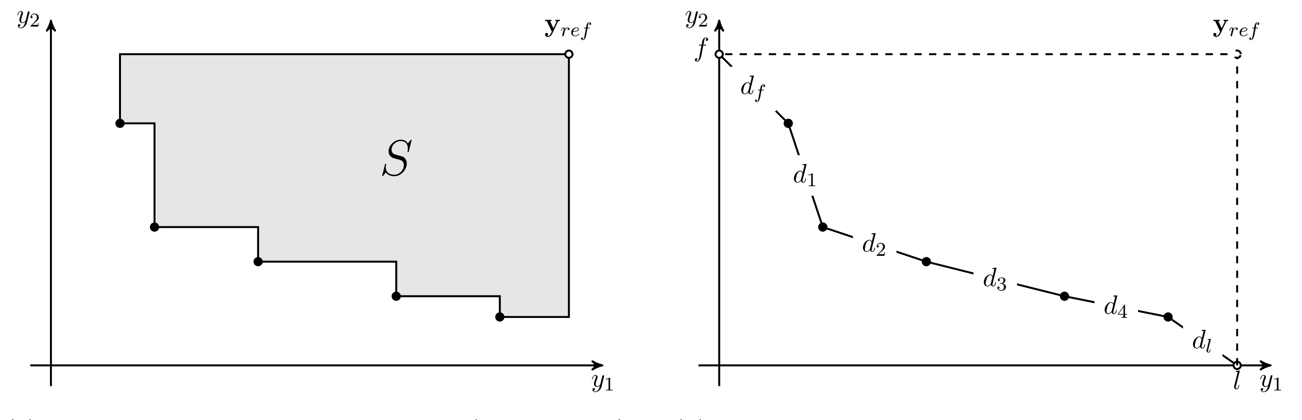
Train and Validation splits: the UCR data sets have pre-defined train-test splits. The UCR defined train set is split again into five stratified train-validation splits. An algorithm is trained on $\frac{4}{5}$ of the data and evaluated on $\frac{1}{5}$ of the data. This is illustrated in Figure 5.1. For each repeated configurator run, a different set of cross-validation folds is generated. However, each experimental condition is run with the same set of cross-validation folds to keep the comparison fair. Due to the fivefold split requirement, we had to exclude eight data sets that did not contain sufficient training examples per class for this split to be made. These were FiftyWords, Fungi, Phoneme, PigAirwayPressure, PigArtPressure, PigCVP, and Symbols. We used the remaining 120 data sets for evaluation.

Configurator runs: because a single configurator run is dependent on the random train/validation splits, and random initialisation, we performed multiple runs to get stable results. For each data set, 25 configurator runs were performed. For practical reasons, we set limits to run times of different stages of the experiment (presented in Table 5.1). The test evaluation is provided with a budget that is significantly bigger than that of the validation. The reason is that the test set can contain much more examples than the validation set. Additionally, we had to set a maximum amount of memory used by an algorithm implementation, which we set to 10GB. This made it impossible to process five more data sets: Crop, ElectricDevices, FordB, FordA, InsectWingbeatSound.

Bootstrapping: each configurator run will result in a Pareto set of configurations based on the validation performance. Note that in the case of the SO configurator, this set will, by definition, contain only a single configuration. To create a distribution of the results, we took a bootstrapping approach by

Configurator time budget	**Config. Validation time limit**	**Config. Test time limit**
120 min	3 min	15 min
Configurator runs	**Bootstrap sample size**	**Bootstrap samples**
25	10	1000

Table 5.1: Key numbers of the evaluation protocol.



(a) A set of non-dominated solutions P (black points) in the objective space $\mathbb{R}^2$. S indicates the hypervolume dominated by P with respect to reference point $\mathbf{y}_{ref}$.

(b) The distances used for the computation of the Δ-spread of a set of non-dominated solutions P (black points). f and l are theoretical extreme solutions.

Figure 5.2: Calculation of the Pareto front metric.

randomly sampling 10 runs from the 25 runs performed. For each subsample, the set of Pareto sets (one for each run) is then combined into a single set of non-dominated configurations (based on validation performance). All configurations ending up in one or more subsample Pareto sets are evaluated using test data resulting in a final Pareto set for each subsample. Evaluation is based on these final Pareto sets. By repeating this approach 1000 times we created a bootstrap distribution.

5.4 Evaluation Metrics

We want to evaluate Pareto sets of configurations in the earliness-accuracy space. Solutions that are both accurate and early are desirable, but we also prefer a “cheap” trade-off, getting much earlier predictions for only a small reduction of accuracy. To quantify the performance of a particular Pareto set, we will turn to the theory of multi-objective optimisation. There is a multitude of Pareto set performance metrics defined each capturing one or more desirable aspects (Audet, 2018). For our purpose, we will be using the dominated hypervolume, the Δ-spread and the HM-metric which we describe here in detail.

- **Hypervolume (HV)**: also called the S-metric, proposed by Zitzler, Deb and Thiele (Zitzler et al., 2000), is the hypervolume in the objective space that is dominated by the Pareto efficient solutions bounded by a reference point $\mathbf{y}_{ref} \in \mathbb{R}^m$ that is dominated by all feasible solutions. Figure 5.2a shows a Pareto front and its dominated hypervolume S in $\mathbb{R}^2$. A commonly mentioned downside of the hypervolume is the need for a reference point that is larger than any point in the Pareto set on all objectives but not too large since that would reduce the precision. In our case, however, it is clear that both error rate and earliness will never be larger than 100%, so we can safely use $(1, 1)$ as the reference point.

  The hypervolume metric has several properties that make it well suited for our purpose. First of all, it is intuitive. Dominating a large part of the objective space is desirable. As denoted in (Audet, 2018), the hypervolume metric is the only known unary Pareto front performance indicator to be strictly monotonic. This means that if set A dominates set B (i.e., all points in B are dominated by at least one point in A) then the hypervolume metric of A is strictly larger than that of B. This is desirable since set domination is a strong indication that one set is preferable over the other and we want the metric to represent this property as well. Furthermore, the hypervolume metric is widely used in the performance evaluation of various multi-objective optimisation algorithms.

- **Δ-spread:** a desirable property that is not very well represented in the hypervolume is the distribution of configurations along the Pareto front. To control the earliness-accuracy trade-off, these configurations should be evenly distributed over a wide range of values. There can be two cases where the hypervolume metric is equal but in one case all hypervolume is dominated by a single configuration and in the other case there are 1000 unique configurations. In that case, it would be preferable to have a choice out of multiple options. We will be using the Δ-spread metric (Zitzler et al., 2000) to quantify the distribution of solutions in the objective space. This metric is based on the Euclidean distances between neighbouring solutions d_i and is formalised as follows:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1) \cdot \bar{d}} \tag{5.2}$$

  Where d_f and d_l are distances between the extreme solutions and the boundary solutions in the Pareto front, and $\bar{d} = \frac{\sum_{i=1}^{N-1} d_i}{N-1}$. See Figure 5.2b for an illustration of the distances. An ideal distribution would make $d_f = d_l = 0$ and all distances d_i equal to $\bar{d}$, resulting in $\Delta = 0$. More clustering will result in higher values of Δ.

- ***HM* metric:** in addition to the hypervolume metric and Δ-spread, we will also look at the *HM* scalarisation as defined in Equation 5.1, to see how this metric compares to our proposed evaluation methods. Since each point in the Pareto set will have its own *HM* value, we will look at the minimum value in each set.

5.5 Results

To provide some intuition on the produced Pareto sets and their metrics we will discuss an illustrative example. A single subsample is constructed by combining ten different validation splits. Each method is run on each split and the resulting Pareto sets are combined into one Pareto set per method. Figure 5.3 shows these Pareto sets for an arbitrary subsample of the BME data set. The BME data set is a synthetic data set created specifically for TSC research and is part of the UCR archive. It is a three-class problem with 10 training examples per class. The test set is balanced containing 150 items.

Looking at Figure 5.3, it is seen that generally, in Pareto fronts, 70% seems to be the upper bound of the error rate at zero earliness (maximally early), which is close to the expected performance of 66%of a random or constant output classifier. With increasing earliness (later classification) most methods can produce classifiers that are increasingly accurate with clearly diminishing returns. Seven out of the eleven methods level out at one-third of the time series observed. EARLIEST and Fixed do improve further around 90% of data observed, but they do not beat most methods in terms of performance at that point. MultiETSC and ECDIRE are able to reduce error rate up to 70% of the data observed achieving the best accuracy of less than 5% error rate.

The visual representation of the trade-off between earliness and accuracy that MultiETSC can provideis a major advantage. Using any EarlyTSC algorithm requires the user to select a hyper-parameter that only indirectly influences the trade-off. Using our approach, a user trying to solve a specific EarlyTSC problem can make an informed choice, making EarlyTSC more accessible for non-experts.

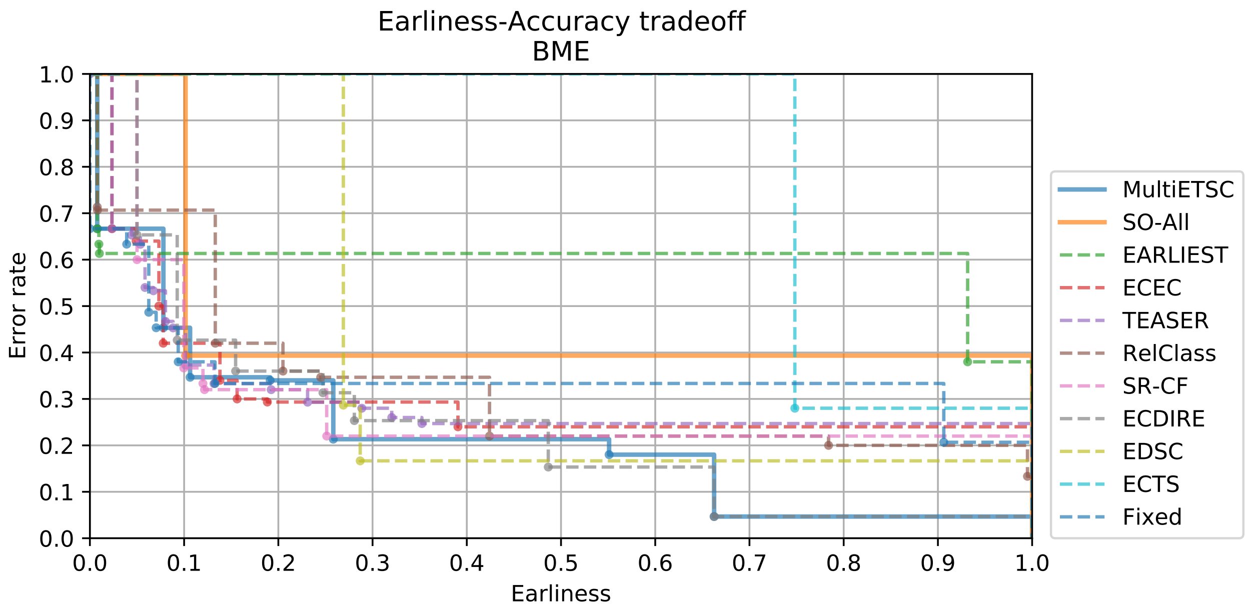


Figure 5.3: Pareto sets for one subsample on BME data set.

method	Fixed	ECTS	EDSC	ECDIRE	SR-CF	RelClass	TEASER	ECEC	EARLIEST	SO-All	MultiETSC
HV	0.653	0.181	0.607	0.757	0.706	0.675	0.694	0.695	0.400	0.545	**0.782**
Δ-spread	1.015	inf	0.925	0.675	0.936	0.723	0.800	0.819	1.132	inf	**0.603**
HM	0.246	0.627	**0.231**	0.267	0.233	0.291	0.246	0.235	0.444	0.276	0.236

Table 5.2: Pareto set metrics for one subsample on the BME data set.

