Automatic Feature Engineering for Time Series Classification: Evaluation and Discussion

Aurélien Renault, Alexis Bondu, Vincent Lemaire, Dominique Gay

Automatic Feature Engineering for Time Series Classification: Evaluation and Discussion

Aurélien Renault
Orange Innovation Paris, France
Email: aurelien.renault@orange.com

Alexis Bondu
Orange Innovation, Paris, France
Email: alexis.bondu@orange.com

Vincent Lemaire
Orange Innovation, Lannion, France
Email: vincent.lemaire@orange.com

Dominique Gay
Université de la Réunion, France
Email: dominique.gay@univ-reunion.fr

***Abstract*—Time Series Classification (TSC) has received much attention in the past two decades and is still a crucial and challenging problem in data science and knowledge engineering. Indeed, along with the increasing availability of time series data, many TSC algorithms have been suggested by the research community in the literature. Besides state-of-the-art methods based on similarity measures, intervals, shapelets, dictionaries, deep learning methods or hybrid ensemble methods, several tools for extracting unsupervised informative summary statistics, aka *features*, from time series have been designed in the recent years. Originally designed for descriptive analysis and visualization of time series with informative and interpretable features, very few of these feature engineering tools have been benchmarked for TSC problems and compared with state-of-the-art TSC algorithms in terms of predictive performance. In this article, we aim at filling this gap and propose a simple TSC process to evaluate the potential predictive performance of the feature sets obtained with existing feature engineering tools. Thus, we present an empirical study of 11 feature engineering tools branched with 9 supervised classifiers over 112 time series data sets. The analysis of the results of more than 10000 learning experiments indicate that feature-based methods perform as accurately as current state-of-the-art TSC algorithms, and thus should rightfully be considered further in the TSC literature.**

I. Introduction

The goal of Time Series Classification (TSC) is to assign a class label y from a set $Y = \{y_i\}$ of predefined labels to an unlabeled time series $X = [x_1, x_2, \ldots, x_m]$ which is an ordered set of real values. The associated machine learning task is to train a classifier function f on a labeled time series data set $\mathcal{D} = \{(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)\}$ in order to map the space of possible input series to the class labels of Y. Then, for an incoming unlabeled time series X, the prediction of the assignment is given by $f(X)$.

Formulated as such, TSC has been identified as a top-10 challenging problem in data mining research [1] and has received much attention in the literature. This particular attention is due to the overwhelming amount of available time series data [2]. Indeed, in many scientific fields, measurements are taken over time. The resulting collection of ordered data is represented as time series. Thus, times series data arise from many real-world applications: e.g., in the UCR/UEA archive [3], audio signal, electrocardiogram, encephalogram, human activity, image or motion classification are among the most frequent tasks. In order to solve these tasks, hundreds of TSC algorithms have been suggested in the past two decades [4]. Various paradigms have been exploited in the TSC literature; from simple distance-based Nearest Neighbors to deep learning architectures and complex hybrid ensemble methods. The very latest contributions [5] indicate that five methods, HIVECOTE (ensemble methods HC1 [6] and HC2 [5]), convolutional kernels based ROCKET [7], tree ensemble TS-CHIEF [8] and deep learning based InceptionTime [9] achieve top predictive performance.

Besides the numerous TSC algorithms, another research area has been developed in the recent years, namely time series Feature Construction (FC). FC approaches are attractive as they regroup methods for extracting informative and interpretable summary statistics from time series. Features might be diverse to capture various properties of the time series, e.g., seasonality, trends, autocorrelation, etc, and thus can be adapted to various application domains. The pioneering work HCTSA [10] allows to extract more than 7000 unsupervised time series features summarizing properties of the distribution of values in a time series, correlation properties, entropy and complexity measures, how properties of a time series change over time, etc. Since then, several unsupervised feature engineering tools have been independently developed in various programming languages: e.g., C/Python-based CATCH22 [11]; Python-based FEATURETOOLS [12], TSFRESH [13], [14], TSFEL [15]; and R-based TSFEATURES [16], FEASTS [17]. Originally designed for descriptive analysis and visualization of time series, their use for TSC problems has been less studied: e.g., B. Fulcher et al. [18] suggest a linear classifier based on the feature set generated by HCTSA and on a much smaller set, CATCH22 [11]. However, to our knowledge, no extensive study has been led on the effectiveness of existing feature engineering tools for TSC problems and how they compare with the leading TSC performers.

In this article, we address these shortfalls and suggest several simple learning processes to assess the effectiveness of feature construction for TSC. Our two-step learning processes consist in : *(i)* transforming the original time series data into feature-based tabular data using an unsupervised feature engineering tool (or a combination of tools), and *(ii)* learning a supervised classifier on the obtained tabular data. Figure 1 summarizes the final results of three selected processes compared with current leaders on 112 UEA/UCR data sets. Using feature construction is then comparable with the current state of the art TSC algorithms in terms of predictive performance (exhaustive details of the experiments are provided in Section III).

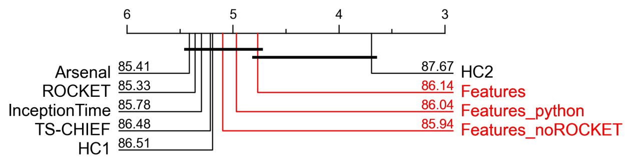


Fig. 1: Critical difference diagram of FC variants against current state of the art on 112 UCR/UEA data sets. The average rank is plotted and each contender is annotated by its mean accuracy results. Thick lines linking group of classifiers indicate no significant difference in predictive performance. This demonstrates that suggested FC processes (in red) are among the best TSC performers.

The rest of the paper is organized as follows: in Section II, we position our feature construction approach w.r.t. related work in TSC and feature engineering. The empirical evaluation and comparison are reported in Section III. We conclude in Section IV with some perspectives for feature construction in TSC problems.

