Machine Learning (2023) 112:4763–4788
https://doi.org/10.1007/s10994-023-06395-w

WEASEL 2.0: a random dilated dictionary transform for fast, accurate and memory constrained time series classification

Patrick Schäfer[1] · **Ulf Leser**[1]

Received: 6 February 2023 / Revised: 26 June 2023 / Accepted: 16 August 2023 /
Published online: 19 September 2023

Abstract
A time series is a sequence of sequentially ordered real values in time. Time series classification (TSC) is the task of assigning a time series to one of a set of predefined classes, usually based on a model learned from examples. Dictionary-based methods for TSC rely on counting the frequency of certain patterns in time series and are important components of the currently most accurate TSC ensembles. One of the early dictionary-based methods was WEASEL, which at its time achieved SotA results while also being very fast. However, it is outperformed both in terms of speed and accuracy by other methods. Furthermore, its design leads to an unpredictably large memory footprint, making it inapplicable for many applications. In this paper, we present WEASEL 2.0, a complete overhaul of WEASEL based on two recent advancements in TSC: Dilation and ensembling of randomized hyper-parameter settings. These two techniques allow WEASEL 2.0 to work with a fixed-size memory footprint while at the same time improving accuracy. Compared to 15 other SotA methods on the UCR benchmark set, WEASEL 2.0 is significantly more accurate than other dictionary methods and not significantly worse than the currently best methods. Actually, it achieves the highest median accuracy over all data sets, and it performs best in 5 out of 12 problem classes. We thus believe that WEASEL 2.0 is a viable alternative for current TSC and also a potentially interesting input for future ensembles.

Keywords Classification · Time series · Dictionary · Dilation

Editors: Fabio Vitale, Tania Cerquitelli, Marcello Restelli, Charalampos Tsourakakis.

✉ Patrick Schäfer
patrick.schaefer@hu-berlin.de

✉ Ulf Leser
leser@informatik.hu-berlin.de

[1] Humboldt Universität zu Berlin, Berlin, Germany

1 Introduction

A time series (TS) is a collection of values sequentially ordered in time. TS emerge in many scientific and commercial applications, like weather observations, wind energy forecasting, industry automation, mobility tracking, etc. Research in TS is diverse and covers topics like storage, compression, clustering, etc.; see Esling and Agon (2012) for a survey. In this work, we study the problem of time series classification (TSC): Given a concrete TS, the task is to determine to which of a set of predefined classes this TS belongs to, the classes typically being characterized by a set of training examples. TSC has applications in many domains; for instance, it is applied to determine the species of a flying insect based on the acoustic profile generated from its wing-beat (Potamitis & Schäfer, 2014), or for identifying the most popular TV shows from smart meter data (Greveler et al., 2012).

To date, among the most accurate approaches are kernel-based methods (Dempster et al., 2020, 2021; Tan et al., 2022), shapelets (Guillaume et al., 2022) and hybrids (Middlehurst et al., 2021b; Shifaz et al., 2020) (heterogeneous ensembles of base TSC). These heterogeneous ensembles of core classifiers, encompassing kernel/convolution, shapelet, dictionary, and interval classifiers, have emerged as leading performers in the field of time series classification. This conclusion has been recently validated through a comprehensive evaluation involving 33 state-of-the-art classifiers (Middlehurst et al., 2023). Particularly noteworthy are two highly effective classifiers: (a) Hydra-MultiRocket (Dempster et al., 2023), which combines the dictionary classifier Hydra with MultiRocket, and (b) HIVE-COTE 2 (Middlehurst et al., 2021b), a heterogeneous ensemble incorporating a dictionary classifier (TDE (Middlehurst et al., 2021a)), among others. Both highlight a crucial role of dictionary classifiers in achieving high classification accuracy over a diverse set of tasks.

So, what makes dictionary-based classification so important? Previous research (Large et al., 2019; Bagnall et al., 2016, 2017) has emphasized that when the task of distinguishing between classes within a dataset relies on the frequency of subsequence repetitions rather than solely their presence or absence, dictionary methods emerge as the optimal choice of classifier. This becomes particularly relevant in the context of time series data exhibiting periodicity, such as recordings of energy consumption in consumer devices or motion data.

Among various datasets, the PigAirwayPressure dataset[1] highlights the significant advantage of dictionary-based classification over other classifiers, including kernel-based approaches. This dataset consists of airway pressure measurements collected from 52 pigs before and after an induced injury, simulating internal bleeding. As time progresses, internal bleeding can lead to symptoms such as low blood pressure, increased heart rate, and elevated breathing rate. The dataset captures the continuous rise in airway pressure from the start of ventilation until it reaches its highest point, known as the peak inspiratory pressure (PIP).

A distinguishing factor between healthy and injured pigs lies in the number of PIP points observed within the recorded time frame, in this dataset ranging from 1 to 3. The number of PIP points directly corresponds to the fluctuations in breathing rate after the injury is induced. This pattern aligns perfectly with the characteristics that dictionary classifiers, like WEASEL 2.0, are designed to capture and utilize. In comparison to other classifiers, such as ROCKET (Dempster et al., 2020) with a test accuracy of only 8.7% and MultiRocket with accuracy of 60.6%, our novel dictionary approach, WEASEL 2.0,

[1] http://www.timeseriesclassification.com/description.php?Dataset=PigAirwayPressure.

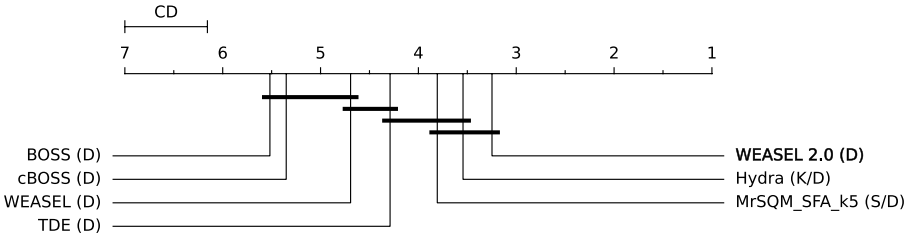


Fig. 1 Critical difference plot on test accuracy for dictionary classifiers on 114 UCR datasets. WEASEL 2.0 is the most accurate dictionary classifier. See Fig. 7 for a comparison to all SotA classifiers

achieves the highest test accuracy among all competitors, with an impressive accurarcy of 93.8% on the default test split. For further details, please refer to the experimental Sect. 5.6.

Notably, classifiers like BOSS, WEASEL, TDE, and MUSE (Schäfer, 2015; Middlehurst et al., 2021a; Schäfer & Leser, 2017a, 2017b) have consistently demonstrated remarkable accuracy and have made significant contributions to the field of time series classification (Bagnall et al., 2016; Ruiz et al., 2021). Yet recently, these dictionary methods have fallen behind in accuracy, and their major drawback is their potentially huge memory-footprint, due to a large variance in generated words. Conceptually, dictionary approaches extract phase-independent subsequences by sliding a window over TS. Each window is transformed into a word, and the frequency of repeating patterns is recorded. A classifier can be learned on the resulting feature vector. WEASEL (Schäfer & Leser, 2017a), which is the basis of this work, uses SFA (Schäfer & Högqvist, 2012) for word generation, and a Ridge regression classifier. The symbolic transformation SFA (Schäfer & Högqvist, 2012) is used in all SotA dictionary-based classifiers. It applies a Fourier transform to subsequences, and discretizes frequencies into symbols.

A recent theme in TSC is to build large, but size-controlled features space of several thousand features, and use these as input to train a linear RIDGE regression classifier. The large feature space is generated from a single high bias transform, by building an ensemble on randomized sets of hyper-parameters (Dempster et al., 2020, 2021; Tan et al., 2022; Guillaume et al., 2022). Both in combination generate a low bias and low variance transformation usable by a linear classifier, for high accuracy and fast prediction. Another recent discovery is the use of a dilation operation applied to a filter, such as the convolutional filter [ROCKET (Dempster et al., 2020, 2021; Tan et al., 2022; Dempster et al., 2023)] or a sliding window operation [R-DST (Guillaume et al., 2022)]. Dilation adds a gap between values, effectively increasing the size of the receptive field of the filter, and also operating similar to a down-sampling operation. Thus, dilation offers features at multiple scales.

In this work, we present a complete overhaul of the dictionary transform WEASEL, using two state-of-the-art techniques: (a) randomized hyper-parameter ensemble and (b) dilation. WEASEL 2.0 addresses many of the shortcomings of current dictionary-based transforms, including the memory foot-print and it inferior accuracy.

Figure 1 shows the advances made by WEASEL 2.0 in comparison to SotA dictionary transforms (Hydra (Dempster et al., 2023) is a hybrid between kernel-based and dictionary, and MrSQM (Le Nguyen & Ifrim, 2022) is a hybrid between Shapelet and Dictionary). On 114 UCR datasets it is the best of its class, and significantly more accurate than its predecessor WEASEL, and furthermore it has a predictable memory footprint (see Fig. 13). Its dictionary is of only some tens of thousand of features, which makes it similar in size to kernel-based methods.

