Distributed FastShapelet Transform: a Big Data time series classification algorithm

Francisco J. Baldán[a,*], José M. Benítez[a]

[a] *Department of Computer Science and Artificial Intelligence, University of Granada, DICITS, iMUDS, DaSCI, 18071 Granada, Spain*

Abstract

The classification of time series is a central problem in a wide range of disciplines. In this field, the state-of-the-art algorithm is COTE (Collective of Transformation-Based Ensembles) which is a combination of classifiers of different domains: time, autocorrelation, power spectrum and shapelets. The weakest point of this approach is its high computational burden which prevents its use in massive data environments. Shapelet Transform is one of the multiple algorithms that compose this ensemble. It has been shown to achieve a good performance over many reference datasets. Nevertheless, its computational complexity is also too high to be used in massive data environments. On the other hand, Big Data has emerged as an approach to manage massive datasets, which also applies to time series. We propose an algorithm for time series classification in a Big Data environment, DFST. It is based on a combination of the FastShapelet and Shapelet Transform ideas and it is the first completely scalable algorithm for time series classification. We have shown that our proposal scales linearly with the number of time series in dataset. In addition, the classification accuracy is equal to or higher than that of comparable sequential algorithms.

Keywords: Time Series, Big Data, Classification, Shapelet.

1. Introduction

Nowadays, we are in the Big Data era. Huge loads of data are created, stored and processed. Their features in many dimensions (volume, velocity, variety, complexity, etc) exceed the computation capabilities of current computers. A new approach to cope with them is necessary.

Advances in computing technologies allow to capture, store and process large amounts of data from varied events and processes: changes in climate, traffic evolution, vital signs, etc. Many of these massive datasets are time series, e.g.

*Corresponding author
Email addresses: `fjbaldan@decsai.ugr.es` (Francisco J. Baldán), `J.M.Benitez@decsai.ugr.es` (José M. Benítez)

the dataset “CCAFS-Climate Data” [7] [20], with up to 6TB of information on temperature, solar radiation, etc, or “Federal Reserve Economic Data - Fred” [16] [21], with up to 20,059 U.S. economic time series.

Time series processing for large datasets is a computationally expensive and complex process [11], specially in the field of time series classification. The leading proposal of the current state-of-the-art is COTE [2] (Collective of Transformation-Based Ensembles). This algorithm is based on the idea that an ensemble made up of time series classifiers from different domains (time, autocorrelation, power spectrum and shapelets) can offer better results than each classifier individually. Through a comprehensive study, a recent work [1] has demonstrated that for a given problem, the simplest algorithms of a domain which adapt to the problem can offer better results than the most complex algorithms of other domains. We can also observe how the different types of time series classification algorithms are described and classified in this study according to the type of processing they perform on the time series.

The shapelets and the Shapelet Transform (ST) algorithm [13] have gained prominence in recent years due to the good results obtained in time series classification problems. Specifically, problems based on the search for independent phase patterns. Shapelet primitive [22] was developed in order to obtain interpretable results in the classification of time series, while achieving the same performance in accuracy than the state-of-the-art algorithms. ST is the data transformation method that is used to convert the raw time series data using the shapelets, allowing the use of any traditional classification algorithm on the new transformed data. Shapelet Transform algorithm obtains the best classification results in the field of shapelets, but it has sacrificed some of the interpretability of the results. In addition, it has the highest computational complexity, $O(n^2m^4)$, of the time series classification algorithms based on shapelets. So far, proposals have been made in two directions: improving accuracy and reducing computational complexity. An example of proposals that reduce the computational complexity of the shapelets search is the FastShapelet (FS) algorithm, which is a heuristic algorithm search and classification of time series, which uses a traditional classification tree and that obtains a computational complexity of $O(nm^2)$. Nevertheless, this approach can not be applied to large time series datasets or Big Data problems due to its complexity. A reasonable improvement can be achieved through a Big Data processing approach.

In this paper, we propose a distributed and scalable time series classification algorithm based on the MapReduce paradigm for Big Data environments named Distributed FastShapelet Transform (DFST). DFST is a hybrid algorithm that uses a re-designed and distributed version of the shapelets search mechanism, proposed by the FS algorithm [19]. DFST includes design elements to render it of a reduced computational complexity enabling the application of the ST in Big Data environments and making it the only completely scalable procedure currently available: with the provision of adequate hardware resources it can process a time series dataset of *any* size. Thus DFST enables the processing of time series datasets that cannot be handled by a sequential approach. The major changes in DSFT with respect to FS are focused on four points: the control of exponential growth in the generation of random projections, the simultaneous selection of several shapelets at each thread, the use of a localized computation of information gain and, the outcome, which allows the use of other classification algorithms, removing the tie to classification trees. As a result of all of this, DFST accuracy results are better than those obtained by FS. The practical incarnation of the algorithm has been developed in Apache Spark and it is available in SparkPackages[1].

The rest of this proposal is structured as follows. Section 2 includes related papers and background on shapelets. In Section 3 we explain our distributed proposal. In Section 4 we show the experimental study carried out to test the effectiveness of our proposal. The conclusions of this paper are presented in Section 5.

2. Related and background work

Our work aims to address problems of classifying time series on large datasets. In Section 2.1 we present the shapelet primitive, which provides interpretable results on time series classification problems, the FS algorithm, that is a heuristic shapelet search algorithm with reduced computational complexity, and the ST, which is a transformation that obtains characteristics from the time series. In Section 2.2 we introduce the MapReduce Model used in Big Data environments, on which we have implemented our proposal.