II. Background

One of the simplest, but also very popular method for TSC problems is k-Nearest-Neighbors coupled with a similarity measure, e.g., Euclidean distance or dynamic time warping. Following these baselines, the TSC literature has grown tremendously. With the main focus on efficiency and predictive performance, several paradigms have been exploited:

- *similarity-based:* generally coupled with 1-Nearest Neighbor (1-NN), several similarity measures have been experimented for TSC problems. Elastic similarity measures provide the best predictive performance as they allow to tackle with local distortions and misalignments in time series [19].
- *interval-based:* from random intervals extracted from time series, a new feature vector is build and a traditional classifier is trained. Successful representatives of this type of method are the ones using multiple intervals and ensemble classifier, see e.g. [20].
- *shapelet-based:* the search for sub-sequences highlighting class separation not depending on when they occur in a serie has attracted many research efforts since [21].
- *dictionary-based:* these algorithms build words from sliding windows, then a classifier is learned on the transformed data made of histograms of word counts [22]
- *deep learning based:* following the success of deep learning for images and videos, several neural network architectures have been suggested for TSC [23].
- *ensemble-based:* to combine the performance of existing approaches, meta-ensemble classifiers have been designed and achieved top predictive performance. For example, the Collective of Transformation-based Ensembles (CoTE [24]) and its successors HC1 [6] and HC2 [5] fall in this category and are also providing structured overviews of the TSC literature.

As the TSC literature already offers well-structured overviews of existing TSC approaches, in the rest of the background part, we will concentrate on the recent top performers. The following algorithms are also the contenders in the experimental section.

A. Top TSC performers

ROCKET family One of the most promising method from the past few years is the ROCKET (Random Convolutional KErnel Transformation) model ([7]), which achieved SOTA performances with just a fraction of the time needed for competing approaches. The idea of ROCKET is to use a high number of random convolutional kernels with random lengths, dilations, padding and weights, then convolve them with the time series samples to finally extract two features per kernel, which are the max and the *ppv* (proportion of positive values) for each sample. A linear classifier is then trained on generated features, either a Ridge or a Logistic regression depending on the data set size we are dealing with: logistic regression will generally be preferred working with bigger data sets.

An extension of the ROCKET framework has been further proposed by the authors in MiniROCKET ([25]), which is now used as the default ROCKET variant in most analysis. The goal of MiniROCKET is to reduce, and almost eradicate randomness from the pipeline. In order to do this, authors have restricted much more the kernels parameters and only extract the *ppv* for each sample, making the algorithm almost deterministic. Hence, a pre-defined set of 84 kernels has been chosen (among the 512 possible ones), which is supposed to balance accuracy and computational cost. Within this framework, MiniROCKET achieved about the same level of accuracy as ROCKET but way faster (up to 75x faster on larger data sets), making this approach even more competitive when it comes to computation time and scalability.

One more ROCKET variant has been developed, this time focusing on predictive performance rather than speed, namely MultiROCKET ([26]). This algorithm is significantly improving the overall performance by extracting four features per kernel: adding to *ppv* the *mpv* (mean of positive values), the *mipv* (mean of indices of positive values) and the *lspv* (longest stretch of positive values). Additionally, those pooling operators are extracted for one alternative time series representation, the first order difference. This improvement comes at the cost of training time, which is approximately 2x slower, generating by default 5x more features than MiniROCKET.

InceptionTime, a Deep Learning approach The InceptionTime algorithm is one of the current most accurate DL architecture for TSC ([9]). InceptionTime is an ensemble of five Inception networks which are using Inception-V4 modules ([27]) which themselves combine classic Inception blocks as introduced in [28] and Residual Networks (ResNet) ([29]). The InceptionTime network architecture consists in two residuals blocks, each of which containing three Inception modules, it is then followed by a global averaging pooling layer and a fully-connected layer with the softmax activation function. For further information about DL for TSC, we refer to the extensive analysis of [23], which review the main DL algorithms for TSC.

Hybrid/Ensemble methods Ensemble approaches aim at combining the performance of several classifiers to built its own prediction. Those types of algorithms are currently holding the state-of-the-art in term of accuracy on the usual UEA & UCR benchmark.

The Time Series Combination of Heterogeneous and Integrated Embedding Forest (TS-CHIEF) ([8]), the second most accurate algorithm, builds random forest of decision trees whose splitting functions are time series specific and based on similarity measures, bag-of-words representations, and interval-based transformations.

The Hierarchical Vote Collective of Transformation-Based Ensembles HIVE-COTE (HC [6], HC2 [5]) is the current best performer, training independently and combining several classifiers, i.e., Shapelet Transform [30], Temporal Dictionary Ensemble aka TDE [31], an ensemble of ROCKET estimators called Arsenal and the interval based Diverse Representation Canonical Interval Forest aka DrCIF [32].

Hybrid ensemble classifiers are the most accurate TSC algorithms to date. However, the top performance comes at the price of high memory and CPU resources. As a result, they are generally at least an order of magnitude slower than the other contenders (see Table III).

B. Feature Extraction tools for time series

Besides previous algorithms dedicated to TSC problems, some research works proposed to automatically extract unsupervised features from time series in order to analyze and visualize the data. Most of them though, are rarely considered it comes to classification task and are less mentioned in TSC literature. In this part, we will briefly review the currently available packages/methods which are extracting some understandable features from time series data and which could actually be useful for classification.

Tsfresh, which stands for Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests, is a library proposing to use 63 usual times series characterization in order to extract, at most, around 800 features per sample ([13]). It actually provides 3 pre-defined features dictionaries whose length ranging from 10 (minimal set-up, very efficient extraction) to 800 features. The package also provides some relevance filter based on some statistical tests to only keep the most informative variables among the ones you have extracted. The features computation can be parallelized, theoretically making the method more scalable.

tsfel:The Time Series Feature Extraction Library is a more recent work which, similarly extracts a bunch of features from time series. Its specification is that it allows you to separate your data into user-specified window length and provides some better tools in order to analyse the temporal complexity of the extracted features. In tsfel, one can compute features according to their domain, the available domains are: temporal, spectral and statistical one or to extract them all at once.

catch22 & hctsa: The CAnonical Time series CHaracteristics (catch22) is a subset of 22 variables from the 7730 available features in the *hctsa* toolbox ([18]), which is originally a matlab package, able to extract up to nearly 7700 time series features. This limited set of features has been empirically selected over a large collection of datasets in order to choose the most explanatory ones and are, as well, minimally redundant. This approach take just a fraction of the computation time needed to compute and analyse the all *hctsa* set of features. Authors exhibit the fact that concerning classification, the average performance reduction is only around 7% ([11]).

tsfeatures: Originally a R package ([16]), the tsfeatures library is used to extract various type of features for time series data. The package implements around 40 methods in order to extract features, ranging from entropy to hurst exponent. A python version of the package is also available, we did not perform any time performances comparison between the two versions.

feasts: The Feature Extraction and Statistics for Time Series (feasts) ([17]), looks very similar to tsfeatures. This R package provides tools for the analysis and visualization of time series. There is no python implementation of the package.

