Balancing Between Scalability and Accuracy in Time-Series Classification for Stream and Batch Settings

Apostolos Glenis(✉) and George A. Vouros

Department of Digital Systems, University of Piraeus, Piraeus, Greece
apostglen46@gmail.com, georgev@unipi.gr

Abstract. As big data sources providing time series increase, and data is provided in increased velocity and volume, we need to efficiently recognize data provided, classifying it according to their type, origin etc. This is a first important step in doing analytics on data provided from disparate data sources, such as archival sources, multiple sensors, or social media feeds. Time series classification is the task labeling time series using a set of predefined labels.

In this paper we present the K-BOSS-VS algorithm for time series classification. The proposed algorithm is based on state-of-the-art symbolic time series classification algorithms, and aims to achieve high accuracy, balancing with computational efficiency. K-BOSS-VS exploits K representatives of each time series class to classify new series. This provides opportunities for representing intra-class differences, thus increasing the classification accuracy, while incurring a small performance overhead compared to methods using one class representative. Additionally, K-BOSS-VS offers a solution for classifying time-series in batch and streaming settings, due to the opportunities for increasing computational efficiency and the low memory requirements.

Keywords: Time series classification · Distributed processing · Streaming data

1 Introduction

As more and more devices are getting smarter, and sensors become ubiquitous, in conjunction to the increase of media channels and their users, time series data, i.e. data that is tagged with time-stamps, become bigger and bigger. This necessitates to recognize time series flowing into a system from disparate data sources efficiently, classifying data according to their type, origin, quality, trustworthiness etc. with high accuracy. This is a first important step in doing analytics on data provided from disparate data sources, such as archival data sources, multiple sensors, or social media feeds. The task, named time series classification

A. Appice et al. (Eds.): DS 2020, LNAI 12323, pp. 265–279, 2020.
https://doi.org/10.1007/978-3-030-61527-7_18

has become an important task in data analytics pipelines, classifying the type of time series flowing into a system using a set of predefined labels. Time series classification has gained popularity in a variety of fields such as signal processing, environmental sciences, health care, and chemistry.

In this paper we address the problem of time series classification both in batch and streaming settings, with the objective to balance between computational efficiency and accuracy. Efficiency is important in dealing with big data sources (either due to volume or velocity), while accuracy is needed for automating the task: Addressing this trade-off is the objective of recent works in time-series classification, e.g. in [3,20,23].

In this paper we present the K-BOSS-VS algorithm for time series classification, which is based on state-of-the-art symbolic time series classification algorithms, such as BOSS-VS [20]. In contrast to these, K-BOSS-VS exploits K representatives of each class to classify new series. This provides opportunities for increasing the degrees of parallelism used - although it incurs a computational overhead compared to methods that use a single representative, while increasing the classification accuracy, due to addressing intra-class time series modalities, in contrast to methods using a single representative per class (e.g. class centroid).

More specifically, our proposed algorithm achieves high accuracy, which is comparable, if not higher, to state of the art algorithms. Additionally, it offers a solution for classifying time-series in batch and streaming settings, due to the opportunities for distributing the task in multiple workers and due to the low memory requirements compared to methods comparing a new series to each class member. However, as already pointed out, it incurs a small performance overhead compared to methods using one class representative, which is due to the need to compare with K representatives per class.

The contributions made in this work are as follows:

- We introduce a new method that scales for big data sources, without sacrificing accuracy on time series classification.
- We evaluate and compare the proposed method against state of the art algorithms, showing its ability to achieve highly accurate results, with a small performance overhead that can be absorbed in distributed settings.
- We show the efficacy of the proposed method in batch and streaming settings.

The structure of the paper is as follows: Sect. 2 describes the time series classification problem, while Sect. 3 describes related work and explains the contributions made in this paper. Section 4 gives a high-level description of the proposed algorithm and Sect. 5 describes the actual implementation on top of Apache Spark. Section 6 provides experimental results of the proposed algorithm, compared against state of the art algorithms. Finally, Sect. 7 concludes the paper.

2 Problem Formulation

In our problem definition a time-series T is defined as a series of values ordered by their timestamp, i.e. $T = t_0, t_1 ..., t_{m-1} ....$

Given a set of n labels $L = \{l_0, l_1..., l_{n-1}\}$ the goal is to train a classifier $f : \mathbf{T} \rightarrow L$ to label time-series in $\mathbf{T}$ (target time series) using a label in L.