2.1. Shapelet primitive, FastShapelet algorithm and Shapelet Transform

This section is structured as follows. In Section 2.1.1 we explain the primitive shapelet. In Section 2.1.2 we show the concept and operation of the Fast Shapelet algorithm. In Section 2.1.3 we explain the idea of the Shapelet Transform.

2.1.1. Shapelet primitive

The shapelet is a primitive [22] used in time series classification problems. It is composed by a subsequence of the time series from which it comes and a threshold distance. The shapelets are used to create a classification tree, where each internal node is composed by one shapelet. Each internal node separates the training instances depending on whether or not they contain the internal node's shapelet. The objective is to obtain a classification tree in which the leaf nodes have instances of a single class. If the shapelet is included in a time series, that time series is classified as belonging to the class of the time series from which this shapelet comes from. We compute the distance between the time series and the shapelet to know if a shapelet is included in a time series. If that distance is less than the threshold distance we consider that the time series contains the processed shapelet. We compute the distance between a time series

[1]Distributed FastShapelet Transform (DFST). `https://spark-packages.org/package/fjbaldan/DFST`

and a shapelet as the minimum distance between the shapelet's subsequence and all possible subsequences of equal length of the time series.

2.1.2. FastShapelet algorithm

The FastShapelet algorithm (FS) [19] was proposed to improve the efficiency of the original extraction algorithm. This algorithm has a complexity of $O(nm^2)$, where n is the number of items or time series to process and m the length of the longest time series. The original shapelet discovery algorithm has a complexity of $O(nm^3)$. FS is a heuristic algorithm that faces the discovery of shapelet by applying a change of representation. For this purpose, FS uses Symbolic Aggregate approXimation (SAX) [12] converting the original real values to discrete values with smaller dimensions. All the extracted sequences are transformed into strings of length 16 with 4 discrete levels per value. This mechanism produces multiple SAX words for each time series. The creation of SAX words has the disadvantage that two subsequences with small differences can generate two different SAX words. Random Masking [6] is used to solve this problem. In this way very similar sequences with different SAX words randomly mask some of their values. After multiple iterations they can be identified as similar even if their SAX words are different. After the generation of random projections a frequency count histogram is built for each class. A score is calculated for each SAX word based on its ability to discriminate between classes by processing the frequency count histogram. The best k SAX words are selected and the actual values of their respective shapelets are retrieved. These shapelets are evaluated according to the following parameters and in this order: the gain of information calculated, the separation achieved between instances of different classes and the number of correctly separated training instances. Once the input dataset has been fully processed, the algorithm returns a decision tree. On this tree, each internal node contains the shapelet and the threshold distance that will be used to classify the new input data.

2.1.3. Shapelet Transform

The Shapelet Transform (ST) [13] does not use the extracted shapelets to classify new time series, but as input characteristics for a classifier. This allows using almost any classification algorithm.

At present, there are different ways to extract the shapelets [22] [15] [19] [3] but once extracted, the operation of the algorithm is the same. For a dataset of n time series and m extracted shapelets, the minimum distance between each time series and each shapelet is calculated, obtaining a new dataset with n instances and m characteristics. This new dataset is the training set of any traditional classification algorithm that you decide to apply. This transformation has to be applied to the test set as well.

It has been shown that this approach improves the results obtained by the classification algorithms based on shapelets [1] in most cases, in addition to maintaining the original interpretability of the Shapelets.

2.2. MapReduce Model

The exponential growth of data generated globally is popularizing the use of Big Data technologies. Most of these technologies are based on the MapReduce framework designed by Google in 2003 [8]. This framework allows creating clusters of common machines that can process large amounts of data. The user does not have to worry about tasks like partitioning the input data, handling machine failures or the inter-machine communication, among others. The MapReduce paradigm is based on two phases:

- The Map phase applies a transformation on each key-value pair of the original dataset locally.
- The Reduce phase unifies the results of the Map phase by means of associative operations and returns a result.

Currently, Apache Hadoop [9] is the most popular framework based on the MapReduce paradigm. This framework suffer from some weaknesses:

- Low memory usage, each executed phase must be written to disk.
- Iterative processes are difficult to implement and have a poor performance.

For these reasons, new alternatives such as Apache Spark [10] have been developed. Apache Spark uses in-memory workloads with memory-intensive usage, increasing the speed of computation by several orders of magnitude. It also allows the implementation of iterative processes in a simple and substantially more efficient way than its predecessors.

Apache Spark [24] is an engine for large-scale data processing. It was developed with the aim of facing tasks that focus on applying parallel operations on a input dataset that is constantly reused. One of its main features is the increase in running speed compared to other options. Spark can run the same program than Apache Hadoop up to 100 times faster. Its distributed processing architecture allows to increase the computing capacities of a cluster by adding new computers transparently to the user. Resilient Distributed Datasets (RDDs) [23] are the data structure on which Apache Spark distributed operations are based. These operations are computed over the local data on each partition. There are two types of operations: Transformations and Actions. The Transformations are not executed until an action is performed. These transformations apply a function on each RDD instance and return a new RDD. The Actions execute all transformations applied on an RDD returning a result. This result depends on the action applied. The RDDs are stored in memory. They are immutable and keep a checkpoint of all the transformations applied to them. This checkpoint is known as “lineage” and it allows recovering any partition in case of error.

3. Distributed FastShapelet Transform proposal

In this section, we present our proposal DFST, for scalable time series classification. Our proposal is based on the MapReduce paradigm, which allows it to be applied to massive time series datasets. In Section 3.1 we explain the computational complexity of our proposal.

