International Journal of Computational Intelligence Systems (2021) 14:186
https://doi.org/10.1007/s44196-021-00036-7

RESEARCH ARTICLE

SCMFTS: Scalable and Distributed Complexity Measures and Features for Univariate and Multivariate Time Series in Big Data Environments

Francisco J. Baldán[1] · Daniel Peralta[2,3] · Yvan Saeys[2,3] · José M. Benítez[1]

Received: 3 September 2021 / Accepted: 12 October 2021

Abstract
Time series data are becoming increasingly important due to the interconnectedness of the world. Classical problems, which are getting bigger and bigger, require more and more resources for their processing, and Big Data technologies offer many solutions. Although the principal algorithms for traditional vector-based problems are available in Big Data environments, the lack of tools for time series processing in these environments needs to be addressed. In this work, we propose a scalable and distributed time series transformation for Big Data environments based on well-known time series features (SCMFTS), which allows practitioners to apply traditional vector-based algorithms to time series problems. The proposed transformation, along with the algorithms available in Spark, improved the best results in the state-of-the-art on the Wearable Stress and Affect Detection dataset, which is the biggest publicly available multivariate time series dataset in the University of California Irvine (UCI) Machine Learning Repository. In addition, SCMFTS showed a linear relationship between its runtime and the number of processed time series, demonstrating a linear scalable behavior, which is mandatory in Big Data environments. SCMFTS has been implemented in the Scala programming language for the Apache Spark framework, and the code is publicly available.[1]

Keywords Time series · Time series features · Feature-based approach · Big Data · Scalability

Abbreviations

AB	AdaBoost
ARIMA	Autoregressive integrated moving average
CV	Cross-validation
DFST	Distributed FastShapelet Transform
DT	Decision tree
FS	FastShapelet
IoT	Internet of things
KNN	K-Nearest neighbors
LDA	Linear discriminant analysis
LOSO	Leave-one-subject-out
MTS	Multivariate time series
RDD	Resilient distributed datasets
RF	Random forest
SCMFTS	Scalable and distributed complexity measures and features for univariate and multivariate time series
ST	Shapelet transform
UCI	University of California Irvine
WESAD	Wearable stress and affect detection

✉ Francisco J. Baldán
fjbaldan@decsai.ugr.es

Daniel Peralta
daniel.peralta@irc.vib-ugent.be

Yvan Saeys
yvan.saeys@ugent.be

José M. Benítez
J.M.Benitez@decsai.ugr.es

[1] Department of Computer Science and Artificial Intelligence, DiCITS, iMUDS, DaSCI, University of Granada, 18071 Granada, Spain

[2] Data Mining and Modelling for Biomedicine, VIB Center for Inflammation Research, Technologiepark-Zwijnaarde 71, 9052 Ghent, Belgium

[3] Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Krijgslaan 281, S9, 9000 Ghent, Belgium

[1] Baldán et al. [1] in Scalable Complexity Measures and Features for Times Series classification. https://github.com/fjbaldan/SCMFTS/.

1 Introduction

Nowadays, we can find devices generating data anywhere and at any time [2]. With the expansion of new technologies, the volume of data generated is growing by leaps and bounds. Until now, the typical time series data comes from well-known fields, for example, from the stock market [3], from industry with power consumption logs [4], or from medical fields with specific applications, such as electrocardiograms [5]. However, nowadays we have access to a lot of new devices like smartwatches which continuously generate information through their incorporated sensors, such as heart rate, temperature or humidity monitors. All the data sources mentioned contain information of high importance that needs to be extracted and used to improve the service offered.

All the examples mentioned above are related to recording one or multiple magnitudes over time, generating a specific type of data named time series. Any magnitude regularly recorded over time is a time series. On the one hand, forecasting future values of a time series has been a very popular topic [6]. Forecasting stock price trends [7] is the most typical example but also one of the hardest. On the other hand, the search for patterns in time series is a task that is attracting more attention each day. Problems like detection of fraudulent energy consumption [8] or detection of heart malfunctions [9] are gathering increasing attention from the research community. With increasingly cheaper sensors and more complex models, new problems of interest appear every day.

There are growing numbers of cases in which multiple variables are recorded at the same time in the same process [10], generating multivariate time series (MTS). MTS problems have additional complexity due to the relationships between the different variables that compose each MTS. An example of this kind of problem is the forecast of the energy demand when additional meteorological variables are available like temperature, humidity, or wind speed, among others.

The data volume generated by the expansion of IoT scales quickly providing a quantity of data that is not processable by the traditional computation model. The concept of Big Data arises in order to face this kind of problem. The MapReduce paradigm [11] proposes a distributed computational model that can face a high volume of data efficiently. Apache Spark [12] is a popular framework that offers high-speed capabilities and includes the MapReduce workflow. Spark has one of the most extensive libraries for machine learning in Big Data environments, MLlib [13], and an extra repository with some untested proposals named spark packages [14]. Although the MLlib has the most representative algorithms for machine learning, the set of available algorithms is still limited. At the time of writing this paper, there are few tools for time series processing in MLlib or spark packages, and in general in Big Data environments.

In this work, we propose a scalable and distributed time series transformation based on well-known time series features, named SCMFTS, to provide an alternative vector-based representation of time series that enables the use of the traditional machine learning techniques available in Big Data environments. We have implemented it in Apache Spark through Scala, guaranteeing a fully scalable behavior, being the first proposal of this type made for Big Data environments. The code is publicly available [1]. SCMFTS allows practitioners to face problems that would otherwise be impossible and to improve the results obtained through the additional information provided by the new time series features. The proposed transformation is applicable for univariate and multivariate time series. It has been tested for effective accuracy and linear scalability.

The remainder of this paper is organized as follows. In Sect. 2, we analyze the works related to our proposal. Section 3 explains the transformation proposed, the time series features selected, and the workflow of our proposal. In Sect. 4, we summarize the obtained results. Finally, we show the conclusions of our work in Sect. 5.

2 Related Works

In this section, we analyze the state-of-the-art of time series processing in Big Data Sect. 2.1 and the main Big Data frameworks Sect. 2.2.

2.1 Time Series in Big Data

For almost a decade, the processing of enormous amounts of time series has been an active research topic. One of the most representative works tries to process trillions of time series subsequences through the dynamic time warping distance measure [15], which has a high complexity. After this first work, we can see a succession of new proposals that try to face the problem of processing time series to a larger scale. For example, the FastShapelet (FS) algorithm [16] which provides a reduction in time complexity of the original proposal at a cost in accuracy, proposals of a generic and scalable framework for automated anomaly detection to deal with large-scale time series data [17], a fast and scalable Gaussian process modeling oriented to astronomical time series [18], or a scalable distance-based classifier for time series named proximity forest [19], show us the growing interest in processing larger and larger sets of time series. However, we can see how the limitations of the traditional computation model and computation systems are also there in these works. Limitations in the available resources to deal with a large problem, becoming impossible its storage in memory or obtaining unacceptable running times, among others.