Featuretools is an open-source Python library, which performs auto-features engineering based on relational tables. The library is built upon a feature discovery algorithm called *Deep Feature Synthesis* ([12]), which uses some aggregation and transform functions applied over several related tables, to create features. Working with temporal data with featuretools is quite natural, it includes many date based operations as well as some of the *tsfresh* functions. While the features computation can be perform on multiple threads, the discovery though, cannot ; which make the method not very scalable when generating a large number of features. Additionally, one can remark that there is no way to prevent features generation from overfitting, generate very complex features may indeed results in learning some training set specificities and thus degrade classification performances.

Generalised signatures: The signature transform is an infinite collection of statistics for sequential data representation and/or feature extraction derived from rough paths theory. The signature can be thought as a moment generating function, as each term in the collection has a particular (geometrical) meaning as a function of data points. Usually, one compute the N truncated signature of $x = (x_1, ..., x_n)$, with $x_i \in \mathbf{R}^d$ and linearly interpolate x into the path $f = (f_1, ..., f_d) : [0, 1] \to \mathbf{R}^d$. The signature of depth N of x is defined as follows:

$$Sig^N(x) = \left(\left(\int_{0\leq t_1 \leq \ldots \leq t_k \leq 1} \cdots \int \prod_{j=1}^{k} \frac{df^{i_j}}{dt}(t_j) dt_1 \ldots dt_k \right)_{1\leq i_1, \ldots, i_k \leq d} \right)_{1\leq k \leq N} \tag{1}$$

The depth-1 terms ($k = 1$), for example, simply represent the total increments (difference between end and start point) over each dimension. When it comes to TSC, the generalised signature pipeline ([33]), which has primarily been designed for multivariate series ($d > 1$), computes the signature transform over some hierarchical dyadic windows, to finally concatenate the outputs into the feature vector. This approach has shown to be competitive compared to state-of-the-art classifiers.

III. Experimental Validation

In order to explore the potential of feature construction for TSC problems, the experiments carried out in this paper have been designed to answer the following questions:

- *(Q_1)* How do the various FC tools compare with each other in terms of predictive performance and time efficiency ?
- *(Q_2)* In which extent using the new features perform better than raw original data ?
- *(Q_3)* Is it beneficial to combine several feature engineering tools ?
- *(Q_4)* How does feature construction based methods compare with state of the art TSC methods ?

Aiming at the full reproducibility of the experiments, we first present the details of our experimental protocol then discuss the obtained results. A dedicated webpage with the source code and scripts to run the experiments is also available [1].

A. Experimental protocol

Data sets. We have conducted our experiments on the usual time series classification benchmark. We ran our experiments on 112 data sets out of the 128 ones which are available in the UCR Time Series Classification Archive ([3], [34]). Indeed, data sets with missing values or with times series of unequal length have been excluded from the current study, as well as the *Fungi* data set which only contains one instance per class in the training set.

Feature engineering tools. As one of the main goal is to compare several feature extraction methods, it is more suitable to separate the feature generation step from the classification one. Thus, we decided to include in this study only the unsupervised methods, which can be easily used with any classifier (see Table I). We removed deep learning approaches from our study, since generative models like AutoEncoders require very high CPU resources – which leads to hardly fair comparison – and, as highlighted in [23], their predictive performance on TSC problems is mitigated. In addition, based on the TSF, CIF algorithms, as in [46], we have included two methods for extracting some set of descriptive statistics on random sampled intervals i.e. subseries. Nevertheless, since we want to be able to use any off-the-shelf classifier, we are extracting the whole set of features on all the sampled intervals rather than learning one Decision tree on some features subsets as is TSF, CIF.

Finally, we tested a large panel of libraries which are the following:

- ROCKET and its variants ([7], [25], [26]),
- Intervals Transform (which simply apply a summary transformation over multiple random intervals)
- Intervals Catch22 (*catch22* features from random intervals)
- TsFreh ([13]),
- Tsfel ([15]),
- Catch22 ([11]),
- TsFeatures ([16]),
- FeatureTools ([12]),
- Signature ([33])
- Feasts ([17]),
- HCTSA ([18]).

Experimental parameters and classifiers. In order to keep the comparison as fair as possible, we evaluated the different methods limiting the number of features to 1000. Of course, not all libraries generate this number of extracted features (see Table I), *tsfresh* for example, cannot generates more than 794 features without explicitly telling it what to compute. When available, the transformations algorithms have been run using the `sktime` package ([47]). Hyperparameters values are reported in Table II. All experiments have been computed using a single thread of a 4-core 2GHz Intel Core i5-1038NG7 CPU, except for *hctsa* which used 4 workers, with 16Go RAM. Once all the features matrices have been computed, we applied 9 different classifiers, Random Forests (100, 500 trees), Logistic Regression (ℓ_2, $elasticnet$ penalties), XGBoost, SVM (linear, rbf kernel), 1 Nearest Neighbors and Rotation Forest with every other parameters being the default ones, from the `sklearn` library ([48]), except for the Rotation Forest classifier, which is in the `sktime` package [49].