In this paper we treat all data sources as sources providing batches of time series data: This means that given a time horizon H, we consider all values that are provided within that time horizon. Thus, the time series provided within H time instants from a specific source is of specific length. Of course, this length may vary between data sources, depending on data source velocity and frequency of sampling. In batch settings we fetch series of values within time windows of duration H, while in streaming settings we get values from a starting time point t_0, until $t_0 + H$. The time horizon may be tuned in different cases, depending on the data sources used, and according to domain-specific requirements concerning the classification speed. We do not deal with this problem in this paper.

Having fixed the time horizon H, the classification task follows the λ architecture [17] paradigm, where, while sources might very well be streams, a preprocessing step converts part of the stream (i.e. the values provided within a specified time horizon) to a batch dataframe.

3 Related Work

As we are focusing on symbolic representations of time series, below we provide state of the art methods in this line of research: Starting from SAX and SAX-VSM, SFA, BOSS, BOSS-VS and WEASEL.

In [23] the authors present the SAX-VSM classification algorithm that uses SAX representation and tf-idf weighting. SAX transforms a series of values into a word. The range of values is first divided into segments usually following the Gaussian distribution and then each segment is mapped to a letter from a given alphabet. A tf-idf vector is created for each class of the training, after the training set has been transformed into SAX words. Then the set of target time series is transformed into SAX words and the Term Frequency (TF) vector is created for each time-series in the target set. Finally, to classify a time series, the cosine similarity between the TF vectors of that series and of classes' centers is computed.

In [19] the authors introduce the Bag of SFA Symbols Ensemble classifier (BOSS) that uses Symbolic Fourier Approximation (SFA) [21] to classify a time series. To compute SFA words, a number of Fourier coefficients are computed, which are grouped based on common prefixes, building histograms per group, discretized, and mapped to an alphabet. Thus, the SFA approximation, and thus BOSS, uses a symbolic representation based on the frequency domain, providing information about the whole series. Properties of this representation lead to significant lower training times compared to using the SAX representation. The BOSS ensemble classifier is based on 1-NN classification using multiple BOSS models at different time series substructural sizes. However, BOSS requires the entire training set to be available while classifying target time series. Because the training set is large, the memory requirements, in addition to the computational complexity incurred, prevent BOSS from being a candidate for big data time series classification, although it is very accurate.

In [20] the authors present the BOSS-VS classification algorithm where each data point in the training set is transformed into SFA words. Then a centroid is created for each class and the cosine similarity is computed between centroids and target series, as in [23]. This significantly reduces computational complexity and the memory footprint of the classification algorithm, since now each target time-series is only compared to each class centroid. This makes BOSS-VS suitable for big data and streaming data sources, but it is less accurate than BOSS. This trade-off between computational complexity/scalability and accuracy of classification motivates our work, addressing issues concerning bid data, streaming and batch data sources.

In [22] the authors propose WEASEL as a middle ground between BOSS-VS [20] and BOSS [19] for time series classification, balancing between accuracy and scalability. It uses SFA, but it does a few novel things: First, WEASEL considers differences between classes during feature discretisation, second it uses windows of variable lengths, also considering the order of windows, and finally it uses statistical feature selection, leading to significantly reduced runtime.

There is not much available work on distributed time series classification. A work that is close to our aims is [3]; where the authors present a distributed algorithm that uses shapelets and a random forest classifier. Their algorithm scales well compared to the centralized version and achieves an average accuracy of 82% for one of the data sets and 99% for the other.

Concerning classification of time series in streaming settings, in [15] the authors present a method for classifying time series data using Time Series Bitmaps (TSBs) based on SAX, which are shown to be maintained in constant time. Given that TSBs are very close to a normalized Term Frequency vector, this work is considered to be using a compact signature of the training set's time series to classify a streaming time series. While TSBs are robust to concept drift and spotting new behaviour, authors in [15] use a single centroid per class, resulting to accuracy that is not as high as that achieved by state of the art methods mentioned above.

Finally, in [16] the authors propose a method based on Piecewise Linear Approximation (PLA). Each streaming time series is transformed to a vector by means of a PLA technique. The PLA vector is a sequence of symbols denoting the trend of the series (either UP or DOWN), and it is constructed incrementally. The author proposes efficient in-memory methods in order to a) determine the class of each streaming time series, and b) determine the streaming time series that comprise a specific trend class. In contrast to that approach we do not explicitly model trends, we use SFA for the symbolic representation of time series, and we do not compute things incrementally. Modelling trends and incremental computations are useful features that we shall consider in future work.

