IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 11, 2022, accepted March 19, 2022, date of publication April 11, 2022, date of current version April 18, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3166525

ATSC-NEX: Automated Time Series Classification With Sequential Model-Based Optimization and Nested Cross-Validation

MIKKO TAHKOLA AND ZOU GUANGRONG
VTT Technical Research Centre of Finland Ltd., 02044 Espoo, Finland

Corresponding author: Mikko Tahkola (mikko.tahkola@vtt.fi)

This work was supported in part by the Integrated Energy Solutions to Smart and Green Shipping (INTENS) Project, through the Business Finland, under Grant 7582/31/2017; and in part by the VesselAI Project, through the European Union's Horizon 2020 Research and Innovation Programme under Grant 957237.

ABSTRACT New methods to perform time series classification arise frequently and multiple state-of-the-art approaches achieve high performance on benchmark datasets with respect to accuracy and computation time. However, often the modeling procedures do not include proper validation but rather rely only on either external test dataset or one-level cross-validation. ATSC-NEX is an automated procedure that employs sequential model-based optimization together with nested cross-validation to build an accurate and properly validated time series classification model. It aims to find an optimal pipeline configuration that includes the selection of input type and settings, as well as model type and hyperparameters. The results of a case study in which a model for the identification of diesel engine type is developed, show that the algorithm can efficiently find a well-performing pipeline configuration. The comparison between ATSC-NEX and some state-of-the-art methods on several benchmark datasets shows that similar accuracy can be achieved.

INDEX TERMS Tme series classification, automated machine learning, configuration optimization, sequential model-based optimization, nested cross-validation.

I. INTRODUCTION

In traditional classification, the order of features in the dataset is irrelevant, whereas in time series classification (TSC) the order matters. The goal in time series classification is to identify which predefined class a time series belongs to by using a classifier that has been trained using labeled data. TSC can be based on a single (univariate TSC) or multiple (multivariate TSC) time series.

While there are various ways to approach the TSC problem and several software packages to perform automated machine learning exist, these two topics have been combined only recently by Parmetier *et al.* [1]. Their approach employs an evolutionary algorithm in hyperparameter optimization and multi-fold cross-validation (CV) to select the best hyperparameters and a fixed test dataset to evaluate the resulting performance. The ATSC-NEX algorithm employs sequential model-based optimization (SMBO) together with nested cross-validation (nCV) procedure to build an accurate and properly validated time series classification model. It has been shown that the nCV gives a less biased estimation of the generalization ability, i.e., how well a model optimized with the algorithm works on data that has not been used in its training [2]. In addition, SMBO outperforms several other often used methods such as random and grid search in hyperparameter optimization [3]. The optimization procedure involves the evaluation of various classifiers and their hyperparameters, as well as multiple input types, sampling rates, and time series sequence lengths. The selection of the optimal configuration is done using a multi-fold CV. To the best of our knowledge, no other approach combines these two features in the context of TSC.

The main contributions of this article are A) a detailed description of the ATSC-NEX algorithm, which aims to automate the efficient development of properly validated time series-based classifiers, and B) an evaluation of the performance of the ATSC-NEX algorithm with an engine identification case study and comparing its performance with state-of-the-art models on several benchmark datasets.

The article is organized as follows. In Section II, literature and methods related to TSC and AutoML are briefly discussed. The ATSC-NEX algorithm is presented

The associate editor coordinating the review of this manuscript and approving it for publication was Sergio Consoli.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

in Section III. There, the configuration optimization, nested cross-validation, data preparation, and the used classifiers are presented. In addition, the case study and six selected benchmark datasets are presented in Section III. In Section IV, we present the results of the case study and evaluate the performance of the ATSC-NEX algorithm on the benchmark datasets. Finally, in Section V, conclusions from the study are presented.

II. RELATED WORK

A. TIME SERIES CLASSIFICATION METHODS

Many methods to perform time series classification exists. One is to compare how similar two time series are, based on the distances between individual samples of the time series at a certain timestep. Euclidean distance is the simplest distance-based similarity measure, whereas elastic distance measures, such as dynamic time warping, can take time-wise shifting into account.

Ye and Keogh [4] proposed shapelets for the classification of time series. Shapelet-based methods rely on finding sequences shorter than the whole time series, called shapelets, to represent different classes. Classification with shapelet-based methods is done based on whether a shapelet or shapelets appear in the sample time series or not, regardless of the time-wise location in the time series. Shapelets preserve the shape of the time series as opposed to transformation-based approaches which provide higher-level approximation of the time series [5]. Similar to shapelets, phase-dependent interval-based TSC approaches make use of shorter sequences than the time series [6]. However, it differs from the shapelet-based approach by taking into account the phase of the discriminatory features, i.e., the time-wise location.

In some cases, the appearance of a pattern can not be used to discriminate which class a time series belongs to, but rather the appearing frequency of the pattern. Dictionary-based TSC methods, such as Bag-of-Patterns [7] and Bag of Symbolic Fourier Approximation Symbols [8], base the classification task on histograms that represent the frequency of different patterns appearing in the time series. These methods transform the time series into multiple words that describe patterns in the time series, and the histogram is computed based on how often these words appear in the time series. The transformation is done to reduce the complexity of the problem by approximation using the words [9].

Different TSC methods, or multiple models employing the same method, can be also combined in form of an ensemble model. For example, HIVE-COTE2 is an ensemble of ensembles that is built of dictionary-based Temporal Dictionary Ensemble, interval-based Diverse Representation Canonical Interval Forest, ROCKET-based ensemble, and Shapelet Transform Classifier [10]. Such an ensemble model is useful if some classes can be discriminated from others based on, for example, a shapelet and a repeating pattern that can be identified with dictionary-based approaches [10].

Feature extraction-based approaches, on the other hand, transform the time series classification into a feature-based classification problem [11]. For example, the random convolutional kernel transformation (ROCKET) method generates a large number of random kernels that are used to transform time series [12]. Two features are computed per transformed time series, which can be then be used to train a classifier. Feature-based approaches can be also based on extracting statistical time series characteristics. For example, tsfresh Python library [11] computes 794 features from the time series and selects significant ones with hypothesis tests. These features include, for example, coefficient of fast Fourier transformation, autocorrelation, number of peaks, and skewness [11]. Further description of the different TSC approaches can be found in [13].

Deep learning (DL) and deep neural networks (DNNs) are yet another methods that can be applied to perform TSC [14]. The simplest DNNs architecture, multilayer perceptron, has multiple layers that are built of neurons, which are either linear or nonlinear functions. Each of the neurons in the input layer of the network takes the weighted sum of the input values as input. Similarly, from each of the neurons, the output is fed as input weighted to each neuron in the next layer, and this is repeated until the output layer is reached. Weights are parameters of a neural network which values are learned during the training process. Among multilayer perceptron and other DNN architectures, convolutional neural network and echo state network are DNN architectures that can be used for the TSC in an end-to-end manner, i.e., they can learn to extract and select relevant features from raw data [14].

B. AUTOMATED MACHINE LEARNING

AutoML aims to automate the development of machine learning-based models. It solves an optimization problem, in which a large search space of candidate solutions is evaluated to find the best one. The process includes data preparation, feature engineering, and hyperparameter optimization. Although several AutoML solutions exist, recent surveys on AutoML show that most of the solutions do not directly support time series classification but rather focus on traditional feature-based classification and regression tasks [15], [16]. Nevertheless, automating the TSC can be performed with these packages by conducting feature engineering beforehand to obtain features that can be used in learning. However, recently Parmentier *et al.* [1] presented an AutoML solution for solving TSC problems.

Exceptions to this are DL-based approaches, such as those implemented in mcfly Python package [17] and multilayer perceptron, fully convolutional neural network, and residual network solutions described in [18]. The authors in [14] similarly use multiple DL-based models for the TSC. The common factor of the DL-based approaches is that they aim to learn to distinguish classes directly from raw time series data. However, only mcfly supports automated hyperparameter optimization out of the box. In addition to DL-based TSC

methods, sktime Python package [19], provides interval, distance, shapelet, dictionary-based for the TSC. Recently Parmetier *et al.* [1] presented an AutoML solution for solving TSC problems by utilizing an evolutionary algorithm to search for an optimal algorithm and hyperparameters from the methods included in sktime.

The difference between AutoTSC and ATSC-NEX is that the latter utilizes nCV to estimate the generalization performance of the model before fixing its hyperparameters within a multi-fold CV procedure, and only after that, an external test dataset is used to test the developed model. In the former, hyperparameter optimization and model selection are done within a multi-fold CV procedure, which can produce an optimistic estimation of the generalization performance.