Evaluation methodology. We use pre-defined train/test sets available from the UCR archive and the accuracy as the predictive performance evaluation measure. This way, the comparison with state of the art methods like HC1, HC2, TSChief and InceptionTime – which are computationally expensive – is feasible as we report the results from the original papers. To compare different approaches over several data sets, critical difference diagrams have been drawn using the post-hoc Nemenyi rank based test. Although [50] suggests to use the Wilcoxon signed-rank test with the Holm correction in place of the Nemenyi one, the diagrams drawn this way have been found to be unreadable quite often as it contained some overlapping cliques, as the Wilcoxon signed-rank test is not based on mean ranks, the analysis could have been confusing. Though, we display, when needed, alongside critical Nemenyi

[1]https://github.com/aurelien-renault/Automatic-Feature-Engineering-for-TSC

Category	Transformers	max features	unsup	embed clf	repr	sktime
Pre-defined	hctsa [18]	7730	✓	✗	✓	✗
	tsfresh [13]	794	✓	✗	✓	✓
	tsfel [15]	390	✓	✗	✓	✗
	tsfeatures [16]	37	✓	✗	✓	✗
	feasts [17]	33	✓	✗	✓	✗
	catch22 [11]	22	✓	✗	✓	✓
Constructed	featuretools [12]	inf	✓	✗	✓	✗
	fears [35]	inf	✗	✓	✓	✗
Signature	gen. signature	inf	✓	✗	✗	✓
Convolution	shapelet [36]	$n\_shapelets$	✗	✗	✗	✓
	rocket [7]	inf	✓	✗	✓	✓
Symbolic (SFA)	BOSS & cie ([37], [38], [39])	c^l	✗	✗	✗	✓
	WEASEL [40]	c^{2l}	✗	✗	✗	✓
(SAX)	BOP [41]	c^l	✓	✗	✗	✓
Intervals	TSF [42]	$3r\sqrt{m}$	✗	✓	✗	✓
	CIF [43]	$25r\sqrt{m}$	✗	✓	✗	✓
Deep Learning	AE/VAE [44]	$latent\_dim$	✓	✗	✗	✗
	TS2Vec [45]		✓	✗	✗	✗

TABLE I: Comparison table of several time series transformation methods. (max features: maximum number of generated features, unsup: unsupervised features extraction, embedded clf: use a specific classifier, i.e. can not be used with any off-the-shelf classifier, repr: use of different TS representations, - sktime: present in sktime framework (0.13.0)

Methods	Parameters	Nb features
rocket	$n\_kernels : 500$	1000
minirocket	$n\_kernels : 1000$	924
multirocket	$n\_kernels : 125$ $n\_features\_per\_kernel : 6$	1008
intervals	$n\_intervals : 100$ $agg : [mean, min, max, sum, med, std$ $count, skew, quant(0.25), quant(0.75)]$	1000
intervals c22	$n\_intervals = 45$	1000
featuretools	$agg : [mean, min, max, sum,$ $median, std, count, skew]$ $transf : [D, DD, CUMSUM,$ $DCUMSUM, ACF, PS]$	50
featuretools_1k	$col\_combinations : [add, substr, mult]$	1000
signature	$window\_name : dyadic$ $window\_depth : 4$	930
hctsa	$n\_features : 1000$	1000
catch22	$default\ parameters$	22
feasts		33
tsfeatures		37
tsfel		142-390
tsfresh		789

TABLE II: Libraries' hyperparameters values

diagrams, the binary matrix showing explicitly the Holm adjusted pairwise Wilcoxon comparison for each considered methods (see subpart (b) of figures). In these charts, the small black squares identify pairs of approaches that do not differ significantly in performance.

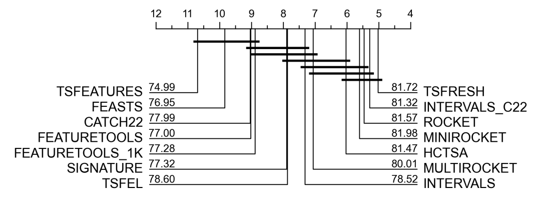
B. Experimental results and Discussion

Performance comparison of feature engineering tools. Due to space limitation, we only report charts for three classifiers: Random Forests, Rotation Forests and Logisitc Regression.
For a default Random Forest classifier, the best performing methods are the ROCKET-like methods, *tsfresh*, *hctsa* and the random intervals extracting the *catch22* features.(see figure 2).

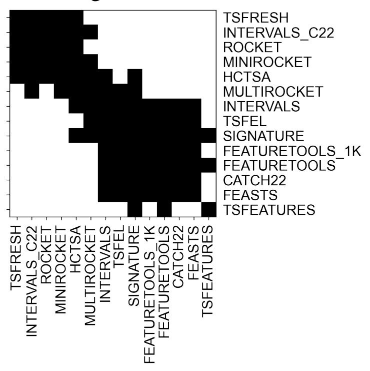
When dealing with linear classifiers though, the ROCKET-like approaches become significantly better than the other tested methods as they are likely to generate independent kernels (see Figure 3). On the other side, the *tsfresh*/*hctsa* features are more likely to be correlated, which can partially explain that best classifiers are non linear ones such as Random Forest or XGBoost. It's also worth noting that data is standardized before being processed by linear classifiers, as they usually embed penalization, which make some of the extracted features useless.

We are also displaying the results obtained for the Rotation Forest classifier [51] (see Figure 4), which is actually the best performer for the great majority of the tested libraries, even if generally less scalable.