The metrics evaluating these Pareto sets are shown in Table 5.2. In this particular example, the configurations found by MultiETSC dominate the largest hypervolume (0.782). Although the competing methods are close, it is clear that MultiETSC corresponds closely to the best of each baseline. If we compare MultiETSC with ECDIRE, we see they have a very similar hypervolume. However, ECDIRE has a higher Δ-spread which indicates more clustered solutions in the earliness-accuracy space which is less favourable. Looking at the HM-metric, we see MultiETSC not achieving the best score. Both SR-CF and EDSC achieved better HM-metrics. These are achieved by the solutions around 0.2 error rate and 0.26 earliness. EDSC in particular is an example of the weakness of the HM-metric. With only two solutions found, it represents only a tiny fraction of the possible trade-off points leaving both earlier and more accurate solutions unexplored.

To compare relative performance between all methods, based on their performance across 115 data sets, we computed the average ranks based on the three selected metrics. Average ranks provide a robust method of comparison without making additional assumptions about normality and symmetry – assumptions we cannot safely make in general. This approach is similar to the empirical analysis done by Bagnall et al. (2016). The ranks are averaged over all subsamples and all datasets. With 1000 subsamples for each of the 115 data sets, these are the averages over 115 000 ranks. We first checked for significant differences between rank means using the Friedman test and subsequently applied the Nemenyi post-hoc test (Nemenyi, 1963) which checks for significant pairwise differences in average ranks. We visualised the outcome of this test using critical difference diagrams (Demšar, 2006), that are commonly used for the comparison of TSC methods to show the result of a statistical comparison of ranking results. These are shown in Figures 5.4, 5.5 and 5.6. Due to our subsampling method, we achieved high statistical power resulting in small CD intervals, which means that most observed differences are statistically significant.

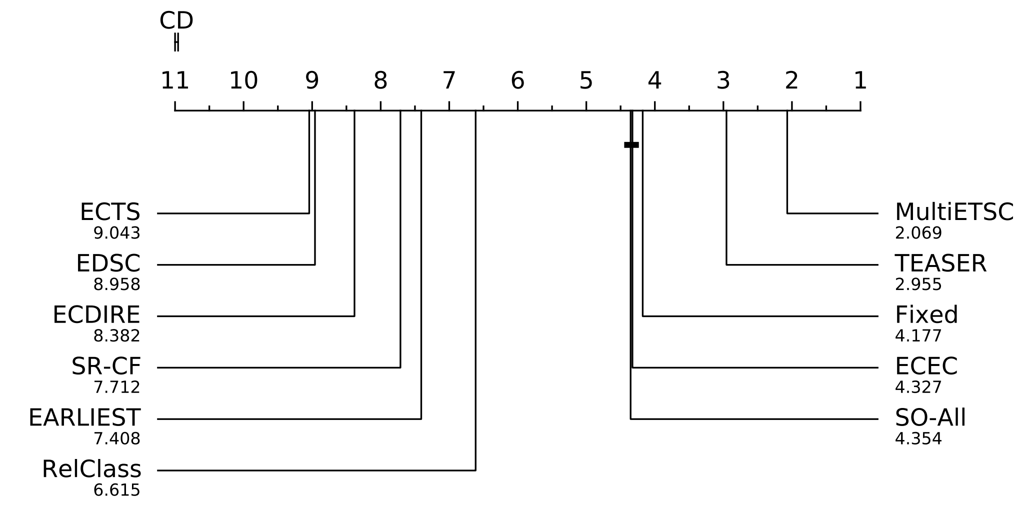


Figure 5.4: CD diagram of the Nemenyi test on the Hypervolume. Numbers represent mean ranks (lower means better). Rank means with non-significant difference are connected with a horizontal line.

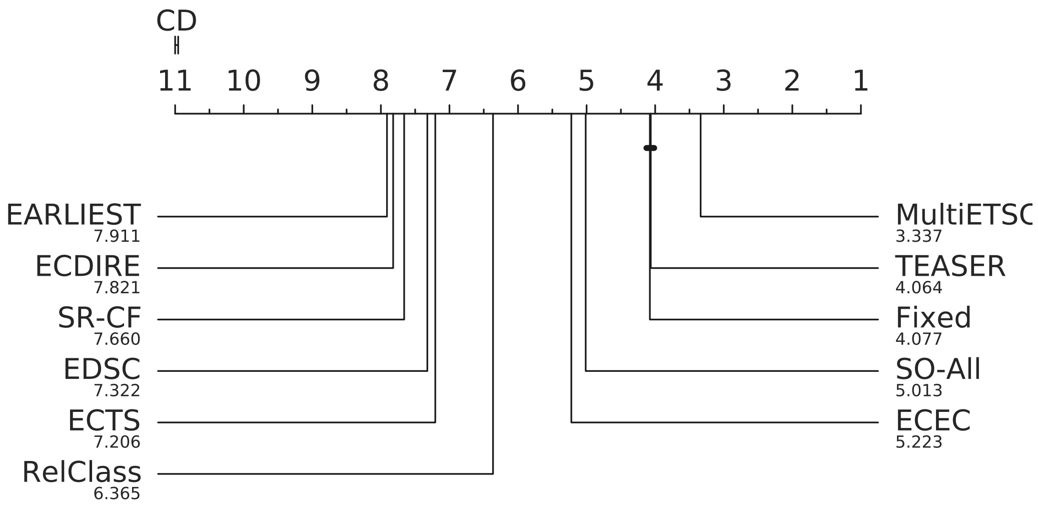


Figure 5.5: CD diagram of the Nemenyi test on the Δ-spread. Numbers represent mean ranks (lower means better). Rank means with non-significant difference are connected with a horizontal line.

According to the comparison of methods based on all metrics (shown in Figures 5.4,5.5,5.6), MultiETSC performs significantly better than any of the algorithms we compared against, finding configurations that together dominate a larger portion of the objective space and distributed more evenly across the trade-off according to the Δ-spread. This answers our first question mentioned earlier in Section 5. Similarly, *MultiETSC performs significantly better than the single-objective CASH method SO-All*, which answers our second question. We also observed that SO-All performs relatively well compared to our baseline algorithms. However, interestingly, SO-All performs worse than MultiETSC on the *HM* metric, even though this is the exact metric that the SO configurator optimises, while MultiETSC does not explicitly consider this metric during configuration. Overall, methods consistently rank according to different metrics.

Table 5.3 shows the performance of the compared methods split out per problem (dataset) type. This table would show any method that is particularly suited or unsuited for a specific problem type. From these results, we observe that MultiETSC is consistently performing well across a broad range of problem types. The results for individual data sets are provided in Appendix A (Tables 5-7).

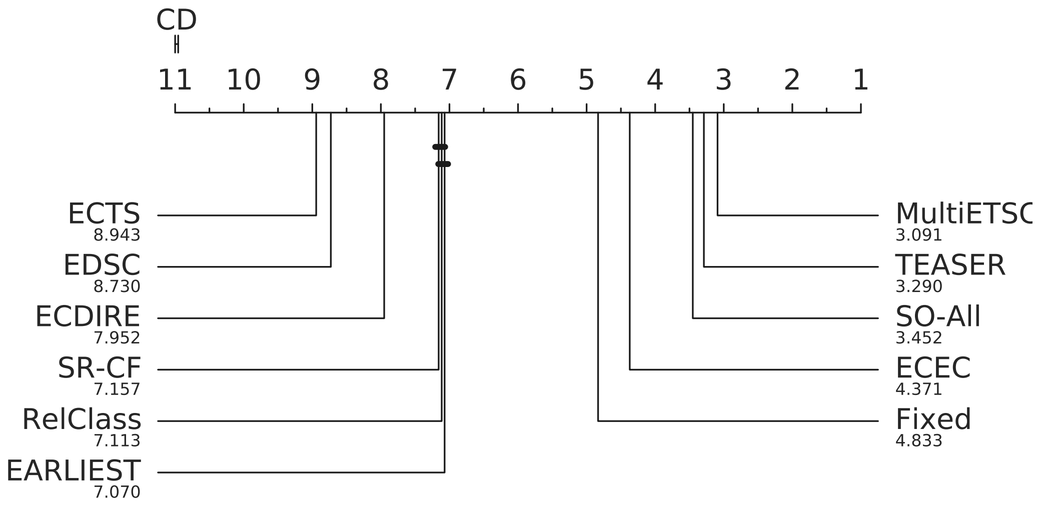


Figure 5.6: CD diagram of the Nemenyi test on the HM metric. Numbers represent mean ranks (lower means better). Rank means with non-significant difference are connected with a horizontal line.

method	EARLIEST	ECDIRE	ECEC	ECTS	EDSC	Fixed	MultiETSC	RelClass	SO-All	SR-CF	TEASER	Counts
DEVICE	0.00	0.00	20.59	0.00	0.00	2.11	**32.26**	0.00	13.37	0.00	31.69	9000
ECG	0.00	0.00	2.93	0.00	0.00	30.90	**54.46**	0.00	0.41	0.00	11.30	7000
EOG	0.00	0.00	0.00	0.00	0.00	**72.35**	30.70	0.00	0.00	0.00	0.00	2000
EPG	4.65	0.00	0.00	0.00	0.00	0.00	**89.00**	22.50	21.80	0.00	0.00	2000
IMAGE	0.33	0.00	3.26	0.00	0.73	5.51	**40.89**	14.90	2.20	0.16	32.01	28000
MOTION	0.00	0.00	7.49	0.00	0.00	6.86	**44.26**	0.12	5.61	0.00	35.72	25000
SENSOR	0.02	0.00	12.36	0.00	0.03	7.87	**46.37**	4.53	5.18	0.06	28.22	19000
SIMULATED	0.00	4.73	1.08	0.00	0.00	15.63	**46.52**	0.36	0.02	0.50	31.16	9000
SPECTRO	0.00	0.15	14.90	0.00	0.05	15.43	**31.19**	9.09	3.17	0.67	25.36	12000
TRAFFIC	0.00	0.00	0.00	0.00	0.00	50.00	**60.60**	0.00	0.00	0.00	38.55	2000
Overall	0.16	0.39	7.89	0.00	0.19	11.14	**43.11**	5.77	4.40	0.16	29.14	
Counts	187	445	9077	0	217	12812	49575	6635	5055	183	33514	115000

Table 5.3: Best performing algorithms by problem type. Entries represent percentages of subsamples that each method achieved the highest hypervolume on. Best performance is presented in bold face. Ties are counted as wins for both methods.