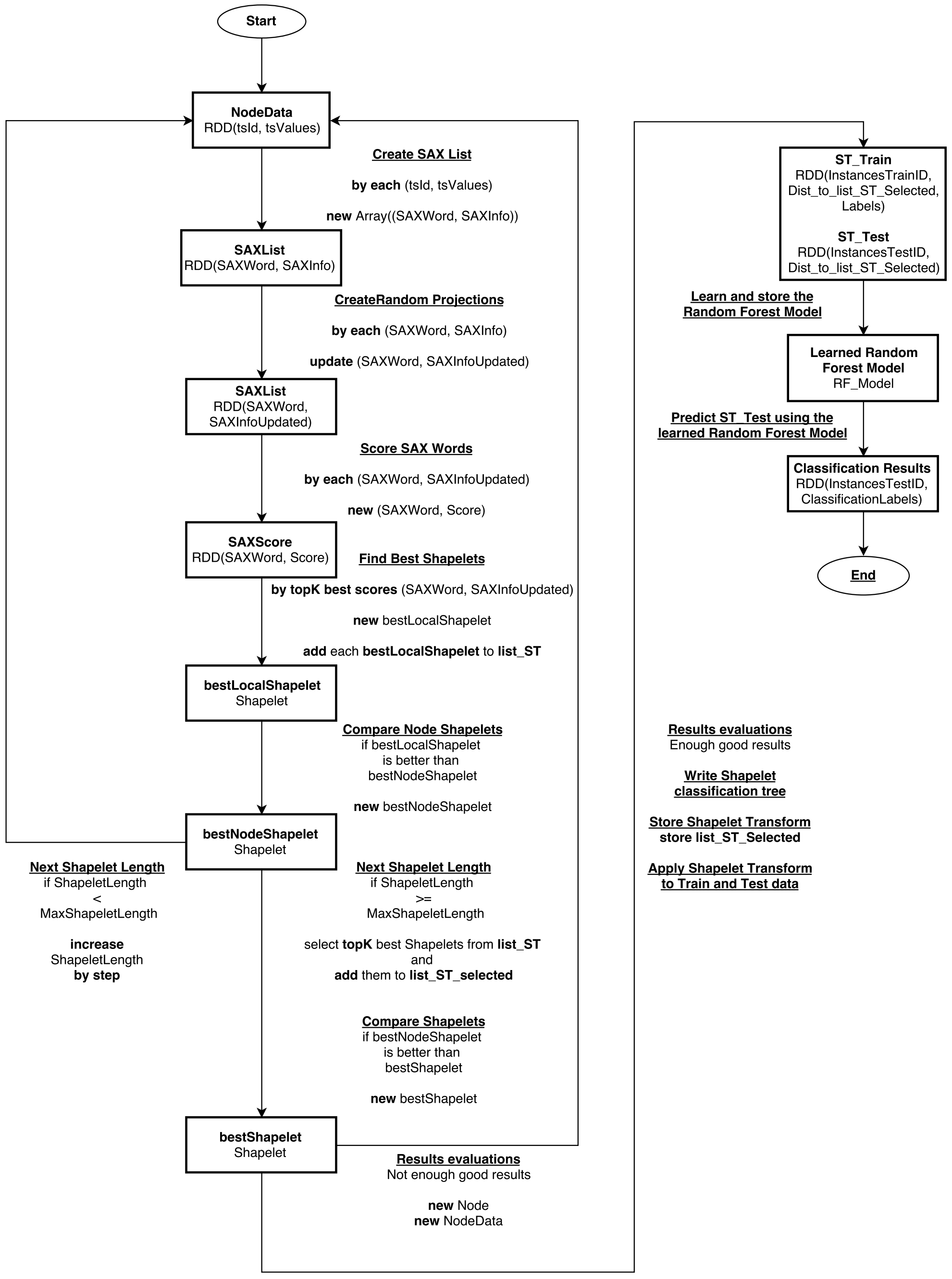
The proposal is expressed in terms of the following Spark primitives:

- *map*: A transformation that applies a function to all elements of the input RDD, returning a new RDD.
- *mapPartitions*: A transformation that applies a function to every partition of the input RDD. This transformation returns a new RDD.
- *reduce*: A transformation that merges the elements of the input RDD using an associative and commutative binary operator.
- *filter*: A transformation that returns a new RDD containing only the elements that satisfy a predicate.
- *sortBy*: A transformation that returns the input RDD sorted by the given key function.
- *count*: An action that returns the number of elements in the input RDD.
- *take*: An action that returns the first `num` elements of the input RDD. *num* is the number of elements to return.
- *lookup*: An action that returns the list of values in the input RDD with the specified key.

The main objective of our proposal is to create an scalable algorithm that can classify time series in Big Data environments. We set as a requirement that the relationship between the increase in running time and the amount of data to be processed must be linear. The second objective is that our proposal must be able to extract and use interpretable features such as shapelets in the best possible way. For this reason, our proposal uses the Shapelet Transform (ST). In addition, our proposal must maintain an accuracy close to the one obtained by similar sequential algorithms. Based on those requirements we have called our proposal Distributed FastShapelet Transform (DFST).

DFST is based on the heuristic shapelet search made in the FS algorithm [18], which is the shapelet search algorithm with the lowest computational complexity of the shapelet search algorithms namely $O(nm^2)$.