Running times comparison. When considering computation time, we observe a clear trend: as expected, all the ROCKET-features based methods outperform the others in term of both time per feature and overall run time. In average the ROCKET-like methods take less than half a second for one given data set. Among the less scalable methods, we find libraries as *hctsa*, *feasts* or *tsfeatures*, as those are designed for analysis and

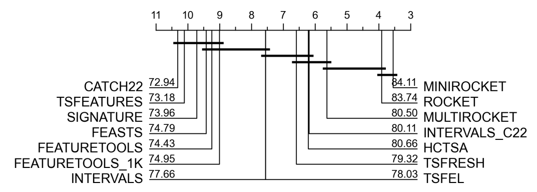


(a) Critical diagram labeled with mean accuracy

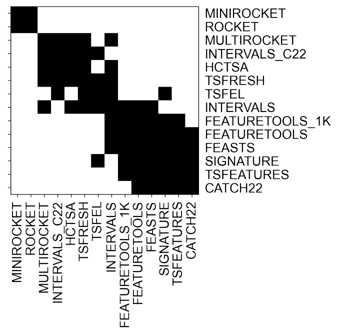


(b) Corrected pairwise comparison

Fig. 2: Random Forest



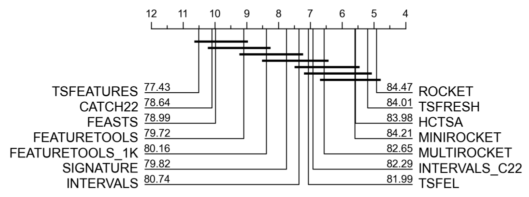
(a) Critical diagram labeled with mean accuracy



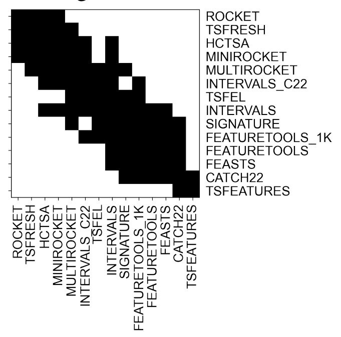
(b) Corrected pairwise comparison

Fig. 3: Logistic Regression

visualization rather than large scale extraction to be include in a automatic classification pipeline. *Hctsa* average run time, for example, is around 10 minutes for one data set (see Figure 5a), *feasts* take more than 4 seconds to extract one feature, which make them the worst scalable libraries among the tested methods. Furthermore, one can notice that bringing *featuretools* from 50 to 1000 generated features slow down the algorithm, doubling the time/feature metric (see Figure 5b), which validate the previously mentioned hypothesis made



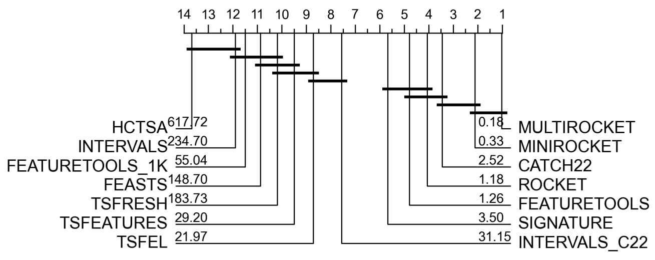
(a) Critical diagram labeled with mean accuracy



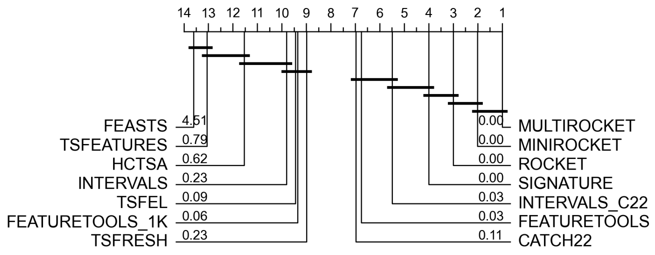
(b) Corrected pairwise comparison

Fig. 4: Rotation Forest

about the library scalability.



(a) Critical diagram labeled with mean run time in seconds

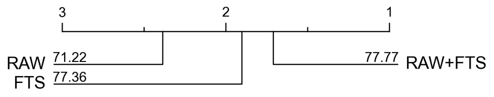


(b) Critical diagram labeled with mean time/feature in seconds per feature

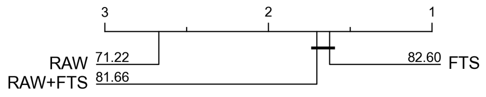
Fig. 5: Computation time performances

C. Raw data, feature construction or both

In order to put some more perspective on our results we analyze the performance when using only the raw time series, i.e. each time point is considered as one feature, as well as the concatenation of the raw data with the extracted features and compare those with the previously obtained results, only using the computed features. Considering these three strategies, the first conclusion is that, extracting features, no matter which method is used, outperforms practically every time a classification only based on the raw time series. As well, Figure 6 demonstrates that, on average, across all strategies for all classifiers, adding the raw data to the feature matrix is performing significantly better.



(a) Strategies comparison: raw data (RAW), with Feature (FTS) and RAW+FTS, for all classifiers



(b) Strategies comparison for minirocket feature, for all classifiers

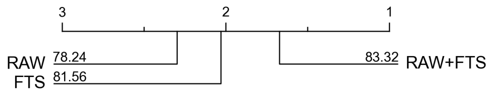
Fig. 6: Performance comparison: adding the raw time series or not labeled with mean accuracy

A more in-depth analysis actually shows that this improvement can be explained by the libraries which tend to generate correlated features. Indeed, in those cases, the best classifiers are the ones that are less sensitive to feature correlation and redundancy, i.e. Random Forest or XGBoost; adding the values of the time series (which could be correlated), when the length of the considered time series remains under a certain threshold, is improving performance in a significant way. However, as the time series length grows, this adding tends to add more noise and, at some point, harm performances, in Figure 7, we are displaying the performance only considering data sets for which the length of the time series are below/above the median length of the UCR repository.

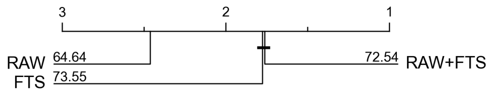
Additionally, one can notice that, only working with data sets whose time series length are beyond the third quartile (around 28 datasets), the overall performance is much degraded and adding the raw data is then ranked behind the FTS strategy. The methods for which linear classifiers were the best never benefit from the raw data: even if some classifiers handle this incorporation better (adding a ℓ_1 penalty for example), they are able, at best, not to degrade the performance too much.