We found a significant number of cases with ties for the best method, most often between Fixed and MultiETSC. In these cases, the set of possible configurations using the Fixed algorithm dominates all other configurations. This set is found by both the Fixed method and MultiETSC, which results in equal hypervolume scores. This causes some rows in Table 5.3 to add up to more than 100% (e.g., TRAFFIC).

As expected, the performance rank of each single algorithm method seems to correspond closely to the order in which these algorithms were originally introduced. However, there are some exceptions to this pattern. EARLIEST, the only neural-network-based algorithm, performs not as well as other methods proposed around the same time. In the original study, EARLIEST (Hartvigsen et al., 2019), is evaluated on data sets containing considerably more training examples than are available in typical UCR data sets and many real-world applications. Additionally, the running time limits (3 minutes for configuration and 15 minutes for testing) might pose a challenge for the configurator to find viable parameter settings. This, however, is a challenge for all methods, not only EARLIEST.

Finally, we would like to address the fact that the naïve fixed-time method we introduced is highly competitive, consistently outperforming six out of the eight EarlyTSC algorithms proposed in the

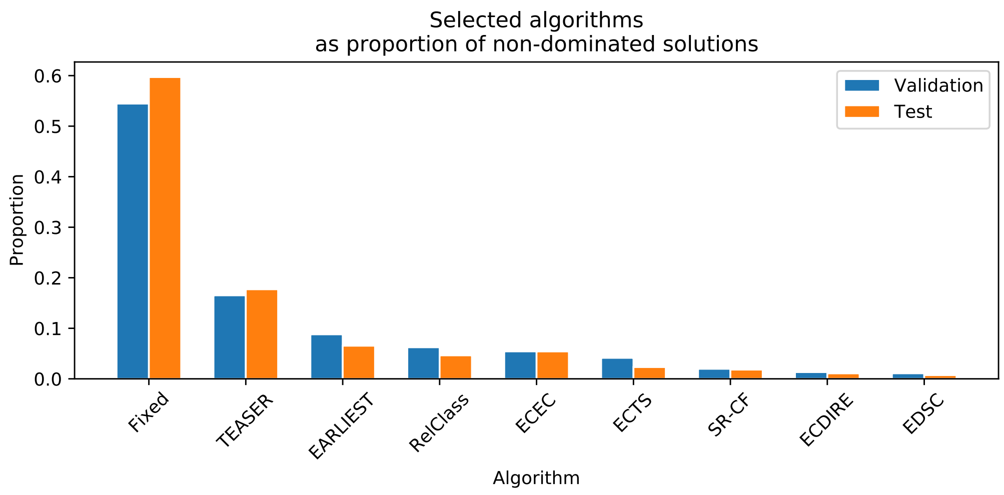


Figure 5.7: Algorithms as selected by MultiETSC as proportion of non-dominated solutions as found based on validation and test evaluations.

literature, according to all three metrics. One explanation for this could be the fact that early research on EarlyTSC has been mainly focused on getting early classification with accuracy similar to that achieved on the full data. This means that these algorithms are very much focused on accuracy. Although there might be hyper-parameters controlling the earliness-accuracy trade-off, once pushed to the early edge of the spectrum, these algorithms can lose accuracy very quickly.

A second explanation for the relatively good performance of the naïve Fixed method is based on a combination of its simplicity and the small size of the search space. Since Fixed is a 1NN method, there is no training phase, and since it is prefix-based, only a fixed portion of each time series needs to be considered, which makes the method very fast. Additionally, there is only one hyper-parameter to tune: the prefix proportion. Due to this, the configurator can exhaustively search the full hyper-parameter space (taking 100 steps of 0.01). This is the only algorithm we considered for which this is possible, and it implies that there are many points along the trade-off between earliness and accuracy considered. For all other algorithms, the configurator needs to spend time on tuning hyper-parameters that are orthogonal to this trade-off.

To get an idea of the relative importance of each included algorithm, we looked at the selection frequency of each algorithm. Since MultiETSC produces sets of non-dominated solutions, we were interested in the selected algorithms as proportions of these solutions. These proportions, which are shown in Figure 5.7, can provide a proxy for the relative performance of each algorithm for the system as a whole. We find the most configurations are using the naïve Fixed algorithm. As stated earlier, this can be explained by the simplicity of its hyper-parameter space. Furthermore, a large portion of algorithms is selected only very few times. This indicates that MultiETSC might benefit from leaving some algorithms out. This would allow for more time to optimise the hyper-parameters of better-performing algorithms.

However, we need to be careful drawing conclusions. Having many solutions does not mean they are good solutions. A single configuration can dominate many sub-optimal configurations. It might be the case that the configurations using Fixed only slightly improve configurations using other algorithms, but that other algorithms give higher improvement the few times that they do end up in the non-dominated set. This means the data in Figure 5.7 is not sufficient to say which algorithms should remain and which should be removed. To get a proper assessment of the relative contribution of each algorithm to the performance of MultiETSC, the Shapley values can be computed for each algorithm as proposed in (Fréchette et al., 2016). We will discuss this further in Section 6.1.

Chapter 6

Conclusions and Future Work

In this work, we have introduced MultiETSC, a systematic approach to automated early time series classification (EarlyTSC). MultiETSC performs automatic algorithm selection and hyper-parameter optimisation to explore the full range of optimised trade-off points between accuracy and earliness. Our approach builds upon recently developed techniques in the area of automated algorithm configuration and uses them in combination with a broad range of well-known EarlyTSC algorithms. Integrating a wide range of recently developed EarlyTSC algorithms, MultiETSC is capable of exploring the combined space of possible algorithms and hyper-parameter configurations efficiently for any given EarlyTSC problem. This enables users to explore and exploit trade-offs between earliness and accuracy, allowing them to make an informed decision on the preferred trade-off point.

We performed an extensive empirical evaluation of our proposed method using 115 data sets from the UCR Time Series Archive. We have shown that by leveraging the performance potential of many existing EarlyTSC algorithms, our approach can outperform any single algorithm, even when those are using optimised hyper-parameter settings. Our results also demonstrate that considering both earliness and accuracy separately produces better results than combining the two into a single objective.

6.1 Possible Improvements to MultiETSC

During the development and evaluation of MultiETSC, we were forced to make several practical decisions to keep the size of this project manageable. However, there is a multitude of ways in which MultiETSC could be improved in the future.

In its current form, MultiETSC includes all currently available EarlyTSC algorithms. Since each algorithm included in MultiETSC adds to the size of the search space, each additional algorithm is a burden to the algorithm configurator. It might therefore be the case that removing an algorithm from the system might increase its overall performance since it allows the configurator to spend more time searching and tuning the other algorithms. In our results, we already found evidence for a small set of algorithms being responsible for a large proportion of the optimised configurations. In future research, the marginal contribution to the performance of each algorithm could be computed. However, marginal contribution penalizes sets of correlated algorithms. Fréchette et al. (2016) propose using the Shapley value to asses relative algorithm importance. Removing all algorithms with a negative Shapley value will likely lead to performance improvement of MultiETSC as a whole.

The current version of MultiETSC is based on pre-existing implementations of algorithms. Some of these algorithms perform similar or even identical computations as part of their learning phase, for example, the computationally expensive distance metrics between all training pairs. We expect that much performance can be gained by exploiting these similarities. This can be done by caching intermediate results. For example, both ECDIRE and SR-CF compute the reliability values of a set of prefix classifiers using cross validation as part of the training process. These reliability values are expensive to compute and only depend on a small subset of hyper-parameters. These values can be stored for reuse when either of these methods are applied to the same data with a different hyper-parameter setting. A similar cache can be used for the prefix classifiers of TEASER and ECEC. This would save computational resources and would allow for a larger number of evaluated configurations in the same time. Using such a caching method would require re-implementation of the EarlyTSC algorithms. Re-implementation of EarlyTSC algorithms, if done right, can have many more benefits over the current version of MultiETSC. Algorithms can be implemented in a single, high-performance language. Source code for certain subroutines, e.g., distance computation, can be shared between algorithms and their solutions can therefore be easily cached. Hyper-parameters for these subroutines can be optimised independently of the algorithm calling them. This all would result in increased efficiency of the algorithm configuration, which in turn leads to better expected results given the same configuration time. Re-implementation would give the opportunity to build a consistent API for EarlyTSC algorithms. Such an API, conforming to best practices of API design and possibly modelled after other, widely used machine learning APIs (e.g., scikit-learn Pedregosa et al., 2011), would make EarlyTSC more accessible to a broader public. Furthermore, re-implementation of EarlyTSC algorithms would allow for adhering to the Programming by Optimization (PbO) paradigm (Hoos, 2012). This would leave design choices (e.g., the type of prefix classifier in SR-CF) to be optimised by the algorithm configurator as hyper-parameters, as opposed to having them hardwired in code.

6.2 Further Future Work

Although the work presented here represents merely a first step into automated machine learning for early time series classification, we have clearly demonstrated the potential of this direction. We see numerous opportunities for future research.

In this work, we have found indications that our Fixed algorithm, a naïve EarlyTSC algorithm that classifies at a fixed proportion of the full time series, can often outperform more sophisticated methods that attempt to use the observed data to find the best time at which to trigger the classification. While this can be ascribed to our method of comparing algorithms, thus far, no standard for comparing EarlyTSC algorithms has been established. In the future, MultiETSC could be extended to a platform for the fair comparison of EarlyTSC algorithms, taking into account earliness and accuracy at the same time. Such a platform would provide a common interface for both problem instances (data sets) and EarlyTSC algorithms. This idea is an extension of the Sparkle platform proposed by Van der Blom et al. (2019). The marginal contribution to performance or Shapley values (as mentioned in the previous section) would provide a metric for the fair comparison of competing algorithms. This platform would greatly facilitate future development of EarlyTSC algorithms.

For automated machine learning, our work clearly shows the potential of using multi-objective algorithm configurators within an integrated system leveraging many state-of-the-art techniques – an idea that can be extended to many other domains in which multiple conflicting performance objectives arise.

One such domain is the relatively simple case of binary classification. Binary classification problems often have asymmetric costs associated with false-positives and false-negatives. Therefore, it can be useful to consider precision and recall as two separate objectives. Tari et al. (2020) used such a multi-objective approach to hyper-parameter optimization of a single binary classification algorithm. The current work suggests the viability of extending their approach to larger configuration spaces that include multiple algorithms.