Figure 1: Distributed FastShapelet Transform (DFST) Schema


Start
NodeData RDD(tsId, tsValues)
Create SAX List
by each (tsId, tsValues)
new Array((SAXWord, SAXInfo))
SAXList RDD(SAXWord, SAXInfo)
CreateRandom Projections
by each (SAXWord, SAXInfo)
update (SAXWord, SAXInfoUpdated)
SAXList RDD(SAXWord, SAXInfoUpdated)
Score SAX Words
by each (SAXWord, SAXInfoUpdated)
new (SAXWord, Score)
SAXScore RDD(SAXWord, Score)
Find Best Shapelets
by topK best scores (SAXWord, SAXInfoUpdated)
new bestLocalShapelet
add each bestLocalShapelet to list_ST
bestLocalShapelet Shapelet
Compare Node Shapelets if bestLocalShapelet is better than bestNodeShapelet
new bestNodeShapelet
bestNodeShapelet Shapelet
Next Shapelet Length if ShapeletLength < MaxShapeletLength
increase ShapeletLength by step
Next Shapelet Length if ShapeletLength >= MaxShapeletLength
select topK best Shapelets from list_ST and add them to list_ST_selected
Compare Shapelets if bestNodeShapelet is better than bestShapelet
new bestShapelet
bestShapelet Shapelet
Results evaluations Not enough good results
new Node
new NodeData
Results evaluations Enough good results
Write Shapelet classification tree
Store Shapelet Transform store list_ST_Selected
Apply Shapelet Transform to Train and Test data
ST_Train RDD(InstancesTrainID, Dist_to_list_ST_Selected, Labels)
ST_Test RDD(InstancesTestID, Dist_to_list_ST_Selected)
Learn and store the Random Forest Model
Learned Random Forest Model RF_Model
Predict ST_Test using the learned Random Forest Model
Classification Results RDD(InstancesTestID, ClassificationLabels)
End

The operation of DFTS is shown in Figure 1. The original ideas of the SAX word creation, the generation of random projections and the score of the SAX words of the FS algorithm have been used in the DFST algorithm, but they have been redesigned to work in a distributed fashion. The main issues in which DFST differs from FS are detailed in the following paragraphs.

A first step in the procedure is the application of a SAX transformation for a simplified representation of the time series. Next step is a generation of random projections of the computed SAX words. This process is prone to exponential growth. So when applied to larger datasets the overall set of word size must be controlled. DFST reduces the length of SAX words as a function of the dataset partition that is processed by each thread.

Another effective improvement is that instead of a single shapelet, DFST selects the *topK* shapelets with the highest scores. Then, for each selected shapelet and locally at each thread, it computes the information gain, the number of misclassified time series, and the gap between the time series that have this shape and those that do not. The search for the best shapelets has been modified so that multiple shapelets are obtained, in each node of the tree, and saved for later use in the ST.

DFST applies the ST to the training and test sets, calculating the distance of each time series to every selected shapelet. This vector of distances is used as a representation of the time series. Thus, at this stage the output is not a classification result but a representation of the time series in terms of the most relevant shape features discovered and with the same length for all. That is, time series are represented in a new way that allows for an easy processing with conventional data mining and machine learning algorithms. In particular, almost any classification algorithm can be used now, removing the tie to the classification tree in the FS algorithm.

The last step in DFST is to learn a model with the transformed training data, ST_Train. It will be used to predict the transformed test data, ST_Test, obtaining the final classification results. So far, we have used Random Forest [5], but this is no restriction of course.

An interesting analysis of the proposal is considering what parts of DFST a sequential and what parts are parallel. In addition, the exchange of data among threads also have a clear impact on the running time. DFST has a first sequential part of reading and delivering the input data. The generation of SAX words and the creation of random projections are performed in a fully parallel fashion, by applying certain transformations independently to each input time series. The score of each shapelet candidate, and its position based on this score, require the overall exchange of information among threads as it is necessary to count the presence or not of each shapelet candidate in all the time series of the dataset and perform this sorting. There is another stage of data exchange in the evaluation of the *topK* shapelets candidates as it needs to process and collect the calculations of information gain for each of them.

DFST is depicted in Algorithm 1. In Algorithm 2 we present the core procedure of our proposal. In this process the best set of shapelets is selected. We must differentiate between distributed and not distributed variables: as general nomenclature, we represent the distributed variables with their first letter in capital.

In Algorithm 1, all original instances are introduced in node 1 (line 3). The number of instances not correctly classified are initialized with the number of original instances (line 4). The algorithm processes a node as long as the number of instances not correctly classified is greater than the threshold (lines 5-34). For each node, the algorithm selects the instances included in that node (line 6). Then it starts to process subsequences of time series from the minimum to the maximum introduced lengths (line 9-26). This process has 4 principal steps:

First, the algorithm obtains a list of all possible SAX words in the current node data (lines 10-17). We use a *map* transformation over the node in order to obtain a HashMap that contains all possible SAX word for each time series (lines 10-13). Each SAX word has a usax item associated with information about that SAX word. Then, we use a *reduce* function (lines 14-17) for combining identical SAX words. The usax item of the first SAX word is updated with extra information about the second SAX word. Secondly, the algorithm computes R random projections for each SAX word and then it counts matches among projections of different SAX words (line 18). The counted values are saved in the match counter of the usax item of the corresponding SAX word. Third, the algorithm computes a score for each SAX word based on their match counter (lines 19-22). Fourth, our proposal selects the *topK* SAX words that have the higher score and it computes the best shapelet among them (lines 23-24). If the computed shapelet improves the last best case, this shapelet is selected as the current best case (line 25). We increase the subsequence length by the desired *step* (line 26).