D. Combining libraries

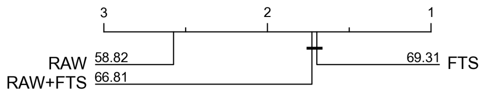
In this part, we define one strategy in order to figure out if performance can be improved by combining several feature engineering tools. To keep things simple, the tested approach consists in gradually stacking the different methods, ranked by one given metric, here accuracy, obtained with a vanilla Random Forest classifier. We are not considering the whole set of ROCKET-like methods any longer, we decided to only retain the 1000 features created by MiniROCKET in a fisrt experiment, before completely deleting MiniROCKET features



(a) $ts\_length < 315$



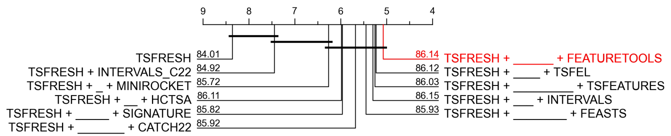
(b) $ts\_length > 315$



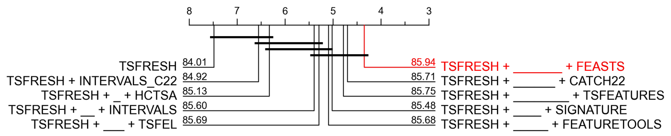
(c) $ts\_length > 720$

Fig. 7: Accuracy results for all strategiess on all classifiers considering time series length

in a second one. Indeed, as ROCKET represents, a TSC approach on its own, we wanted to address whether or not our features set could still be competitive not using this great performing classifier. Additionally, to reduce computation time and redundancy, the expanded version of *featuretools* (with 1000 features) is also removed, as it did not provide any more significant information than its lighter version (with 50 features).



(a) Rotation Forest, the best ranked strategy is called: Features



(b) Rotation Forest, without ROCKET methods, the best ranked strategy is called: Features_noROCKET

Fig. 8: Critical diagrams labeled with mean accuracy
Only the first and last library are displayed, one underscore stands for one in-between library like in the following :
TSFRESH + _ + HCTSA = TSFRESH + INTERVALS_C22 + HCTSA.

It seems intuitive that stacking the different features ones onto others will probably add redundancy; in this way classifier like random/rotation forest seem to better fit this purpose ([49]), we only reports the results for the Rotation Forest in what follows as it provided best performance so far. For this classifier, the best stacking strategy both in terms of ranks and mean accuracy (see Figure 8a) is the ones using

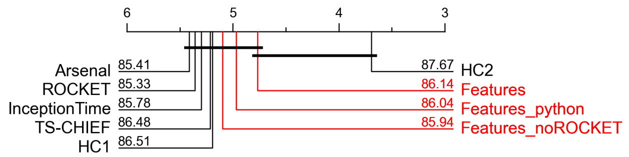
8 tools from the all library set, excluding the ones which are not performing well on their own i.e. *catch22*, *feasts* and *tsfeatures*, which are also the ones creating a limited number of features. When removing MiniROCKET features, some of those libraries become informative and the best strategy, without MiniROCKET, is now containing the all libraries set, only excluding *tsfeatures* (see Figure 8b). The top performers of both of these experiments are called *Features* and *Features_noROCKET* respectively in what follows. Those strategies though, are not performing significantly better that stacking the two or three best libraries ones onto others.

E. Comparison with state of the art TSC approaches

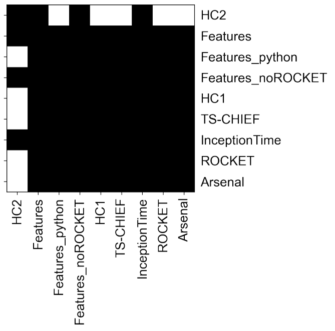
We are now interested in comparing the obtained performance using our previously defined feature generation libraries with state-of-the-art methods, we can see that our methods are not the worst performers (See Figure 9). With the Rotation Forest algorithm (with default parameters) our main approach, noted *Features*, is ranked second among the six others methods, with a the fourth mean accuracy on 112 data sets from the UCR Archive with default train/test splits. Actually, under Nemenyi test, the method is the only one not significantly performing worst than the current most accurate method, namely HIVE-COTE 2.0 (HC2). Figure 10 gives additional results by comparing *Features* to four of the best TSC using 1vs1 scatter plots with standard ±5% to highlight notable difference of performance.

As the selected approach contains in itself some SOTA method with MiniROCKET features, we are also providing results when optimizing combining strategy without it. The *Features_noRocket* approach is, under Holm's correction, as InceptionTime, not significantly worst than HC2: it remains competitive, only loosing 0.2% of mean accuracy comparing to its MiniROCKET version. Replacing MiniROCKET features comes also at the price of around 12 hours more in total training time from features extraction step.

Concerning scalability, Table III demonstrates that the optimal combination of features take a relatively long time to train on the default splits, indeed, it includes the worst scalable methods, i.e. the non-python ones such as *hctsa* and *feasts*. When using only the python available libraries (*Features_python*), we are roughly dividing the total time by 2, it comes at the cost of exactly 0.1% of average accuracy, not loosing any rank compared with the SOTA classifiers. This approach though, becomes significantly worst than HC2. Moreover, it is worth-noting that these train time values do not include features extraction step on the test set, which would actually considerably increase the total time (see Table IV). That being said, even by considering complete feature extraction on both train and test set, our approaches remain substantially faster than HC2 for similar predictive performance on the default split, according to both Nemenyi test and Holm's corrected Wilcoxon signed-rank test.



(a) Critical diagram labeled with mean accuracy



(b) Corrected pairwise comparison

Fig. 9: Predictive performance comparison with state-of-the-art TSC algorithms

Algorithm	Total (h)	Average (min)
ROCKET	2.85	1.53
Features_python	25.15	13.47
Arsenal	27.91	14.95
Features	46.16	24.73
Features_noROCKET	47.62	25.51
InceptionTime	86.58	46.38
HC2	340.21	182.26
HC1	427.18	228.84
TS-CHIEF	1016.87	544.75

TABLE III: Run time to train on a single split on the 112 UCR datasets. For all competitors algorithms, the value for each dataset is the median taken over 30 different resamples. For our approaches, written in red, the value is the one for the default train/test split.