An objective that could be worth considering for a wide range of machine learning models is computational cost. There are situations where models are applied in an environment with limited computing resources, for example, on lightweight SoC's. In other situations, ML models might be competing with other software for computing resources, like the recently developed Deep Learning Super Sampling

(DLSS) within the graphics pipeline. For models deployed on a large scale, even environmental considerations might become a reason to prefer simpler models to reduce overall energy consumption. For all mentioned cases, it would be valuable to know the loss in performance of an ML model as a result of limiting its complexity. This can be accomplished by considering performance and model simplicity as two competing objectives.

Furthermore, our approach can be extended to a larger number of objectives, which would enable the exploration of additional trade-offs. However, interpretation of the resulting Pareto fronts will become less intuitive since they will be hard to visualize, especially with more than three objectives.

Finally, we hope that this work will inspire further exploration in this direction and ultimately lead to significant improvements in the state of the art in solving a broad set of machine learning problems that involve multiple competing performance objectives, as is the case in early time series classification.

Bibliography

Abdelghani SA, Rosenthal TM, Morin DP (2016) Surface electrocardiogram predictors of sudden cardiac arrest. The Ochsner Journal 16(3):280–289, URL https://pubmed.ncbi.nlm.nih.gov/27660578

Agrawal R, Faloutsos C, Swami A (1993) Efficient similarity search in sequence databases. In: Lomet DB (ed) Foundations of Data Organization and Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 69–84, DOI 10.1007/3-540-57301-1_5

Antonucci A, Scanagatta M, Mauá DD, de Campos CP (2015) Early classification of time series by hidden markov models with set-valued parameters. In: Proceedings of the NIPS Time Series Workshop, pp 1–5, URL https://sites.google.com/site/nipsts2015/home

Audet C (2018) Performance Indicators in Multiobjective Optimization. Les Cahiers du GERAD, GERAD HEC Montréal, URL https://books.google.nl/books?id=uKepzQEACAAJ

Bagnall A, Lines J (2020) The UEA TSC website. URL http://www.timeseriesclassification.com/

Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2016) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery 31, DOI 10.1007/s10618-016-0483-9

Baydogan MG, Runger G, Tuv E (2013) A bag-of-features framework to classify time series. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(11):2796–2802, DOI 10.1109/TPAMI.2013.72

Bengio Y (2000) Gradient-based optimization of hyperparameters. Neural Computation 12(8):1889–1900, DOI 10.1162/089976600300015187

Bergstra J, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. In: Proceedings of the 24th International Conference on Neural Information Processing Systems, Curran Associates Inc., USA, NIPS'11, pp 2546–2554, URL http://dl.acm.org/citation.cfm?id=2986459.2986743

Van der Blom K, Luo C, Hoos HH (2019) Sparkle: Towards automated algorithm configuration for everyone. Configuration and Selection of Algorithms (COSEAL), URL https://www.researchgate.net/publication/335421409

Blot A, Hoos HH, Jourdan L, Kessaci-Marmion M, Trautmann H (2016) MO-ParamILS: A multi-objective automatic algorithm configuration framework. In: Proceedings of the 10th International Conference on Learning and Intelligent Optimization (LION 10), Springer, Lecture Notes in Computer Science, vol 10079, pp 32–47, DOI 10.1007/978-3-319-50349-3_3, URL https://doi.org/10.1007/978-3-319-50349-3_3

Chen L, Özsu MT, Oria V (2005) Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA, SIGMOD '05, p 491–502, DOI 10.1145/1066157.1066213

Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2015) The UCR time series classification archive. URL `www.cs.ucr.edu/~eamonn/time_series_data/`

Christ M, Kempa-Liehr AW, Feindt M (2017) Distributed and parallel time series feature extraction for industrial big data applications. `1610.07717`

Christ M, Braun N, Neuffer J, Kempa-Liehr A (2018) Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package). Neurocomputing 307, DOI 10.1016/j.neucom.2018.03.067

Dachraoui A, Bondu A, Cornuéjols A (2015) Early classification of time series as a non myopic sequential decision making problem. In: Appice A, Rodrigues PP, Santos Costa V, Soares C, Gama J, Jorge A (eds) Machine Learning and Knowledge Discovery in Databases, Springer International Publishing, Cham, pp 433–447, DOI 10.1007/978-3-319-23528-8_27

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

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7(1):1–30, URL `http://jmlr.org/papers/v7/demsar06a.html`

Deng H, Runger G, Tuv E, Vladimir M (2013) A time series forest for classification and feature extraction. Information Sciences 239:142 – 153, DOI https://doi.org/10.1016/j.ins.2013.02.030, URL `http://www.sciencedirect.com/science/article/pii/S0020025513001473`

Emmerich M, Yang K, Deutz A, Wang H, Fonseca C (2015) A Multicriteria Generalization of Bayesian Global Optimization, vol 107, Springer International Publishing, pp 229–242. DOI 10.1007/978-3-319-29975-4_12

Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F (2015) Efficient and robust automated machine learning. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R (eds) Advances in Neural Information Processing Systems, Curran Associates, Inc., vol 28, pp 2962–2970, URL `https://proceedings.neurips.cc/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf`

Fréchette A, Kotthoff L, Michalak TP, Rahwan T, Hoos HH, Leyton-Brown K (2016) Using the shapley value to analyze algorithm portfolios. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16), AAAI Press, pp 3397–3403, URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12495`

Freund Y, Schapire RE (1999) A short introduction to boosting. In: In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp 1401–1406, DOI 10.1051/matecconf/201713900222

Fulcher B, Jones N (2014) Highly comparative feature-based time-series classification. IEEE Transactions on Knowledge and Data Engineering 26, DOI 10.1109/TKDE.2014.2316504

Gardner S, Golovidov O, Griffin J, Koch P, Thompson W, Wujek B, Xu Y (2019) Constrained multi-objective optimization for automated machine learning. In: 2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp 364–373, DOI 10.1109/DSAA.2019.00051

Grabocka J, Schilling N, Wistuba M, Schmidt-Thieme L (2014) Learning time-series shapelets. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 392–401, DOI 10.1145/2623330.2623613

Grabocka J, Wistuba M, Schmidt-Thieme L (2016) Fast classification of univariate and multivariate time series through shapelet discovery. Knowledge and Information Systems 49(2):429–454, DOI 10.1007/s10115-015-0905-9

Hartvigsen T, Sen C, Kong X, Rundensteiner E (2019) Adaptive-halting policy network for early classification. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 101–110, DOI 10.1145/3292500.3330974

Hatami N, Chira C (2013) Classifiers with a reject option for early time-series classification. 2013 IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL) pp 9–16, DOI 10.1109/CIEL.2013.6613134

Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2013) Classification of time series by shapelet transformation. Data Mining and Knowledge Discovery 28, DOI 10.1007/s10618-013-0322-1

Hoos HH (2012) Programming by optimization. Commun ACM 55(2):70–80, DOI 10.1145/2076450.2076469, URL `http://doi.acm.org/10.1145/2076450.2076469`

Hutter F, Hoos HH, Leyton-Brown K, Murphy K (2009) An experimental investigation of model-based parameter optimisation: SPO and beyond. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO 2009), pp 271–278, DOI 10.1145/1569901.1569940

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION 5), pp 507–523, DOI 10.1007/978-3-642-25566-3_40

Jin H, Song Q, Hu X (2019) Auto-keras: An efficient neural architecture search system. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, pp 1946–1956, DOI 10.1145/3292500.3330648

Koch P, Golovidov O, Gardner S, Wujek B, Griffin J, Xu Y (2018) Autotune: A derivative-free optimization framework for hyperparameter tuning. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Association for Computing Machinery, New York, NY, USA, KDD '18, p 443–452, DOI 10.1145/3219819.3219837

Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing SAX: A novel symbolic representation of time series. Data Mining and Knowledge Discovery 15:107–144, DOI 10.1007/s10618-007-0064-z

Lin J, Khade R, Li Y (2012) K rotation-invariant similarity in time series using bag-of-patterns representation. Journal of Intelligent Information Systems 39, DOI 10.1007/s10844-012-0196-5

Lindauer M, Eggensperger K, Feurer M, Falkner S, Biedenkapp A, Hutter F (2017) SMAC v3: Algorithm configuration in python. URL `https://github.com/automl/SMAC3`

Lines J, Davis L, Hills J, Bagnall A (2012) A shapelet transform for time series classification. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 289–297, DOI 10.1145/2339530.2339579

Lv J, Hu X, Li L, Li P (2019) An effective confidence-based early classification of time series. IEEE Access 7:96113–96124, DOI 10.1109/ACCESS.2019.2929644

Mori U, Mendiburu A, Keogh E, Lozano J (2016) Reliable early classification of time series based on discriminating the classes over time. Data Mining and Knowledge Discovery 31, DOI 10.1007/s10618-016-0462-1

Mori U, Mendiburu A, Dasgupta S, Lozano JA (2018) Early classification of time series by simultaneously optimizing the accuracy and earliness. IEEE Transactions on Neural Networks and Learning Systems 29(10):4569–4578, DOI 10.1109/TNNLS.2017.2764939

Mori U, Mendiburu A, Miranda I, Lozano J (2019) Early classification of time series using multi-objective optimization techniques. Information Sciences 492:204 – 218, DOI 10.1016/j.ins.2019.04.024, URL http://www.sciencedirect.com/science/article/pii/S0020025519303317

Nemenyi P (1963) Distribution-free Multiple Comparisons. Princeton University, URL https://books.google.nl/books?id=nhDMtgAACAAJ

Olson RS, Bartley N, Urbanowicz RJ, Moore JH (2016) Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, New York, NY, USA, GECCO '16, pp 485–492, DOI 10.1145/2908812.2908918

Parrish N, Anderson HS, Gupta MR, Hsiao DY (2013) Classifying with confidence from incomplete information. Journal of Machine Learning Research 14:3561–3589, URL http://jmlr.org/papers/v14/parrish13a.html

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830

Rakthanmanon T, Keogh E (2013) Fast shapelets: A scalable algorithm for discovering time series shapelets. In: Proceedings of the 2013 SIAM International Conference on Data Mining, pp 668–676, DOI 10.1137/1.9781611972832.74

Renard X, Rifqi M, Erray W, Detyniecki M (2015) Random-shapelet: An algorithm for fast shapelet discovery. In: 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp 1–10, DOI 10.1109/DSAA.2015.7344782

Rodríguez Diez JJ, Alonso González CJ (2002) Boosting Interval-Based Literals: Variable Length and Early Classification, World Scientific, pp 149–171. DOI 10.1142/9789812565402_0007

Schäfer P, Leser U (2017) Fast and accurate time series classification with WEASEL. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, Association for Computing Machinery, New York, NY, USA, CIKM '17, p 637–646, DOI 10.1145/3132847.3132980