In addition to the above approaches, there are recent proposals for time series classification using deep learning methods:

While deep learning approaches, such as [7,10,13,14,24,25], report results that are better than baseline approaches and close to the state of the art time series classification methods, their computational complexity and

sample-efficiency while training, together with their memory requirements and accuracy scores, impose specific limitations.

Other methods, such as the Leveraging Bagging method in [5], combining bagging with randomization to the input and output of the classifiers using the ADWIN [4] change detector, the Adaptive Random Forests (ARF) in [12], including a theoretically sound resampling method and adaptive operators that can cope with different types of concept drift, Kappa Updated Ensemble (KUE) [6] using an ensemble classification algorithm for drifting data streams, address explicitly the concept drift problem that we do not address in this work.

4 Algorithm Description

As described in Sect. 3, BOSS is highly accurate but requires that the entire training set is compared to each target time-series using an 1-NN classifier. This, implying that the entire training set is available during classification, incurs specific scalability limitations, which are crucial for big data, batch and streaming settings. On the other hand, BOSS-VS and SAX-VSM use a single centroid for each class label. This makes them more suitable for big data and streaming settings since the computational complexity in terms of comparisons needed, as well as their memory footprint, is significantly reduced compared to BOSS. The problem with using a single centroid to represent a class label is the reduction on the accuracy achieved, given that a single centroid may not be representative of all time series patterns in a class (this is apparent when comparing the accuracy of BOSS-VS and SAX-VSM against BOSS).

The main intuition behind the method proposed here is that instead of having a single centroid to represent each class label, we can have K representatives per class. Our approach requires that K is a bound constant so that the memory and computation time remains close to BOSS-VS and SAX-VSM, while at the same time the use of K representatives preserves, or even increases, the classification accuracy of BOSS-VS. We choose SFA as the symbolic representation for our algorithm, similarly to BOSS, given its representation flexibility and superiority compared to SAX. In addition, BOSS-VS has been proved superior to SAX-VSM in terms of accuracy, something which is also shown in our experiments.

We apply K-means to each class label of the training set to obtain K representatives per class. After we have obtained the representatives for each class, to classify a target time series we compute the cosine similarity between the normalized term frequency vector of that series and the normalized term frequency vector of each of these representatives. The target time series is assigned the label of the closest class representative, using an 1-NN classifier.

5 Implementation

We implemented the proposed method in Apache Spark [29] using Spark's MLLIB [18] for computing the tf-idf vector of each time series, as well as for determining the K representatives per class using K-means. To implement the

1-NN classification step we use the Dataframe API. For the streaming use case we used Spark Streaming, more specifically the Spark Structured Streaming [1] API. The Structured Streaming API makes Dataframe operations of Apache Spark available in a streaming environment.

Before delving into the implementation, we provide very succinctly some preliminaries on Apache Spark.

Apache Spark [29] is a unified engine for distributed data processing. Spark uses the MapReduce [8] programming model, but extends it with an abstraction called Resilient Distributed data sets (RDDs) [27]. RDDs provide a distributed memory abstraction that allows in-memory computations on large clusters in a fault-tolerant manner. Using RDDs, Spark can express a vast array of workloads that needed separate processing engines. Example workloads include SQL [2,9], streaming workloads [28], machine learning [18] and graph processing [26]. Apart from RDDs another Spark abstraction that is relevant to our case is Spark DataFrames which are RDDs of records with a known schema. Spark DataFrames get most of their functionality from the DataFrame abstraction for tabular data in R and Python, and essentially model a database table. Dataframes provide methods for filtering data, computing new columns, and aggregating data. In Spark Dataframes, operations map down to operations of the Spark SQL engine and as such, they use all available optimizations.

Coming to the implementation of K-BOSS-VS, Algorithm 1 shows the auxilliary function for data pre-processing. Line 1 first groups the data from each data source to H-time instances' chunks to create the time series. Then it sorts the target column (the column containing the measurements) by the column containing the time stamp, to account for data points that are out of (temporal) order in the data set. This is important, because real-world time series might contain out-of-order values. It must be noted that we have set an horizon of 24 h, that fits better in the data sets used in our experiments. Different time horizons could be applied for other data sets.