In summary, our contributions are as follows:

1. We introduce a novel dilation mapping. This allows to turn any subsequence-based method into a dilated algorithm, i.e. it is not limited to dictionary-based methods. This mapping can be implemented using just two lines of python code, and can be applied prior to the down-stream classification task.
2. We introduce changes to the symbolic transformation SFA (Schäfer & Högqvist, 2012), used in word generation, to significantly reduce the feature space to just 256 words, including a novel variance-based feature selection strategy. SFA is used in the SotA classifiers and ensembles such as (Le Nguyen & Ifrim, 2022; Middlehurst et al., 2021a, b; Shifaz et al., 2020), thus any improve in SFA potentially benefits these, too.
3. The refined word generation in combination with random ensembling over multiple hyper-parameters produces a predictable size of the feature space. Thereby, we solve the major shortcoming of dictionary methods.
4. Through extensive experiments on the UCR datasets, we show that WEASEL 2.0 is as fast and not significantly different in accuracy than the non-ensemble SotA, namely ROCKET (Dempster et al., 2020), MiniRocket (Dempster et al., 2021), MultiRocket (Tan et al., 2022), or R-DST (Guillaume et al., 2022). Furthermore, WEASEL 2.0 is the best dictionary-based method.
5. Finally, given the multitude of classifiers available and the absence of a universal superior option for all datasets, we aim to provide insights into the characteristics of datasets in which WEASEL 2.0 excels in Sect. 5.6.

Therefore, we believe that WEASEL 2.0 presents itself as a promising alternative for the current time series classification (TSC) landscape, and interesting input for future heterogeneous ensembles.

The rest of the paper is organized as follows: In Sect. 2 we present the background and in Sect. 3 the related work on time series, classification, and dictionary approaches. Section 4 presents the novel WEASEL 2.0. Section 5 shows our experimental evaluation and Sect. 6 concludes the paper.

2 Background

We will first formally define the basic concepts.

Definition 1 *Time Series (TS)*: A time series $T = \left(t_1, t_2, \ldots, t_n\right)$ is an ordered sequence of n real values. We denote the i-th value of T by t_i.

Such a TS is also refereed to as *univariate time series*. If each point represents multiple variables (e.g. humidity, temperature and pressure) we call it a *multivariate data series*.

Most SotA TSC algorithms make use of subsequences of the data.

Definition 2 *Subsequence*: A subsequence $T_{i,l}$ of $T = (t_1, \ldots, t_n)$, with $1 \leq i \leq n$ and $1 \leq i + l \leq n$, is a subseries of length l, consisting of l contiguous points from T starting at offset i: $T_{i,l} = (t_i, t_{i+1}, ..., t_{i+l-1})$

We may extract subsequences from a TS by the use of a sliding window.

Definition 3 *Sliding Window*: A time series T of length n has $(n - l + 1)$ sliding windows of length l, when increment is 1, given by:

$$sliding\_windows(T) = \{T_{1,l}, \dots, T_{(n-l+1),l}\}$$

Time series classification (TSC) is the task of predicting a class label for a TS whose label is unknown. A TS classifier is a function that is learned from a training dataset of discrete, labeled time series, takes an unlabeled time series as input and outputs a label.

Definition 4 *Dataset*: A dataset $D = \left(T^{(i)}, y^{(i)}\right)_{i \in [1,\dots m]}$ is a collection of m time series, each assigned to one of a predefined set of classes Y. We denote the size of D by m, and the i^{th} instance by $T^{(i)} \in \mathbb{R}^n$, and its label by $y^{(i)} \in Y$.

A common operation to reduce the length of a very large TS, is to take every d-th value, dropping all other values, and referred to as down-sampling.

Definition 5 *Down-Sampling*: A time series T of length n is down-sampled by factor d, by taking every d-th value from T:

$$down\_sample(T, d) = T_{1::d} = (t_1, t_{1+d}, t_{1+2\times d}, \dots)$$

Definition 6 *Dilated Subsequence*: A dilated subsequence of T with dilation d and offset i, with $1 \leq i \leq n$ and $1 \leq i + l \times d \leq n$, is a subseries of length l, obtained from down-sampling the time series starting from offset i and then extracting the first l continuous points:

$$dilated\_window(T, i, d) = \left(t_i, t_{i+d\times 1}, t_{i+d\times 2}, ..., t_{i+d\times(l-1)}\right)$$

3 Related work

In this section, we will first introduce the techniques used in time series classification (Sect. 3.1). Next, we will put the focus on dictionary-based methods, the symbolic transformation SFA (Sect. 3.2), and WEASEL (Sect. 3.3).

3.1 Time series classification (TSC)

The techniques used for TSC can be broadly categorized into the following categories (Bagnall et al., 2016). We present a brief overview of algorithms, based on first movers and state-of-the-art in their field.

1. *Distance based* classifiers use a distance function, to measure the similarity, and a classification algorithm on the distances. Historically, distance functions have been mostly used with nearest neighbor (NN) classifiers. Commonly, 1-NN Dynamic Time Warping (DTW) (Rakthanmanon et al., 2012) was used as a baseline in comparisons (Lines & Bagnall, 2014; Bagnall et al., 2016), but is now significantly worse than SotA (Bagnall

et al., 2016; Ruiz et al., 2021). Typically, these techniques work well for short but fail for noisy or long TS (Schäfer, 2015). Furthermore, DTW has a computational complexity of $O(n^2)$ for TS of length n. Recent advances include the Matrix Profile Distance (MPDist) (Gharghabi et al., 2018), which defines two time series to be similar, if they share many phase-independent subsequences.

2. *Exploratory transformations* are a popular recent theme. These extract descriptive statistics as features from time series (subsequences) to be used in classifiers. Several toolkits exist for extracting features, including hctsa (Christ et al., 2018), with over 7700 features. catch22 (Lubba et al., 2019) contains a subset of 22 dominant features from hctsa. *tsfresh* (Christ et al., 2018) is a collection of roughly 800 features.
3. *Shapelets* are phase independent discriminatory subsequences, which *presence* or *absence* can be an indicator of a class of a TS. The expression of a shapelet is found by sliding the shapelet across the TS, and minimizing the Euclidean distance between each subsequence in the TS and the shapelet. The currently most accurate Shapelet approach is R-DST (Guillaume et al., 2022). It combines randomization on hyper-parameters with dilation to increase its diversity. A total of $10k$ shapelets with three features each are extracted, and are fed into a RIDGE regression model. MrSEQL (Agarwal et al., 2021) is an ensemble classifiers that looks for the absence of presence of patterns. But other than the previous method, which minimizes the distance between raw subsequences, MrSEQL discretizes subsequences into words using SFA (Schäfer & Högqvist, 2012) and searches for matches of words. A set of discriminative words is selected through Sequence Learner (SEQL) (Ifrim & Wiuf, 2011).
4. *Dictionary approaches* use phase-independent subsequences by sliding a window over time series, too. But rather than to measure the distance to a subsequence, as in shapelets, each window is transformed into a word, and the frequency of occurrence of repeating patterns is recorded. These methods discriminate based on the frequency of patterns. BOSS (Schäfer, 2015) converts each sliding window into a word by the use of the Symbolic Fourier transformation (SFA) (Schäfer & Högqvist, 2012). A classifier can be built using a non-symmetric distance function in combination with a 1-NN classifier. Temporal Dictionary Ensemble (TDE) (Middlehurst et al., 2021a) is an ensemble of 1-NN classifiers, each transforming the time series into a histogram of word counts using SFA. Conceptually, TDE combines properties of different flavors of dictionary classifiers, like WEASEL (Schäfer & Leser, 2017a), or SpatialBOSS (Large et al., 2019).
5. *Kernel/Convolution* classifiers extend on the idea of Shapelets. Shapelets can be realised through a convolution operation, followed by a min-pooling operation on the array of windowed Euclidean distances. This was first observed by Grabocka et al. (2014). The main difference between convolutions and shapelets is that shapelets are searched for from the candidate space of subsequences extracted from the training data. Yet, convolution filters are found from the entire space of possible real-values. The most well known approach is ROCKET (Dempster et al., 2020), with its successors MiniROCKET (Dempster et al., 2021), MultiROCKET (Tan et al., 2022), and Hydra (Dempster et al., 2023). ROCKET generates tens of thousands of randomly parameterized convolutional kernels, and applies two pooling operations to the output: the maximum and the proportion of positive values. These are used as features in RIDGE regression. The first extension MiniROCKET is significantly faster with an equal accuracy, and reduces the randomization of parameters of ROCKET, making it almost deterministic. MultiROCKET builds on MiniROCKET adding new pooling operations and first order differences. MultiROCKET is one of the most accurate classifiers to date (Tan et al., 2022).

6. *Hybrid:* The most accurate current TSC algorithms are ensembles of multiple types of classifiers. HIVE-COTE v1 (Lines et al., 2016) incorporates classifiers from five domains. It was recently updated to version 2.0 (Middlehurst et al., 2021b) to address scalability issues, and reflect recent innovations. It refined the ensemble to use more accurate classifiers, such as ROCKET [Arsenal (Middlehurst et al., 2021b)] or TDE. To date, this is the most accurate classifier. But it is one to two orders of magnitude slower than kernel-based classifiers while not being significantly more accurate (Tan et al., 2022).
7. *Deep Learning:* InceptionTime (Ismail Fawaz et al., 2020) is an ensemble of deep convolutional neural networks based on the Inception architecture. According to its authors, it is currently the most accurate deep learning approach for TSC. LSTM-FCN (Karim et al., 2017) combines a recurrent neural network with a fully connected neural network. At the time of its publication, it was among the most accurate approaches.