Schäfer P (2015) The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery 29, DOI 10.1007/s10618-014-0377-7

Schäfer P, Leser U (2020) TEASER: Early and accurate time series classification. Data Mining and Knowledge Discovery 34:1336–1362, DOI 10.1007/s10618-020-00690-z

Snoek J, Larochelle H, Adams RP (2012) Practical bayesian optimization of machine learning algorithms. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds) Advances in Neural Information Processing Systems 25, Curran Associates, Inc., pp 2951–2959, URL http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

Tari S, Szczepanski N, Mousin L, Jacques J, Kessaci ME, Jourdan L (2020) Multi-objective automatic algorithm configuration for the classification problem of imbalanced data. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp 1–8, DOI 10.1109/CEC48606.2020.9185785

Thornton C, Hutter F, Hoos HH, Leyton-Brown K (2013) Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In: Dhillon IS, Koren Y, Ghani R, Senator TE, Bradley P, Parekh R, He J, Grossman RL, Uthurusamy R (eds) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013, ACM, pp 847–855, DOI 10.1145/2487575.2487629, URL http://doi.acm.org/10.1145/2487575.2487629

Wang X, Smith-Miles K, Hyndman R (2006) Characteristic-based clustering for time series data. Data Mining and Knowledge Discovery 13:335–364, DOI 10.1007/s10618-005-0039-x

Xing Z, Pei J, Yu PS, Wang K (2011a) Extracting interpretable features for early classification on time series. In: Proceedings of the Eleventh SIAM International Conference on Data Mining, SIAM / Omnipress, pp 247–258, DOI 10.1137/1.9781611972818.22

Xing Zz, Pei J, Yu P (2011b) Early classification on time series. Knowledge and Information Systems 31, DOI 10.1007/s10115-011-0400-x

Ye L, Keogh E (2011) Time series shapelets: A novel technique that allows accurate, interpretable and fast classification. Data Mining and Knowledge Discovery 22:149–182, DOI 10.1007/s10618-010-0179-5

Zhao C, Liu S, Pan L, Ji C, Yang C (2019) Selecting superior candidates from a suitable set: A selective extraction algorithm for accelerating shapelet discovery in time series data. In: 2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp 404–409, DOI 10.1109/CSCWD.2019.8791861

Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation 8(2):173–195, DOI 10.1162/106365600568202

Appendix A

Additional, detailed results

This appendix provides tables of per-dataset median values of the three Pareto set performance metrics that were computed. These metrics being hypervolume (see Section 5.4), Δ-spread (Eq. 5.2) and HM metric (Eq. 5.1). The reported values are the medians of the 1000 bootstrap subsamples of size 10, resampled from 25 runs (see Section 5.3). In some cases, not a single viable configuration was found in any of the subsamples. In these cases the table entry is left blank.

method	Fixed	ECTS	EDSC	ECDIRE	SR-CF	RelClass	TEASER	ECEC	EARLIEST	SO-All	MultiETSC
dataset											
ACSF1	0.674	0.085			0.712		0.747	0.728	0.360	**0.748**	0.717
Adiac	0.562	0.162					**0.591**		0.076	0.526	0.563
AllGestureWiimoteX	0.100	0.000					**0.399**	0.332	0.160	0.306	0.363
AllGestureWiimoteY	0.100	0.011					**0.472**	0.409	0.185	0.350	0.415
AllGestureWiimoteZ	0.100	0.007					**0.379**	0.306	0.150	0.317	0.340
ArrowHead	0.661	0.129	0.523	0.639	0.551	0.655	0.731	**0.733**	0.381	0.601	0.682
BME	0.616	0.185	0.604	**0.732**	0.706	0.683	0.724	0.701	0.351	0.642	0.726
Beef	0.503	0.125	0.430	0.487	0.603	0.614	**0.744**	0.684	0.399	0.680	0.742
BeetleFly	0.747	0.203	0.463	0.617	0.617	0.549	**0.827**	0.645	0.748	0.633	0.625
BirdChicken	0.710	0.223	0.530	0.630	0.586	0.731	**0.792**	0.692	0.549	0.665	0.775
CBF	0.748	0.192	0.683	0.763	0.808	0.763	0.820	0.810	0.442	0.780	**0.823**
Car	0.678	0.179	0.557	0.658	0.602	0.507	**0.863**	0.819	0.283	0.763	0.836
Chinatown	**0.945**	0.169	0.763	0.927	0.925	0.854	0.827	0.821	0.931	0.933	**0.945**
ChlorineConcentration	0.593	0.191				0.627	0.667	0.654	0.533	0.648	**0.667**
CinCECGTorso	0.701	0.394		0.556	0.596	0.177	0.835	0.779	0.327	0.788	**0.847**
Coffee	0.893	0.280	0.826	0.818	0.844	**0.898**	0.862	0.874	0.534	0.816	0.895
Computers	0.575	0.049		0.593	0.593	0.511	0.744	0.738	0.531	0.741	**0.745**
CricketX	0.484	0.143				0.102	0.503	0.373	0.153	0.443	**0.534**
CricketY	0.490	0.124				0.315	**0.515**	0.419	0.166	0.446	0.504
CricketZ	0.493	0.134				0.102	**0.557**	0.343	0.148	0.464	0.497
DistalPhalanxOutlineAgeGroup	0.632	0.105	0.616			0.671	0.685	0.667	0.667	0.652	**0.707**
DistalPhalanxOutlineCorrect	0.693	0.107	0.650	0.661	0.661	0.665	0.744	0.713	0.648	0.675	**0.757**
DistalPhalanxTW	0.564	0.129	0.549			**0.631**	0.596	0.596	0.298	0.601	0.626
DodgerLoopDay	0.150						**0.420**	0.406		0.399	0.412
DodgerLoopGame	0.522						**0.527**	0.516			0.522
DodgerLoopWeekend	0.739						0.901	**0.927**		0.875	0.739
ECG200	0.854	0.362	0.737	0.812	0.817	0.795	0.849	0.832	0.633	0.846	**0.860**
ECG5000	0.904	0.070	0.718			0.851	0.900	0.893	0.610	0.902	**0.915**
ECGFiveDays	0.612	0.215	0.529	0.583	0.604	0.613	0.759	0.668	0.570	0.600	**0.764**
EOGHorizontalSignal	**0.306**	0.102					0.128	0.101	0.102	0.160	0.299
EOGVerticalSignal	0.295	0.068					0.086	0.082	0.086	0.213	**0.296**
Earthquakes	**0.748**	0.008		0.711	0.711	0.747	0.736	0.744	0.747	0.747	0.747
EthanolLevel	0.272	0.028				0.264	**0.382**	0.380	0.252	0.376	0.379
FaceAll	0.698	0.229				0.093	0.785	0.723	0.258	0.738	**0.786**
FaceFour	0.763	0.189	0.618	0.539	0.759	0.808	0.765	0.758	0.306	0.749	**0.811**
FacesUCR	0.647	0.077	0.521			0.351	**0.781**	0.723	0.192	0.705	0.751
Fish	0.696	0.218			0.353	0.581	**0.844**	0.777	0.308	0.798	0.758
FreezerRegularTrain	0.937	0.146	0.822	0.911	0.912	0.762	0.940	0.933	0.521	0.942	**0.947**
FreezerSmallTrain	0.724	0.209	0.828	0.642	0.636	0.763	0.900	0.926	0.750	0.750	**0.933**