Lines 2 to 8 transform time series to SFA words and then create windows of the SFA representation (SFA windows: In our experiments the SFA window length is 4 h). Finally, the algorithm returns a dataframe with the transformed column together with its label.

Algorithm 2 shows the algorithm for computing the representatives per class. In line 2 the algorithm pre-processes the data set using the auxiliary functions provided above. Then in line 3 it creates the hashing term frequency for each document corresponding to the SFA windows of a single time-series of time horizon H. Then in line 4 each term frequency vector is normalized using L2-normalization. The next step of the algorithm is to compute the representatives for each class label (column) of the data set (line 5) using MLLIB's K-means. After that, each column (class) of the data set has K class representatives.

Algorithm 1: processColumnTrain Algorithm

Input : timestampColumn: The name of the column containing the timestamp
Input : inputDF: The input Dataframe to be processed
Input : field_groupedBy: The field to group by the measurement, usually the data source ID
Input : outputFunction: The function that transforms the time-series into the symbolic representation
Input : outputColumnName: The column of the input Dataframe containing the measurements
Output: someDF: the Dataframe with the transformed column and associated label

```
1 grouped2 ← inputDF. groupBy (inputDF.col (field_groupedBy), window
  (inputDF.col (timestampColumn),"1 day")) .agg ( collect_list
  (struct(inputDF.col (timestampColumn), inputDF.col (outputColumnName)))
  .as ("columnSorted")) .withColumn
  (outputColumnName+"SortedByTimestamp", getValueFromTuple(sort_array
  ("columnSorted", asc = false)))
2 withAppendedColumnsRdd ← grouped2.rdd.map (row => {
3 myArray ← row.get(outputColumnName+"SortedByTimestamp")
4 newColumn2 ← outputFunction(myArray)
5 getColumns ← List(row.get(field_groupedBy))
6 Row.fromSeq(getColumns :+newColumn2) } )
7 someDF ← withAppendedColumnsRdd.toDF
8 return someDF.withColumn ("variable_name", lit(outputColumnName))
```

To classify a set of target time series, we use the following process:

1. The data set is grouped into intervals of length H and then converted into SFA windows similarly to the training set.
2. Each group of SFA windows is converted to a term frequency vector and normalized as in the training set.
3. The similarity between the normalized term frequency vector of the target time series and the class representatives is computed.

The proposed K-BOSS-VS approach borrows from BOSS-VS and SAX-VSM the low memory footprint, as it uses only a subset of training examples as class representatives. Indeed, it uses a number of representatives per class to balance between scalability (of BOSS-VS) and accuracy (of BOSS): The number of class representatives is small compared to all the series available in the training set, but can be large enough to represent different intra-class modalities and provide accuracy improvement compared to BOSS-VS and SAX-VSM. Furthermore, the additional number of class representatives must be small enough that it can be stored in memory.

It must be noted that both, BOSS-VS and the K-BOSS-VS implementations are inspired by the Lambda Architecture [17], where models are trained in batch setting and tested on an setting that can be either batch or streaming.

Algorithm 2: ComputeRepresentatives Algorithm

Input : training: The Dataframe containg the training set
Input : columns: List of column names
Input : timestampColumn: The name of the column containing the timestamp
Input : fieldGroupBy: The field to group by the measurement, usually the data source ID
Input : windowsColumnName: The label of the column to contain the result of the output function
Input : outputFunction: The function that transforms the time-series into the symbolic representation
Output: retVector: The class representatives together with their label

```
for column ← columns do
    currDataframe ← processColumnTrain(timestampColumn, training,
    fieldGroupBy, outputFunction, windowsColumnName, column)
    rescaledData2 ← hashingTF.transform (currDataframe)
    l2NormTrain ← normalizer.transform (rescaledData2)
    model ← kmeans.fit(l2NormTrain)
    retVector += (column -> model.classRepresentatives)
end for
return retVector
```

6 Evaluation

In our evaluation we examine SAX-VS, BOSS, BOSS-VS, K-BOSS-VS, in terms or their accuracy, execution time and scalability. We implemented parallel versions of SAX-VSM, BOSS and BOSS-VS in Apache Spark [29] using Spark's MLLIB [18] for tf-idf and term frequency weighting, so as to have a fair comparison with the parallel version of the proposed K-BOSS-VS method.