III. METHODS

Automated machine learning aims to automate data preprocessing, feature engineering, hyperparameter optimization, and model selection processes. Here, we call this procedure configuration optimization. The methods that ATSC-NEX uses are described in this section.

A. CONFIGURATION OPTIMIZATION

Most machine learning models include hyperparameters that configure the model itself (e.g., number of layers in an artificial neural network), and ones that affect the learning process (e.g., the learning rate of the optimizer used to train an artificial neural network). Hyperparameters are chosen before the model training process starts, and thus differ from the actual model parameters, which are learned during the model training process. A set of hyperparameters found with optimization are data-specific, meaning that the hyperparameters that produce sufficient results for a dataset, probably won't work well for another dataset. Hence, a procedure called hyperparameter optimization is often included in the process of developing a machine learning-based model [20].

Hyperparameter optimization methods can be divided into model-free, i.e., unguided, and model-based, i.e., guided, approaches. Grid search and random search are model-free methods, in which the trials are independent of the previous trials. Hence, these methods are often easy to implement and parallelize. Contrary to model-free search, SMBO methods perform guided search by employing results of previous trials to select a set of hyperparameters for the following trial. To make the selection, a surrogate model of the objective function is built. For example, random forest (RF), Gaussian process, and Tree of Parzen Estimator (TPE) models can be used as surrogate. The pseudo-code for the SMBO approach is given in Algorithm 1. Metaheuristic approaches, such as genetic algorithm and particle swarm optimization, and gradient-based algorithms can be also used in hyperparameter optimization [20].

We have used the TPE-based approach implemented in the Hyperopt Python-library [3] and included multiple options

Algorithm 1 Sequential Model-Based Optimization

Input: Number of initial configurations N_i, number of configurations with guided search N_g, time limit, early stopping criteria
Output: Configuration c_b with the lowest loss l
Random search, i.e., initialization:
1: Initialize L and C as empty lists
2: **for** $i = 1$ to N_i **do**
3: Fit model m_i with a random configuration c_i
4: Append loss of m_i to L and c_i to C
5: **end for**
6: Fit model M_s that maps C to L
Guided search:
7: **for** $j = 1$ to N_g **do**
8: Using M_s, get configuration c_j that minimizes l
9: Fit model m_j with configuration c_j
10: Append loss of m_j to L and c_j to C
11: Update M_s that maps C to L
12: **end for**
13: $c_b =$ configuration in C with the lowest loss
14: **return** c_b

regarding data input type, feature engineering, and classifiers together with their model and learning algorithm hyperparameters. A complete list of model-specific hyperparameters explored in this study can be found in the Appendix.

B. NESTED CROSS-VALIDATION

Cross-validation is a commonly used method in hyperparameter optimization to find a set of model hyperparameters that minimize a certain loss metric. One-level CV is often used in the development of ML-based models because the computational cost of nCV can be significantly higher since N x M models are trained instead of N models. However, in a one-level multifold CV, the same data is used for tuning the hyperparameters and evaluating the models. Therefore, the estimate of model generalization performance is optimized during the hyperparameter optimization process, which can lead to optimistically biased estimation [21]. The nCV procedure can be used to avoid that bias by separating the hyperparameter tuning and the model generalization performance estimation. The pseudo-code for the nCV is given in Algorithm 2. The nCV procedure includes two levels – an outer and inner loop, as shown in Figure 1. The dataset is first divided into M equally-sized folds called outer folds. The same hyperparameter or pipeline optimization procedure is executed M times, each time using a different fold out of the M folds as the outer validation dataset and the rest $M - 1$ folds as the outer development dataset. In the inner loop, the outer development dataset is further divided into N folds, which form the inner training and validation folds. While the inner training and validation folds are used to find optimal model hyperparameters or pipeline, the outer validation folds are only used to evaluate the found

configuration. Therefore, the nCV procedure essentially estimates the performance of the pipeline or hyperparameter optimization method. This performance is quantified with nCV score, which is the expected generalization performance of a model or pipeline that the algorithm can find with the dataset in question.

A numerical example of the nCV procedure: If $M = N = 5$ and the number of samples in a dataset D is 1000, D is first divided into M outer folds, which each contains $1000/M = 200$ samples. Therefore, the 800 samples ($M - 1 = 4$ folds) are fed to the inner loop. In the inner loop, the 800 samples are divided further into $N = 5$ inner folds, which each contains $800/N = 160$ samples. In each $N = 5$ iteration of the inner loop, a different inner fold of 180 samples is used to validate a model trained with the remaining inner folds. Similarly, in each M iteration of the outer loop, a different outer fold of 200 samples is used to validate a model trained with the remaining outer folds. The model hyperparameters are optimized in the inner loop and the model with the best hyperparameters is retrained with outer training folds to be validated with the current outer validation fold. As a result, there are $M = 5$ estimations for the performance, which form the nCV score.

The ATSC-NEX algorithm makes use of nCV procedure in estimating the generalization performance of the models. This procedure includes the configuration optimization discussed in Section III-A. One-level multi-fold CV is executed after the nCV procedure to find the configuration, i.e., the feature engineering configuration and the model hyperparameters, for the final model. The final model is evaluated using an external test dataset, which has been excluded from the original dataset and not used in the model development, as shown in Figure 1.

Accuracy is not a reliable metric when dealing with imbalanced datasets [22]. For example, if there are 100 samples for class A and 10 samples for class B, 90% accuracy could be reached with a model that always predicts A. In the ATSC-NEX algorithm, the predictive performance of the classifiers is measured using balanced accuracy (BAC) score in order to take possible class imbalances into account. The balanced accuracy score is defined by

$$BAC = \frac{1}{K}\sum_{i=1}^{K} r_i, \tag{1}$$

where K is the number of classes, and r_i is the recall, i.e., the number of true positive predictions divided by the sum of true positive and false negative predictions. The BAC is rescaled as described by

$$BAC_{\mathrm{r}} = \frac{BAC - \frac{1}{K}}{1 - \frac{1}{K}}, \tag{2}$$

With rescaling, an accuracy corresponding to random guessing would result in BAC score of 0. For each trial, i.e., for each set of tested hyperparameters and pipeline configuration, N rescaled BAC_r scores are obtained from the inner CV.

Algorithm 2 Nested Cross-Validation

Input: Dataset D, number of outer folds M, number of inner folds N, Number of configurations N_C
Output: Estimation of generalization performance L

1: Divide D into M folds in stratified manner
Outer loop:
2: Initialize L_o as empty list
3: **for** fold m in M folds **do**
4: m = outer validation set
5: Remaining $M - 1$ folds = outer training set
6: **for** $n = 1$ to N_C **do**
7: Get configuration C_n using an SMBO algorithm
8: Initialize L_i as empty list
Inner loop:
9: **for** fold n in $M - 1$ **do**
10: n = inner validation set
11: Remaining $M - 1$ folds excluding n = outer training set
12: Initialize a model with configuration C_n
13: Fit the model on inner training data
14: Append loss of the model on fold n to L_i
15: **end for**
16: L_c = average of values in L
17: **end for**
18: Initialize a model with configuration C_n with the lowest L_c
19: Fit the model on outer training set
20: Append loss of the model on fold m to L_o
21: **end for**
22: L = average of values in L_o
23: **return** L

In order to take the robustness of the hyperparameters and pipeline into account, each configuration is ranked by standard deviation weighted BAC score BAC_w obtained from the inner CV results, as defined by

$$BAC_{\mathrm{w}} = BAC_{\mathrm{r_mean}} - 2\log_{10}(1 + BAC_{\mathrm{r_std}}), \tag{3}$$

where

$$BAC_{\mathrm{r_mean}} = \frac{1}{N}\sum_{i=1}^{N} BAC_{\mathrm{r_N}} \tag{4}$$

and

$$BAC_{\mathrm{r_std}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(BAC_{\mathrm{r_N}} - BAC_{\mathrm{r_mean}})^2} \tag{5}$$

Finally, the nCV score is computed as the mean of the weighted BAC scores obtained from the M iterations of the outer CV loop. Also, the standard deviation is reported. After obtaining the nCV score, the final classifier and its hyperparameters are fixed in one-level cross-validation using the same optimization procedure as in the nCV. The final classifier is trained using the hyperparameters and pipeline configuration determined by the CV. Finally, the developed