To face the limitations mentioned above the Distributed FastShapelet Transform (DFST) algorithm [20] has been introduced, the first time series classification algorithm developed in a distributed way. DFST joins the low complexity of the FastShapelet (FS) algorithm with the Shapelet Transform (ST) [21] performance. ST proposes to use the distance between the selected shapelets and each time series in the dataset as the input features. For this reason, we can consider it as a feature-based method. The performance of ST depends on the machine learning algorithm used on the transformed dataset, but it provides competitive results concerning the best proposals of the state-of-the-art. In addition, the DFST method lets us apply the traditional vector-based algorithms already available in Apache Spark to time series problems, expanding the tools to process this kind of data. But this approach can only be used in supervised problems.

The time series analysis problem has additional characteristics with respect to the traditional vector-based problems. Characteristics like time dependency, trend, seasonality, or stationarity of a time series, among others, must be considered in the algorithm proposals, raising the complexity of the methods or adding some limitations to them and making the application of the proposed methods in distributed environments impossible. For example, on the one hand, FS analyzes the entire dataset sequentially and provides the best decision tree, based on shapelets founded, evaluating each shapelet with the complete dataset. On the other hand, DFST evaluates the shapelets candidates in a distributed way on the data available in each node and saves the most valuated. This is because the shapelets evaluation process has a high complexity, which makes it impossible to apply it to the complete dataset in Big Data environments.

Following the feature-based approach used in DFTS and ST, but without depending on shapelets, we can find multiple works based on extracting time series features. For example, the application of multiple types of mathematical operations over a time series to obtain valuable information from it that explains the underlying structure of their behavior [22, 23], the selection from a theoretical point of view of the most representative features of a time series that could explain their behavior [24], or the proposal of a set of 22 characteristics [25] through exhaustive experimentation which guarantee these features as the most representative of the original set of features, among others.

Unsupervised feature extraction has been applied in other domains [26]. In the particular case of time series, recently, it has been demonstrated that a set of well-known complexity measures and time series features is able to provide competitive results concerning the state-of-the-art of univariate [27] and multivariate [28] time series classification. To extrapolate this approach to a distributed Big Data environment is necessary to filter and prepare the selected features to be totally independent of each other and do not require relationships between different time series or additional information. These conditions allow their inclusion in a distributed environment, increasing the limited amount of tools for time series processing in Big Data environments.

2.2 Big Data Frameworks

Even with the previous scalable—but not distributed—proposals, some problems are impossible to process when storage needs become untractably large for a single computer. The MapReduce paradigm addresses these issues by proposing a distributed computation model that joins the capabilities of multiple computers to obtain the necessary resources transparently for the user. The MapReduce paradigm is based on two types of operations:

- The *map* operation distributes throughout the cluster the computation needed over each instance of the dataset. This operation is applied independently over each instance, and there is no possible interaction between different instances.
- The *reduce* operation brings to the cluster driver the generated results for a previous map operation. In this case, there are interactions between the different instances of the dataset.

The most popular framework that includes this paradigm was Apache Hadoop [29], written in Java. However, its limitations, such as the necessity of writing to disk every step in the workflow, the impossibility to implement iterative behaviors efficiently, or the necessity to hand code every operation, among others, have led to the emergence of new frameworks, like Apache Spark [12] or Apache Flink [30]. Spark proposes a framework that includes in-memory processing, increasing the processing speed with several orders of magnitude. In addition, Spark introduces the new Resilient Distributed Datasets (RDD) data structure [31] and lazy evaluation. Every transformation and action applied over the RDD is recorded in the RDD lineage. This lineage is a register of each operation applied to the RDD. It allows the RDD to be recovered in any of its previous states, giving a high fault tolerance to the system.

Although Spark provides the framework to process enormous quantities of data and the time series processing is evolving towards processing ever-increasing amounts of data, we cannot find enough tools for processing time series in Spark with official support yet. If we analyze MLlib, we cannot find specific time series algorithms for classification, clustering, or forecasting tasks. In spark packages [14], we only can find two time series proposals. The first one is the spark-ts[2] package, which provides statistical modeling for time series in a distributed way. However, it is oriented to forecasting tasks, and it has been discontinued for a long time. The second one is the Distributed FastShapelet Transform referred to in the previous section. MLlib includes tools for handling data streaming, but only a streaming linear regression model is available.

3 Scalable and Distributed Time Series Transformation Proposal

In this work, we propose, through a MapReduce framework, a scalable and distributed transformation for univariate and multivariate time series (SCMFTS) based on well-known time series features for Big Data environments. The proposed transformation provides a traditional vector-based representation for time series data. SCMFTS allows us to apply algorithms that are not time series specific to time series problems. Table A1, in Appendix, contains the selected set of features used in this work.

In sequential and supervised scenarios, the above approach has proven to obtain competitive results concerning the main algorithms of the state-of-the-art [27, 28]. In addition, SCMFTS is able to process MTS with multiple frequencies and lengths, allowing practitioners to add new features easily.

Our proposal is based on the extraction of the features of each time series. More formally, it can be stated as follows. Given a time series dataset represented in the temporal domain, a new, vectorial representation of the dataset is obtained as follows:

- Unidimensional case. For each time series in the dataset, all the features are computed. The time series is represented by the vector composed of the values for the computer features. The order of the feature values is the same for all the time series in the dataset.
- Multidimensional case. For each time series in the dataset, for each variable in the multivariate time series, all the features are computed. The individual time series is represented by the vector composed of the values for the computer features. The order of the feature values is the same for all the time series in the dataset. The multivariate time series is represented by a vector formed as the concatenation of the corresponding vectors to each of the individual time series composing the multivariate time series.

This new vector-based representation of time series opens a huge landscape of applicable methods for time series. But it is also rather robust allowing for easy usage of complex methods.

While these transformations follow trivially from our proposal for time series representations in previous papers [27, 28], in the current work, we dive deep into its effective application in a Big Data scenario. To effectively study this, we have designed and developed an actual implementation of the representation conversion in a software package able to face real-world problems. This particular implementation is what we term SCMFTS. Its most relevant design issues are detailed in the rest of this section. Its performance and scalability are further analyzed in the remainder of the paper.

Our implementation unifies the distributed computation provided by Spark with the powerful statistical tools available in R to obtain the desired transformation in Big Data environments. We have developed SCMFTS in Scala/Spark, and the communications between Spark and R have been done through rscala [32].

To process the time series correctly, a number of considerations must be made:

- Each multivariate time series has a unique key that identifies it (tsKey).
- Each variable of an MTS has a unique key (varKey). The combination of a time series key with a variable key is unique.
- The input data must have the following format: (tsKey, varKey, tsClass [optional], tsData_1, tsData_2, … , tsData_n).
- Due to the possible differences between the multiple variables that compose an MTS, we chose to process each variable individually. For example, one variable could have hundreds of data points, but another variable could have thousands of data points. Because the communication between Spark and R must be done through a simple data structure like Array[Array[PrimitiveDataType]], including both variables in the same RDD forces us to fill the shortest time series with 0.0 wasting a vast amount of memory resources.

Figure 1 shows the workflow of SCMFTS for the multivariate case. We have included the class column to illustrate a typical supervised problem. In the case of univariate time series, the process is the same, but we do not need the filter and joins steps. First, the framework reads the data in the correct format or processes it until we obtain this format (step 1). Due to this, we can unequivocally identify each time series and variable. Second, we generate an RDD for each problem variable, filtering the input data by the varKey column (step 2). Next, we process each RDD/Variable in a distributed way, generating a new RDD for each variable with the extracted features (step 3), which are then joined

[2] https://spark-packages.org/package/sryza/spark-timeseries.