For the batch setting we compared our algorithm against BOSS (since it provides state-of-the-art classification accuracy), SAX-VSM and BOSS-VS (since they are more efficient and scalable than BOSS). For the streaming case we compared K-BOSS-VS with BOSS-VS (since, it outperforms SAX-VSM in all the cases in the batch setting, and it is more suitable for streaming settings), and we evaluated the execution time and scalability of the algorithms for both the training phase and the testing phase. As part of the evaluation we measured both the total execution time and speedup, while for the streaming case we also provide results concerning the testing time. For the K-BOSS-VS method the scalability and execution time measurements both in the batch and in the streaming use-case are performed with K equal to 16. This is further justified below.

6.1 Experimental Setup and Data Sets

We tested all algorithms in a cluster with 10 computing nodes in total, where one node of the cluster is reserved as the driver-node in Spark, and the remaining 9 nodes are used as workers. The hardware specifications for each computing node is as follows: 2*XEON e5-2603v4 6-core 1.7 GHz with 15 MB cache, with 128 GB of RAM and 256 GB SSD.

Details of the configurations of parallelism are depicted in Table 1.

Table 1. Configurations of parallelism

Number of executors	Number of cores per executor	Parallelism
9	4	36
4	4	16
1	4	4
1	1	1

To evaluate the algorithms we have used the following data sets:

1. The *power measurement* data set[1] used in [11]. This data set contains two classes, one for watts and another for temperature.
2. The Intel Data Lab Sensor (*lab data*) data set[2]. This data set contains 2.3 Million readings from sensors with classes temperature, humidity, light and voltage.

The time horizon in our experiments for both data sets is equal to 24 in order to create time-series per day. This created time series of different length for each data data set: In Intel Lab data set we have about 58415 data points per day for a total of 38 days (0.67 datapoints per second). In power measurements data we have a data point every 10 s which means we have 8640 points per day.

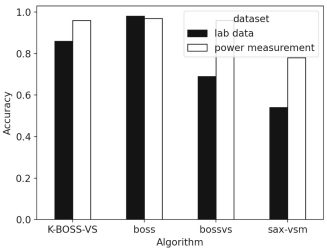
In our tests 80% of each data set is randomly assigned to the training set, and the remaining 20% to the test set.

In the streaming case we used the training set to obtain the representative vectors per class, and then we used the test set to simulate a streaming setting: For this purpose the test set in JSON was fed into the Apache Spark structured streaming readStream function.

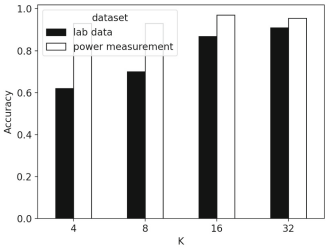
6.2 Evaluation Results

As we can see from Fig. 1a the K-BOSS-VS method is a middle ground between BOSS and BOSS-VS in terms of accuracy, while it outperforms SAX-VSM. This

[1] https://github.com/UniSurreyIoT/KAT/raw/master/logic/data.csv.
[2] http://db.csail.mit.edu/labdata/labdata.html.



(a) Accuracy of the methods

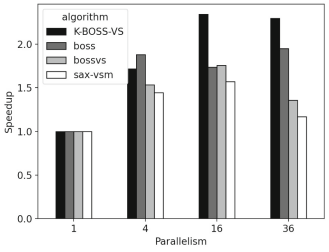


(b) Accuracy of K-BOSS-VS varying K

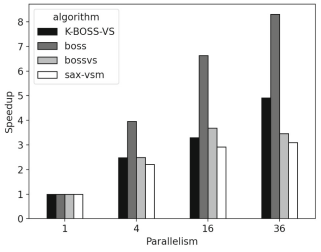
Fig. 1. Accuracy results

proves that having multiple class representatives per class (in contrast to BOSS-VS and SAX-VSM) provides increased accuracy especially on classification tasks with multiple labels, as in the Intel Data Lab Sensor data set. All the algorithms provide high accuracy in the power measurement data set, but the performance of the algorithms differs in the lab data case. The K-BOSS-VS algorithm provides an accuracy of 96% for the power measurement data set and 86% for the lab data data set for $K = 16$ compared to 96% and 98% accuracy scores, respectively, from BOSS.