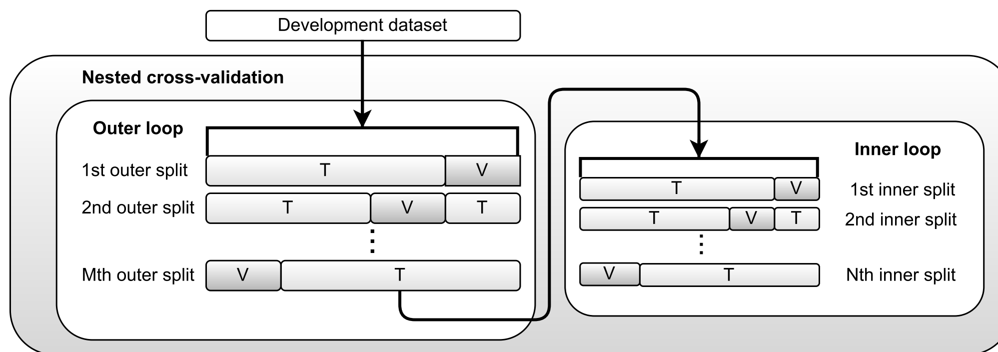


FIGURE 1. The nested cross-validation procedure. T and V stands for training and validation data, respectively.

classifier is tested using the external test dataset, that was excluded from the development dataset before starting the modeling procedure.

C. DATA PREPROCESSING AND FEATURE ENGINEERING

The pipeline optimization algorithm is allowed to explore multiple input types, namely raw time series, features extracted from the time series, and random convolutional kernel transformed (ROCKET) [12] time series. The algorithm utilizes tsfresh Python library [11] to extract over 700 features from the time series. The feature extraction is done before the model development loop. However, having a large feature set that includes irrelevant features can have a negative impact not only on the training and inference cost of the model but also on its accuracy [23]. Therefore as a further option, the algorithm is allowed to choose whether or not to apply feature selection when the feature-based data is used. The Boruta algorithm [23] is used for feature selection. The feature selection is done based on the current training data in both the outer and inner loops of the nCV procedure, which is introduced in Section III-B. Boruta is a wrapper type of feature selection algorithm for finding all relevant features. By default, it employs a random forest, an ensemble model, to compute feature importances. Randomization involved in the procedure decreases the effect of random fluctuation and correlations. However, it should be noted that all relevant features may include highly correlated and redundant features, and even if there is a correlation between a feature and the output, it is not guaranteed that their relation is causal. Hence, the authors of Boruta suggest using other feature selection libraries in order to further narrow down the feature set to include only highly relevant and uncorrelated features [23].

Here, the pipeline optimization algorithm is allowed, as an option, to choose the ROCKET method to transform the time series data. While convolutional neural networks learn convolutional kernels from data, the ROCKET method transforms time series using a large number of random convolutional kernels to create features for training [12]. The random generation decreases the training time, and transforming input data allows the use of simpler model types. The kernels have random length, dilation, padding, weights, and bias. ROCKET computes the proportion of positive values, *ppv*, for each time series and kernel. These values represent how the input time series match the different random kernels.

In some cases, the available dataset can be imbalanced, i.e., there is a large difference between the number of samples of multiple classes. Naturally, some applications require high accuracy for the minority class, for example, when a rarely occurring fault in a system is being predicted and most of the data represent healthy operation. There are multiple ways to solve the class imbalance issue. One approach is to set weights for the training samples. The higher the sample weight, the more that sample affects the computed loss, and therefore, the learning of the model. In the weighting approach, minority classes are given higher weights than majority classes. The weights can be defined, for example, as the inverse of the class incidences in the training dataset. Another approach to address class imbalance is to resample the dataset either by oversampling and/or undersampling. In oversampling, the number of minority class samples is increased, while in undersampling, majority class samples are removed [24]. It is also possible to combine oversampling with data cleaning methods, i.e., undersampling in the sense they remove noisy samples that were generated with oversampling [25]. Yet another way to address class imbalances is ensemble learning, where each ensemble is trained using different subsets of majority class samples [24]. Here, we have utilized the synthetic minority oversampling technique (SMOTE) algorithm [22]. SMOTE first computes k nearest neighbors for the original minority class samples (all or a fraction). Then, for each of the input features (i.e., attributes), the difference between the feature value of the original sample and that of the randomly chosen nearest neighbor is computed. A new feature value for a new synthetic sample is then computed by adding up the original feature value with the computed difference and by multiplying it with a randomly chosen value between 0 and 1 [22]. Finally, despite the input type, the training and validation input data was standardized by subtracting the value of each sample with the mean of the training

TABLE 1. Classifiers evaluated in this work.

Classifier	Abbreviation
AdaBoostClassifier[a]	ADAB
DecisionTreeClassifier[a]	DT
kNeighborsClassifier[a]	KNN
LogisticRegression[a]	LOGREG
PassiveAggressiveClassifier[a]	PASAGR
RandomForest[a]	RF
RidgeClassifier[a]	RIDGE
RidgeClassifierCV[a]	RIDGECV
C-Support Vector Classifier[a]	SVC
SGDC[a]	SGDC
BalancedBaggingClassifier[b]	BBAG
Gradient boosting decision tree[c]	GBDT
Multilayer perceptron[d]	ANN

[a]Scikit-learn, [b]Imbalanced-learn,
[c]LightGBM, [d]Tensorflow-Keras v.2

samples and dividing by the standard deviation of the training samples.

D. CLASSIFIERS

Thirteen classifiers were included in the framework to be compared in the case studies. Ten classifiers are from the Scikit-learn library [26], balanced bagging classifier from Imbalanced-learn library [24]. In addition, gradient boosting decision tree algorithm implemented in LightGBM library [27] was included in the framework, as well as multilayer perceptron classifier using the Keras API of Tensorflow library version 2.1 [28]. The included classifiers are listed in Table 1 together with their abbreviations used in this work. Each Scikit-learn classifier was set up to configure weights for the samples representing different classes as the inverse of the class incidences in the training dataset.

1) LINEAR CLASSIFIERS

Linear models included in the classifier evaluation are logistic regression, passive-aggressive classifier, ridge, and ridge with internal CV, and stochastic gradient descent classifier. For each, the Scikit-learn implementation is used.

The logistic regression model predicts class probability Pr_k as described by

$$Pr_k = \frac{e^{\beta_k^T x}}{1 + \sum_{l=1}^{K-1} e^{\beta_l^T x}}, \tag{6}$$

where k is the class index, K is the number of classes, x is the independent variable value vector, and β^T is the transposed weight vector that is learned during model fitting [29].

The passive-aggressive classifier is a model that is updated online, taking one input sample at a time and updating the weights accordingly [30]. The Scikit-learn implementation of the algorithm uses the hinge loss function by default in the learning process. The ridge classifier involves the regularization of the size of the model coefficients during the learning phase, which can reduce the variance of the predictions. The regularization strength is defined by the parameter λ. Here, both ridge classifiers with and without cross-validation are considered. The ridge classifier with cross-validation tunes λ internally, whereas the optimization algorithm tunes it for the ridge classifier without cross-validation. The stochastic gradient descent classifier implemented in Scikit-learn can fit multiple types of linear classifiers using the steepest gradient descent in the training. The model type can be chosen with a parameter that defines the loss function. Here, the default loss function (hinge) was used, which results in fitting a linear support vector machine.

2) TREE-BASED CLASSIFIERS

Five tree-based classifiers, namely decision tree, random forest, balanced bagging classifier, Adaboost, and gradient boosting decision tree, were included in the search as well. The decision tree is the most basic tree-based estimator that is built of simple if-else rules. These are called splits and each branch can be split again into multiple branches until a leaf, i.e., decision, node is reached. The Scikit-learn implementation of the decision tree considers all features by default when searching for the best one to be used for splitting.

The other tree-based estimators used here are ensembles, that combine a number of base tree estimators to achieve better predictive performance than a single classifier can achieve. The random forest and balanced bagging classifier both rely on bagging. Whereas the decision trees use the whole training dataset in constructing a tree and all features in splitting the nodes, a randomly sampled subsets of the training dataset are used in bagging approaches. In addition, the maximum number of features to be considered in splitting the nodes can be limited in bagging. The final classification result is based on majority voting. Bagging can potentially reduce the variance of an estimator without increasing bias if the model is unstable [29]. The balanced bagging classifier implementation of imbalanced-learn Python library is used with decision tree and random forest base estimators. The balanced bagging classifier balances the imbalance of the training dataset internally using random undersampling by default. For the random forest, the Scikit-learn implementation is used in this work.