GestureMidAirD1	0.226	0.018					**0.281**	0.177	0.153	0.262	0.240
GestureMidAirD2	0.161	0.001					**0.233**	0.138	0.069	0.229	0.222
GestureMidAirD3	0.139	0.003					0.138	0.085	0.084	0.134	**0.141**
GesturePebbleZ1	0.243	0.046					**0.446**	0.413	0.180	0.369	0.419
GesturePebbleZ2	0.249	0.040					**0.375**	0.292	0.164	0.308	0.357
GunPoint	0.881	0.491	0.742	0.825	0.816	0.776	0.856	0.873	0.662	0.803	**0.899**
GunPointAgeSpan	0.936	0.461	0.819	0.894	0.886	0.808	0.937	0.927	0.503	0.905	**0.968**
GunPointMaleVersusFemale	0.934	0.615	0.898	0.895	0.867	0.906	0.944	0.939	0.522	0.908	**0.958**
GunPointOldVersusYoung	0.976	0.675	0.956	0.917	0.936	0.974	0.965	0.967	0.520	0.952	**0.984**
Ham	0.635	0.071	0.424	0.664	0.656	0.713	0.719	0.715	0.513	0.653	**0.723**
HandOutlines	0.843	0.109		0.664	0.811		0.657	0.659	0.640	0.826	**0.844**
Haptics	0.378	0.016			0.355	0.211	0.381	**0.385**	0.260	0.358	0.377
Herring	0.594	0.058	0.521	0.564	0.623	**0.643**	0.611	0.621	0.593	0.593	0.624
HouseTwenty	0.703	0.055		0.735	0.716	0.487	0.914	**0.917**	0.580	0.880	0.912
InlineSkate	0.265	0.037		0.083	0.259	0.120	0.357	**0.364**	0.218	0.319	0.314
InsectEPGRegularTrain	0.995	0.555	0.991	0.926	0.950	0.998	0.946	0.931	0.998	0.998	**0.999**
InsectEPGSmallTrain	0.995	0.280	0.995	0.933	0.950	0.998	0.837	0.839	0.998	0.998	**0.999**
ItalyPowerDemand	0.791	0.200	0.666	0.787	0.793	0.794	0.712	0.709	0.658	0.682	**0.822**
LargeKitchenAppliances	0.464	0.070				0.341	**0.537**	0.528	0.333	0.439	0.527
Lightning2	0.685	0.079	0.663	0.573	0.639	0.591	**0.694**	0.665	0.671	0.671	0.693
Lightning7	0.452	0.060	0.443	0.194	0.449	0.490	**0.577**	0.509	0.355	0.495	0.532
Mallat	0.684	0.230		0.380	0.554	0.230	0.585	0.550	0.125	0.587	**0.686**
Meat	0.861	0.482	0.708	0.870	0.836	0.897	0.914	**0.926**	0.333	0.851	0.918
MedicalImages	0.693	0.291	0.546			0.587	0.691	0.664	0.509	0.678	**0.698**
MelbournePedestrian	0.100						**0.719**	0.684		0.654	0.706
MiddlePhalanxOutlineAgeGroup	0.514	0.107	0.561			0.553	**0.574**	0.526	0.564	0.557	0.567
MiddlePhalanxOutlineCorrect	0.689	0.130	0.613	0.642	0.711	**0.768**	0.729	0.674	0.563	0.647	0.752
MiddlePhalanxTW	0.467	0.081	0.453			**0.553**	0.508	0.497	0.526	0.494	0.511
MixedShapesRegularTrain	0.801	0.221				0.136	**0.850**	0.820	0.276	0.833	0.840
MixedShapesSmallTrain	0.715	0.240		0.567	0.708	0.182	**0.833**	0.731	0.262	0.725	0.822
MoteStrain	0.810	0.109	0.544	0.728	0.754	0.801	0.842	0.794	0.674	0.670	**0.858**
NonInvasiveFetalECGThorax1	**0.787**	0.142							0.063	0.741	0.765
NonInvasiveFetalECGThorax2	**0.849**	0.172							0.104	0.821	0.838
OSULeaf	0.498	0.077			0.324	0.373	**0.684**	0.617	0.313	0.596	0.653
OliveOil	0.800	0.384	0.673	0.672	0.789	0.874	**0.923**	0.866	0.399	0.852	0.919
PLAID	0.544	0.002					**0.715**	0.647	0.272	0.676	0.690
PhalangesOutlinesCorrect	0.706	0.120			0.617	**0.745**	0.718	0.703	0.633	0.707	0.743
PickupGestureWiimoteZ	0.505	0.001					**0.670**	0.632	0.180	0.637	0.668
Plane	0.933	0.560	0.808		0.923	0.867	0.930	0.942	0.586	0.899	**0.948**
PowerCons	0.807	0.199	0.606	0.801	0.805	0.745	0.830	0.764	0.563	0.713	**0.852**
ProximalPhalanxOutlineAgeGroup	0.766	0.140	0.790			0.778	0.818	0.800	0.780	0.820	**0.839**
ProximalPhalanxOutlineCorrect	0.768	0.131	0.678	0.714	0.771	0.746	0.817	0.793	0.702	0.725	**0.836**
ProximalPhalanxTW	0.693	0.115	0.726			0.654	0.770	0.756	0.347	0.761	**0.773**
RefrigerationDevices	0.434	0.032			0.466	0.272	**0.552**	0.550	0.333	0.549	0.547
Rock	0.456	0.145		0.487	0.380		**0.629**	0.459	0.420	0.555	0.449
ScreenType	0.409	0.022				0.309	0.402	**0.412**	0.338	0.400	0.394
SemgHandGenderCh2	0.793	0.129		0.810	0.820	0.673	0.694	0.695	0.551	0.800	**0.838**
SemgHandMovementCh2	**0.501**	0.059					0.321	0.327	0.233	0.452	0.500
SemgHandSubjectCh2	**0.617**	0.082				0.222	0.350	0.335	0.266	0.536	0.617
ShakeGestureWiimoteZ	0.595	0.000					0.671	0.629	0.239	0.623	**0.704**
ShapeletSim	0.505	0.018	0.446	0.475	0.491	0.504	**0.705**	0.651	0.499	0.503	0.630
ShapesAll	**0.647**	0.230				0.037			0.070	0.541	0.647
SmallKitchenAppliances	0.560	0.026				0.450	0.719	0.718	0.333	0.721	**0.721**
SmoothSubspace	0.702	0.069		0.518	0.730	0.758	0.671	0.659	0.311	0.614	**0.796**
SonyAIBORobotSurface1	0.761	0.152	0.586	0.750	0.676	**0.833**	0.800	0.810	0.618	0.745	0.801
SonyAIBORobotSurface2	0.754	0.366	0.502	0.700	0.657	0.730	0.748	0.717	0.660	0.657	**0.792**
StarLightCurves	0.887	0.104					0.883	0.884	0.826	0.882	**0.893**
Strawberry	0.928	0.449		0.824	0.881	0.859	0.931	0.922	0.641	0.908	**0.941**
SwedishLeaf	0.714	0.139				0.290	**0.785**		0.260	0.719	0.778
SyntheticControl	0.788	0.068	0.593			0.819	0.889	0.860	0.643	0.858	**0.900**
ToeSegmentation1	0.624	0.057	0.702	0.521	0.525	0.567	**0.827**	0.803	0.507	0.513	0.762
ToeSegmentation2	0.829	0.205	0.559	0.774	0.623	0.664	0.789	0.834	0.483	0.658	**0.840**
Trace	0.745	0.334	0.668	0.622	0.712	0.622	**0.865**	0.729	0.667	0.736	0.857
TwoLeadECG	0.776	0.214	0.727	0.802	0.827	0.720	0.819	0.843	0.611	0.829	**0.873**
TwoPatterns	**0.516**	0.121				0.456	0.456	0.362	0.259	0.425	0.486
UMD	0.576	0.158	0.532	0.589	0.682	0.594	**0.742**	0.637	0.338	0.593	0.722
UWaveGestureLibraryAll	**0.784**	0.156					0.386		0.285	0.674	0.779
UWaveGestureLibraryX	0.574	0.069				0.111	0.478	0.262	0.291	0.523	**0.583**
UWaveGestureLibraryY	0.519	0.056				0.147	0.443		0.279	0.462	**0.523**
UWaveGestureLibraryZ	0.545	0.064				0.152	0.436		0.284	0.489	**0.547**
Wafer	0.989	0.558		0.920	0.920	0.975	0.979	0.975	0.886	0.986	**0.997**
Wine	0.640	0.471	0.670	0.623	0.581	**0.717**	0.694	0.694	0.498	0.554	0.678
WordSynonyms	0.468	0.068				0.048	0.460	0.374	0.219	0.425	**0.474**
Worms	0.499	0.070			0.494	0.311	0.637	0.593	0.428	0.644	**0.649**
WormsTwoClass	0.711	0.078		0.538	0.565	0.571	0.689	0.568	0.571	0.596	**0.721**
Yoga	0.769	0.201		0.748	0.745	0.575	0.815	0.802	0.545	0.775	**0.817**