Once the current node has been processed, the algorithm analyzes the results and prepares the next iteration. With the best shapelet of the current node, each time series is assigned to one of the possible next nodes (line 28). DFST obtains the time series incorrectly classified and the stopping threshold of the algorithm (line 29) for the next nodes. The *topK* shapelets with the highest gain are selected and added to the final list of shapelets (line 30). This list contains the shapelets to be used in the ST. Finally, we increase the node indicator (line 31). After all the relevant shapelets have been extracted, ST is performed on the training dataset. The minimum distance of each time series to each of the selected shapelets is calculated (line 33). Transformed training data is used as input for the classification algorithm, e.g. RandomForest (line 34). Finally, DFST returns the learning model obtained on the training data set and the list of shapelets used in the ST (line 35).

In Algorithm 2, our proposal searchs for the best shapelet of the current SAX word list. We take the *topK* best scored SAX words to process them (line 1). Each shapelet is processed independently (lines 3-28). The algorithm retrieves information about the original subsequence of the SAX word processed (line 4). It uses this information to obtain the original values of this subsequence (line 5). We use *mapPartitions* transformation to evaluate independently each shapelet over each partition of the data (lines 6-22). Our proposal computes the local frequency of each class in the partition (line 9). Then it computes the entropy of

Algorithm 1 DFST Algorithm

```
    Input:
        $OrgData$: RDD with (TsId, LabeledPoint(features, label))
        $minLen$: minimum length of shapelet processed
        $maxLen$: maximum length of shapelet processed
        $step$: the increment of length of shapelet processed
        $topK$: number of best shapelets evaluated by iteration
        $R$: number of random projections computed
    Output:
        $RF\_Model$: RF learned model
        $list\_ST\_selected$: list of shapelet selected for the ST
 1: Node $\leftarrow$ 1
 2: bestSh, threshold, list_ST_selected $\leftarrow$ 0
 3: NodeTsList[Node] $\leftarrow$ getAllTsId(OrgData)
 4: incorrectlyClassifiedData $\leftarrow$ count(NodeTsList[Node])
 5: while incorrectlyClassifiedData $>$ threshold do
 6:     NodeData $\leftarrow$ filter(OrgData, NodeTsList[Node]==getTsId(OrgData))
 7:     subSeqLen $\leftarrow$ minLen
 8:     list_ST $\leftarrow$ 0
 9:     while subSeqLen $<$ maxLen do
10:         SaxMapWords $\leftarrow$
11:             map ts $\in$ NodeData
12:                 HashMap[saxWord, usax] $\leftarrow$ createSaxList(ts, subSeqLen)
13:             end map
14:         SaxMapWordsReduced $\leftarrow$
15:             reduce (SaxListX, SaxListY) $\in$ SaxMapWordsReduced
16:                 combineSaxList(SaxListX, SaxListY)
17:             end reduce
18:         RPWords $\leftarrow$ createRP(SaxMapWordsReduced, R, subSeqLen)
19:         ScoreList $\leftarrow$
20:             map (word, usax) $\in$ RPWords
21:                 (word, calScore(usax, labels(NodeData)))
22:             end map
23:         (sh, list_ST) $\leftarrow$ findBestShapelet(topK, subSeqLen, ScoreList,
24:                         NodeData, list_ST)
25:         if (sh $>$ bestSh) then bestSh $\leftarrow$ sh end if
26:         subSeqLen $\leftarrow$ subSeqLen + step
27:     end while
28:     NodeTsList $\leftarrow$ setNextNodes(NodeTsList, Node, OrgData, bestSh)
29:     (incorrectlyClassifiedData, threshold) $\leftarrow$ evaluate(Node, OrgData, bestSh)
30:     list_ST_selected $\leftarrow$ addBestShapelets(topK, list_ST, list_ST_selected)
31:     Node $\leftarrow$ Node + 1
32: end while
33: ST_Train $\leftarrow$ calcST(Org_Data, list_ST_selected)
34: RF_Model $\leftarrow$ trainRF(ST_Train)
35: return (RF_Model, list_ST_selected)
```

local data given the number of instances of the partition and the class frequencies (lines 10-11). We compute the distance between the subsequence selected and each time series in the partition (line 12). These results are sorted in ascending order (line 13) and then it searches on these for the best shapelet (lines 14-21). For this, we compute the information gain, the number of instances in a different position of the correct and the inter-class gap (lines 19-21). Then it takes as the best shapelet of the partition the case that has the maximum gain, the minimum number of differences and the maximum inter-class gap (line 19), respectively. As $DistributedShapeletCalculated$ RDD contains the best shapelet of each partition, we must take the best of these cases following the previous criteria. We take the first of those cases (line 23). Our proposal saves the selected best shapelet for the ST afterwards (line 24). We compare this shapelet with the best case at the moment and we take the best one (line 25). Therefore, the algorithm returns the best shapelet of the $topK$ processed and the list with the node's shapelets for the ST (line 28).

DFST uses the shapelet extraction mechanism proposed in the FS algorithm. This proposal selects the $topK$ shapelets extracted in each internal node as shapelets to be applied in the ST. For example, for a tree with 4 internal nodes and a $topK$ value of 10, our proposal would extract 40 shapelets.

In DFST, for a dataset with n time series of length m from which it has been extracted s shapelets, once already applied the ST we will obtain a dataset with n instances of length s. This approach has proven to be the most recommended within the field of time series classification by shapelets. The main advantage of this transformation is the possibility of applying any automatic learning algorithms, e.g. decision trees, to classification problems of time series. In our proposal, we have chosen to use the Random Forest algorithm [5] included in the MLlib of Spark with default parameters and 1000 trees.