Whereas random forests and balanced bagging classifiers employ bagging, the Adaboost and gradient boosting decision tree classifiers utilize a technique called boosting. In contrary to bagging, boosting procedure includes sequential construction of several tree-based estimators. After fitting one tree, its predictive performance is evaluated sample by sample. Based on the result, weights are assigned for the training samples and the weighted samples are then used to fit the next tree. The final classification is formed using majority voting of the weighted predictions of each tree [29]. For the gradient boosted decision trees (GBDT), the implementation of the LightGBM Python library is used here. LightGBM is essentially a gradient boosting algorithm but feature-wise and sample-wise bagging can be enabled as well [27].

3) OTHER CLASSIFIERS

The neighbors-based classifier is simple as it saves the training examples and the class prediction is based on majority voting of the nearest neighbors. The number of nearest neighbors k defines how many nearest neighbors are considered in the classification phase, and it is one of the parameters of the k-nearest neighbors classifier that can be tuned [31]. The weights for the neighbor points can be computed in different ways. Here, the uniform and the distance weighting are considered in the hyperparameter optimization. Whereas the distance-based weighting takes the inverse of the distances between the input sample and its k nearest neighbors into account, the uniform assigns the same weight for each neighbor as the name suggests. C-SVC is a type of support vector machine (SVM) that can employ, for example, linear, polynomial, radial basis function, and sigmoid kernels [32]. Fitting an SVC is based on structural risk minimization and aims to find a hyperplane that maximizes the distance between the hyperplane and the data points of different classes that are closest to it [33]. A kernel is a function that is used to compute how similar two input samples are [32].

Multilayer perceptron classifier is a feed-forward neural network model. Adam optimization algorithm is used in this study for optimizing the network weights. Optimized hyperparameters of the multilayer perceptron are here the number of layers and hidden neurons in the network, learning rate, batch size, and probability of dropout that is one type of regularization approach.

E. CASE STUDY - ENGINE TYPE IDENTIFICATION

In the case study, the aim is to identify a ship's diesel engine type from its exhaust gas temperature behavior. The engine type identification dataset consists of eight measurement signals of engine exhaust gas temperatures after a turbocharger. From the turbocharger, the exhaust gas continues to an exhaust gas economizer as shown in Figure 2. In the economizer heat is recovered from it to improve the efficiency of the system. Four of the eight signals are from the ship's four main engines (MEs) of the same type, and four signals from four different auxiliary engines of two different types (AE1 and AE2, respectively). The length of the signals is approximately 3350 timesteps with a six-minute sampling interval, corresponding to approximately 14 days worth of data. The signals were first divided into 17 time series subsets with 197 timesteps in each. These time series were shuffled so that the signals in one subset are not from the same period of time, since the data from each engine is collected simultaneously and the operation of engines is coupled to some extent as shown in Figure 3. Next, two of the 17 subsets, i.e., 16 time series in total, were chosen randomly and excluded from the dataset to be used as an external test dataset. The rest of the subsets, i.e., 120 time series, were used in the classifier development. The time series in the development and external test datasets are shown

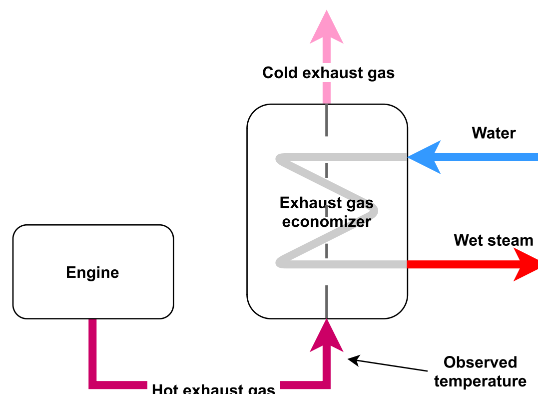


FIGURE 2. Simplified layout of the engine and heat recovery system.

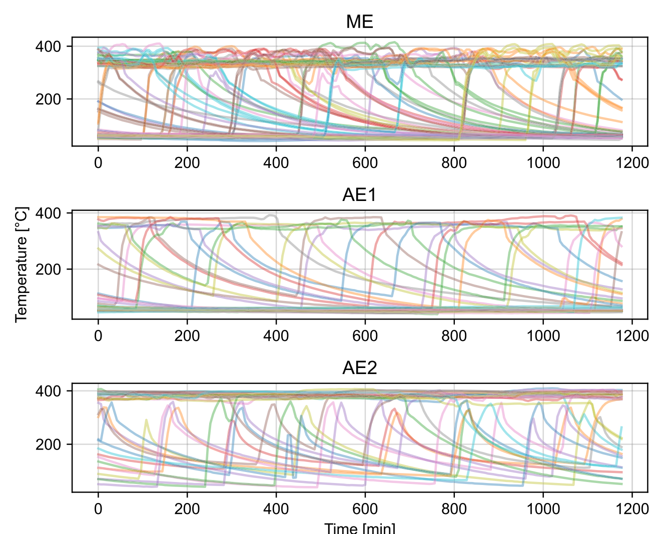


FIGURE 3. Exhaust gas temperature measurements from three types of engines used as classifier development data.

in Figures 3 and 3, where ME time series represent the main engine exhaust gas temperature behaviour, and AE1 and AE2 corresponding behaviour of the two auxiliary engine types. Similar kinds of patterns can be seen between the three engine types, as well as within the engine types, although the different time series start from the different operation points.

Multiple options for the input time series length, the sampling rate and the amount of overlap between two successive input sequences in the training data were included in the configuration optimization. These options, as well as the classifier input type and resampling options, are shown in Table 2. Tsfresh was used to extract over 700 features from the time series as described in the methods section. As the final preparation step before classifier development, the subsets that form the outer and inner validation folds in the nCV procedure are fixed to ensure comparability of the results. The 15 development subsets form five outer folds with three subsets per fold. This means that in each iteration, twelve subsets are used for the model development and three for validation. The inner folds for nCV are created

by dividing each set of outer development datasets into six folds.

F. BENCHMARK DATASETS

Six benchmark datasets from the UCR time series classification archive [34] are also used to evaluate the algorithm and the results with our algorithm are compared to the reference ones. It should be noted that the reference results are average values from 100 repetitions where the whole dataset is randomly resampled into the development and test datasets, whereas here, the development and test datasets are used as provided on the website. Thus, a direct comparison of the results can not be made, but the result should nevertheless give an idea of the performance of the ATSC-NEX algorithm.

The ACSF1 dataset represents power consumption of home appliances including coffee machines, computers, freezers, fridges, Hi-Fi systems, lamps, microwaves, mobile phone chargers, printers, and TVs. The time series in the BME dataset has three segments. The first or last segment has a positive bell arising, but the middle segment does not. The segments that do not have the bell can have different levels. The FordA dataset represents measurement data from an automotive subsystem. The ItalyPowerDemand dataset represents the electricity demand in Italy in 1997. The training data includes 67 days' worth of data. The target is to distinguish days recorded between April and September from the period from October to March. The SyntheticControl is a simulated dataset that represents control chart patterns, i.e., the level of a machine parameter changing over time. The target is to distinguish six different patterns, for example, a cyclic one or an increasing trend. The Trace dataset is a simulated dataset that represents instrumentation failures in a nuclear power plant.

IV. RESULTS

The automated classifier development approach is evaluated in the context of ship engine type identification, and the results are presented in this section. First, the algorithm is run with only one classifier type included at a time to find out which input configuration and hyperparameters work the best for specific classifier types. The results of the classifier evaluation set a baseline for the algorithm evaluation, here referred to as ALL_IN experiment, in which all the classifiers are included in the search space and the algorithm attempts to find well-performing classifiers for the ship engine problem. Each experiment is repeated five times, which are here referred to as runs. The classifier development and testing are made on a machine with Intel i9-9900k CPU and 16 Gb of RAM. In this study, graphics processing units (GPUs) were not used in the computations, although some models could benefit from it.