Figure 1b shows the accuracy of K-BOSS-VS for varying values of parameter K. This algorithm reaches high accuracy for the power measurement data set even for low values of K. For the lab data data set the increase of K proves beneficial. For $K = 16$ K-BOSS-VS uses 41% of the time series per class for the lab data and 48% of the time series per class for the power measurements data set. While increasing K would result in accuracy similar to BOSS, the comparisons needed per target time series would also be similar to BOSS, as the percentage of representatives per class would increase. These results supports our intuition that a sufficient number of class representatives is beneficial to



(a) power measurements data set

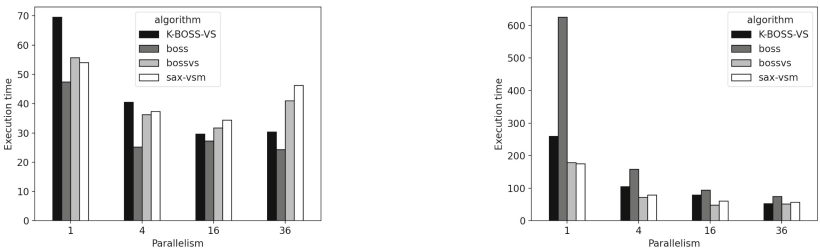


(b) lab data set

Fig. 2. Speedup of methods in the batch setting, per data set.

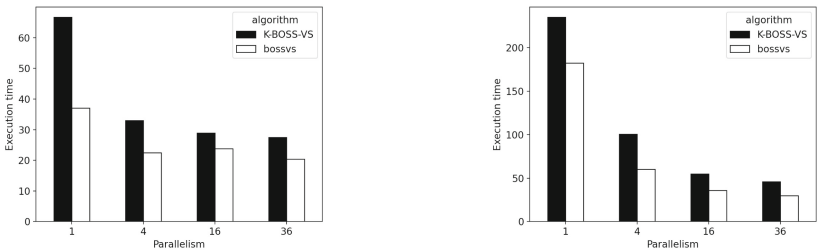
the accuracy of the algorithm, while we can effectively reduce the comparisons needed per target case.

Figures 2a and 2b show the speedup for the batch settings for all four algorithms on the two data sets. As we can see, based on our implementations, BOSS scales better than SAX-VSM and BOSS-VS as parallelism increases, given that it can distribute comparisons of test cases with each of the class members effectively. For the power measurement data set the K-BOSS-VS method provides the best speedup, while is has the second best speedup after BOSS for the lab data set.



(a) power measurements data set (b) lab data set

Fig. 3. Total methods' execution time in the batch setting, per data set.



(a) power measurements data set (b) lab data set

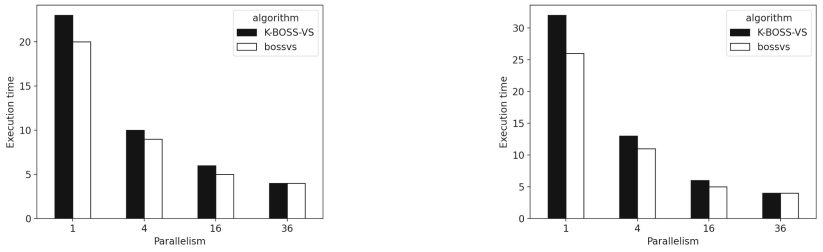
Fig. 4. Training time in the streaming setting for K-BOSS-VS and BOSS-VS, per data set.

Figures 3a and 3b show the total execution time (i.e., the time for the training and test-phase of the algorithms) in the batch setting. On the power measurements data set the results are not what we would expect, especially when comparing BOSS with BOSS-VS and SAX-VSM (and with K-BOSS-VS): This may be due to the fact that the data set is small and auxiliary operations add

non-trivial execution time. On the larger and more complex data set the picture is different. BOSS, as expected, has the largest execution time in the low parallelism settings, but as parallelism increases it becomes competitive to the other algorithms. The K-BOSS-VS algorithm reports high execution time in low parallelism settings, compared to BOSS-VS and SAX-VSM, although it is much faster than BOSS. In high parallelism setting K-BOSS-VS matches the performance of BOSS-VS and SAX-VSM. It must be noted that for parallelism equal to 36 the training time for K-BOSS-VS is 47 s and the test time is 6 s. More results regarding the test time are provided in the streaming settings, where these are more relevant.