Algorithm	Extraction train (h)	Extraction test (h)	Classifier Fit (h)
Features_python	15.82	37	9.33
Features	34.05	79.58	12.11
Features_noROCKET	38.74	90.1	8.88

TABLE IV: Run time details of the proposed approaches. Training time is the sum of first and third column.

IV. Conclusion

In this paper, we aim at exploring the potential predictive power of feature construction (FC) for time series classification (TSC). To this end, we have designed simple processes to branch existing feature engineering tools with standard classifiers. We have led extensive experiments resulting in the comparison of 11 feature engineering tools branched with

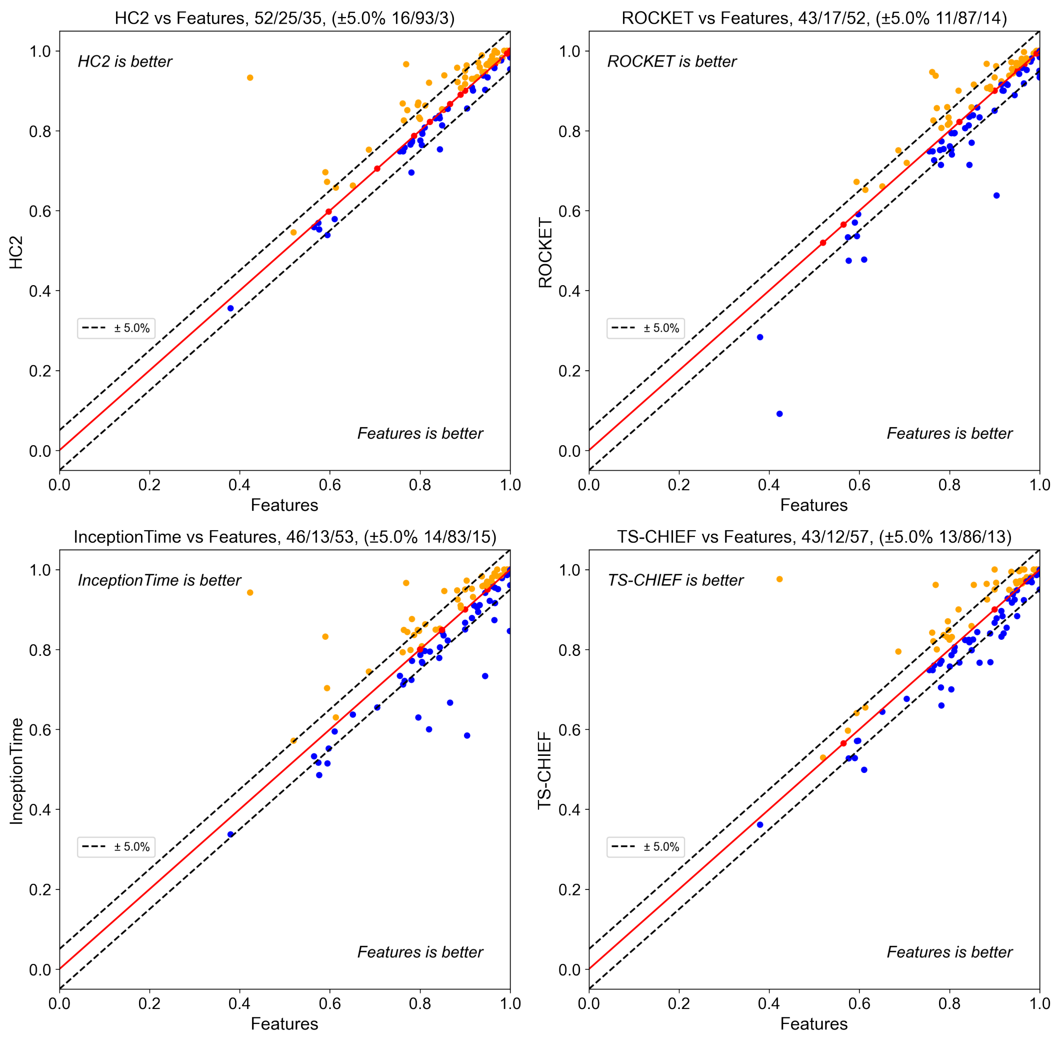


Fig. 10: Scatter plot of accuracy results comparison: our approach "Features" vs state of the art TSC algorithms HiveCote2 (HC2), ROCKET, InceptionTime and TS-CHIEF

9 classifiers over 112 TSC problems – totalling more than 10000 learning experiments. The analysis of the experimental results indicates that *(i)* using feature engineering leads to better predictive performance for a given standard classifier, *(ii)* combining several unsupervised feature engineering tools with the rotation forest classifier is comparable with the best TSC performers in terms of predictive performance while demanding reasonable computing resources. The predictive and efficiency results of the suggested approaches also indicate that feature construction fot TSC is a valuable option to pursue. Indeed, it might be naturally extended to multivariate TSC problems – extracting the same feature set over each dimension or, e.g., randomly sampling some subsets of features in the case of high dimensional data sets. Also, another perspective would be to filter redundant or correlated features to speed up the learning phase.

References

[1] Q. Yang and X. Wu, "10 challenging problems in data mining research," *Int. J. Inf. Technol. Decis. Mak.*, vol. 5, no. 4, pp. 597–604, 2006.

[2] P. Esling and C. Agón, "Time-series data mining," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 12:1–12:34, 2012.

[3] A. Bagnall, J. Lines, W. Vickers, and E. Keogh, "The UEA & UCR time series classification repository." [Online]. Available: www.timeseriesclassification.com

[4] 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 and knowledge discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[5] 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," *Machine Learning*, vol. 110, no. 11, pp. 3211–3243, 2021.

[6] J. Lines, S. Taylor, and A. Bagnall, "Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles," *ACM Transactions on Knowledge Discovery from Data*, vol. 12, no. 5, 2018.

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

[8] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, "Ts-chief: a scalable and accurate forest algorithm for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 3, pp. 742–775, 2020.

[9] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "Inceptiontime: Finding alexnet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, 2020.