3.1. Computational Complexity

COTE algorithm is the most popular and widely used algorithm for time series classification. This approach achieves excellent results in most cases. However, COTE is a computationally expensive technique. The computational complexity of COTE is set by the classifier with greater computational complexity that forms part of this ensemble. The algorithm with highest computational complexity included in COTE is the ST, with a computational complexity of $O(n^2m^4)$. The computational complexity limits the number of time series that this approach is capable of processing in conventional computers with limited resources. This high computational complexity order prevents its use in Big Data environments.

The DFST learning computational complexity in time is defined by the sum of the complexity of the FS based search algorithm, the shapelet transformation applied to the training set and the computational complexity of learning a Random Forest model. The computational complexity of the shapelet search algorithm used, which is based on FS, is $O(nm^2)$, being n the number of time series and m the length of time series. The transformed shapelet applied to the training set calculates the distance of each shapelet to each time series. This distance has a computational complexity $O(timeSeries_{length} - shapelet_{length})$,

Algorithm 2 *findBestShapelet*

```
Input:
    $topK$: number of better SAX words to consider
    $subSeqLen$: length of shapelet processed
    $ScoreList$: RDD with the scores for word
    $NodeData$: RDD that contains the data of the current node
    $list\_ST$: list with the node's shapelets for the ST
Output:
    $bestSh$: best Shapelet found
    $list\_ST$: updated list with the node's shapelets for the ST
1:  ScoreListLocal ← take(sortBy(ScoreList, "score", "decrescent"), topK)
2:  k, bestSh ← 0
3:  while $k < topK$ do
4:      tsCandInfo ← ScoreListLocal(k)
5:      tsCand ← lookup(Data, tsIndex(tsCandInfo))
6:      DistributedShapeletCalculated ←
7:          mapPartitions DataPartition ∈ NodeData
8:              localBestSh ← 0
9:              localClassFreq ← countClassLabels(labels(DataPartition))
10:             localClassEntropy ←
11:                 entropyArray(localClassFreq, size(DataPartition))
12:             distTs ← calcNNDist(DataPartition)
13:             orderedDistTs ← sortBy(distTs, "NNDist", "ascending")
14:             i ← 0
15:             while i < (size(orderedDistTs) - 1)
16:                 sh ← calcShInfo(tsCandInfo, orderedDistTs(i),
17:                         orderedDistTs(i+1), localClassFreq,
18:                             localClassEntropy)
19:                 if (sh > localBestSh) then localBestSh ← sh end if
20:             end while
21:             localBestSh
22:         end mapPartitions
23:     sh ← getBestShapelet(DistributedShapeletCalculated)
24:     list_ST ← addShapelet(sh, list_ST)
25:     if (sh > bestSh) then bestSh ← sh end if
26:     k ← k+1
27: end while
28: return (bestSh, list_ST)
```

since the minimum distance between the shapelet is calculated to all the sequences of the time series of equal length to that of the shapelet. To simplify, we replace $(timeSeries_{length} - shapelet_{length})$ with p in the following computational complexity equations. This calculation is repeated for each time series of the training set, n, and as many times as shapelets have been extracted, s.

Table 1: Classification Problems Data

Classification Problem 1: Random sequences of 0 and 1 processed by 4 ARIMAs

Class	Model	Coeff
0	ARIMA(1,0,1)	AR(0.6), MA(0.1)
1	ARIMA(1,0,2)	AR(0.5), MA(-0.6,0.6)
2	ARIMA(2,0,2)	AR(0.5,-0.7), MA(-0.6,0.5)
3	ARIMA(2,1,2)	AR(0.5,-0.7), MA(-0.6,0.5)

Dataset size: 16,000,000 Time series length: 100

Classification Problem 2: 6 Real Datasets with 10% of White Noise

Class	Dataset
0	ECG5000
1	PhalangesOutlinesCorrect
2	Two_Patterns
3	Gun_Point
4	wafer
5	ElectricDevices

Dataset size: 8,000,000 Time series length: 150

For this reason, the computational complexity of the ST applied is $O(pns)$. The computational complexity of the Random Forest learning has been theoretically demonstrated [14] as $O(tk\tilde{n}\log\tilde{n})$, being t the number of randomized trees, k the number of variables randomly included at each node and n the number of samples of the training partition. $\tilde{n} = 0.632n$ due to the 63.2% of unique samples, on average, [4] extracted by bootstrap. Finally, the DFST learning computational complexity in time is: $O(nm^2) + O(pns) + O(tk\tilde{n}\log\tilde{n})$.

The DFST prediction time complexity is defined by the sum of the complexity of the ST applied to the testing set and the computational complexity of Random Forest model for prediction. The computational complexity of the ST applied to the test set is $O(pls)$, being l the number of samples in the test partition. The computational complexity of prediction of the Random Forest is $O(t\log l)$. Finally, the DFST prediction time complexity is: $O(pls) + O(t\log l)$.

Comparing our proposal with FS, we see that our proposal can be applied in Big Data environments and obtain better classification results due to the inclusion of ST in the training and classification phase. The use of this transformation together with traditional classification algorithms has proven to offer better results than the use of a traditional classification tree. If we compare our proposal with the ST, which has a computational complexity $O(n^2m^4)$, we see that DFST obtains a lower order complexity, $O(nm^2)+O(pns)+O(tklog)$, which allows it to be used in Big Data environments with similar accuracy results.