Figures 4a and 4b show the execution time for training on the two data sets for BOSS-VS and the K-BOSS-VS method for the streaming case. As we can see the K-BOSS-VS algorithm takes longer time to train, as expected, due to the cost incurred on computing the K representatives per class. However, it becomes more competitive with increased parallelism.

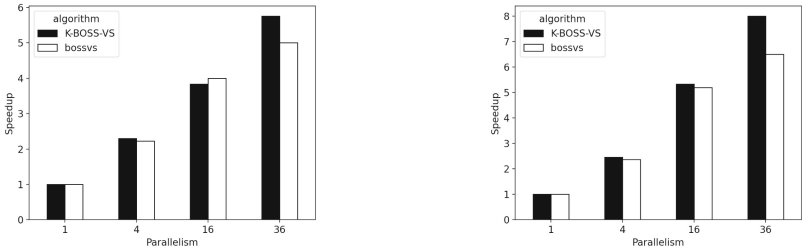
Figures 5a and 5b show the execution time for the test part of the algorithms in streaming settings. The time needed for online classification is important - especially in the streaming case, since, the training phase of the streaming algorithm can be performed offline in the fashion of the Lambda Architecture [17]. As we can see, although the K-BOSS-VS algorithm is competitive to BOSS-VS, it is still slightly slower in low parallelism settings, as it incurs additional cost for testing with multiple representatives per class. The performance benefits of BOSS-VS are less apparent with increased parallelism and the two algorithms perform identically for parallelism equal to 36 (given that K = 16), although K-BOSS-VS achieves similar (in the power measurements data set) or better (in the lab data set) accuracy scores, as shown above.



(a) power measurements data set (b) lab data set

Fig. 5. Testing time in the streaming setting, for K-BOSS-VS and BOSS-VS, per data set.

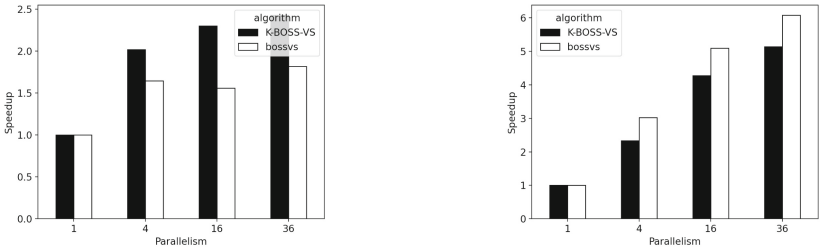
Figures 6a and 6b show the scalability of the test phase in the streaming setting. As we can see the K-BOSS-VS method scales better than BOSS-VS due to higher opportunities for parallelism, given that it can compare each target time series to K representatives for each of the classes, in parallel.



(a) power measurements (b) lab data set

Fig. 6. Speedup for testing in the streaming setting, for K-BOSS-VS and BOSS-VS, per data set.

Figures 7a and 7b compare the scalability of BOSS-VS with the scalability of the K-BOSS-VS algorithm in the training phase. As we can see, in the smaller data set K-BOSS-VS scales better, whereas in the larger data set BOSS-VS scales better, given the cost incurred by K-BOSS-VS in comparing each target series with K representatives per class. However, despite the fact that the algorithms are close in terms of scalability, K-BOSS-VS is significantly more accurate than BOSS-VS in the later case.



(a) power measurements (b) lab data set

Fig. 7. Speedup for training in the streaming setting, for K-BOSS-VS and BOSS-VS, per data set.

7 Conclusions

In this paper we have introduced the K-BOSS-VS algorithm for time-series classification in batch and streaming settings. Our proposed algorithm uses K representatives per class to compare each test case using an 1-NN classifier, providing accuracy similar or better than state of the art classifiers that compare each test

case with every member of each class. On the contrary, although the algorithm incurs a performance overhead compared to algorithms using a single representative (centroid) per class, maintaining K representatives per class, achieves higher accuracy. However, on high parallelism settings it matches the performance of algorithms that use a single centroid due to better scalability. The small memory footprint of the algorithm, the high accuracy, and the efficiency of the classification despite the performance overhead that it incurs - mainly in the training phase, make it suitable for streaming applications.