Table A.1: Median hypervolume metric for compared methods on different datasets in the UCR Archive. Best values are marked. Cases where no configurations are found are left blank.

method	Fixed	ECTS	EDSC	ECDIRE	SR-CF	RelClass	TEASER	ECEC	EARLIEST	SO-All	MultiETSC
dataset											
ACSF1	0.663				**0.530**	0.965	0.942	0.901	0.832	0.924	0.872
Adiac	1.019	0.938					**0.656**		0.975	0.894	0.968
AllGestureWiimoteX							0.762	0.838	0.949	0.735	0.842
AllGestureWiimoteY		0.983					0.750	0.848	0.921	0.974	0.725
AllGestureWiimoteZ							0.805	0.863	0.964	0.771	0.810
ArrowHead	0.918	0.996	0.822	0.872	0.813	0.887	0.838	0.825	0.982	0.892	**0.800**
BME	0.896	0.919	0.925	0.799	0.863	**0.737**	0.764	0.786	0.976	0.898	0.833
Beef	0.784		0.915	0.764	**0.758**	0.792	0.818	0.922	0.956	0.769	0.773
BeetleFly	0.942	0.892	0.936	0.941		**0.440**	0.718	0.939	0.859	0.941	0.882
BirdChicken	0.913	0.908	0.859	0.877	**0.704**	0.723	0.711	0.921	0.970	0.882	0.843
CBF	0.895	0.875	0.917	0.975	0.918	0.905	**0.823**	0.909	0.995	0.876	0.883
Car	**0.750**	0.981	0.933	0.895	0.936	0.924	0.797	0.851	0.946	0.898	0.756
Chinatown	**0.638**	0.956	0.958			0.989	0.978	0.977	0.870	0.951	**0.638**
ChlorineConcentration	0.975	0.998				**0.683**	0.956	0.906	0.998	0.944	0.848
CinCECGTorso	0.869	0.971		0.874	0.957	1.000	0.867	0.934	0.996	0.950	**0.847**
Coffee	0.853		0.845	0.852	**0.827**	0.838	0.853	0.858	0.957	0.959	0.872
Computers	**0.960**	0.974		0.988	0.975	0.998	1.125	0.970	0.971	0.965	0.971
CricketX	0.728	0.953				**0.419**	0.867	0.888	0.996	0.937	0.794
CricketY	0.754					**0.752**	0.876	0.858	0.983	0.921	0.828
CricketZ	0.793	0.977				**0.363**	0.788	0.897	0.993	0.900	0.811
DistalPhalanxOutlineAgeGroup	0.907	0.943	0.913			0.947	0.758	**0.682**	0.878	0.884	0.790
DistalPhalanxOutlineCorrect	0.941	0.958	0.844	0.876	**0.629**	1.018	0.841	0.796	0.981	0.954	0.836
DistalPhalanxTW	0.872	0.989	0.875			0.930	0.851	**0.655**		0.938	0.842
DodgerLoopDay							0.748	0.893		0.853	0.843
DodgerLoopGame							0.865	0.791		0.861	0.508
DodgerLoopWeekend							0.852	0.953		0.994	0.885
ECG200	0.933	0.962	0.934	0.948	0.917	**0.847**	0.930	1.024	0.994	0.938	0.898
ECG5000	0.900	0.988	**0.856**			0.868	1.098	0.968	0.993	0.934	0.995
ECGFiveDays	0.977	0.856	0.918	0.940		**0.785**	0.890	1.037	0.991	0.943	0.938
EOGHorizontalSignal	**0.828**	0.966				0.998	0.940	0.982	0.987	0.960	0.829
EOGVerticalSignal	0.912	0.990					0.953	0.983	0.996	0.964	**0.892**
Earthquakes		0.995					0.772				0.579
EthanolLevel	0.978					0.995	1.021	**0.958**	0.996	0.979	0.984
FaceAll	0.766	0.955				**0.364**	0.774	0.991	0.966	0.987	0.803
FaceFour	0.879		0.864	0.817	**0.661**	0.887	0.826	0.919	0.978	0.905	0.858
FacesUCR	**0.729**	0.968	0.773			0.906	0.775	0.933	0.997	0.778	0.803
Fish	0.773	0.949				0.835	0.771	0.939	0.891	0.863	**0.766**
FreezerRegularTrain	0.878		0.943	0.889	1.035	**0.800**	0.962	1.037	0.986	0.922	0.939
FreezerSmallTrain	**0.802**		0.898	0.975	0.950	0.998	1.027	0.963	0.996	0.815	0.814
GestureMidAirD1	0.896	0.968					**0.817**	0.873	0.965	0.835	0.945
GestureMidAirD2	0.896						**0.790**	0.901	0.979	0.834	0.906
GestureMidAirD3	0.956	0.996					**0.807**	0.875	0.984	0.871	0.978
GesturePebbleZ1	0.958	0.917					**0.673**	0.808	0.980	0.718	0.807
GesturePebbleZ2	0.945	0.929					**0.680**	0.877	0.991	0.737	0.895
GunPoint	**0.841**	0.944	0.873	0.913	0.926	0.872	0.889	0.872	0.971	0.907	0.851
GunPointAgeSpan	1.040	**0.700**	0.897	0.977	0.956	0.912	1.070	0.910	0.992	0.961	0.953
GunPointMaleVersusFemale	0.930	0.987	0.881	**0.845**	0.952	1.001	1.031	0.983	0.969	0.934	0.898
GunPointOldVersusYoung	1.051		**0.950**	0.988	0.969	0.975	1.079	0.985	0.971	0.990	1.023
Ham	0.936		**0.834**	0.910	0.954	0.902	0.957	0.876	0.982	0.946	0.890
HandOutlines	0.928	0.948		0.975	0.836	0.997	0.990	0.987	**0.835**	0.934	0.912
Haptics	0.896				0.958	**0.880**	0.965	0.954	0.972	0.931	0.937
Herring			0.727		0.991	0.890	0.757	0.959			0.953
HouseTwenty	0.946	0.947		0.953	0.878	0.972	**0.847**	0.917	0.904	0.924	0.920
InlineSkate	0.910	0.978			**0.828**	0.999	0.945	0.988	0.963	0.904	0.971
InsectEPGRegularTrain	**0.736**		0.915				1.026	0.979			0.737
InsectEPGSmallTrain	**0.736**						0.868	0.987	0.797		0.737
ItalyPowerDemand	**0.742**	0.966	0.782	0.899	0.953	0.897	0.809	0.834	0.995	0.979	0.817
LargeKitchenAppliances	**0.860**	0.953					0.979	0.994	0.991	0.941	0.886
Lightning2	0.805	0.943	0.834	0.958	0.982	**0.673**	0.899	0.963	0.981	0.952	0.924
Lightning7	0.823	0.991	0.756	0.905	0.962	0.787	**0.739**	0.965	0.991	0.805	0.789
Mallat	0.889	0.961		0.942	0.912	0.929	0.904	0.930	0.999	0.926	**0.750**
Meat	**0.706**	0.892	0.883	0.942	0.928	0.931	1.061	0.895	0.806	0.983	0.921
MedicalImages	0.911	0.889	**0.835**			0.868	0.850	0.938		0.911	1.059
MelbournePedestrian							0.686	0.938		0.913	0.891
MiddlePhalanxOutlineAgeGroup	0.897	0.962	0.978			0.886	0.959	**0.761**	0.767	0.793	0.802
MiddlePhalanxOutlineCorrect	0.828	0.968	0.828	0.961	0.885	0.945	0.772	0.867		0.941	**0.712**
MiddlePhalanxTW	0.985		0.961			1.028	0.879	0.953	0.920	0.852	**0.786**
MixedShapesRegularTrain	0.940	0.950				0.997	1.034	0.963	0.967	0.965	**0.931**
MixedShapesSmallTrain	**0.831**	0.918		0.920	0.988	0.994	0.937	0.962	0.991	0.977	0.858
MoteStrain	0.951		0.937	0.953	0.871	0.997	0.943	0.944	0.995	**0.834**	0.918
NonInvasiveFetalECGThorax1	1.099	**0.971**							0.994	1.163	1.077
NonInvasiveFetalECGThorax2	1.221	0.925							0.965	0.981	**0.819**
OSULeaf	0.884	0.926			0.953	0.881	0.906	0.890	0.929	0.874	**0.837**
OliveOil	0.830	0.949	0.811	0.828	0.856	0.889	**0.740**	0.859		0.920	0.806
PLAID	0.988	0.999					**0.795**	0.949	0.926	0.824	0.934
PhalangesOutlinesCorrect	0.912	0.972			0.960	0.874	**0.808**	0.842	0.999	0.944	0.860
PickupGestureWiimoteZ	0.739						0.699	0.832	0.974	0.667	**0.571**
Plane	0.944	0.981	0.911		0.995	0.965	1.008	**0.831**	0.966	0.903	0.938
PowerCons	1.006	0.882	0.811	0.862	0.867	0.965	**0.767**	0.892	0.960	0.872	0.780
ProximalPhalanxOutlineAgeGroup	0.928		0.982			0.833	0.958	0.914	0.832	0.942	**0.770**
ProximalPhalanxOutlineCorrect	0.922	0.959	0.845	0.987	0.905	0.878	0.843	0.854	0.992	0.971	**0.820**
ProximalPhalanxTW	0.878	0.982	0.938			0.950	0.943	0.918	**0.778**	0.895	0.797
RefrigerationDevices	0.935	0.995			0.983	0.911	1.025	0.966	0.957	**0.875**	0.929
Rock	0.915	**0.843**		0.923	0.939		0.936	0.980	0.975	0.863	0.878
ScreenType	0.984					**0.683**	0.938	0.972	0.996	0.953	0.977
SemgHandGenderCh2	0.909	0.956		0.871	0.935	0.999	0.993	0.977	0.887	**0.825**	0.876
SemgHandMovementCh2	0.802	0.956					0.976	0.980	0.988	0.842	**0.793**
SemgHandSubjectCh2	0.793	0.960					1.010	0.978	0.962	**0.739**	0.784
ShakeGestureWiimoteZ	0.780						0.677	0.831	0.932	0.769	**0.561**
ShapeletSim	0.969	0.979	0.946		0.969	0.871	**0.642**	0.738	0.997	0.706	0.741
ShapesAll	0.782	0.895							0.967	0.945	**0.768**
SmallKitchenAppliances	0.968						1.082	0.989	0.998	**0.835**	0.895
SmoothSubspace	**0.506**			0.888	0.926	0.725	0.794	0.871	0.871	0.898	0.817
SonyAIBORobotSurface1	0.830	0.955	0.957	0.940	**0.454**	0.972	0.874	0.793	0.986	0.922	0.806
SonyAIBORobotSurface2	0.950	0.977	0.973	0.987	0.912	0.987	0.993	0.974	0.993	0.993	**0.901**
StarLightCurves	**0.950**	0.997					1.168	0.983	0.999	0.975	0.950
Strawberry	0.868	0.870		0.944	0.959	0.911	1.124	0.911		0.905	**0.866**
SwedishLeaf	**0.872**	0.994				1.129	0.899		0.908	0.970	0.885
SyntheticControl	0.716	0.980	0.905			0.838	0.792	0.897	0.955	0.855	**0.656**
ToeSegmentation1	0.912		0.854	0.803	0.957	**0.764**	0.797	0.935	0.997	0.937	0.899
ToeSegmentation2	0.920	0.954	0.861	0.887	**0.625**	0.849	0.694	0.848	0.844	0.931	0.769
Trace	0.646	0.891	0.845	0.748	0.942	0.753	**0.619**	0.966	0.942	0.948	0.673
TwoLeadECG	0.846	0.835	0.850	**0.724**	0.766	0.940	0.920	0.774	0.993	0.833	0.871
TwoPatterns	**0.588**	0.998				0.850	0.833	0.917	0.997	0.840	0.948
UMD	0.937	0.941	0.901	0.854	0.858	0.699	0.703	0.808	0.977	0.837	**0.675**
UWaveGestureLibraryAll	**0.616**	0.970					0.867	0.993	0.997	0.997	0.649
UWaveGestureLibraryX	**0.682**	0.984				0.996	0.913	0.951	0.995	0.901	0.769
UWaveGestureLibraryY	**0.739**	0.996				0.998	0.855	0.879	0.999	0.919	0.821
UWaveGestureLibraryZ	**0.766**	0.984				1.000	0.915		0.992	0.948	0.819
Wafer	0.951	0.989		0.998	0.991	**0.939**	1.198	0.992		0.954	0.944
Wine	0.936	**0.577**	0.887	0.882	0.876	0.950	0.856	0.883	0.966	0.950	0.892
WordSynonyms	0.770	0.965				**0.304**	0.854	0.964		0.817	0.831
Worms	0.951				0.951	0.935	0.960	0.933		**0.897**	0.919
WormsTwoClass	0.908			0.907	**0.754**		0.978	0.936	0.914	0.914	0.884
Yoga	1.040	0.970		0.937	0.981	**0.915**	0.955	0.919	0.994	0.926	0.923

Table A.2: Median Δ-spread for compared methods on different datasets in the UCR Archive. Best values are marked. Cases where no configurations are found are left blank.