[10] B. D. Fulcher, M. A. Little, and N. S. Jones, "Highly comparative time-series analysis: The empirical structure of time series and their methods," *J. Roy. Soc. Interface*, vol. 10, 2013.

[11] C. H. Lubba, S. S. Sethi, P. Knaute, S. R. Schultz, B. D. Fulcher, and N. S. Jones, "catch22: Canonical time-series characteristics," *Data Mining and Knowledge Discovery*, vol. 33, no. 6, pp. 1821–1852, 2019.

[12] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *2015 IEEE int. conf. on Data Science and Advanced Analytics (DSAA)*. IEEE, 2015, pp. 1–10.

[13] M. Christ, A. W. Kempa-Liehr, and M. Feindt, "Distributed and parallel time series feature extraction for industrial big data applications," *arXiv preprint arXiv:1610.07717*, 2016.

[14] 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, 2018.

[15] M. Barandas, D. Folgado, L. Fernandes, S. Santos, M. Abreu, P. Bota, H. Liu, T. Schultz, and H. Gamboa, "Tsfel: Time series feature extraction library," *SoftwareX*, vol. 11, p. 100456, 2020.

[16] R. Hyndman, Y. Kang, P. Montero-Manso, T. Talagala, E. Wang, Y. Yang, and M. O'Hara-Wild, "tsfeatures: Time series feature extraction," *R package version*, vol. 1, no. 0, 2019.

[17] M. O'Hara-Wild, R. Hyndman, E. Wang, D. Cook, T. Talagala, L. Chhay, and M. M. O'Hara-Wild, "feasts," 2019. [Online]. Available: https://cran.r-project.org/web/packages/feasts

[18] B. D. Fulcher and N. S. Jones, "hctsa: A computational framework for automated time-series phenotyping using massive feature extraction," *Cell systems*, vol. 5, no. 5, pp. 527–531, 2017.

[19] J. Lines and A. J. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Min. Knowl. Discov.*, vol. 29, no. 3, pp. 565–592, 2015.

[20] M. G. Baydogan and G. C. Runger, "Time series representation and similarity based on local autopatterns," *Data Min. Knowl. Discov.*, vol. 30, no. 2, pp. 476–509, 2016.

[21] L. Ye and E. J. Keogh, "Time series shapelets: a new primitive for data mining," in *Proceedings of the 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*. ACM, 2009, pp. 947–956.

[22] J. Large, A. J. Bagnall, S. Malinowski, and R. Tavenard, "On time series classification with dictionary-based classifiers," *Intell. Data Anal.*, vol. 23, no. 5, pp. 1073–1089, 2019.

[23] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[24] A. J. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with COTE: the collective of transformation-based ensembles," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 9, pp. 2522–2535, 2015.

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

[26] C. W. Tan, A. Dempster, C. Bergmeir, and G. I. Webb, "Multirocket: Multiple pooling operators and transformations for fast and effective time series classification," *Data mining and knowledge discovery (DMKD)*, 2022.

[27] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[30] A. Bostrom and A. J. Bagnall, "Binary shapelet transform for multiclass time series classification," *Trans. Large Scale Data Knowl. Centered Syst.*, vol. 32, pp. 24–46, 2017.

[31] M. Middlehurst, J. Large, G. C. Cawley, and A. J. Bagnall, "The temporal dictionary ensemble (TDE) classifier for time series classification," in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 12457. Springer, 2020, pp. 660–676.

[32] M. Middlehurst, J. Large, and A. J. Bagnall, "The canonical interval forest (CIF) classifier for time series classification," in *2020 IEEE International Conference on Big Data (IEEE BigData 2020), Atlanta, GA, USA, December 10-13, 2020*. IEEE, 2020, pp. 188–195.

[33] J. Morrill, A. Fermanian, P. Kidger, and T. Lyons, "A generalised signature method for multivariate time series feature extraction," *arXiv preprint arXiv:2006.00873*, 2020.

[34] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The UCR time series archive," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 6, pp. 1293–1305, 2019.

[35] A. Bondu, D. Gay, V. Lemaire, M. Boullé, and E. Cervenka, "Fears: a feature and representation selection approach for time series classification," in *Asian Conf. on Machine Learning*. PMLR, 2019, pp. 379–394.

[36] L. Ye and E. Keogh, "Time series shapelets: a new primitive for data mining," in *Proceedings of the 15th ACM SIGKDD int. conf. on Knowledge discovery and data mining*, 2009, pp. 947–956.

[37] P. Schäfer, "The boss is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.

[38] J. Large, A. Bagnall, S. Malinowski, and R. Tavenard, "On time series classification with dictionary-based classifiers," *Intelligent Data Analysis*, vol. 23, no. 5, pp. 1073–1089, 2019.

[39] M. Middlehurst, W. Vickers, and A. Bagnall, "Scalable dictionary classifiers for time series classification," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2019, pp. 11–19.

[40] P. Schäfer and U. Leser, "Fast and accurate time series classification with weasel," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 637–646.

[41] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, 2003, pp. 2–11.

[42] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Information Sciences*, vol. 239, pp. 142–153, 2013.

[43] M. Middlehurst, J. Large, and A. Bagnall, "The canonical interval forest (cif) classifier for time series classification," in *2020 IEEE international conference on big data (big data)*. IEEE, 2020, pp. 188–195.

[44] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[45] Z. Yue, Y. Wang, J. Duan, T. Yang, C. Huang, Y. Tong, and B. Xu, "Ts2vec: Towards universal representation of time series," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8980–8987.

[46] M. Middlehurst and A. Bagnall, "The freshprince: A simple transformation based pipeline time series classifier," in *Int. Conf. on Pattern Recognition and Artificial Intelligence*. Springer, 2022, pp. 150–161.

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

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[49] A. Bagnall, M. Flynn, J. Large, J. Line, A. Bostrom, and G. Cawley, "Is rotation forest the best classifier for problems with continuous features?" *arXiv preprint arXiv:1809.06705*, 2018.

[50] A. Benavoli, G. Corani, and F. Mangili, "Should we really use post-hoc tests based on mean-ranks?" *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 152–161, 2016.

[51] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012.