The Rocket-family of approaches utilizes pooling operators to assess both the frequency of subsequence occurrences (PPV) and the presence or absence of subsequences (MAX). PPV bears a resemblance to dictionary-based methods, while MAX mirrors shapelet-based approaches. In fact, the problem of shapelet discovery can be reformulated as convolving the shapelet with the data, followed by a MAX-pooling operation (Guillaume et al., 2022; Grabocka et al., 2014). By integrating these techniques, the Rocket-family approaches effectively harness the advantages of dictionary-based and shapelet-based analyses, enhancing overall classification performance.

3.2 Dictionary-based approaches

Algorithms following the dictionary model build a classification function by:

1. Extracting subsequences, aka *windows*, from a TS;
2. Transforming each window of real values into a discrete-valued *word* (a sequence of symbols over a fixed alphabet);
3. Building a sparse feature vector from word counts, and
4. Finally using a classification method from the machine learning repertoire on these feature vectors.

Fig. 2 illustrates these steps from a raw time series to a dictionary model using overlapping windows.

Dictionary-based methods differ in the concrete way of transforming a window of real-valued measurements into discrete words (discretization). For example, the basis of the BOSS model, TDE (Middlehurst et al., 2021a), or MrSQM (Le Nguyen & Ifrim, 2022), is a symbolic representation called SFA. SFA works as follows (Schäfer & Högqvist, 2012):

1. Values in each window are normalized to have standard deviation of 1 to obtain amplitude invariance.
2. Each normalized window of length w is subjected to dimensionality reduction by the use of the truncated Fourier transform, keeping only the first $l < w$ coefficients for further

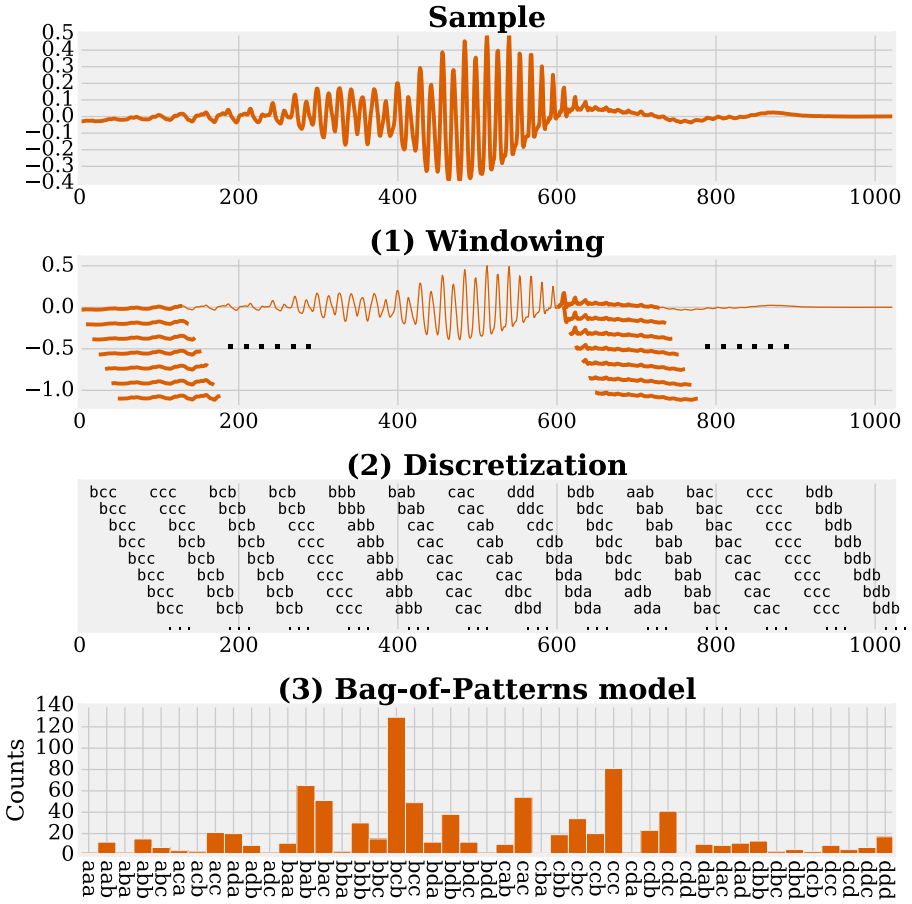


Fig. 2 Transformation of a TS into the dictionary-based model [from (Schäfer & Leser, 2017a)] using overlapping windows (second to top), discretization of windows to words (second from bottom), and word counts (bottom)

analysis. This step acts as a low pass filter, as higher order Fourier coefficients typically represent rapid changes like dropouts or noise.

3. Each coefficient is discretized to a symbol of an alphabet of fixed size α to achieve further robustness against noise.

Figure 3 exemplifies this process from a window of length 128 to its DFT representation, and finally the word *ABDDABBB*.

3.3 WEASEL 1.0

WEASEL, as published in (Schäfer & Leser, 2017a), refined the dictionary approaches to add supervision using class labels. It is the basis of the WEASEL 2.0 model presented in this paper. Thus, we will refer to WEASEL as WEASEL 1.0 in the following.

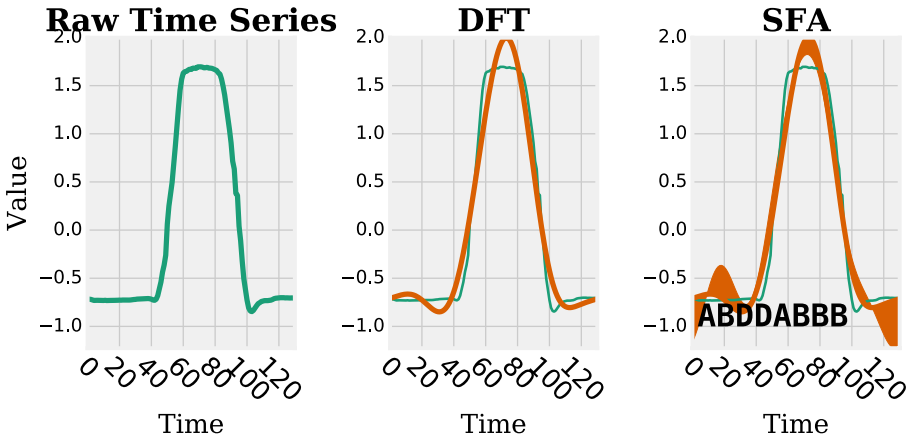


Fig. 3 The Symbolic Fourier Approximation (SFA) [from (Schäfer & Leser, 2017a)]: A time series (left) is approximated using the truncated Fourier transform (center) and discretized to the word *ABDDABBB* (right) with the four-letter alphabet ('a' to 'd'). The inverse transform is depicted by an orange area (right), representing the tolerance for all signals that will be mapped to the same word

WEASEL 1.0 is composed of the building blocks depicted in Fig. 4: a supervised SFA representation for discriminative word generation, and a large sparse dictionary of word counts. First, WEASEL extracts normalized subsequences (windows) of varying lengths from a TS. Next, each window is approximated using the Fourier transform, and the real and imaginary Fourier values are kept that best separate TS from different classes, determined using the ANOVA F-test. These selected Fourier values are then discretized into a word based on information gain binning, using class labels to choose those discretization boundaries to best separate the TS classes; This process is similar to a decision tree split. Finally, a single, large sparse dictionary is built from the words (unigrams), neighboring words (bigrams), over all chosen window lengths. To filter irrelevant words, and reduce the size of the dictionary, a Chi-squared test is applied. A RIDGE regression classifier is trained on the retained dictionaries.

WEASEL 1.0 is still among the fastest classifiers, see Sect. 5.3, but its accuracy is significantly worse than the SotA, and its dictionary can result in excessive memory consumption (Sect. 5.5).

4 WEASEL 2.0 - A random dilated dictionary classifier

We observed two design issues (DI) present in current dictionary classifiers, not just limited to WEASEL 1.0 but also MrSQM, TDE, BOSS:

1. *DI 1: Memory footprint*: A major shortcoming of dictionary based approaches is that the feature space is *huge, but sparse*. This allows for high accuracy classification using linear classifiers, but results in large amounts of memory to be allocated even for small datasets.
2. *DI 2: Sensitivity*: A negative effect of supervision in word generation is that a minor change in two very similar subsequences can result in two distinct words, when it causes a Fourier value close to a discretization boundary to change. This is typically compen-

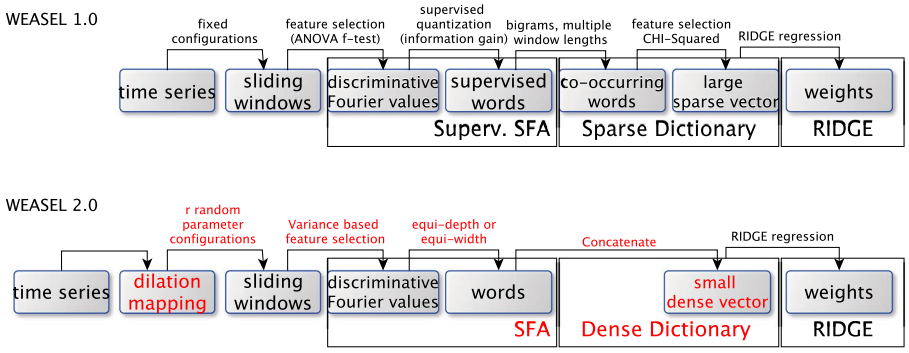


Fig. 4 The WEASEL 1.0 [from (Schäfer & Leser, 2017a)] and WEASEL 2.0 pipelines. Differences are highlighted in red. WEASEL 2.0 adds a dilation mapping prior to the sliding window operator, carefully chooses parameters to control the size and memory consumption of the dictionary, and adds randomization to increase variance

sated by increasing the number of words generated using different parameterizations of SFA. If we however restrict the dictionary size, the increased sensitivity to small changes can deteriorate accuracy.