method	Fixed	ECTS	EDSC	ECDIRE	SR-CF	RelClass	TEASER	ECEC	EARLIEST	SO-All	MultiETSC
dataset											
ACSF1	0.194	0.759			0.162		0.153	0.158	0.471	**0.144**	0.165
Adiac	0.330	0.630					**0.262**		0.858	0.325	0.330
AllGestureWiimoteX	0.818	0.992					**0.501**	0.533	0.724	0.539	0.534
AllGestureWiimoteY	0.818	0.937					**0.454**	0.471	0.687	0.471	0.475
AllGestureWiimoteZ	0.818	0.949					**0.526**	0.556	0.739	0.535	0.556
ArrowHead	0.300	0.733	0.354	0.303	0.314	0.280	**0.243**	0.250	0.447	0.292	0.292
BME	0.264	0.627	0.231	0.267	**0.231**	0.291	0.253	0.238	0.479	0.265	0.236
Beef	0.465	0.707	0.379	0.349	0.317	0.355	0.192	0.198	0.429	0.196	**0.166**
BeetleFly	0.150	0.603	0.332	0.228	0.228	0.291	0.148	0.219	**0.144**	0.221	0.251
BirdChicken	0.303	0.551	0.312	0.277	0.324	0.213	0.177	0.188	0.291	0.205	**0.169**
CBF	0.210	0.643	0.216	0.193	**0.179**	0.243	0.218	0.189	0.385	0.185	0.180
Car	0.303	0.642	0.287	0.330	0.269	0.369	0.172	0.200	0.559	**0.168**	0.189
Chinatown	**0.034**	0.701	0.144	0.037	0.038	0.076	0.091	0.094	0.035	**0.034**	**0.034**
ChlorineConcentration	0.305	0.590				0.306	0.277	0.286	0.303	0.300	**0.276**
CinCECGTorso	0.337	0.411		0.382	0.328	0.700	0.126	0.136	0.507	0.124	**0.123**
Coffee	0.104	0.562	0.157	0.175	0.149	0.112	0.119	0.118	0.303	0.109	**0.103**
Computers	0.280	0.852		0.247	0.247	0.323	0.154	0.155	0.306	0.153	**0.152**
CricketX	0.415	0.653				0.814	0.343	0.439	0.733	0.408	**0.324**
CricketY	0.418	0.687				0.541	**0.341**	0.385	0.714	0.409	0.378
CricketZ	0.385	0.672				0.814	**0.309**	0.470	0.741	0.361	0.385
DistalPhalanxOutlineAgeGroup	0.245	0.731	0.238			0.279	0.217	0.216	**0.198**	0.216	0.216
DistalPhalanxOutlineCorrect	0.217	0.752	0.229	0.197	0.197	0.281	0.205	0.205	0.212	0.197	**0.187**
DistalPhalanxTW	0.300	0.684	0.335			**0.251**	0.267	0.258	0.537	0.265	0.258
DodgerLoopDay	0.739						**0.443**	0.451		0.459	0.451
DodgerLoopGame	**0.314**						0.330	0.317			**0.314**
DodgerLoopWeekend	0.150						0.056	**0.055**		0.065	0.150
ECG200	0.138	0.446	0.165	0.134	0.125	0.197	0.129	0.131	0.223	**0.124**	0.133
ECG5000	0.092	0.862	0.156			0.102	**0.069**	0.070	0.243	0.092	0.090
ECGFiveDays	0.246	0.599	0.344	0.255	0.239	0.335	**0.186**	0.311	0.272	0.248	0.246
EOGHorizontalSignal	**0.571**	0.700					0.771	0.815	0.815	0.653	**0.571**
EOGVerticalSignal	**0.552**	0.756					0.842	0.847	0.842	0.568	0.558
Earthquakes	**0.144**	0.978		0.163	0.163	0.145	0.153	0.146	0.145	0.145	0.145
EthanolLevel	0.581	0.855				0.582	**0.449**	0.450	0.597	0.456	0.452
FaceAll	0.236	0.573				0.774	0.174	**0.157**	0.588	0.236	0.236
FaceFour	0.198	0.640	0.268	0.314	**0.155**	0.165	0.178	0.170	0.531	0.173	0.173
FacesUCR	0.319	0.819	0.409			0.542	**0.217**	0.229	0.676	0.232	0.236
Fish	0.266	0.591			0.466	0.301	0.190	0.196	0.529	**0.185**	0.194
FreezerRegularTrain	0.114	0.717	**0.112**	0.116	0.115	0.284	0.129	0.127	0.315	0.115	0.114
FreezerSmallTrain	0.201	0.587	0.149	0.211	0.215	0.158	0.145	**0.137**	0.143	0.142	0.142
GestureMidAirD1	0.622	0.894					**0.573**	0.680	0.733	0.599	0.615
GestureMidAirD2	0.710	0.980					**0.624**	0.742	0.871	0.630	0.626
GestureMidAirD3	0.741	0.959					**0.724**	0.832	0.844	0.752	0.741
GesturePebbleZ1	0.615	0.806					**0.502**	0.503	0.695	0.503	0.507
GesturePebbleZ2	0.611	0.827					**0.553**	0.559	0.717	0.593	0.559
GunPoint	0.171	0.331	0.184	**0.168**	0.169	0.254	0.177	0.180	0.202	0.187	0.172
GunPointAgeSpan	**0.063**	0.360	0.231	0.073	0.085	0.180	0.096	0.099	0.329	0.065	**0.063**
GunPointMaleVersusFemale	0.132	0.235	0.164	0.122	0.149	0.145	**0.094**	0.119	0.313	0.105	0.105
GunPointOldVersusYoung	0.037	0.193	0.046	0.052	0.052	0.034	**0.025**	0.026	0.314	0.027	0.027
Ham	0.257	0.806	0.365	0.256	0.256	0.254	0.221	**0.216**	0.321	0.218	0.226
HandOutlines	**0.172**	0.780		0.201	0.176		0.207	0.206	0.219	0.191	**0.172**
Haptics	0.477	0.924			0.464	0.652	0.447	**0.443**	0.588	0.480	0.487
Herring	0.255	0.825	0.292	0.269	**0.224**	0.271	0.251	0.237	0.255	0.255	0.244
HouseTwenty	0.207	0.854		0.213	0.173	0.334	**0.084**	0.084	0.266	0.092	0.099
InlineSkate	0.626	0.832		0.825	0.593	0.786	0.476	**0.466**	0.642	0.516	0.522
InsectEPGRegularTrain	0.005	0.286	0.004	0.038	0.026	**0.001**	0.039	0.037	0.001	**0.001**	**0.001**
InsectEPGSmallTrain	0.005	0.563	0.003	0.034	0.026	**0.001**	0.090	0.087	0.001	**0.001**	**0.001**
ItalyPowerDemand	0.226	0.653	0.272	**0.186**	0.192	0.213	0.251	0.251	0.201	**0.186**	0.192
LargeKitchenAppliances	0.371	0.778				0.491	**0.303**	0.309	0.500	0.389	0.310
Lightning2	0.298	0.806	0.226	0.261	0.213	0.271	0.215	0.211	**0.197**	**0.197**	0.197
Lightning7	0.411	0.828	0.408	0.619	**0.378**	0.389	0.389	0.391	0.475	0.401	0.389
Mallat	**0.284**	0.600		0.432	0.294	0.624	0.312	0.328	0.778	0.288	0.290
Meat	0.119	0.354	0.218	0.122	0.112	0.108	0.084	0.077	0.500	0.090	**0.077**
MedicalImages	**0.216**	0.478	0.317			0.335	0.229	0.237	0.323	0.225	**0.216**
MelbournePedestrian	0.817						0.264	0.267		**0.262**	0.270
MiddlePhalanxOutlineAgeGroup	0.351	0.710	**0.265**			0.318	0.289	0.309	0.277	0.309	0.309
MiddlePhalanxOutlineCorrect	0.259	0.722	0.251	0.231	**0.224**	0.241	0.238	0.270	0.277	0.238	0.238
MiddlePhalanxTW	0.382	0.762	0.401			**0.287**	0.351	0.336	0.308	0.336	0.349
MixedShapesRegularTrain	0.230	0.611				0.760	**0.094**	0.105	0.568	0.100	0.106
MixedShapesSmallTrain	0.279	0.566		0.319	0.241	0.691	**0.156**	0.160	0.585	0.163	0.163
MoteStrain	0.147	0.782	0.297	0.170	0.162	0.163	0.133	**0.130**	0.193	0.196	0.143
NonInvasiveFetalECGThorax1	**0.152**	0.715							0.881	0.153	**0.152**
NonInvasiveFetalECGThorax2	0.126	0.680							0.811	**0.125**	0.128
OSULeaf	0.388	0.762			0.501	0.454	0.273	0.269	0.522	0.277	**0.265**
OliveOil	0.205	0.426	0.258	0.240	0.164	**0.094**	0.104	0.101	0.429	0.097	0.110
PLAID	0.308	0.975					**0.219**	0.224	0.573	0.224	0.227
PhalangesOutlinesCorrect	0.240	0.738			0.229	0.227	0.230	0.232	**0.223**	0.230	0.238
PickupGestureWiimoteZ	0.331	0.970					**0.300**	0.314	0.695	0.310	0.316
Plane	0.087	0.279	0.126		**0.076**	0.112	0.086	0.101	0.259	0.098	0.095
PowerCons	**0.171**	0.657	0.281	0.172	0.176	0.311	0.186	0.192	0.279	0.176	0.176
ProximalPhalanxOutlineAgeGroup	0.144	0.712	0.116			0.208	0.116	0.120	0.123	0.119	**0.109**
ProximalPhalanxOutlineCorrect	0.188	0.729	0.229	**0.167**	0.171	0.243	0.175	0.177	0.173	0.175	0.168
ProximalPhalanxTW	0.217	0.733	**0.166**			0.284	0.167	0.169	0.482	0.169	0.175
RefrigerationDevices	0.396	0.868			0.353	0.542	0.294	0.304	0.500	**0.293**	0.294
Rock	0.391	0.636		0.345	0.437		**0.254**	0.370	0.409	0.292	0.408
ScreenType	0.437	0.898				0.520	0.432	**0.427**	0.494	0.436	0.437
SemgHandGenderCh2	0.212	0.751		**0.182**	0.184	0.195	0.201	0.195	0.289	0.194	**0.182**
SemgHandMovementCh2	**0.384**	0.830					0.515	0.504	0.622	0.390	**0.384**
SemgHandSubjectCh2	**0.311**	0.814				0.636	0.486	0.499	0.579	0.318	**0.311**
ShakeGestureWiimoteZ	0.265	0.995					**0.251**	0.275	0.613	0.253	0.263
ShapeletSim	0.333	0.934	0.349	0.345	0.331	0.329	0.330	**0.316**	0.334	0.334	0.331
ShapesAll	**0.310**	0.566				0.929			0.869	**0.310**	**0.310**
SmallKitchenAppliances	0.285	0.876				0.379	0.164	0.167	0.500	**0.163**	0.167
SmoothSubspace	**0.277**	0.865		0.306	0.282	0.286	0.291	0.301	0.509	**0.277**	**0.277**
SonyAIBORobotSurface1	0.142	0.672	0.235	0.141	0.183	**0.108**	0.124	0.124	0.234	0.155	0.142
SonyAIBORobotSurface2	0.201	0.427	0.301	**0.176**	0.200	0.178	0.181	0.181	0.204	0.205	0.181
StarLightCurves	0.109	0.787					**0.075**	0.076	0.095	0.078	0.085
Strawberry	0.111	0.368		0.131	0.116	0.142	0.097	0.109	0.218	0.106	**0.095**
SwedishLeaf	0.259	0.710				0.665	**0.173**		0.585	0.223	0.259
SyntheticControl	0.201	0.859	0.308			0.192	**0.146**	0.164	0.235	0.169	**0.146**
ToeSegmentation1	0.299	0.852	0.257	0.323	0.301	0.317	0.243	**0.242**	0.326	0.319	0.269
ToeSegmentation2	**0.102**	0.629	0.316	0.187	0.300	0.316	0.208	0.114	0.348	0.218	**0.102**
Trace	0.154	0.442	0.274	0.225	0.208	0.384	**0.144**	0.156	0.199	0.156	0.178
TwoLeadECG	0.207	0.584	0.232	0.198	0.206	0.229	**0.186**	0.209	0.239	0.198	0.198
TwoPatterns	0.508	0.764				0.455	**0.446**	0.511	0.586	0.463	0.453
UMD	0.302	0.672	0.286	0.275	**0.253**	0.372	0.257	0.300	0.493	0.290	0.310
UWaveGestureLibraryAll	**0.249**	0.718					0.435		0.556	0.250	0.251
UWaveGestureLibraryX	0.384	0.833				0.799	**0.367**	0.568	0.549	0.384	0.384
UWaveGestureLibraryY	0.422	0.848				0.743	**0.391**		0.563	0.423	0.422
UWaveGestureLibraryZ	**0.393**	0.829				0.735	0.401		0.556	0.393	**0.393**
Wafer	**0.013**	0.282		0.041	0.041	0.060	0.021	0.022	0.060	**0.013**	**0.013**
Wine	0.263	0.365	0.209	0.234	0.272	**0.207**	0.208	0.257	0.334	0.269	0.263
WordSynonyms	0.471	0.811				0.867	**0.402**	0.451	0.640	0.433	0.433
Worms	0.365	0.774			0.328	0.526	0.243	0.259	0.400	**0.241**	**0.241**
WormsTwoClass	**0.216**	0.797		0.297	0.318	0.273	0.231	0.277	0.273	0.261	**0.216**
Yoga	0.184	0.621		0.182	0.179	0.293	**0.164**	0.174	0.294	0.169	0.166

Table A.3: Median HM metric for compared methods on different datasets in the UCR Archive. Best values are marked. Cases where no configurations are found are left blank.