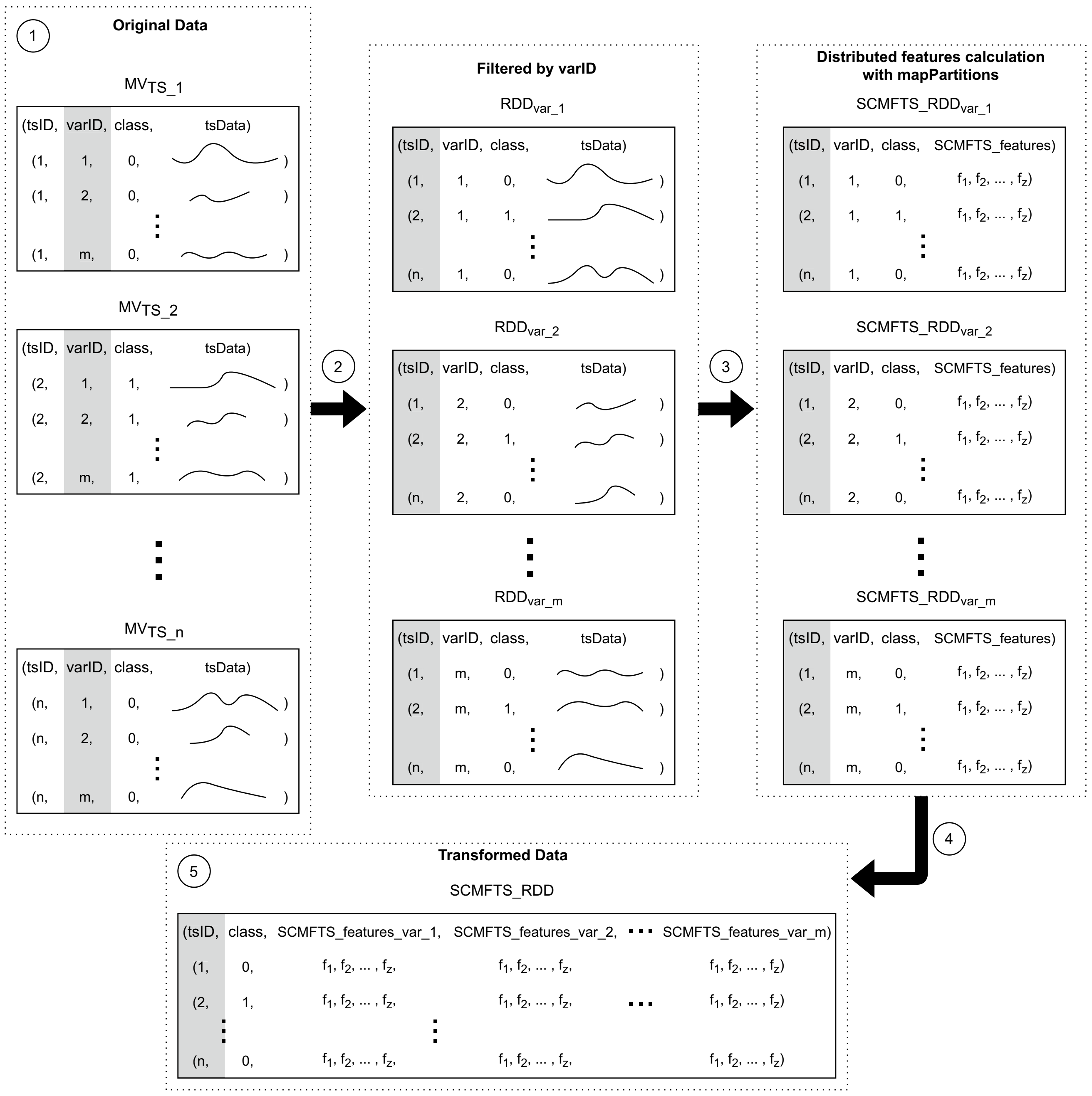
1
Original Data
MVTS_1
(tsID, varID, class, tsData)
MVTS_2
MVTS_n
2
Filtered by varID
RDDvar_1
RDDvar_2
RDDvar_m
3
Distributed features calculation with mapPartitions
SCMFTS_RDDvar_1
SCMFTS_RDDvar_2
SCMFTS_RDDvar_m
(tsID, varID, class, SCMFTS_features)
f1, f2, ... , fz
4
5
Transformed Data
SCMFTS_RDD
(tsID, class, SCMFTS_features_var_1, SCMFTS_features_var_2, ... SCMFTS_features_var_m)

Fig. 1 Workflow example of SCMFTS. The shaded column in each case represents the key value of the typical MapReduce paradigm schema (key, value). The remaining unshaded columns belong to the value field

iteratively (step 4), obtaining a final RDD/dataset where each instance contains the ordered extracted features from each variable (step 5).

In Algorithm 1, we show the pseudocode of SCMFTS, utilizing the operations in Scala/Spark and R. In line 1, we initialize a list that will contain the computed RDDs with the time series features. In lines 2 to 5, our proposal obtains the input data of each variable that composes the MTS processed. We chose a mapPartition transformation because the initialization of the R environment is a time-consuming process. Using this transformation, we only initialize an R environment for each data partition, processing every time series in that partition in the same R session. In line 6, we initialize the output RDD that will contain, in order, the RDDs with the calculated features for each variable that composes the input MTS. In this first step, we include the feature-RDD of the first variable. In lines 7 to 9, we join, iteratively and in order, the output RDD with each feature-RDD of the rest of the variables. We use the tsID as the unique key for the join, and we remove the varID and class columns because we included them in the first RDD for this output. Finally, in line 10, we return the final RDD that contains the features calculated for every variable that composes each MTS. The output RDD can be used as input of any traditional vector-based machine learning algorithm available in Spark. Naturally, it can also be exported to a CSV-like format so that it can be published or further processed on a different platform.

Algorithm 1 SCMFTS procedure

```
Input:
    inputData: list of input RDDs with (tsID, varID, class, tsData)
    structure

Output:
    outputData: output RDD that contains the time series features
    calculated for each input time series with (tsID, class,
    scmftsFeaturesVars) structure

1: scmftsVars ← [ ]
2: for each variable in inputData do
3:     varData ← inputData.filter(variable==varID)
4:     scmftsVars[variable] ← varData.mapPartitions{rscala::scmfts(varDataPartition)}
5: end for
6: outputRDD ← scmftsVars[first]
7: for each scmftsRDD in scmftsVars[-first] do
8:     outputRDD ← outputRDD.join(scmftsRDD[-varID,-class], by=tsID)
9: end for
10: return (outputRDD)
```

Sometimes, the time series features used are not present in the processed time series. For example, a time series without a seasonal pattern or too short to calculate an autocorrelation coefficient cannot offer values for these features. To face this problem, we have included an imputation process after the application of Algorithm 1, explained in Algorithm 2. This procedure grants our proposal a robust behavior to a wide variety of scenarios such as time series that are too short, non-seasonal, trendless, among others.