4. Empirical study

To asses the effectiveness and performance of our proposal, we have developed a thorough experimental setup. It is described in this section along with the analysis of the experimental results. In Section 4.1, we present the experimental framework as well as the details of the datasets and the parameters

Table 2: Experimentation Configuration Values

Parameter	Value
minLen	10
maxLen	100 (Problem 1) / 150 (Problem 2)
step	10
R	1
topK	10

used in the methods. In Section 4.2, we present the results of performance over huge datasets in a Big Data environment. These datasets can not be processed/handled by the original algorithm or typical computers. In Section 4.3, we present the accuracy results of DFST on the datasets used in the previous section. In Section 4.4, we show the utility of shapelets as input characteristics to the classification models created by DFST.

The source code of our proposal, the datasets creation files, results and additional material are available online [2].

4.1. Experimental Framework

Since no publicly available large time series classification dataset could be found, we have created two classification problems[3]. For each problem, multiple datasets have been created with different numbers of instances. The number of instances varies from 100,000 to 20 million. The length of the time series created are 100, for first problem, and 150, for the second problem.

As a first problem, we have simulated 4 ARIMA models with `arima.sim()` function from "stats" package of the R language [17] over random sequences of 0 and 1. Each model has been assigned a classification label, Table 1. For the second problem, we have selected 6 datasets of time series classification problems from the UCR repository: ECG5000, PhalangesOutlinesCorrect, Two_Patterns, Gun_Point, wafer and ElectricDevices. These datasets are representative of the large groups of existing problems: ECG (Electrocardiograms), Image, Motion, Sensor, Simulated and Device, respectively. Each dataset has been assigned a classification label, Table 1.

For our experiments, we have used a Big Data cluster composed of one master node and 20 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, 2 TB HDD (1 TB HDFS), 64 GB RAM. We have used the following software configuration: CentOS 6.9, Hadoop 2.6.0-cdh5.4.3 from Cloudera open source Apache Hadoop distribution, Apache Spark and MLlib 1.6.0, 23 threads/node, 1040 RAM GB (52 GB/node).

To obtain the sequential results in a comparable setting we have run the sequential algorithm in one of the nodes.

[2]Additional material on the Distributed FastShapelet Transform (DFST) proposal. `http://dicits.ugr.es/papers/DFST/`

[3]Popular dimensions datasets for benchmark, for example UCR datasets, lacks behind the values that currently qualify as starting Big Data dimensions

Table 3: Sequential FS and DFST Running Time vs Number of Time Series Problem 1

Number of time series	Sequential FS Running time (s)	DFST Running time (s)
100,000	2,088.06	12,190.91
200,000	4,119.02	12,310.29
300,000	6,699.75	9,748.17
400,000	7,909.84	11,222.16
500,000	11,643.08	9,025.84
600,000	13,407.48	12,057.48
700,000	16,317.37	8,965.22
800,000	18,454.49	9,798.00
900,000	20,764.48	12,226.55
1,000,000	22,513.41	11,855.29
1,100,000	24,622.18	9,651.16
1,200,000	26,634.61	11,536.39
1,300,000	30,618.57	13,686.43
1,400,000	30,524.07	13,741.24
1,500,000	33,556.26	13,075.48
1,600,000	36,532.96	14,814.28
1,700,000	38,695.64	14,998.85
1,800,000	41,606.93	15,583.09
1,900,000	35,099.40	16,625.55
2,000,000	49,435.64	15,128.27
2,100,000	48,421.24	15,692.12
2,200,000	53,036.16	15,667.97
3,000,000	NC	20,560.59
4,000,000	NC	26,823.02
5,000,000	NC	27,244.44
6,000,000	NC	33,295.01
7,000,000	NC	35,709.65
8,000,000	NC	42,389.48
9,000,000	NC	44,831.42
10,000,000	NC	52,685.68
11,000,000	NC	61,208.60
12,000,000	NC	67,723.17
13,000,000	NC	75,183.40
14,000,000	NC	75,973.21
15,000,000	NC	98,655.94
16,000,000	NC	103,155.57

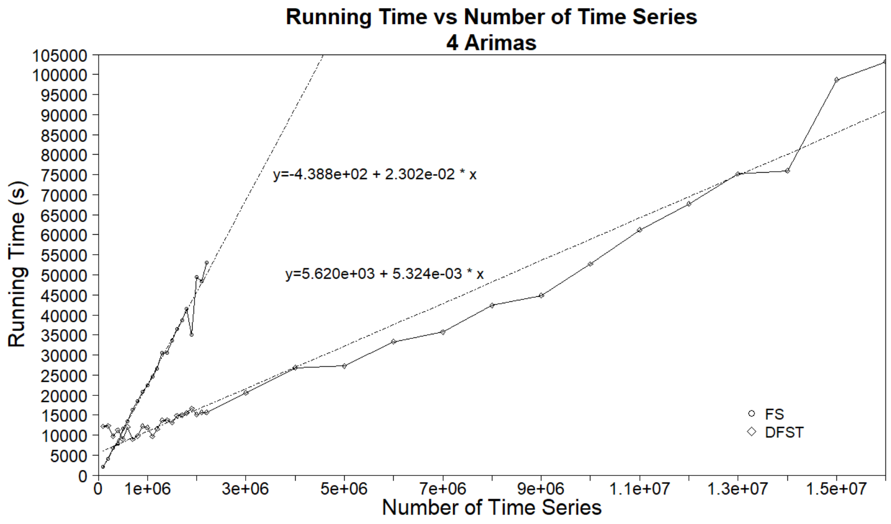
NC indicates cases not computable by sequential algorithm.