A. ENGINE TYPE IDENTIFICATION

The nCV scores obtained with each classifier and the ALL_IN experiment are shown in the Figure 6, including all

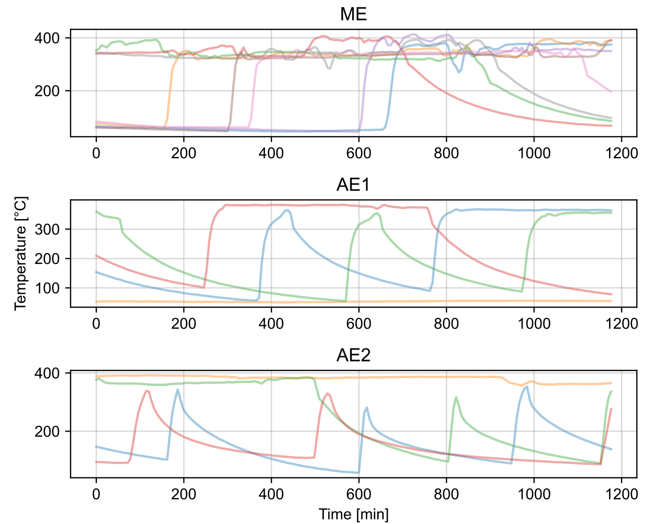


FIGURE 4. Exhaust gas temperature measurements from three types of engines used as external test data.

five runs. The maximum deviations from the average values are observed with GBDT and PASAGR, whereas particularly small deviations are obtained with ADAB, DT, LOGREG, RIDGE, RIDGECV, and SVC classifiers. However, Figure 6 shows that the balanced accuracy on the outer validation folds has relatively high variance. For example, the difference between the minimum and maximum outer validation folds BAC is approximately 65% with GBDT. The large minimum and maximum ranges are due to the differences between outer validation folds and how different accuracy the models reached. However, Figure 6 shows that the nCV scores (unweighted BAC) are very close to each other, showing the robustness of the algorithm in individual classifier evaluation. Similar to the inner CV scores, the ridge and ridge CV models achieve the highest accuracy. To take a closer look at what causes this variance, Figure 7 shows the five-run BAC average of each obtained with each outer fold separately. For this purpose, the same samples were used to construct the five outer validation folds within each run.

Figure 7 shows a clear trend that the average BAC for the fourth fold was worse compared to the other folds in each experiment, suggesting that the samples in the fourth outer fold are more difficult on average to classify than the samples in the other outer validation folds. The average BAC for the first and third outer folds were the highest two in most of the experiments.

Within each run, the five outer development folds were used to find five best-performing configurations using SMBO in the inner loop of the nCV procedure as described in Section III-B. 23 out of 25 optimization runs resulted in either RIDGE (9 times) or RIDGECV (14 times) being the best model of all. In two optimization runs, ADAB and LOGREG resulted in the best performance. The average BAC of each of these classifiers is shown in Figure 8. The results show superior performances for RIDGE and RIDGECV classifiers

TABLE 2. Options for the input time series data properties.

Data property	Options	Additional information
Input time series sequence length (SL)	25%, 40%, 55%	Fraction of original time series length
Sampling rate (SR)	6 min, 12 min	
Input time series sequence overlapping (OL)	25%, 50%, 75%	Percentage how much two successive sequences overlap
Input type	feature-based, time series	
Is feature selection applied	true, false	Always false when input is time series
Is ROCKET used to transform the input data	true, false	Always false input is feature-based
Is oversampling with SMOTE applied	true, false	

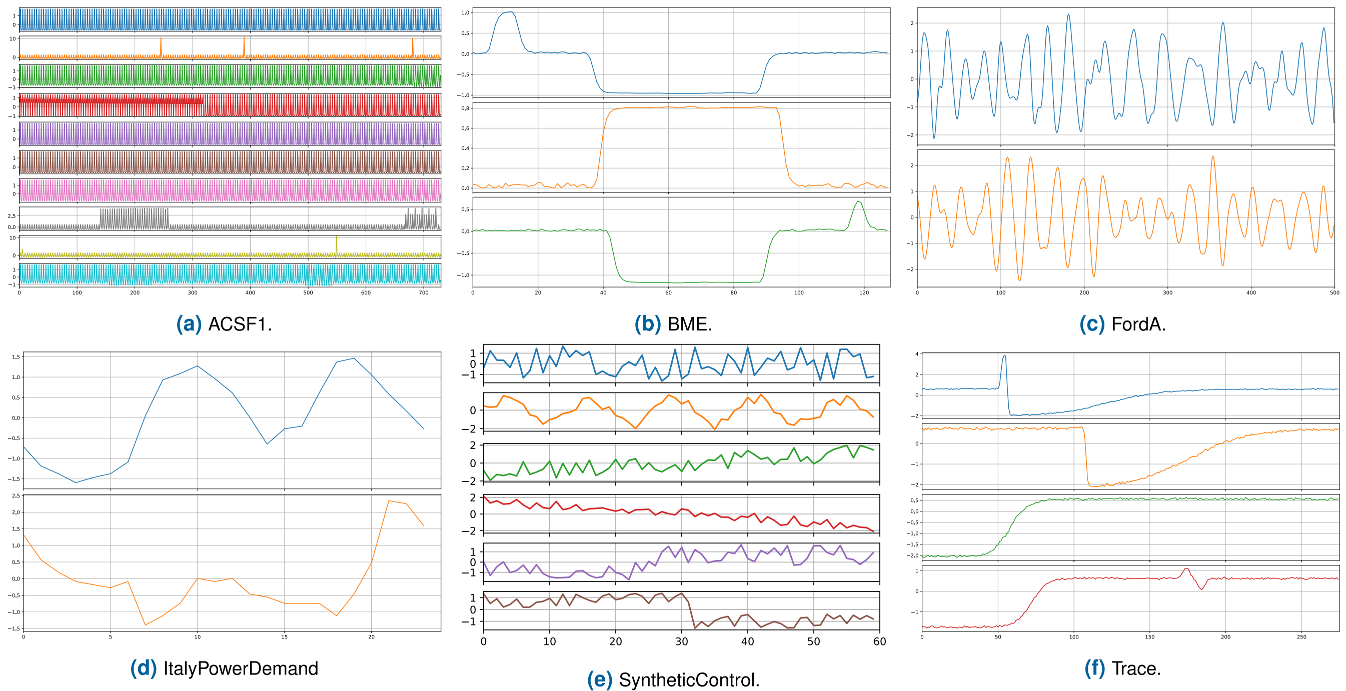


FIGURE 5. (a) ACSF1, (b) BME, (c) FordA, (d) ItalyPowerDemand, (e) SyntheticControl, and (f) Trace benchmark datasets.

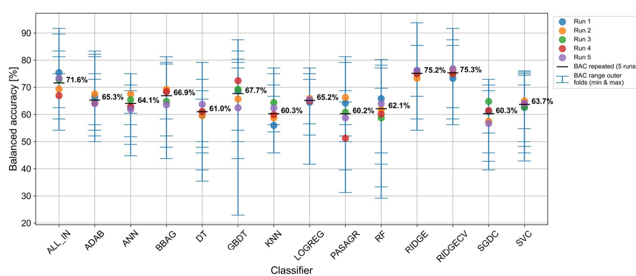


FIGURE 6. The nCV scores of individual classifiers and classifiers in ALL_IN experiment.

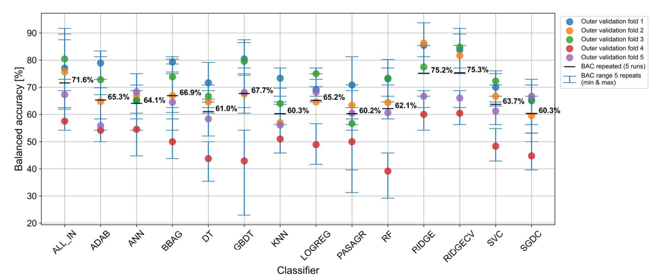


FIGURE 7. Average balanced accuracy on outer validation folds.

in comparison to other evaluated classifiers. From the two, RIDGE had higher average BACs for all but the fourth fold. The average balanced accuracy scores in the inner CV over the five runs and folds were 79.2%, and 77.5% for RIDGE and RIDGECV models, respectively. The average scores for the five folds were close to each other with RIDGE, but much higher deviance is visible with the RIDGECV. A nearly as high average BAC was obtained in the ALL_IN experiments, which is due to the search successfully ending up in finding a RIDGE or RIDGECV model with a well-performing configuration. Nevertheless, the variance in the ALL_IN experiments is naturally higher since the search space is much larger and those experiments would have required more iterations within the hyperparameter optimization to converge to similar configurations as in RIDGE or RIDGECV in the individual classifier experiments. The RIDGECV average scores are spread more than those of RIDGE, most likely caused by the internal cross-validation of RIDGECV

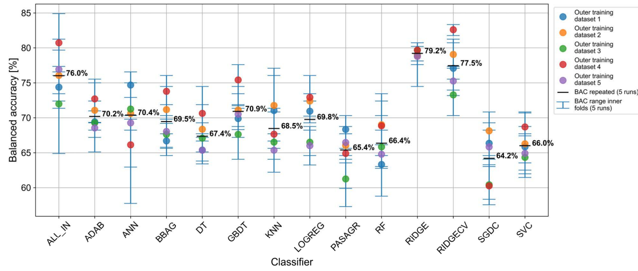


FIGURE 8. Average balanced accuracy of the best models found in the inner loop with the five outer training datasets.