Table 1 MTS characteristics of the WESAD dataset

Variable	Time series length	Frequency (Hz)
c_ACCx	3500	700
c_ACCy	3500	700
c_ACCz	3500	700
c_ECG	42,000	700
c_EDA	42,000	700
c_EMG	42,000	700
c_RESP	42,000	700
c_TEMP	42,000	700
w_ACCx	160	32
w_ACCy	160	32
w_ACCz	160	32
w_BVP	3840	64
w_EDA	240	4
w_TEMP	240	4

Algorithm 2 Imputation procedure

```
Input:
    SCMFTS_RDD: input RDD that contains the time series
    features calculated for each input time series (tsID, class,
    scmftsFeaturesVars) structure
Output:
    imputedSCMFTSDataset: output RDD with (tsID, class,
    scmftsFeaturesVars) structure without the non-desirable values

1: for each column in SCMFTS_RDD[-tsID, -class] do
2:     for each value in SCMFTS_RDD[ , column.index] do
3:         if is.infinite(value) then
4:             if value $\geq$ 0 then
5:                 value $\leftarrow$ max(SCMFTS_RDD[ , column.index], ignore.inf)
6:             else
7:                 value $\leftarrow$ min(SCMFTS_RDD[ , column.index], ignore.inf)
8:             end if
9:         end if
10:    end for
11:    column $\leftarrow$ imputeMean(SCMFTS_RDD[ , column.index])
12: end for
13: imputedSCMFTSDataset $\leftarrow$ SCMFTS_RDD
14: return (imputedSCMFTSDataset)
```

In lines 1–12, we focus our imputation process on the time series features columns. For each value in each column, we identify if this value is infinite or -infinite, replacing this value by the maximum or minimum finite value of the column, respectively (lines 2–10). After this imputation, the rest of the non-desirable values are imputed using the mean of the column, ignoring the non-desirable values in the mean calculation of each column (line 11). Finally, we obtain the final RDD without non-desirable values (lines 13–14).

In summary, the proposal computes multiple features out of MTS in a robust and scalable way thanks to its MapReduce workflow, whose design, combined with an imputation strategy, enables the tackling of MTS with different lengths and frequencies for each variable with no memory nor runtime overhead, resulting in a fixed length vector for each MTS. The combination of R, Scala, and Spark ensures the efficiency, failure tolerance, and extensibility of the software.

Table 2 Setup for four class problem

Class	Variable 1	Variable 2	Variable 3
0	ARIMA(0, 1, 0)	ARIMA(1, 1, 1): AR(0.5), MA(0.5)	sin(var1 + var2)
1	ARIMA(0, 1, 0)	ARIMA(1, 1, 1): AR(0.25), MA(0.5)	cos(var1 + var2)
2	ARIMA(0, 1, 0)	ARIMA(2, 1, 2): AR(0.2, 0.1), MA(0.1, 0.1)	sin(var1) + cos(var2)
3	ARIMA(0, 1, 0)	ARIMA(2, 1, 2): AR(0.5, 0.25), MA(0.1, 0.1)	cos(var1) + sin(var2)

4 Empirical Study

In this section, we conduct an empirical evaluation of the performance and scalability of SCMFTS. Section 4.1 details the empirical evaluation design. In Sect. 4.2, we show the results obtained by SCMFTS and the selected proposals for comparison.

Table 3 Hyperparameters for used models

Model	Setup
DT	MaxDepth = 10, MinInstancesPerNode = 20, Seed = 1
RF	MaxDepth = 10, MinInstancesPerNode = 20, numTrees = 100, maxBins = 32, Seed = 1
KNN	K = 9, Euclidean distance, Normalized data with mean 0 and standard deviation 1

4.1 Experimental Design

With the aim to evaluate the scalability and performance of SCMFTS, we thoroughly designed and rigorously conducted an empirical analysis. Section 4.1.1 includes the motivation of the selected dataset along with their description. Section 4.1.2 specifies the measures used to evaluate our proposal and the methodology performed. In Sect. 4.1.3, we describe the models to which SCMFTS is compared. Finally, Sect. 4.1.4 includes the hardware and software used for the experiments.

4.1.1 Datasets

To evaluate the performance of SCMFTS in a real world, we have selected the multivariate time series dataset with the highest number of instances in the UCI Repository [33], the Wearable Stress and Affect Detection (WESAD) dataset [34]. In this dataset, we try to identify the patient's state through 14 different sensors that measure biological parameters of the subject, such as blood volume pulse or respiration, among others, and its movement through acceleration sensors in the x, y, and z axes. We have information from 15 different patients. WESAD dataset is used to address two different problems:

- The first problem tries to differentiate three states: baseline vs. stress vs. amusement.
- The second problem differentiates stress vs. non-stress states, joining baseline and amusement states under the same label non-stress.

For data extraction, we followed the segmentation process with a sliding window explained by the original authors: window shift of 0.25 s, a window size of 5 s for the ACC signals, and 60 s for the physiological signals. For subject 14, we also removed the first 136 time series because they contain multiple missing values. In this way, we have generated 132,477 MTS composed of 14 variables each. Table 1 shows the names of each variable and its most representative characteristics.

To better evaluate the scalability, a larger dataset is necessary. Since there is no one publicly available, we have created a synthetic dataset composed of MTS with three variables, with 100 data points per variable. We have generated two different sets of time series:

- The first set of ten datasets containing from 100,000 to 1,000,000 MTS with increments of 100,000 MTS between each dataset.
- A second set of 10 datasets contained from 1,000,000 to 10,000,000 MTS with increments of 1,000,000 MTS between each dataset

The problem generated has four classes obtained from combining random walk processes with AutoRegressive Integrated Moving Average (ARIMA) models in different ways. In Table 2, we show the setup for the variables of these four classes. For variable one, we use an ARIMA(0, 1, 0) model to simulate a random walk process. Variable two is composed of different ARIMA models, and variable three contains combinations of variables one and two through multiple functions.

4.1.2 Measures and Methodology

To evaluate the performance of our proposal, we compare SCMFTS in two and three class sub-problems from the WESAD dataset against the original work using all the

Table 4 Accuracy and F1-score results for WESAD dataset

	Three classes		Two classes	
	Accuracy	F1-score	Accuracy	F1-score
WESAD proposal				
DT	63.56	58.05	83.60	80.83
RF	74.97	64.08	87.74	85.71
AB	79.57	68.85	87.00	83.88
LDA	75.80	71.56	92.28	90.74
KNN ($k = 9$)	56.14	48.70	74.20	69.14
SCMFTS				
DT	64.08	62.69	85.38	84.71
RF	**81.62**	**77.16**	**92.67**	**91.96**
KNN ($k = 9$)	77.78	75.79	89.89	89.90

The best results in each case are in bold

Table 5 Run times for WESAD dataset

Variable	Time SCMFTS (s)	Time series length	Frequency (Hz)
c_ACCx	990.51	3500	700
c_ACCy	1008.76	3500	700
c_ACCz	994.42	3500	700
c_ECG	23,007.39	42,000	700
c_EDA	22,727.54	42,000	700
c_EMG	22,263.16	42,000	700
c_RESP	23,639.61	42,000	700
c_TEMP	22,396.20	42,000	700
w_ACCx	151.24	160	32
w_ACCy	150.72	160	32
w_ACCz	145.75	160	32
w_BVP	813.51	3840	64
w_EDA	242.28	240	4
w_TEMP	244.43	240	4
All variable join	183.93		
Total time	118,959.47		

available variables, using as much data as possible due to the Big Data approach of our proposal. Although a number of papers with algorithmic proposals working on varied versions of the dataset have been published [35–39], they do not follow the original segmentation process nor the leave-one-subject-out (LOSO) cross-validation (CV) approach. Due to this, their results are not comparable with those reported in the original work. We have opted to follow the specifications provided in [34], using the same measures and the same validation criteria. To compare the different models, we use the accuracy and F1-score [40]. The accuracy is obtained by dividing the number of correctly classified instances by the total number of instances in the test set. The F1-score is defined as the harmonic mean of model precision and recall, and it is obtained by combining model precision and recall. F1-score represents a measure of thoroughness. We have evaluated all models using the LOSO CV procedure.