Our main goals in the design of WEASEL 2.0 were thus to constrain the memory consumption, and add robustness (improve the sensitivity) in word generation. WEASEL 2.0 differs from WEASEL 1.0 in multiple aspects highlighted in red in Fig. 4:

1. Dilation is applied to each time series using a dilation mapping (see Sect. 4.1). This adds state of the art techniques to WEASEL;
2. A fixed-size dictionary of just 256 words is generated for each parameter-set to reduce the impact of minor changes in extracted windows (see Sect. 4.2). This addresses DI 1;
3. A variance-based real and imaginary Fourier value selection strategy for SFA is introduced (see Sect. 4.4). We add first order differences of the time series, just as in MUSE (Schäfer & Leser, 2017b). This addresses DI 2;
4. We apply randomization to choose parameter configurations and thereby increase variance (see Sect. 4.5). This addresses DI 2;
5. The final fixed-size feature vector is small and used for classification through a fast RIDGE regression classifier (see Sect. 4.6).

4.1 Dilation mapping

Dilation is a technique that allows a filter, such as a convolution filter or sliding window, to cover more of the time series data by inserting *gaps* between the entries in the filter. These gaps allow the filter to increase the size of the receptive field, while keeping the total number of values constant. For example, a dilation of $d = 2$ would insert a gap of 1 between every pair of values. This effectively doubles the size of the receptive field, and allows the filter to process the data at different scales, similar to a down-sampling operation.

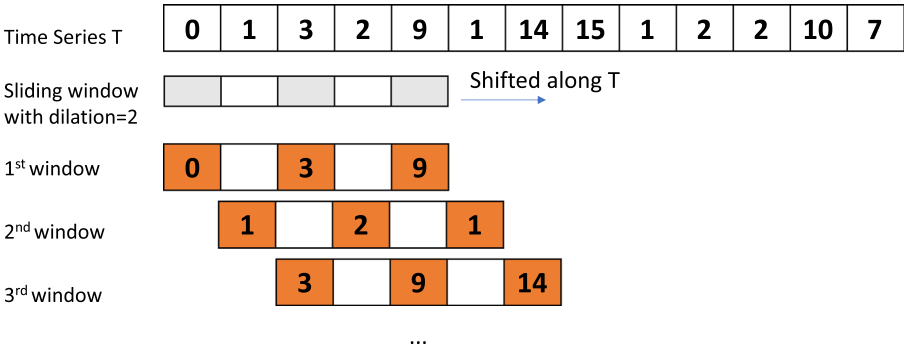


Fig. 5 A dilated windowing operation applied to a time series with $d = 2$

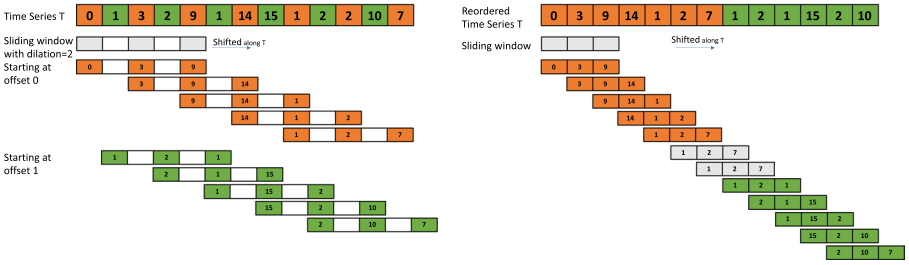


Fig. 6 (Left): A dilated sliding window operation applied to an input time series. This yields two sets of subsequences starting at uneven (orange) and even (green) offsets. (Right): Reordering the green and orange values leads to a sliding window operation with equivalent results as when using dilation

Dilation is one core mechanism related to the recent increase in scalable and accurate classifiers. In TSC it was first proposed by the ROCKET-family of classifiers. Dilation provides analysis at different scales, while keeping the number of values of the filter constant. In ROCKET filter sizes of 7, 9 or 11 are common. In prior methods, the window size was increased up to hundredths of values, increasing computations and memory footprint, and thereby reducing scalability.

Introducing dilation to an algorithm involves a (major) rewrite of its code-base. Large TSC libraries such as sktime (Löning et al., 2019), contain dozens of classifier. Touching every classifier, is not feasible. One of our main contributions is to show that a simple transformation on the time series - which effectively reorders all values - will result in a dilation operation to be applied by the down-stream classification model. This transformation can be applied as a pre-processing step, and can be achieved using just *two lines of python code*. Using this transformation, we do not have to modify the code of any sliding window-based model (such as shapelets, dictionaries, interval) apart from adapting its hyper-parameters.

Figure 5 illustrates the concept of dilation, when used in conjunction with sliding windows. A dilation of $d = 2$ is added to the sliding window, effectively adding a gap of 1 between each value. The dilated sliding window then filters every d^{th} value of the time series, and finally the dilated window can be fed into the downstream processing task.

When coloring the windows extracted from a dilated sliding window with dilation $d = 2$, we find two sets of overlapping subsequences (Fig. 6). The first set of windows

starts at every even offsets (orange). The second set starts at every even offset (green). Within the even or uneven windows, each window overlaps with its successor by all but the *first* and *last* value. This is conceptually equivalent to an ordinary sliding window on a re-ordered TS. In the case of $d = 2$, we first need to take every uneven index $(1, 3, 5, \ldots)$, and then concatenate this subsequence to every even index $(2, 4, 5, \ldots)$. The same concept applies to higher dilation factors $d > 2$: we build groups for the first d indices, take every d-th value in each, and concatenate the resulting d subsequences to form a reordered new TS.

Theorem 1 *The windows extracted by a dilated sliding window, with dilation d, are a subset of those windows extracted from a reordered time series T′, where T′ is constructed by a d-rate down-sampling of indices i* $\in [0, .., d)$:

$$dilated\_window(T, d) \subset sliding\_window\left(T_{0::d} \cup T_{1::d} \cup T_{(d-1)::d}\right)$$

We apply this dilation mapping in the implementation of WEASEL 2.0.

Algorithm 1 Dilation Mapping

```
procedure APPLY_DILATION(TIME SERIES T, DILATION RATE d)
    for i ∈ range(0, d) do
        T' ← np.concatenate([T', T[i :: d]])
    end for
    return T'
end procedure
```

The main advantage of this operation is that we do not have to alter any aspect of the downstream algorithm. We simply reorder the input time series, and train the original model. In principle this operation allows to turn any algorithm into a dilated algorithm. The major disadvantage is that there are a few additional windows generated at the intersections of the re-ordering (the two gray subsequences on the right in Fig. 6). The longer the time series becomes, however, these hardly have any impact.

The dilation mapping is a linear time and space operation in the length of the time series. The dilated time series can be discarded once the classifier has been trained.

To dispel any misunderstanding, the runtime of an algorithm is not significantly affected by the addition of dilation, as the number of extracted subsequences remains relatively stable (see Fig. 6). For any time series with a length of n and a window size of w, we can extract $n - w + 1$ sliding windows with a stride of 1. Similarly, when employing a dilation factor of d, we can extract $n - d \cdot w + 1$ dilated sliding windows. Consequently, the overall reduction in the number of windows is only a small constant, specifically $d \cdot w$.

However, the possibility of accelerating computations arises from the adjustment of hyperparameters in combination with dilation. By expanding the receptive field using dilation, we can utilize smaller window sizes and reduce the number of required window sizes as hyperparameters. This adaptation directly influences the runtime, offering potential improvements in efficiency. Specifically, for dictionary classifiers like BOSS, the maximum window size, denoted as $w_{\max}$, is typically defined as $w_{\max} = n/4$, where n represents the length of the time series. Each window within this range undergoes a Fourier

transform. By utilizing a larger dilation factor, we can effectively cover the same receptive field while reducing the number and size of windows tested in training.

4.2 Dictionary construction

Several methods achieve SotA results (Dempster et al., 2020; Tan et al., 2022; Dempster et al., 2021) using only some thousands of features for high accurate classification though linear models. The feature space of WEASEL 1.0 can grow up to millions of features for even small TS. While statistical feature selection (Chi-squared) has been applied to counter-act this, the number of features above a p-value remains unpredictable. We address this, and build predictable size dictionaries through two design decisions:

1. *Dense Dictionary*: We build a high bias, dense dictionary representation of the time series containing just 256 words. Thus, all subsequences are mapped to one of these words, increasing robustness. However, the low number of words limits the overall classification accuracy.
2. *Ensembling Parameters*: We increase diversity (variance) and reduce bias, through ensembling over multiple parameter configurations (50, 100 or 150) using randomization. This results in a controlled fixed-size dense dictionary of size $256 \cdot 50 \approx 12.8k$ to $256 \cdot 150 \approx 38k$,

4.3 Dense dictionary

SFA generates its dictionary over an alphabet size α and a word length l. The upper bound on the dictionary size is computed as α^l. All subsequences are mapped to one of these words. Using bigrams explodes this feature space to a theoretical upper bound of α^{2l}.