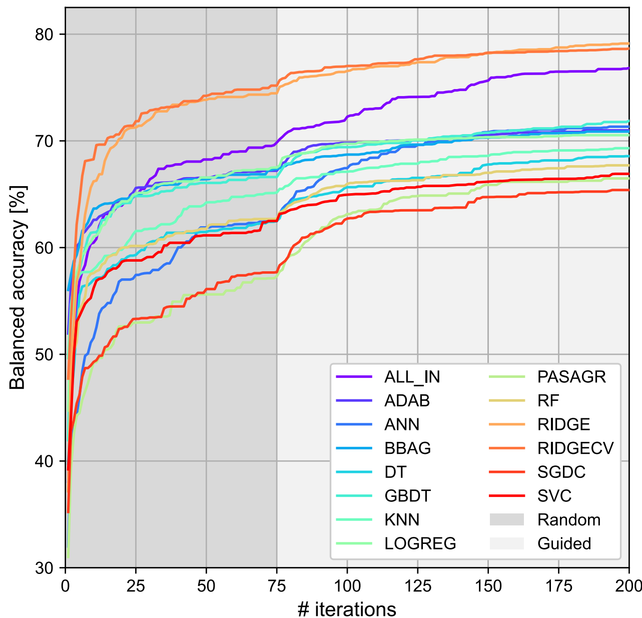


FIGURE 9. Evolution of the balanced accuracy score within hyperparameter optimization in the inner cross-validation loop (lines represents the average of five outer training datasets and five runs.

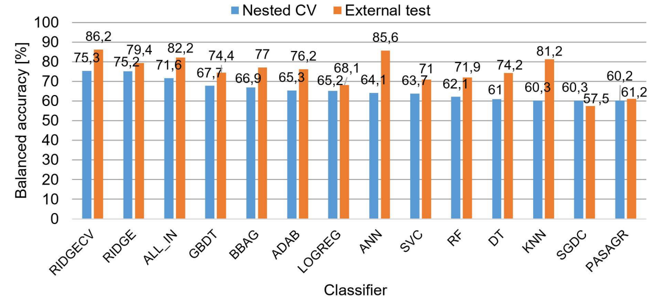


FIGURE 10. Balanced accuracy of developed classifiers in nCV and on external test dataset, sorted by the former. Both are the average of five runs.

and having fewer samples for training in comparison to RIDGE.

1) FINAL CLASSIFIERS

The ALL_IN runs resulted twice in both RIDGECV and RIDGE classifiers and once in LOGREG classifier. Balanced accuracy scores of 79.7-90.6% were obtained with the former, each with ROCKET transformed time series input data. LOGREG was an exception to this, as only 68.8% accuracy was reached using feature-based input data. This suggests that more iterations would be required for the algorithm to converge to the same solution each time, which is logical, as the dimension of the configuration optimization search space is much higher compared to the runs that included only one classifier at a time.

The balanced accuracy of the classifiers within nCV and on external test dataset is compared in Figure 10. With all except the SGDC classifier, the balanced accuracy is higher compared to the corresponding average nCV score, which suggests that the samples in the external test dataset are somewhat easier to classify than the ones in the development dataset. The largest differences between nCV score and external test accuracy are shown with ANN and KNN, for which the difference is 21.5% and 20.9%, respectively. The corresponding differences for other classifiers are 1–13.2%, except for SGDC which accuracy on the test dataset is 2.8% worse on average than the nCV score. Compared to other methods, the ANNs often require more samples for training, which might be the reason why the difference between the nCV score and external test score is high. The corresponding difference of the kNN model suggests that it is also more sensitive to the amount of training data than the other studied models.

The evolution of the configuration optimization during the final cross-validation for the different classifiers and in the experiment where all classifiers were included in the same search, is shown in Figure 11. It shows that, after evaluating 75 random configurations, the balanced accuracy starts to increase when the guided search with Hyperopt begins, even though with some classifiers the evolution seemed to stall before that. This shows the power of SMBO approaches.

The stability of the configuration optimization algorithm is evaluated by comparing the configurations found for each classifier type and in the experiment in which all classifiers were included in the search. The results show, as expected, that the highest overlapping (75%) between the successive time series used in the model development results in the best accuracy-wise performance, as 97.1% of all runs (68 out of 70). An exception to this was one of the five runs with the RF and RIDGECV models, for which the algorithm found overlapping of 50% to be optimal. Similarly, in 80% of the runs (56 out of 70), the longest sequence length of 55% was found to be optimal. The sequence length of 40% appeared in the results once with the RIDGECV in the ALL_IN experiment, and the shortest 25% sequence length was resulted in the best accuracy 13 times out of 70 (18.6%). The shortest sequence length worked the best with the BBAG classifier in each of the five runs and four out of five runs for the ADAB model. Also within the majority of the GBDT runs (60%), it was found to be the best sequence length. At the same time, the original sampling rate was found to be the best 65 times out of 70 (92.9%). Two of these appeared

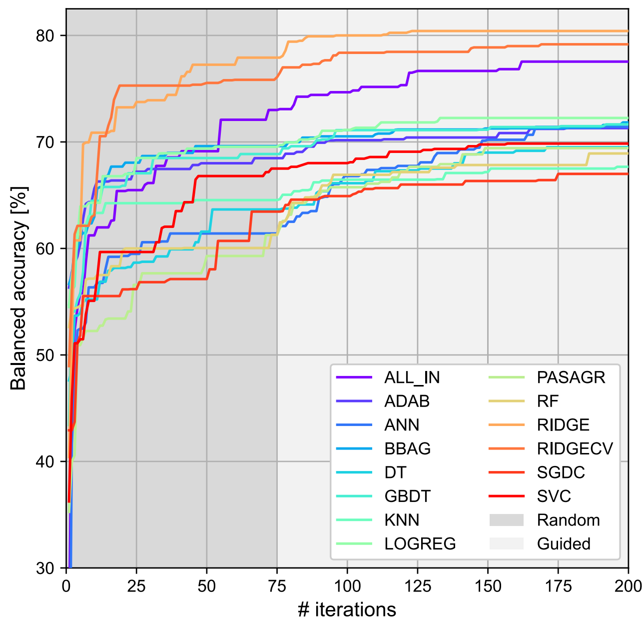


FIGURE 11. Evolution of the balanced accuracy score within hyperparameter optimization in the final CV (lines represent the average of five runs).

with the LOGREG classifier and once with three other classifiers.

The optimization algorithm found the feature selection to generally result in better performance as, with ANN, PASAGR, SHDC, SVC, LOGREG, DT, and RF models, the final pipeline configuration was found with the feature selection enabled when feature-based input was used. With GBDT, the feature-based input worked better without feature selection, and with BBAD and ADAB models, the feature selection was enabled three out of five and two out of three times, respectively.

The final ANN and RIDGECV models achieved similar average accuracy on the external test dataset, but feature-based input worked better for the former in all five runs and ROCKET transformed time series for the latter similarly in all five runs. The KNN and RIDGE models also achieved the best accuracy using ROCKET transformed time series, whereas three out of five times it worked the best for DT, GBDT, and RF. Similar to ANN, the feature-based input was the best for the PASAGR classifier, but its average BAC was only 61%. In fact, in 51% of the runs, feature-based input was found to result in better accuracy than ROCKET transformed time series, but the average BAC of the former was 67% with a standard deviation of 12%, and 81% with a standard deviation of 6% for the latter. Either feature or ROCKET-based input was found to be best in 60% of runs with ADAB, DT, GBDT, LOGREG, and RF models. There was no clear pattern in the configurations with these models, which suggests that more iterations should be run in the configuration optimization.