The scalability of SCMFTS is evaluated through two different approaches. First, we measure runtimes by increasing the number of processed time series. In this case, we seek to obtain a linear relationship between the runtimes and the number of processed time series. Second, we will measure runtimes by varying the number of workers in the cluster and the number of cores per worker independently. In both cases, we will compare the evolution of runtimes with the ideal and unachievable case of scalability expressed by Amdahl's law [41].

4.1.3 Models

Since the main target of our proposal is Big Data scenarios, we have focused our experimentation on all the available variables to process as much data as possible. The study in [34] used the following models: Decision Tree (DT), Random Forest (RF), AdaBoost (AB), Linear Discriminant Analysis (LDA),[3] and K-Nearest Neighbors (KNN) with $k = 9$. For the (DT, RF, AB) models, the referred to work set the minimum number of samples required to split a node to 20, and they set the number of base estimators to 100 for (RF and AB). To ensure a fair comparison and reproducibility, we have chosen the models available in the MLlib of Apache Spark (DT, RF, KNN). In addition, we have used the same hyperparameters specified in [34]. Some of these models have additional parameters that were not specified in the mentioned work. All hyperparameters used in our experimentation are listed in Table 3.

To maintain the number of data partitions over the entire process, we have set this number in all experiments performed to three times the number of total available cores, which is the typical recommendation for Spark.

Considering the LOSO approach used in [34], we have applied the imputation process explained in Algorithm 2 to each subject data independently. Due to the data source of the WESAD dataset, we have an additional column that contains the identification number of the subject that provides each time series. Using this information, we can filter the data of each subject and process it independently. Obviously, this value is not used within the training datasets.

[3] The LDA method available in Spark corresponds to Latent Dirichlet Allocation and is not related to the Linear Discriminant Analysis method used in WESAD work.