For the default parameters in WEASEL 1.0 ($\alpha = 4$, $l = 6$, $bigrams = True$) this feature space can create $4^{12} = 16M$ distinct words. However, as each subsequence can only generate one word and one bigram, the resulting feature space is sparse. While this diversity can be favorable to identify single words to distinguish between time series, it required large amounts of memory and increases time for training or prediction.

To build a dense dictionary with guaranteed memory consumption, we decided to default $\alpha = 2$, $l = 8$ and disable bigrams. This reduces the size of the dictionary to just $2^8 = 256$. I.e. all subsequences are mapped to one of 256 different words. Technically, this is implemented by using an array of 256 Integers. However, this tiny feature space makes it hard for any linear classifier to find discriminating features, resulting in high bias. To reduce bias and increase variance, we ensemble different hyper-parameter configurations using randomization (Sect. 4.5). In the next section we introduce a variance-based of Fourier values selection strategy to derive robust words using SFA.

4.4 Robust word generation

The supervision introduced in SFA, such as *information-gain* binning, *ANOVA F-test* coefficient selection, and CHI-squared word selection, provide high variance, but are sensitive

to minor changes in values of the time series. This would make SFA very brittle when combined with reducing the overall number of words generated to 256.

To compensate for this, we introduce a novel *variance*-based Fourier value selection strategy. This strategy first computes the variance within each real and imaginary value of the Fourier transform, and then chooses the top l real and imaginary values by highest variance. The larger the variance, the larger the discretization interval (bins) may become. Thus, we avoid minor value changes in similar subsequences leading to different words.

Other than in WEASEL 1.0, which uses information-gain to derive bins, in WEASEL 2.0 we restrict SFA to only use *equi-width* or *equi-depth* binning, as these showed to be the most robust against overfitting.

4.5 Ensemble generation

Our goal was to restrain the number of distinct features (words) to a fixed, predictable number, with which a linear RIDGE classifier can still find distinctive features with high accuracy. We achieve this through randomization on hyper-parameters, where the number of drawn parameter-configurations, aka ensemble size in the following, determines the overall number of features generated per TS. WEASEL 2.0 has three key hyper-parameters to set:

1. **Minimal window length** *w_min*: Typically defaulted to 4
2. **Maximal window length** *w_max*: Typically chosen from 24, 44 or 84 depending on the time series length.
3. **Ensemble size** r_{max}: Typically chosen from 50, 100, 150, to derive a feature vector of roughly $20k$ up to $70k$ features (distinct words).

Other than ROCKET or R-DST, which limit the size of its filter to $\{7, 9, 11\}$, our window size is in a range of 4 up to 84. For a meaningful approximation via SFA, we require large windows. SFA is applied to the windows, extracts Fourier values, and generates words of length 8 and $\alpha = 2$. If we used the Rocket size of filters, the SFA transformation would eventually be a 1:1 mapping from values to symbols, providing no benefits from the Fourier transform.

We use a simple rule of thumb as default for r_{max} based on the dataset size m and ts length n:

$$r_{max} = \begin{cases} 50, & \text{if } m \leq 250 \\ 100, & \text{if } (m > 250) \,\&\, (n \leq 100) \\ 150, & \text{else} \end{cases}$$

The rationale is to use a larger ensemble, the more or the longer the time series in the dataset become.

We then randomly initializes one parameter-set for each of the r_{max}-many configurations using randomization:

1. **Window length** w: Randomly chosen from interval $[w\_min, \ldots, w\_max]$.
2. **Dilation** d: Randomly chosen from interval $[1, \ldots, 2^{\frac{\log(n-1)}{w-1}}]$. The formula is inherited from the ROCKET-family.

3. **Word length** l: Randomly chosen from $\{7, 8\}$.
4. **Binning strategy**: Randomly chosen from {"equi-depth","equi-width"}.
5. **First order differences**: We extract words from both, the raw time series, and its first order difference, effectively doubling the feature space.

For example, when using $r_{max} = 50$ configurations, the feature space has a size of $256 \cdot 50 \approx 10k$, and is roughly in the range of the ROCKET-family. First order differences double the space requirements, effectively generating $20k$ to $70k$ features. By increasing r_{max} we can increase the size of the feature space.

We will next introduce the pseudocode of WEASEL 2.0, illustrating the steps outlined before.

4.6 Pseudocode

WEASEL 2.0 is based on the dilation mapping, and randomization over different hyperparameter configurations, where r_{max} controls the number of configurations and thus the number of features generated.

Algorithm 2 WEASEL 2.0 transform

1: **procedure** TRANSFORM(D: a list of m data series of length m; $\mathbf{D} = \left(\mathbf{T^{(i)}}\right)_{i \in [1...m]}$; w_max and w_min: the maximal and minimal word length; r_{max}: the ensemble size)
2: Let $\mathbf{H}$ be a vector of word counts of dim $m \times (r_{max} \times 256)$
3: **for** $r \leftarrow 1$ to r_{max} **do**
4: $w \leftarrow random.choose([w\_min, ..., w\_max])$
5: $d \leftarrow random.choose([1, \ldots, 2^{\frac{\log(n-1)}{w-1}}])$
6: $l \leftarrow random.choose([7, 8])$
7: $binning \leftarrow random.choose(["equidepth", "equiwidth"])$
8: Let $\mathbf{H'}$ be a vector of word counts with dim $m \times 256$
9: **for** $i \leftarrow 1$ to m **do**
10: $T' \leftarrow$ **apply_dilation**$(T^{(i)}, d)$
11: **for** $j \leftarrow 1$ to $n - w + 1$ **do**
12: $\mathbf{w} \leftarrow \text{SFA\_transform}(T'_{j:w}, l, \alpha = 2, norm = False)$
13: $H'^{(i)}[w] \leftarrow H'^{(i)}[w] + 1$
14: **end for**
15: **end for**
16: $H.concatenate(H')$ ▷ append counts for r-th config
17: **end for**
18: **end procedure**

Algorithm 2 shows the complete WEASEL 2.0 transform. Given r_{max}, w_{max} and w_{min}, we randomly choose the parameters for each configuration. Each configuration then generates a dense vector (dictionary) for each of m time series (line 7), with a total size

of $m \times 256$. Dilation is applied to each time series through the dilation mapping (line 9). By default, we apply dilation to the first order differences, too, effectively doubling the feature space. Next, windows are extracted from each time series (line 10), and the windows are transformed to words of length l using an alphabet of size 2 (line 11). Other than in the previous dictionary-base methods using SFA, the alphabet size is fixed to 2 and the mean-normalization is fixed to *False* to control the size of the feature space. SFA then chooses the best Fourier coefficients based on maximal variance. I.e. those real and imaginary Fourier values that have the largest spread are favorable to obtain robust words. Next, word counts are increased for each time series (line 12). Finally, the dictionaries of all configurations are concatenated into one dense vector (line 13).

At the end of transformation, the resulting feature vector is used as input to a RIDGE regression classifier. Its parameters are learned through cross-validation from $alphas = np.logspace(-1, 5, 10)$ and $normalize = False$.

4.7 Complexity

The major advantage of WEASEL 2.0 is its controlled size of the feature space. Depending on the value r_{max}, we generate up to $256 \times r_{max}$ distinct words, which can be stored in a dense vector. Using first order differences, we double the number of features to $2 \times 256 \times r_{max}$. For input parameter $r_{max} = 50$ we thus generate $25k$ features. Given m time series, the memory required by WEASEL 2.0 is thus equivalent to $256 \times r_{max} \times m \times 2$ using first order differences. E.g., given a dataset with $m = 10k$ and $n = 1024$, WEASEL 2.0 needs roughly $976MB$ of memory for $r_{max} = 50$. This is comparable to the memory needed by ROCKET et al. (see Sect. 5.4)

The runtime required for classification depends on the machine learner used. Yet, RIDGE regression can be implemented in linear time, depending on the solver used, which makes WEASEL 2.0 also very fast.

5 Experimental evaluation

Datasets: We compare our *WEASEL 2.0* classifier to the SotA using the UCR benchmark of 114 TSC problems (Dau et al., 2019). Each UCR dataset provides a train and test split, which we use unchanged to make our results comparable to prior publications. We visualize comparisons with critical difference diagrams, which compare mean ranks of approaches. A horizontal bar indicates cliques, for which there is no statistical significant difference between approaches in rankings. These cliques are computed using a Wilcoxon-Holm post-hoc analysis and p value of 0.05.

Competitors: We compare WEASEL 2.0 to up to 15 state-of-the-art TSC methods. **Dictionary (D):** BOSS, cBOSS, WEASEL, TDE; **Hybrid (H):** HiveCote 2.0, HiveCote 1.0, TS_CHIEF, **Deep-Learning (DL):** InceptionTime; **Shapelets (S):** R-DST, MrSQM_SFA_k5; **Kernel (K):** Arsenal, MiniRocket, MultiRocket, Rocket, Hydra. We used implementations available in sktime (Löning et al., 2019), or published by the authors (Dempster et al., 2023; Le Nguyen & Ifrim, 2022; Guillaume et al., 2022). All reported numbers are accuracy on the test split.

In all figures, we append the names of the methods with its type of approach: D: dictionary, S: Shapelets, K: Kernel, H: Hybrid, DL: Deep Learning.