Nevertheless, the results show that there is no input type that works the most optimally for all data-driven model types,

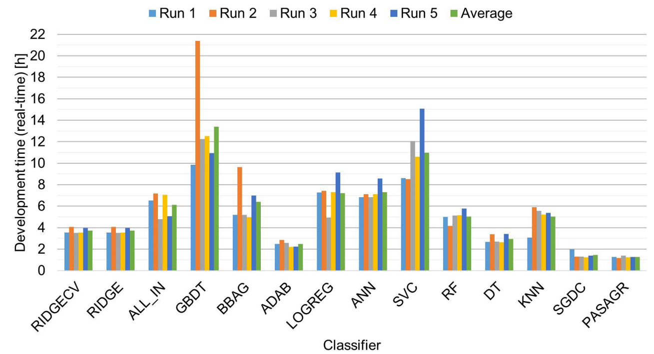


FIGURE 12. Total classifier development time for each classifier and experiment where all classifiers were included in the search.

which is also why an automated algorithm to find optimal model and data processing methods is useful. The found input configuration shows that the algorithm is relatively stable but with some classifiers, the algorithm would most likely require more iterations.

2) CLASSIFIER DEVELOPMENT TIME

Figure 12 shows the development time of the classifiers measured as real-time, sorted from left to right by the nCV scores. The development time for the most accurate classifiers, according to the nCV, i.e., RIDGECV and RIDGE classifiers, was slightly less than four hours on average. The ALL_IN run resulted in the third-highest nCV score with a development time of six hours on average. The computations were performed on one machine. However, the computational performance of the algorithm can be optimized by utilizing parallel computing and some model types could utilize GPUs to reduce the training time. The folds of the outer loop in the nCV procedure could be performed in parallel. In addition, the iterations of the SMBO algorithm to initialize the surrogate model can be parallelized since the hyperparameters for these iterations are selected randomly. In this study, the number of outer folds was five, i.e., the nCV procedure could be performed at least five times faster with parallelization.

B. BENCHMARK RESULTS

In this study, six benchmark datasets described in Section III-F were used to evaluate the performance of the ATSC-NEX algorithm. The results obtained with the ATSC-NEX algorithm are shown in Table 3.

Out of the thirteen classifier types, the algorithm found the RF classifier to work best on the ACSF1 and ItalyPowerDemand datasets. In both, the algorithm also found that transforming the time series data with the ROCKET method produces better results than the feature-based approach. With ACSF1 and ItalyPowerDemand datasets, the BACs on nCV were 71.1% with a standard deviation of 12.4% and 90.8% with a standard deviation of 14.2%, while the corresponding BACs on the external test set were 78.9% and 88.1%, respectively. The ROCKET method was also found to

TABLE 3. Performance of classifiers developed with ATSC-NEX on the selected benchmark datasets.

Benchmark dataset	Classifier	Input type	Balanced accuracy		Development time	
			nCV	External test	Real	CPU
ACSF1	RF	ROCKET	71.1% (± 12.4%)	78.9%	3.3 h	6.1 h
BME	RIDGE	ROCKET	100.0% (± 0.0%)	92.0%	0.2 h	0.5 h
FordA	BBAG	Features[a]	100.0% (± 0.0%)	100.0%	0.2 h	1.0 h
ItalyPowerDemand	RF	ROCKET	90.8% (± 14.2%)	88.1%	1.5 h	2.3 h
SyntheticControl	KNN	ROCKET	98.0% (± 4.8%)	99.6%	0.5 h	2.3 h
Trace	BBAG	Features[a]	100.0% (± 0.0%)	100.0%	0.04 h	0.1 h

[a]Statistical features computed from time series data

TABLE 4. Comparison of balanced accuracy of the selected state-of-the-art models [34] and ATSC-NEX on the selected benchmark datasets.

Classifier	ACSF1	BME	FordA	ItalyPowerDemand	SyntheticControl	Trace
TS-CHIEF	80.7%	99.6%	94.8%	96.2%	99.9%	100.0%
HIVE-COTE v1.0	85.0%	98.2%	94.4%	95.8%	99.4%	100.0%
InceptionTime	82.7%	99.6%	95.9%	96.0%	99.6%	100.0%
ROCKET	80.7%	99.7%	94.2%	96.2%	99.8%	100.0%
ResNet	82.4%	99.9%	93.1%	95.7%	99.4%	100.0%
STC	83.8%	93.0%	93.4%	95.4%	99.2%	100.0%
ProximityForest	63.8%	99.9%	85.0%	95.6%	99.8%	100.0%
WEASEL	81.8%	94.8%	96.9%	94.7%	98.7%	100.0%
S-BOSS	81.5%	86.5%	90.7%	86.9%	96.5%	100.0%
BOSS	76.8%	86.6%	92.1%	87.1%	96.7%	100.0%
cBOSS	75.7%	78.5%	91.6%	92.7%	95.1%	100.0%
TSF	63.5%	96.2%	81.6%	95.9%	99.2%	99.3%
RISE	76.0%	78.6%	94.0%	94.5%	67.8%	98.5%
Catch22	77.8%	90.5%	90.9%	87.8%	96.7%	100.0%
Best	85.7%	99.9%	96.9%	96.2%	99.9%	100.0%
Average	78.0%	93.0%	92.0%	93.6%	96.3%	99.8%
ATSC-NEX	78.9%	92.0%	100.0%	88.1%	99.6%	100.0%

perform better than the feature-based approach on the BME and SyntheticControl datasets, where the best results were obtained with the RIDGE and KNN models, respectively. The nCV BAC on the BME and SyntheticControl datasets were 100% and 98.0%, respectively, with a standard deviation of 4.8% for the latter. The corresponding BACs on the external test datasets with the BME and SyntheticControl datasets were 92% and 99.6%, respectively. With the FordA and Trace datasets, the BBAG classifier with feature-based input outperformed other options and achieved 100% BAC in both nCV and external test dataset.

Although nCV is computationally less efficient than, e.g., multi-fold CV, the development time (real-time and CPU-time) in Table 3 shows that even with extensive optimization in a relatively large search space, a good solution can be found with the ATSC-NEX in a relatively short time.

The benchmark results obtained with the ATSC-NEX and fourteen other methods are shown in Table 4. As mentioned in Section III-F, a direct comparison between the ATSC-NEX and other methods shown in Table 4 can not be made. Still, the comparison can be used to estimate how well the ATSC-NEX performs. With the ACSF1, BME, and ItalyPowerDemand datasets, the BAC obtained with the ATSC-NEX algorithm is 6.1-8.1% lower than with the best state-of-the-art classifiers. With the other three datasets, the ATSC-NEX algorithm achieves similar performance (ItalyPowerDemand and SyntheticControl), or has a slightly higher BAC (3.1% with the FordA dataset). However, when comparing the BACs obtained with the ATSC-NEX with the average values obtained with the other models, the differences are lower. This suggests that similar performance can be achieved with the ATSC-NEX algorithm. One benefit of the ATSC-NEX is that the algorithm can find the best working methods for a specific problem through optimization. Extending the ATSC-NEX algorithm to cover other signal processing algorithms, feature engineering methods, and model types can potentially improve the performance of the developed models further, which makes it an intriguing topic to study in the future.

V. CONCLUSION

We have presented an algorithm to develop a time series classification model. The algorithm employs the SMBO to optimize the pipeline configuration, including the selection of an optimal input data shape and preprocessing method, and model type from multiple options available. The algorithm is designed in such a way that it is straightforward to implement other preprocessing methods and model types to be explored. The results of the case study show that the proposed algorithm can efficiently find an optimal pipeline configuration among 14 model types and several input configurations. The comparison of ATSC-NEX to state-of-the-art TSC methods shows that ATSC-NEX can achieve similar performances. Since ATSC-NEX employs nCV in the estimation of generalization performance, the estimate can be expected to be more reliable compared to those methods that only perform one-level CV. Naturally, the more

TABLE 5. Model specific hyperparameters and their allowed values in this study.