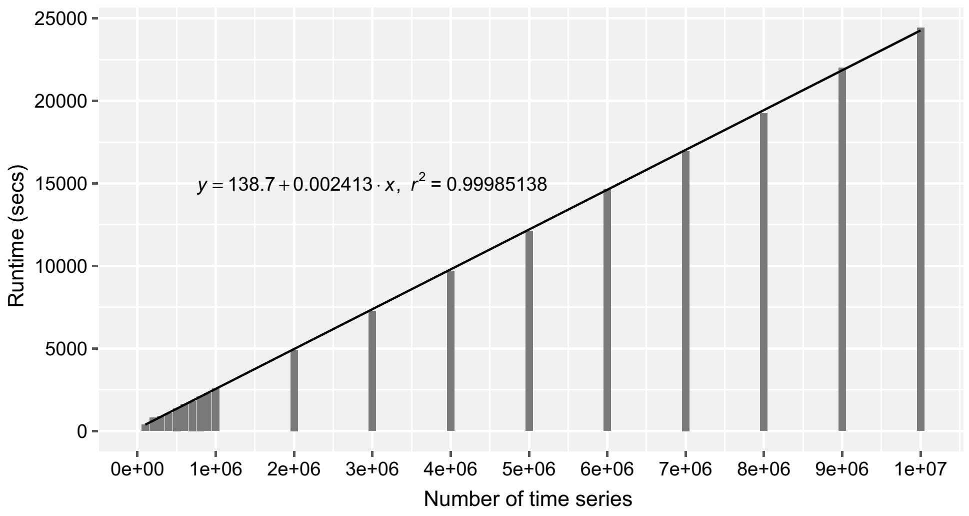

Fig. 2 Scalability experiment: Runtime vs Number of time series

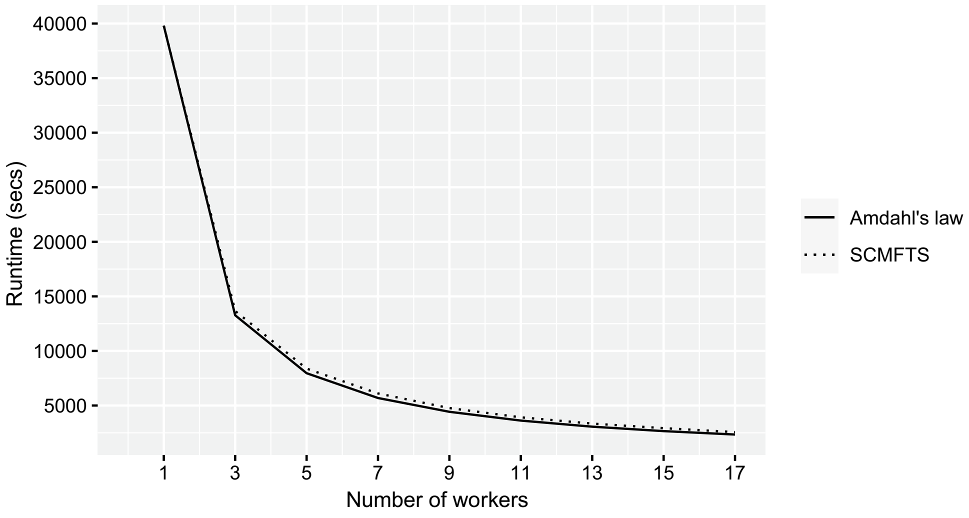

Fig. 3 Scalability experimentation: Runtime vs Number of workers

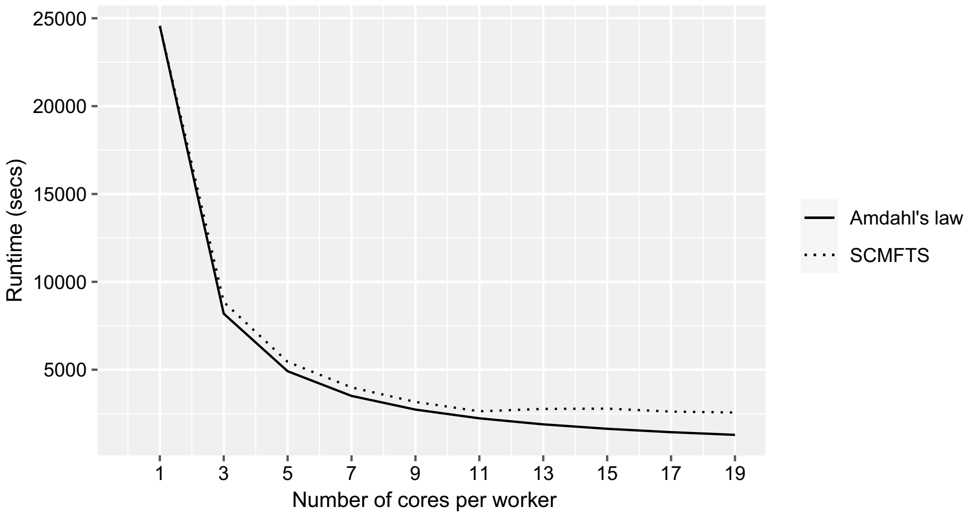

Fig. 4 Scalability experiment: Runtime vs Number of cores per worker

4.1.4 Hardware and Software

We have performed the experimentation in a Big Data cluster composed of one driver/master node and 17 slave/computing nodes. The computing nodes hold the following characteristics: 2 × Intel(R) Xeon(R) CPU E5-2620 processors, 6 cores per processor with HyperThreading, 2.00 GHz, 64 GB RAM, 2 TB HDD (1 TB HDFS). We have used the following software configuration: Ubuntu 18.04.5 LTS, Apache Hadoop 2.7, Apache Spark 3.0.1, 19 threads/node, 833 RAM GB (48 GB/node).

The source code of SCMFTS is publicly available [1].

4.2 Results

In this section, we evaluate the two main aspects of SCMFTS: performance and scalability. Section 4.2.1 includes the performance results of SCMFTS on the WESAD dataset. In Sect. 4.2.2, we analyze the scalability of SCMFTS on the synthetic dataset.

4.2.1 Performance Results on WESAD

To assess the performance of the SCMFTS proposal, we have applied it to solve the two- and three-classes problems of WESAD and compared it to the ML procedures analyzed in the original work. The empirical results, expressed in terms of accuracy and F1-score, are displayed in Table 4.

The results show that SCMFTS provides consistently better results than the WESAD work, with the available Spark machine learning models, in both cases. In addition, we can see in the WESAD work [34], Table 3, that the best results for the three classes problem are provided by the AB model but only using the chest physiological modalities (c_ECG, c_EDA c_EMG, c_RESP, and c_TEMP): 80.34 of accuracy and 72.51 of F1-score, and still SCMFTS+RF provides better results. In the two classes problem, the WESAD work provides the LDA model with the chest physiological modalities as the best results with 93.12 of accuracy and 91.47 of F1-score, Table 4 in WESAD work [34], outperforming in accuracy our best model but not in F1-score. In this case, the WESAD work uses fewer variables and a model that is not available in Spark, so we cannot make a direct comparison. It is important to note that the LDA model provides the best results in 20 out of 32 cases in the WESAD work, so this particular model clearly offers better results than the others. In addition, the AB model in the three-class problem and the LDA model in the two-class problem provide results significantly better than DT, RF, and KNN models in the WESAD work.

There are relevant differences between the multiple variables that compose the MTS of this problem. Due to the segmentation applied to the original time series proposed in the WESAD work, we have variables that contain from 160 to 42,000 data points in the same problem. These differences between variables generate significant variations in runtime for feature calculation between the different variables, as we can see in Table 5. As usual, a high number of data points generates high runtimes, but if we compare runtimes for variables c_ACCx, c_ACCy, or c_ACCz with w_BVP, this does not happen. It is so because of the differences in the frequency value of these variables, which is included in the time series features calculation affecting the runtime. These phenomena are not related to the Spark implementation performed, but it depends on the structure of the input time series.

4.2.2 Scalability Results on Synthetic Dataset

In this section, we analyze the scalability performance of SCMFTS. Particularly, we focus on the three most frequently considered dimensions: number of instances to process, number of machines available, and number of cores per machine. In Fig. 2, we show the relationship between the runtime of SCMFTS and the number of MTS to process. For this experimentation, we have used the 17 available workers and 19 cores/threads per worker. Figure 2 allows us to graphically identify a linear relationship between the runtime and the number of time series to be processed through SCMFTS. This feature is a mandatory requirement for the scalability considerations in Big Data environments.

In Fig. 3, we can compare the runtime of SCMFTS with different numbers of workers. In this case, we perform the experimentation with a dataset composed of 1,000,000 MTS. As usual, an increase in the number of workers entails a reduction in the runtime. For example, if we compare the one and three workers cases, we can appreciate that the reduction obtained is close to three times. This behavior is present in the rest of the comparison in the Fig. 3. In this kind of process, the ideal case is to obtain a time reduction equal to the number of the added workers as Amdahl's law [41] specifies, but it is a theoretical limit and in general impossible to achieve in practice. SCMFTS is near to the optimal case. Furthermore, we have to note the existence of additional procedures related to adding workers to the cluster, like extra workers communications, data transmission, among others, that do not let us reach the performance of the ideal case.

Each worker has 12 real/physical cores, 24 with hyper-threading technology. To evaluate the core's performance of SCMFTS, we process 1,000,000 MTS using the 17 workers available in our cluster, but we vary the number of used cores in each worker. Figure 4 shows the relationship between the runtime of SCMFTS and the number of cores per worker. We can appreciate that the reduction in runtime has similar behavior to the one observed previously, Fig. 3, but with a greater gap regarding the optimal case. If we compare the case of one core per worker with the cases of three or five cores per worker, among others, we can see that the amount of runtime reduction is directly related to the number of cores used: one core case has a runtime close to 25,000 s, three cores case has a runtime close to 7500 s, five cores case has a runtime slightly higher to 5000 s, etc. But in this case, we can appreciate that the runtime stops decreasing with the number of cores with 11 cores per worker. This issue is due to the number of physical cores by machine, which is 12. Although hyper-threading technology allows us to increase the efficiency of a core to provide an additional virtual core, we cannot reach the maximum desired performance in computationally intensive tasks.

Based on the results obtained in this section, in which our proposal shows behaviors close to the ideal, we can conclude that SCMFTS has a high scalability performance.

5 Conclusions

In this paper, we have presented a scalable and distributed method, named SCMFTS, for transforming univariate and multivariate time series into a vector of well-known features. This method lets us apply the traditional vector-based algorithms already available in Big Data to time series problems, allowing us to address problems that would otherwise be impossible. SCMFTS extends considerably the limited number of algorithms available to process time series in Big Data environments. Our proposal is able to process MTS with multiple frequencies and lengths and allows practitioners to add new features easily.

SCMFTS has improved the results obtained, under the same conditions, by the state-of-the-art on the biggest multivariate time series dataset available in the UCI Machine Learning Repository, wearable stress and affect detection (WESAD). The results obtained by SCMFTS on a general problem improved those obtained by the WESAD work solution, applying the proposed transformation without additional considerations and allowing it to be a tool of interest to a large number of researchers in multiple areas. In addition, SCMFTS has shown a totally scalable behavior through exhaustive experimentation, with a linearly scalable relationship in runtime concerning the number of time-series processed.

Our proposal has been implemented in the Scala programming language for the Apache Spark framework, and the code is publicly available. The implementation of SCMFTS has followed the principles of FAIR [42] (Findability, Accessibility, Interoperability, and Reuse) and Open Science.