As future work we plan to investigate trends' representations by means of class representatives, as well as incremental computations, aiming to further increase the computational efficiency of the method while increasing further its accuracy. We also plan to investigate time series classification methods following paradigms that are different to the methods used here, such as decision trees, also addressing the concept drift problem. Finally, we plan to investigate time series classification purely on a stream setting without having to train on batch.

Acknowledgement. This work is partially supported by the University of Piraeus Research Center.

References

1. Armbrust, M., et al.: Structured streaming: a declarative API for real-time applications in apache spark. In: Proceedings of the 2018 International Conference on Management of Data, pp. 601–613 (2018)
2. Armbrust, M., et al.: Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1383–1394. ACM (2015)
3. Baldán, F.J., Benítez, J.M.: Distributed fastshapelet transform: a big data time series classification algorithm. Inf. Sci. **496**, 451–463 (2019)
4. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM International Conference on Data Mining, pp. 443–448. SIAM (2007)
5. Bifet, A., Holmes, G., Pfahringer, B.: Leveraging bagging for evolving data streams. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010. LNCS (LNAI), vol. 6321, pp. 135–150. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15880-3_15
6. Cano, A., Krawczyk, B.: Kappa updated ensemble for drifting data stream mining. Mach. Learn. **109**(1), 175–218 (2020)
7. Cui, Z., Chen, W., Chen, Y.: Multi-scale convolutional neural networks for time series classification. arXiv preprint arXiv:1603.06995 (2016)
8. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
9. Engle, C., et al.: Shark: fast data analysis using coarse-grained distributed memory. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 689–692. ACM (2012)
10. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Transfer learning for time series classification. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 1367–1376. IEEE (2018)

11. Ganz, F., Barnaghi, P., Carrez, F.: Automated semantic knowledge acquisition from sensor data. IEEE Syst. J. **10**(3), 1214–1225 (2014)
12. Gomes, H.M., et al.: Adaptive random forests for evolving data stream classification. Mach. Learn., 1469–1495 (2017). https://doi.org/10.1007/s10994-017-5642-8
13. Hüsken, M., Stagge, P.: Recurrent neural networks for time series classification. Neurocomputing **50**, 223–235 (2003)
14. Karim, F., Majumdar, S., Darabi, H., Chen, S.: LSTM fully convolutional networks for time series classification. IEEE Access **6**, 1662–1669 (2018)
15. Kasetty, S., Stafford, C., Walker, G.P., Wang, X., Keogh, E.: Real-time classification of streaming sensor data. In: 2008 20th IEEE International Conference on Tools with Artificial Intelligence, vol. 1, pp. 149–156. IEEE (2008)
16. Kontaki, M., Papadopoulos, A.N., Manolopoulos, Y.: Continuous trend-based classification of streaming time series. In: Eder, J., Haav, H.-M., Kalja, A., Penjam, J. (eds.) ADBIS 2005. LNCS, vol. 3631, pp. 294–308. Springer, Heidelberg (2005). https://doi.org/10.1007/11547686_22
17. Marz, N., Warren, J.: Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co., New York (2015)
18. Meng, X., et al.: MLlib: machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
19. Schäfer, P.: The boss is concerned with time series classification in the presence of noise. Data Min. Knowl. Disc. **29**(6), 1505–1530 (2015)
20. Schäfer, P.: Scalable time series classification. Data Min. Knowl. Disc. **30**(5), 1273–1298 (2015). https://doi.org/10.1007/s10618-015-0441-y
21. Schäfer, P., Högqvist, M.: SFA: a symbolic Fourier approximation and index for similarity search in high dimensional datasets. In: Proceedings of the 15th International Conference on Extending Database Technology, pp. 516–527. ACM (2012)
22. Schäfer, P., Leser, U.: Fast and accurate time series classification with weasel. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 637–646. ACM (2017)
23. Senin, P., Malinchik, S.: SAX-VSM: interpretable time series classification using SAX and vector space model. In: 2013 IEEE 13th International Conference on Data Mining, pp. 1175–1180. IEEE (2013)
24. Smirnov, D., Nguifo, E.M.: Time series classification with recurrent neural networks
25. Wang, Z., Yan, W., Oates, T.: Time series classification from scratch with deep neural networks: a strong baseline. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 1578–1585. IEEE (2017)
26. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: GraphX: a resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems, p. 2. ACM (2013)
27. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, p. 2. USENIX Association (2012)
28. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 423–438. ACM (2013)
29. Zaharia, M., et al.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016)