Classifier	Hyperparameter and options
ADAB	Base_estimator: (RF, DT) N_estimators: (25, 50, ..., 250) Learning_rate: Random uniform float (1e-3, 1)
DT	Cp_alpha: Random uniform float (1e-3, 1e-1) Max_depth: (4, 5, ..., 20) Max_leaf_nodes: (2, 3, ..., 40) N_estimators: (50, 75, ..., 1000)
KNN	N_neighbors: (3, 4, 5, 6) Weights: ('uniform', 'distance')
LOGREG	C: Random uniform float (1e-1, 1) Max_iter: (50, 75, ..., 1000)
PASAGR	C: Random uniform float (1e-1, 2) Max_iter: (50, 75, ..., 1000)
RF	Ccp_alpha: Random uniform float (1e-3, 1e-1) Max_depth: (4, 5, ..., 20) Max_leaf_nodes: (2, 3, ..., 40) N_estimators: (50, 75, ..., 1000)
RIDGE	Alpha: Random uniform float (1e-3, 10) Max_iter: (50, 75, ..., 500)
SVC	C: Random uniform float (1e-1, 1) Degree: (2, 3) Kernel: ('linear', 'poly', 'rbf')
SGDC	Alpha: Random uniform float (1e-5, 1e-2) L1_ratio: Random uniform float (1e-3, 3e-1) Lr: 'optimal' Max_iter: (50, 75, ..., 1000) Penalty: 'elasticnet', 'l1', 'l2', None
BBAG	Base_estimator: (RF, DT) N_estimators: (6, 8, ..., 40) Max_features: Random uniform float (5e-1, 8e-1)
GBDT	Bagging_fraction: Random uniform float (5e-1, 9e-1) Bagging_freq: (5, 10, 25, 50) Feature_fraction: Random uniform float (5e-1, 9e-1) Lr: Random uniform float (1e-4, 1e-2) Max_depth: (4, 5, ..., 20) N_estimators: (50, 75, ..., 300) Num_leaves: (2, 3, ..., 20) Reg_alpha: (0, 1e-1, 2e-1, 3e-1) Reg_lambda: (0, 1e-1, 2e-1, 3e-1) Valid_sets: 10 % of training samples Early_stopping_rounds=10
ANN	Batch_size: (32, 64, 96, 128) Dropout_rate: Random uniform float (1e-2, 4e-1) Lr: Random uniform float (1e-4, 1e-2) N_epochs: (1000, 1500, 2000) N_neurons_per_hidden_layer: (16, 24, ..., 64) N_layers: (1, 2, 3, 4) Validation_split: 0.2 Callbacks: ReduceLROnPlateau(factor=0.2, patience=5, min_lr=1e-5) EarlyStopping(patience=30) ModelCheckpoint(save_best_only=True)

reliable estimate comes with a higher computational cost. The computational cost, however, can be improved by executing the outer loop of the nCV procedure in a parallel manner, and by parallelizing the model training iterations that are used to initialize the surrogate model used by the SMBO algorithm. As future work, expanding the search space to include more signal processing, feature engineering methods, and model algorithms and studying the effect of such expansion on the resulting model performance would be of interest.

APPENDIX

A complete list of model specific hyperparameters and their allowed values in this study are shown in Table 5.

REFERENCES

[1] L. Parmentier, O. Nicol, L. Jourdan, and M.-E. Kessaci, "AutoTSC: Optimization algorithm to automatically solve the time series classification problem," in *Proc. IEEE 33rd Int. Conf. Tools with Artif. Intell. (ICTAI)*, Nov. 2021, pp. 412–419.

[2] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC Bioinf.*, vol. 7, no. 1, pp. 1–8, Dec. 2006.

[3] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. ICML*, 2013, pp. 115–123.

[4] L. Ye and E. Keogh, "Time series shapelets: A new primitive for data mining," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2009, pp. 947–955.

[5] M. Arul and A. Kareem, "Applications of shapelet transform to time series classification of earthquake, wind and wave data," 2020, *arXiv:2004.11243*.

[6] J. J. Rodríguez, C. J. Alonso, and J. A. Maestro, "Support vector machines of interval-based features for time series classification," *Knowl.-Based Syst.*, vol. 18, nos. 4–5, pp. 171–178, Aug. 2005.

[7] J. Lin and Y. Li, "Finding structural similarity in time series data using bag-of-patterns representation," in *Proc. Int. Conf. Sci. Stat. Database Manage.*, vol. 21, M. Winslett, Ed. Berlin, Germany: Springer, 2009, p. 648.

[8] P. Schäfer, "The BOSS is concerned with time series classification in the presence of noise," *Data Mining Knowl. Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015, doi: 10.1007/s10618-014-0377-7.

[9] J. Large, P. Southam, and A. Bagnall, "Can automated smoothing significantly improve benchmark time series classification algorithms?" in *Hybrid Artificial Intelligent Systems* (Lecture Notes in Computer Science), vol. 11734. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-29859-3_5.

[10] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall, "HIVE-COTE 2.0: A new meta ensemble for time series classification," *Mach. Learn.*, vol. 110, 2021, doi: 10.1007/s10994-021-06057-9.

[11] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series FeatuRe extraction on basis of scalable hypothesis tests (tsfresh—A Python package)," *Neurocomputing*, vol. 307, pp. 72–77, Sep. 2018, doi: 10.1016/j.neucom.2018.03.067.

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

[13] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining Knowl. Discovery*, vol. 31, no. 3, pp. 606–660, May 2017.

[14] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Müller, "Deep learning for time series classification: A review," *Data Mining Knowl. Discovery*, vol. 33, no. 4, pp. 917–963, Mar. 2019, doi: 10.1007/s10618-019-00619-1.

[15] X. Chen, K. Men, B. Chen, Y. Tang, T. Zhang, S. Wang, Y. Li, and J. Dai, "CNN-based quality assurance for automatic segmentation of breast cancer in radiotherapy," *Frontiers Oncol.*, vol. 10, pp. 1–9, Apr. 2020.

[16] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622, doi: 10.1016/j.knosys.2020.106622.

[17] D. van Kuppevelt, C. Meijer, F. Huber, A. van der Ploeg, S. Georgievska, and V. T. van Hees, "Mcfly: Automated deep learning on time series," *SoftwareX*, vol. 12, Jul. 2020, Art. no. 100548, doi: 10.1016/j.softx.2020.100548.

[18] H. Wang, C. Li, H. Sun, Z. Guo, and Y. Bai, "Shapelet classification algorithm based on efficient subsequence matching," *Data Sci. J.*, vol. 17, pp. 1–12, Mar. 2018.

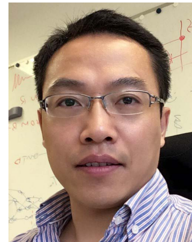
[19] M. Löning, A. Bagnall, S. Ganesh, V. Kazakov, J. Lines, and F. J. Király, "Sktime: A unified interface for machine learning with time series," 2019, *arXiv:1909.07872v1*.

[20] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020, doi: 10.1016/j.neucom.2020.07.061.

[21] J. Wainer and G. Cawley, "Nested cross-validation when selecting classifiers is overzealous for most practical applications," 2018, *arXiv:1809.09446*.

[22] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," 2011, *arXiv:1106.1813*.
[23] M. B. Kursa and W. R. Rudnicki, "Feature selection with the Boruta package," *J. Stat. Softw.*, vol. 36, no. 11, pp. 1–13, 2010.
[24] G. Lemaître, F. Nogueira, W. S. West, O. Mv, and C. K. Aridas, "Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning," *J. Mach. Learn. Res.*, vol. 40, pp. 1–5, Jan. 2015.
[25] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, p. 20–29, Jun. 2004.
[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.
[27] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2017, pp. 3147–3155.
[28] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.
[29] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning, Data Mining, Inference, and Prediction*. New York, NY, USA: Springer, 2009, doi: 10.1007/978-0-387-84858-7.
[30] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *J. Mach. Learn. Res.*, vol. 7, pp. 551–585, Dec. 2006.
[31] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.
[32] T. Sahay, A. Aggarwal, A. Bansal, and M. Chandra, "SVM and ANN: A comparative evaluation," in *Proc. 1st Int. Conf. Next Gener. Comput. Technol. (NGCT)*, Sep. 2015, pp. 960–964.
[33] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[34] A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh, "The UEA multivariate time series classification archive, 2018," 2018, doi: 10.48550/arXiv.1811.00075.

MIKKO TAHKOLA received the B.Eng. degree in energy technology from the Oulu University of Applied Sciences, in 2017, and the M.Sc. (Tech.) degree in process engineering from the University of Oulu, in 2019. Currently, he is a Research Scientist with the VTT Technical Research Centre of Finland, AI-aided Systems Engineering Research Team. His research interests include related to developing ML-based tools for engineering and ML-based surrogate modeling.



ZOU GUANGRONG is a former Marie Curie Researcher and a Senior Scientist with the VTT Technical Research Centre of Finland. He has been dedicating to the decarbonization, digitalization, and automation in maritime shipping, since 2010, with special focus on simulation and data-driven design and operation optimization of ship/fleet energy systems.