This proposal opens promising research lines in this topic, as exploring the semi-supervised approach based on the proposed set of features. In Big Data environments, the volume of processed data is high, and the labeling is limited. In those environments, the semi-supervised approach offers very interesting solutions. Another research line is the study of the improvement in the expressivity and performance of the selected set of features in Big Data environments.

Appendix: Time Series Complexity Measures and Features Selected

See Table A1.

Table A1 Time series complexity measures and features selected

Fea.	Name	Description
f_1	lempel_ziv	LempelZiv (LZA)
f_2	approximation_entropy	Approximation Entropy
f_3	sample_entropy	Sample Entropy (DK Lake in Matlab)
f_4	permutation_entropy	Permutation Entropy (tsExpKit)
f_5	shannon_entropy_CS	Chao-Shen Entropy Estimator
f_6	shannon_entropy_SG	Schurmann–Grassberger Entropy Estimator
f_7	spectral_entropy	Spectral Entropy
f_8	nforbidden	Number of forbidden patterns
f_9	kurtosis	Kurtosis, the "tailedness" of the probability distribution
f_{10}	skewness	Skewness, asymmetry of the probability distribution
f_{11}	x_acf1	First autocorrelation coefficient
f_{12}	x_acf10	Sum of squares of the first 10 autocorrelation coefficients
f_{13}	diff1_acf1	Differenced series first autocorrelation coefficients
f_{14}	seas_acf1	First autocorrelation coefficient of the seasonal component
f_{15}	diff1_acf10	Differenced series sum of squares of the first 10 autocorrelation coefficients
f_{16}	diff2_acf1	Twice differenced series first autocorrelation coefficients
f_{17}	diff2_acf10	Twice differenced series sum of squares of the first 10 autocorrelation coefficients
f_{18}	max_kl_shift	Maximum shift in Kullback–Leibler divergence between two consecutive windows
f_{19}	time_kl_shift	Instant of time in which the Maximum shift in Kullback–Leibler divergence between two consecutive windows is located
f_{20}	outlierinclude _mdrmd	Calculates the median of the medians of the values, while adding more outliers
f_{21}	max_level_shift	Maximum mean shift between two consecutive windows
f_{22}	time_level_shift	Instant of time in which the maximum mean shift between two consecutive windows is located
f_{23}	ac_9	Autocorrelation at lag 9
f_{24}	crossing_points	The number of times a time series crosses the median line
f_{25}	max_var_shift	Maximum variance shift between two consecutive windows
f_{26}	time_var_shift	Instant of time in which the maximum variance shift between two consecutive windows is located
f_{27}	nonlinearity	Modified statistic from Teräsvirta's test
f_{28}	embed2_incircle	Proportion of points inside a given circular boundary in a 2-d embedding space
f_{29}	spreadrandomlocal _meantaul	Mean of the first zero-crossings of the autocorrelation function in each segment of the 100 time-series segments of length l selected at random from the original time series
f_{30}	flat_spots	Maximum run length within any single interval obtained from the ten equal-sized intervals of the sample space of a time series
f_{31}	x_pacf5	Sum of squares of the first 5 partial autocorrelation coefficients
f_{32}	seas_pacf	Sum of squares of the first 5 partial autocorrelation of the seasonal component
f_{33}	diff1x_pacf5	Differenced series sum of squares of the first 5 partial autocorrelation coefficients
f_{34}	diff2x_pacf5	Twice differenced series sum of squares of the first 5 partial autocorrelation coefficients
f_{35}	firstmin_ac	Time of first minimum in the autocorrelation function
f_{36}	std1st_der	Standard deviation of the first derivative of the time series
f_{37}	stability	Stability variance of the means
f_{38}	firstzero_ac	First zero crossing of the autocorrelation function
f_{39}	trev_num	The numerator of the trev function, a normalized nonlinear autocorrelation, with the time lag set to 1
f_{40}	alpha	Smoothing parameter for the level-alpha of Holt's linear trend method
f_{41}	beta	Smoothing parameter for the trend-beta of Holt's linear trend method
f_{42}	nperiods	Number of seasonal periods (1 for no seasonal data)
f_{43}	seasonal_period	Seasonal periods (1 for no seasonal data)
f_{44}	trend	Strength of trend
f_{45}	spike	Spikiness variance of the leave-one-out variances of the remainder component
f_{46}	linearity	Linearity calculated based on the coefficients of an orthogonal quadratic regression
f_{47}	curvature	Curvature calculated based on the coefficients of an orthogonal quadratic regression

Table A1 (continued)

Fea.	Name	Description
f_{48}	e_acf1	First autocorrelation coefficient of the remainder component
f_{49}	seasonal_strength	Strength of seasonal
f_{50}	peak	Strength of peaks
f_{51}	trough	Strength of trough
f_{52}	e_acf10	Sum of the first then squared autocorrelation coefficients
f_{53}	walker_propcross	Fraction of time series length that walker crosses time series
f_{54}	hurst	Long-memory coefficient
f_{55}	unitroot_kpss	Statistic for the KPSS unit root test with linear trend and lag one
f_{56}	histogram_mode	Calculates the mode of the data vector using histograms with 10 bins (It is possible to select a different number of bins)
f_{57}	unitroot_pp	Statistic for the PP unit root test with constant trend and lag one
f_{58}	localsimple_taures	First zero crossing of the autocorrelation function of the residuals from a predictor that uses the past train-Length values of the time series to predict its next value
f_{59}	lumpiness	Lumpiness variance of the variance
f_{60}	motiftwo_entro3	Entropy of words in the binary alphabet of length 3. The binary alphabet is obtained as follows: Time-series values above its mean are given 1, and those below the mean are 0

Acknowledgements This work has been partially carried out while Francisco J. Baldán was on a research stay at the University of Ghent. Francisco J. Baldán is grateful for the valuable feedback and knowledge received from the members of the Data Mining and Modeling for Biomedicine (DAMBI) group.

Author Contributions Paper writing and experiment implementation: FJB; methodology: FJB and DP; revising and reviewing: DP, JMB, and YS; supervision: DP, JMB, and YS; funding acquisition: JMB and YS.

Funding This research has been partially funded by the following grants: TIN2016-81113-R from the Spanish Ministry of Economy and Competitiveness, P12-TIC-2985 and P18-TP-5168 from Andalusian Regional Government, Spain, and EU Commission with FEDER funds. Francisco J. Baldán holds the FPI grant BES-2017-080137 from the Spanish Ministry of Economy and Competitiveness. D. Peralta is a Postdoctoral Fellow of the Research Foundation of Flanders (170303/12X1619N). Y. Saeys is an ISAC Marylou Ingram Scholar.

Availability of Data and Materials The code of our proposal and synthetic time series generation is publicly available (https://github.com/fjbaldan/SCMFTS/). The Wearable Stress and Affect Detection dataset is publicly available already (https://ubicomp.eti.uni-siegen.de/home/datasets/icmi18/).

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

Ethics Approval Not applicable.

Consent to Participate Not applicable.

Consent for Publication The authors consent to this work for publication.