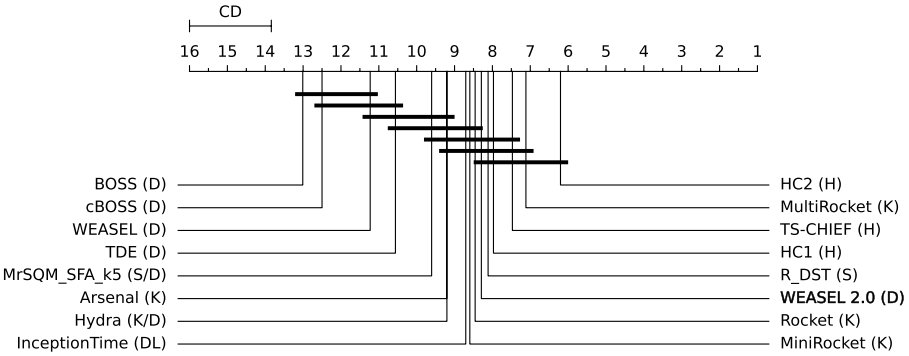


Fig. 7 Critical difference plot on average ranks on test accuracy on 114 UCR datasets. Smaller is better

We will compare methods on (a) accuracy, (b) runtimes, (c) memory-footprint, and (d) the dependency on application domains.

Hardware: All experiments ran on a server running LINUX with four 10 core Intel(R) Xeon(R) CPU E7-4870 at 2.40GHz, using sktime v1.4 and python 3.8.3. We also CPU time as runtime of all implementations, to address parallel and single threaded codes.

To ensure reproducible results, we provide the WEASEL 2.0 source code and the raw measurement sheets. (WEASEL, 2022). WEASEL 2.0 applies the rule of thumb presented in Sect. 4.5 to set the two hyper-parameters r_{max} and w_{max}, and all other parameters are set through randomization.

5.1 Accuracy

Figure 7 shows a critical difference diagram on the average ranking of each competitor method on the 114 datasets. Lower ranks indicate that a method is better, and a horizontal bar indicates that two methods are not significantly different. WEASEL 2.0 is significantly better than WEASEL, and the most accurate dictionary method (compare Fig. 1). WEASEL 2.0 is further in the same group as other non-ensemble dilation-based approaches, such as ROCKET, MiniRocket, R-DST. Among the most accurate methods are hybrids such as HC2, HC1 and TS-CHIEF, which ensembles include variants of dictionary classifiers. Thus, WEASEL 2.0 could be a promising candidate to further improve these.

The box plots in Fig. 8 show how close the accuracy of current SotA approaches has become. Approaches in this figure are sorted by median accuracy. Top ranking approaches have a low IQR (inter-quartile range), with few outliers, and the majority of datasets scores above 90%.

Figure 9 shows a pairwise comparison of WEASEL 2.0 to selected competitors. Each point represents the test accuracy of one dataset. A point below (above) the diagonal indicates that WEASEL 2.0 scores higher (lower) than its competitor. Firstly, WEASEL 2.0 performs much better than WEASEL. Most points are below the diagonal. Secondly from the other competitors most points are along or slightly above/below the diagonals. When compared to the Rocket-family the *PigAirwayPressure* datasets stands out. On this dataset, WEASEL 2.0 has a much higher test accuracy than its competitors. We will discuss the properties of this dataset that make ROCKET fail in Sect. 5.6.

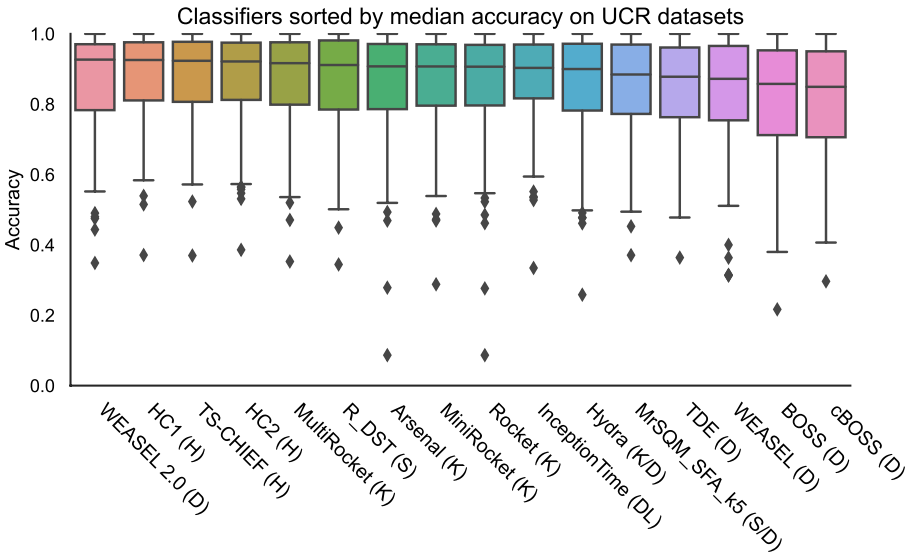


Fig. 8 Box-plot on test accuracy on 114 UCR datasets. Methods are sorted by median accuracy

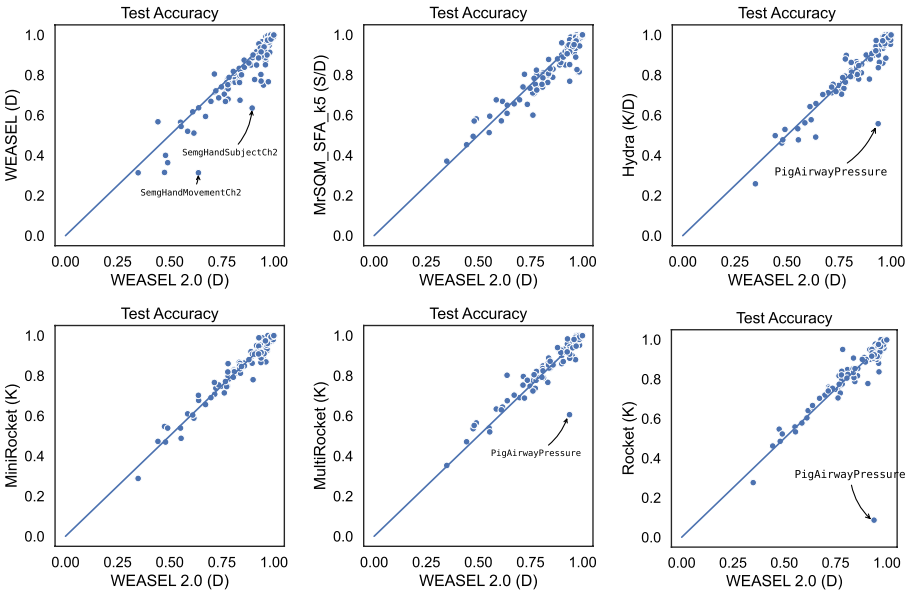


Fig. 9 Pairwise comparison of approaches. Each dataset accounts for one point. A point below (above) the diagonal indicates that WEASEL is more (less) accurate than its competitor on one dataset

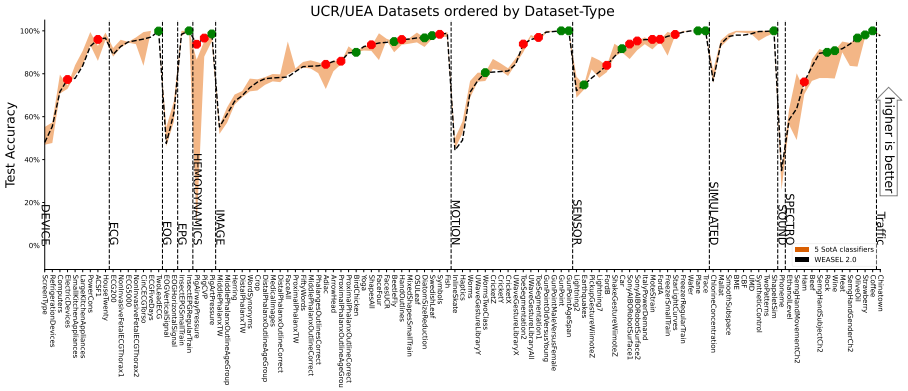


Fig. 10 Classification accuracies for WEASEL 2.0 vs the best five core classifiers (R-DST, MultiRocket, MiniRocket, Rocket, and Hydra). The orange area represents the six core classifiers' range of accuracies. Red (green) dots indicate where WEASEL 2.0 wins (evens out) against the other classifiers

Table 1 Wins and ties by method on 114 UCR datasets

	WEASEL 2.0	MultiR	MiniR	Rocket	Hydra	R-DST
win/tie	**18 / 38**	25 / 46	2 / 22	10 / 31	5 / 29	18 / 38

5.2 Accuracy by dataset and by domain

In this experiment, we thoroughly examine the performance of WEASEL 2.0 across different application domains. To assess domain-specific strengths and weaknesses, we categorized datasets based on their respective application types. This categorization follows the predefined groupings outlined in the benchmark data types established by Dau et al. (2019). In total, there are 12 predetermined groups, namely Device, ECG, EOG, EPG, Hemodynamics, Image, Motion, Sensor, Simulated, Sound, Spectro, and Traffic.

For this particular experiment, we focused on the top 5 non-ensemble competitors. Figure 10 displays the accuracy comparison between WEASEL 2.0 (represented by the black line) and the five leading core classifiers, namely R-DST, MultiRocket, MiniRocket, Rocket, and Hydra. The orange area indicates the range of accuracies achieved by all 5 competitors. A red (or green) dot signifies where WEASEL 2.0 outperforms (or matches) its competitors. Overall, WEASEL 2.0 delivers highly competitive performance across nearly all datasets.