References

1. Baldán, F.J., Peralta, D., Saeys, Y., Benítez, J.M.: Scalable complexity measures and features for times series classification package repository (2021). https://github.com/fjbaldan/SCMFTS/
2. Kobusińska, A., Leung, C., Hsu, C.-H., Raghavendra, S., Chang, V.: Emerging trends, issues and challenges in Internet of Things, Big Data and cloud computing. Future Gener. Comput. Syst. **87**, 416–419 (2018)
3. Lee, S.W., Kim, H.Y.: Stock market forecasting with super-high dimensional time-series data using ConvLSTM, trend sampling, and specialized data augmentation. Expert Syst. Appl. **161**, 113704 (2020)
4. Kim, T.-Y., Cho, S.-B.: Predicting the household power consumption using CNN-LSTM hybrid networks. In: Intelligent Data Engineering and Automated Learning—IDEAL 2018, pp. 481–490 (2018)
5. Aarthy, S., Iqbal, J.M.: Time series real time Naive Bayes electrocardiogram signal classification for efficient disease prediction using fuzzy rules. J. Ambient Intell. Humaniz. Comput. **12**(5), 5257–5267 (2021)
6. Nguyen, T., Nguyen, T., Nguyen, B.M., Nguyen, G.: Efficient time-series forecasting using neural network and opposition-based coral reefs optimization. Int. J. Comput. Intell. Syst. **12**(2), 1144–1161 (2019)
7. Wu, B., Duan, T.: A performance comparison of neural networks in forecasting stock price trend. Int. J. Comput. Intell. Syst. **10**(1), 336–346 (2017)

8. Viegas, J.L., Cepeda, N.M., Vieira, S.M.: Electricity fraud detection using committee semi-supervised learning. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–6 (2018)
9. Haddi, Z., Ananou, B., Trardi, Y., Pons, J.-F., Delliaux, S., Deharo, J.-C., Ouladsine, M.: Advanced machine learning coupled with heart-inter-beat derivatives for cardiac arrhythmia detection. In: 2020 American Control Conference (ACC), pp. 5433–5438 (2020)
10. Handhika, T., Murni, Lestari, D.P., Sari, I.: Multivariate time series classification analysis: state-of-the-art and future challenges. In: IOP Conference Series: Materials Science and Engineering, vol. 536, p. 012003 (2019)
11. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation, vol. 6, p. 10 (2004)
12. Hamstra, M., Karau, H., Zaharia, M., Konwinski, A., Wendell, P.: Learning Spark: Lightning-Fast Big Data Analytics. O'Reilly Media, Inc., Sebastopol (2015)
13. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: MLlib: machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
14. Packages, S.: 3rd Party Spark Packages (2019). https://spark-packages.org/
15. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Searching and mining trillions of time series subsequences under dynamic time warping. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 262–270 (2012)
16. Rakthanmanon, T., Keogh, E.: Fast shapelets: a scalable algorithm for discovering time series shapelets. In: Proceedings of the 2013 SIAM International Conference on Data Mining, pp. 668–676 (2013)
17. Laptev, N., Amizadeh, S., Flint, I.: Generic and scalable framework for automated time-series anomaly detection. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1939–1947 (2015)
18. Foreman-Mackey, D., Agol, E., Ambikasaran, S., Angus, R.: Fast and scalable Gaussian process modeling with applications to astronomical time series. Astron. J. **154**(6), 220 (2017)
19. Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., Petitjean, F., Webb, G.I.: Proximity forest: an effective and scalable distance-based classifier for time series. Data Min. Knowl. Discov. **33**(3), 607–635 (2019)
20. Baldán, F.J., Benítez, J.M.: Distributed FastShapelet Transform: a Big Data time series classification algorithm. Inf. Sci. **496**, 451–463 (2019)
21. Lines, J., Davis, L.M., Hills, J., Bagnall, A.: A shapelet transform for time series classification. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 289–297 (2012)
22. Fulcher, B.D., Little, M.A., Jones, N.S.: Highly comparative time-series analysis: the empirical structure of time series and their methods. J. R. Soc. Interface **10**(83), 20130048 (2013)
23. Fulcher, B.D.: Feature-based time-series analysis (2017). arXiv preprint. arXiv:1709.08055
24. Kang, Y., Hyndman, R.J., Li, F., et al.: Efficient generation of time series with diverse and controllable characteristics. Technical report, Monash University, Department of Econometrics and Business Statistics (2018)
25. Lubba, C.H., Sethi, S.S., Knaute, P., Schultz, S.R., Fulcher, B.D., Jones, N.S.: catch22: CAnonical Time-series CHaracteristics. Data Min. Knowl. Discov. **33**(6), 1821–1852 (2019)
26. Peralta, D., Saeys, Y.: Robust unsupervised dimensionality reduction based on feature clustering for single-cell imaging data. Appl. Soft Comput. **93**, 106421 (2020)
27. Baldán, F.J., Benítez, J.M.: Complexity measures and features for times series classification (2020). arXiv preprint arXiv:2002.12036
28. Baldán, F.J., Benítez, J.M.: Multivariate times series classification through an interpretable representation. Inf. Sci. **569**, 596–614 (2021)
29. White, T.: Hadoop: The Definitive Guide. O'Reilly Media, Inc., Sebastopol (2012)
30. Flink, A.: Apache Flink (2019). http://flink.apache.org/
31. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pp. 15–28 (2012)
32. Dahl, D.B.: Integration of R and Scala using rscala. J. Stat. Softw. **92**(1), 1–18 (2020)
33. Dua, D., Graff, C.: UCI Machine Learning Repository (2017). http://archive.ics.uci.edu/ml
34. Schmidt, P., Reiss, A., Duerichen, R., Marberger, C., Van Laerhoven, K.: Introducing wesad, a multimodal dataset for wearable stress and affect detection. In: Proceedings of the 20th ACM International Conference on Multimodal Interaction, pp. 400–408 (2018)
35. Bobade, P., Vani, M.: Stress detection with machine learning and deep learning using multimodal physiological data. In: 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), pp. 51–57 (2020)
36. Indikawati, F.I., Winiarti, S.: Stress detection from multimodal wearable sensor data. In: IOP Conference Series: Materials Science and Engineering, vol. 771, p. 012028 (2020)
37. Lin, J., Pan, S., Lee, C.S., Oviatt, S.: An explainable deep fusion net-work for affect recognition using physiological signals. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 2069–2072 (2019)
38. Saeed, A., Salim, F.D., Ozcelebi, T., Lukkien, J.: Federated self-supervised learning of multisensor representations for embedded intelligence. IEEE Internet Things J. **8**(2), 1030–1040 (2020)
39. Samyoun, S., Sayeed Mondol, A., Stankovic, J.A.: Stress detection via sensor translation. In: 2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 19–26 (2020)
40. Espíndola, R.P., Ebecken, N.F.: On extending f-measure and g-mean metrics to multi-class problems. WIT Trans. Inf. Commun. Technol. **35** (2005)
41. Hill, M.D., Marty, M.R.: Amdahl's law in the multicore era. Computer **41**(7), 33–38 (2008)
42. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L.B., Bourne, P.E., et al.: The fair guiding principles for scientific data management and stewardship. Sci. Data **3**(1), 1–9 (2016)