In total, WEASEL 2.0 achieves 18 victories (represented in red) and secures the top score in 38 instances (combining red and green). Notably, WEASEL 2.0 exhibits the highest percentage of top scores in the following groups: Hemodynamics (3 out of 3), Image Outlines (9 out of 31), Motion (5 out of 16), Sensor (10 out of 18), and Spectro (6 out of 12). However, it is outperformed in terms of wins and top scores by MultiRocket, which boasts 25 victories and 46 top scores. For a detailed breakdown, refer to Table 1.

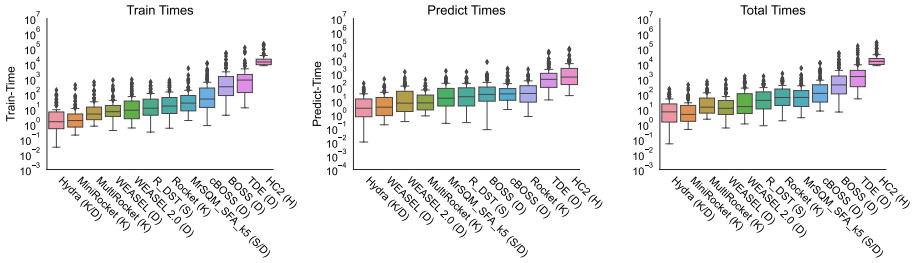


Fig. 11 Runtime (Training, Prediction, Total) on the 114 UCR datasets by method in seconds

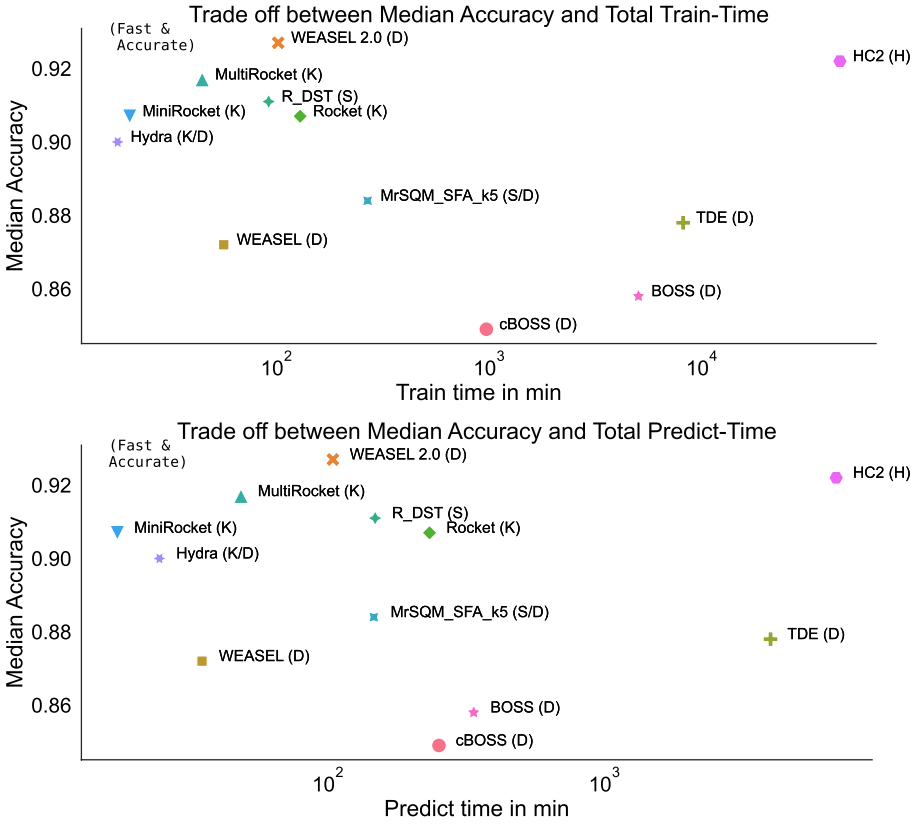


Fig. 12 Total runtime on 114 UCR datasets by approach in minutes against median accuracy

5.3 Scalability

Figure 11 shows the runtimes for training and predictions of each competitor for non-hybrid methods. Hybrid methods typically have a runtime that is at least 10 – 100 times higher. The total fit times vary from 30 minutes (Hydra, MiniRocket) to 130 (for TDE)

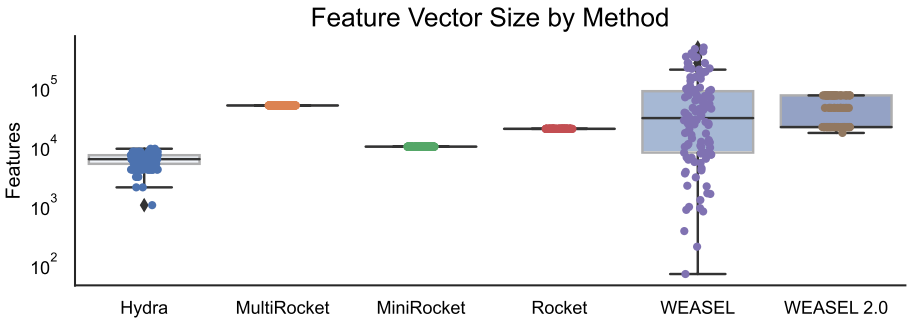
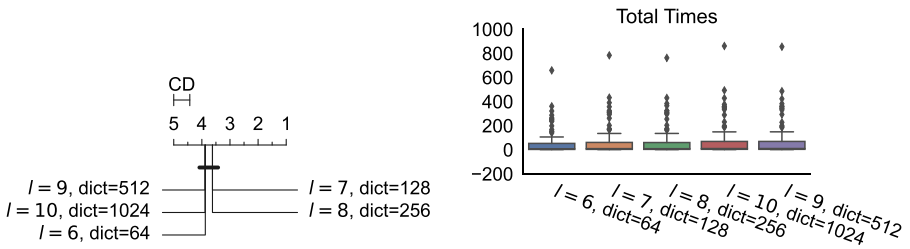


Fig. 13 Size of the feature vector on the UCR datasets. The size of WEASEL 2.0 depends on the parameterization of r_{max}, chosen from 3 values



(a) Accuracy (b) Runtime

Fig. 14 Critical difference plot on average ranks (left) and total runtime (right) for different dictionary sizes

and 730 (for HC2) hours on the full 114 UCR datasets. WEASEL 1.0 requires a total of 1 hour, and WEASEL 2.0 requires 1.6 hours. Predict times are in a similar range of a total of 30 minutes (MiniRocket), 1.7 hours for WEASEL 2.0 and 66 (for TDE) or 110 (for HC2) hours. Thus, while runtimes have not improved for WEASEL 2.0 over WEASEL 1.0, it has the advantage of a constant sized dictionary, and the SotA accuracy. We will highlight the differences in the memory-footprint in the following Fig. 13.

Figure 12 plots the trade-off between the median accuracy and total runtime (train+predict). The most desirable method has a low runtime and a high accuracy (top left corner). The best Pareto-optimal methods, closest to the upper left edge, are WEASEL 2.0, MultiRocket, MiniRocker and Hydra. Inferior methods are BOSS, TDE or HC 2.0 in at least one dimension (accuracy or runtime).

5.4 Size of the feature space

Figure 13 shows the size of the feature space per time series compared to the ROCKET family. For WEASEL the size is defined by the size of the dictionary. WEASEL 1.0 showed excessive memory consumption, which made it unusable for many scenarios. Meanwhile, WEASEL 2.0 has a controlled size of the dictionary, which depends on the

value of r_{max} and the rule of thumb used (Sect. 4.5). Depending of the parameterization of $r_{max} = 50(150)$ WEASEL 2.0 generates $25k(76k)$ features. The size of the feature space for ROCKET and its variants, using default parameters, varies between approaches from $1k$ to $50k$.

5.5 Influence of dictionary size

For each parameter configuration, we decided to choose the word length from 7 and 8, equivalent to a dictionary of 128 to 256 words (compare Sect. 4.5. Within this experiment, we examined the influence of varying word lengths on overall accuracy and runtime.

We conducted tests using five different word lengths, ranging from $l = 6$ to $l = 10$, resulting in the generation of 64 to 1024 words when considering an SFA alphabet of size $\alpha = 2$. Figure 14 shows the critical difference plot (left) and a boxplot on runtimes (right) for these configurations. Our experiment shows that the best accuracy (lowest rank) was achieved with word lengths of 7 and 8. Deviating from these lengths, whether increasing or decreasing, resulted in decreased accuracy. If the dictionary becomes excessively large, similar subsequences are assigned different words. Conversely, if it becomes too small, dissimilar subsequences are mapped to the same word. The optimal range appears to be between 7 and 8.

Regarding runtimes (Fig. 14 right), increasing the word length, i.e. dictionary size, led to a moderate increase in runtime as well. However, it is important to note that an expanding dictionary also requires more memory for WEASEL 2.0 (see Sect. 4.5 for details). Yet, one of our objectives was to construct a dense dictionary to minimize overall memory requirements.

5.6 Understanding when to use WEASEL 2.0

We aim to provide insights into the characteristics of datasets in which WEASEL 2.0 excels. According to previous studies (Large et al., 2019; Bagnall et al., 2016, 2017), dictionary methods perform optimally when classes can be distinguished based on the frequency of subsequence repetition rather than solely their presence or absence.

Figure 15 illustrates the datasets in which WEASEL 2.0 outperforms the top five core classifiers, namely R-DST, MultiRocket, MiniRocket, Rocket, and Hydra. These datasets encompass various domains (as depicted in Fig. 10). Most of them exhibit periodicity, such as recordings of energy consumption in consumer devices (ACSF1, ElectricDevices), motion data (ToeSegmentation 1 and 2, SonyAIBORobotSurface), engine sounds (FordA and FordB), and health data (PigAirwayPressure, PigCVP).

PigAirwayPressure stands out as a dataset where WEASEL 2.0 exhibits a significant performance advantage of 40 to 80 percentage points compared to MultiRocket and Rocket (refer to Fig. 9). This dataset comprises airway pressure measurements collected from 52 pigs before and after an induced injury, specifically a controlled bleeding at a fixed rate of 20 mL/min, simulating internal bleeding.

As time progresses, internal bleeding can lead to symptoms such as low blood pressure, increased heart rate, and elevated breathing rate. The dataset captures the continuous rise in airway pressure from the start of ventilation until it reaches its highest point, known as the peak inspiratory pressure (PIP). The first 10 classes of this dataset are displayed in Fig. 15, and the complete set of classes can be found at (Dau et al., 2019). One of the

Fig. 15 Datasets for which WEASEL 2.0 performs best when compared to the five top ranking core classifiers R-DST, MultiRocket, MiniRocket, Rocket, and Hydra. We show for each dataset one representative time series per class

distinguishing factors between healthy and injured pigs is the number of peaks observed within the recorded time frame, which can range from 1 to 3. These peaks are a result of the fluctuations in blood pressure, heart rate, or breathing rate after inducing an injury. This particular pattern lends itself to an ideal scenario for the application of dictionary classifiers, such as WEASEL 2.0.

Yet, it is important to acknowledge that machine learning does not offer a universal solution, as each classification algorithm possesses its own set of strengths and weaknesses. This observation has been recently reinforced by an extensive assessment involving 33 state-of-the-art classifiers (Middlehurst et al., 2023). Determining the most appropriate algorithm for a specific problem remains an active area of research. Nevertheless, when confronted with datasets characterized by repetitive, phase-invariant, and noisy patterns, we highly recommend considering WEASEL 2.0 as a sensible initial choice.

6 Conclusion

In this work, we have presented WEASEL 2.0, a novel TSC method following the dictionary approach, which achieves highly competitive classification accuracy. It is fast, and other than its predecessors (BOSS, WEASEL, TDE), has a predictable, constant memory footprint. This makes it applicable in domains with high runtime and accuracy constraints.

The novelty of WEASEL 2.0 lies in combining dilation and randomization with the dictionary model, along with a carefully refined symbolic representation for extracting words. WEASEL 2.0 uses small, 256 word, dictionaries derived for each set of input features. While this causes high bias in combination with linear classification, we reduce bias and add variance (diversity) through ensembling random hyper-parameter configurations.

In our evaluation on the UCR datasets, WEASEL 2.0 is significantly more accurate than its predecessors and the best in its dictionary-based class. It is in the group of the best and fastest SotA methods, including the ROCKET-family, and has a predictable constant memory footprint.

Given the multitude of classifiers available and the absence of a universally superior option, we highly recommend considering WEASEL 2.0 as an excellent initial choice when confronted with datasets containing repetitive, phase-invariant, and noisy patterns. Its performance, along with its ability to handle such data characteristics, makes it a compelling option for practical applications.

Author Contributions PS conceived this research, implemented the presented algorithms, performed experiments and analysis, and wrote the initial draft of the manuscript; UL provided supervision, participated in the design and interpretation of the experiments, and participated in the revision of the manuscript. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. No funds, grants, or other support was received.

Data availability The UCR datasets are available at http://timeseriesclassification.com.

Code Availability The WEASEL v2.0 source code is available at https://github.com/patrickzib/dictionary. Other classifiers are available at https://www.aeon-toolkit.org.

Declarations

Conflicts of interest All the authors declared that they have no conflict of interest.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

Agarwal, S., Nguyen, T.T., Nguyen, T.L., et al. (2021). Ranking by aggregating referees: Evaluating the informativeness of explanation methods for time series classification. In *International Workshop on Advanced Analytics and Learning on Temporal Data*, Springer, pp. 3–20.

Bagnall, A., Lines, J., Bostrom, A., et al. (2016). The great time series classification bake off: An experimental evaluation of recently proposed algorithms. Extended Version. Data Mining and Knowledge Discovery pp. 1–55

Bagnall, A., Bostrom, A., Large, J., et al. (2017). Simulated data experiments for time series classification part 1: Accuracy comparison with default settings. arXiv preprint arXiv:1703.09480

Christ, M., Braun, N., Neuffer, J., et al. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh-a python package). *Neurocomputing, 307*, 72–77.

Dau, H. A., Bagnall, A., Kamgar, K., et al. (2019). The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica, 6*(6), 1293–1305.

Dempster, A., Petitjean, F., & Webb, G. I. (2020). Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery, 34*(5), 1454–1495.

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

Dempster, A., Schmidt, D.F., & Webb, G.I. (2023). Hydra: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery* pp. 1–27.

Esling, P., & Agon, C. (2012). Time-series data mining. *ACM Computing Surveys, 45*(1), 12:1-12:34.

Gharghabi, S., Imani, S., Bagnall, A., et al. (2018). Matrix profile xii: Mpdist: a novel time series distance measure to allow data mining in more challenging scenarios. In *2018 IEEE International Conference on Data Mining (ICDM), IEEE*, pp. 965–970.

Grabocka, J., Schilling, N., Wistuba, M., et al. (2014). Learning time-series shapelets. In *Proceedings of the 2014 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM*, pp. 392–401.

Greveler, U., Glösekötterz, P., Justusy, B., et al. (2012). Multimedia content identification through smart meter power usage profiles. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*, p. 1.

Guillaume, A., Vrain, C., & Elloumi, W. (2022). Random dilated shapelet transform: A new approach for time series shapelets. In *International Conference on Pattern Recognition and Artificial Intelligence*, Springer, pp. 653–664.

Ifrim, G., & Wiuf, C. (2011). Bounded coordinate-descent for biological sequence classification in high dimensional predictor space. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 708–716.

Ismail Fawaz, H., Lucas, B., Forestier, G., et al. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery, 34*(6), 1936–1962.

Karim, F., Majumdar, S., Darabi, H., et al. (2017). LSTM fully convolutional networks for time series classification. *IEEE Access, 6*, 1662–1669.

Large, J., Bagnall, A., Malinowski, S., et al. (2019). On time series classification with dictionary-based classifiers. *Intelligent Data Analysis, 23*(5), 1073–1089.

Le Nguyen, T., & Ifrim, G. (2022). Fast time series classification with random symbolic subsequences. AALTD https://project.inria.fr/aaltd22/files/2022/08/AALTD22_paper_5778.pdf

Lines, J., & Bagnall, A. (2014). Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery, 29*(3), 565–592.

Lines, J., Taylor, S., & Bagnall, A. (2016). HIVE-COTE: The hierarchical vote collective of transformation-based ensembles for time series classification. In *IEEE ICDM 2016 Conference*.

Löning, M., Bagnall, A., Ganesh, S., et al. (2019). sktime: A unified interface for machine learning with time series. arXiv preprint arXiv:1909.07872

Lubba, C. H., Sethi, S. S., Knaute, P., et al. (2019). catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery, 33*(6), 1821–1852.

Middlehurst, M., Large, J., Cawley, G., et al. (2021a). The temporal dictionary ensemble (tde) classifier for time series classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, pp. 660–676.

Middlehurst, M., Large, J., Flynn, M., et al. (2021b). HIVE-COTE 2.0: A new meta ensemble for time series classification. *Machine Learning, 110*(11), 3211–3243.

Middlehurst, M., Schäfer, P., & Bagnall, A. (2023). Bake off redux: A review and experimental evaluation of recent time series classification algorithms. arXiv preprint arXiv:2304.13029

Potamitis, I., & Schäfer, P. (2014). On classifying insects from their wing-beat: New results. In *Ecology and acoustics: Emergent properties from community to landscape*, Paris, France.

Rakthanmanon, T., Campana, B., Mueen, A., et al. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 2012 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 262–270.

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

Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery, 29*(6), 1505–1530.

Schäfer, P., & Högqvist, M. (2012). SFA: A symbolic fourier approximation and index for similarity search in high dimensional datasets. In *Proceedings of the 2012 International Conference on Extending Database Technology, ACM*, pp. 516–527.

Schäfer, P., & Leser, U. (2017a). Fast and accurate time series classification with weasel. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 637–646.

Schäfer, P., & Leser, U. (2017b). Multivariate time series classification with WEASEL+MUSE. arXiv preprint arXiv:1711.11343

Shifaz, A., Pelletier, C., Petitjean, F., et al. (2020). Ts-chief: A scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery, 34*(3), 742–775.

Tan, C. W., Dempster, A., Bergmeir, C., et al. (2022). Multirocket: Multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery, 36*(5), 1623–1646.

WEASEL 2.0 - Classifier Source Code and Raw Results (2022). https://github.com/patrickzib/dictionary