4.2. Performance in Big Data environments

We are mainly concerned with accuracy and scalability of the proposed approach. Since no other scalable algorithm is available we will compare DFST to the FastShapelet algorithm, because they share some basic ideas.

The experimental configuration used is presented in Table 2. In Table 3 and 4 we show the total running times of our proposal for Big Datasets. In both cases we can see that starting from 500,000 time series onwards our distributed proposal needs lower running times than those obtained by the sequential algorithm. From this point on, as the number of time series processed increases, so does the difference between the two algorithms. Figures 2 and 3 show these results graphically. In both figures we can see the point from which the DFST improves over the sequential algorithm. We can also appreciate a linear relationship between the running time and the number of time series. The expressions of the line that defines the execution time behavior of each algorithm have been

Figure 2: Sequential FS and DFST Running Time vs Number of Time Series Graph Problem 1



included. The execution times of problem 2 show increments in each step greater than those obtained in problem 1. This is mainly due to the fact that the time series of problem 2 have a length of 150, which is 50% greater than that of problem 1, that is 100.

Regarding the amount of data, for the first problem we have done experiments with DFST on the cluster with up to 16 millions of time series with a length of 100. The sequential algorithm can process with up to 2,200,000 time series with a length of 100. For the second problem, DFST on the cluster with up to 8 millions of time series with a length of 150. The sequential algorithm can process with up to 1,100,000 time series with a length of 150. Sequential or iterative algorithms have limitations in terms of the amount of data they can process. DFST has shown to be able to cope with datasets of any size. This is the definition of scalability.

A complementary view of the proposal could be provided through the speedup. Because of different issues this value is not very meaningful in this case. To begin with, the large difference between the size of the dataset processable by the sequential approach with respect to the parallel one. In addition, the effective gain in performance for the parallel approach is better realized for bigger case sizes, where the sequential approach cannot be applied —we would have to wait for it to finish an unacceptably long time or we would run out of main memory.

Next, the difference between the platforms. Big Data platforms are designed with scalability and easiness of use in mind, with high performance at a second level of importance. A completely parallel solution (based on communications through main memory) is scalable up to a limited size. A distributed approach,

Table 4: Sequential FS and DFST Running Time vs Number of Time Series Problem 2

Number of time series	Sequential FS Running time (s)	DFST Running time (s)
100,000	3,801.29	13,608.42
200,000	8,155.19	24,660.67
300,000	13,564.75	18,829.35
400,000	17,335.96	19,032.37
500,000	22,330.90	20,188.38
600,000	28,371.62	16,507.55
700,000	35,450.22	21,773.58
800,000	36,850.60	16,073.29
900,000	39,122.43	16,882.79
1,000,000	47,092.72	20,669.58
1,100,000	51,151.38	28,544.63
2,000,000	NC	27,774.42
3,000,000	NC	39,545.70
4,000,000	NC	70,036.37
5,000,000	NC	71,690.11
6,000,000	NC	72,842.22
7,000,000	NC	88,590.94
8,000,000	NC	112,239.49

NC indicates cases not computable by sequential algorithm.

e.g. based on message passing, is more scalable although the required design and code effort is larger. In addition, its deployment is more troublesome and not feasible for the average programmer. Instead, Big Data platforms have become very popular at the price of lower performance with respect the optimal usage of resources. We selected a Big Data platform to implement DFST with idea of having a wider set of potential users.

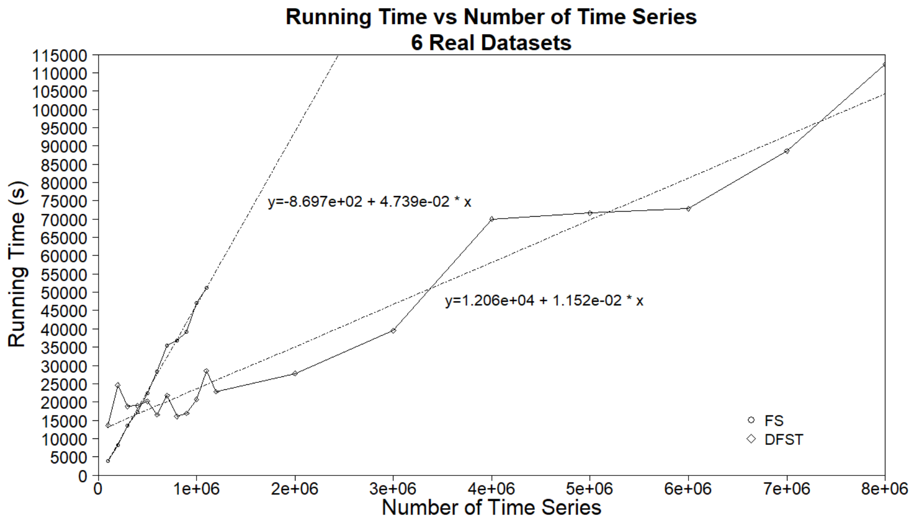
Finally, the programming language also has an impact. The sequential algorithm is coded in C++, allowing for an effective usage of the underlying hardware platform. On the other hand, Big Data platforms are coded either on Java or languages that compile to Java Virtual Machine. Their performance falls behind that of C++, and while this effect in terms of complexity would fall under a constant term it is usually a large one.

After the considerations made above, we made a computation of the running times ratios sequential vs MapReduce for the largest size that the sequential approach can handle and obtained a value of 4. This entails us to conclude the following advice: when the running time of the sequential approach is acceptable use it, otherwise use DFST.

4.3. Accuracy

To the best of our knowledge, there is currently no other proposal that can be applied to such massive time series datasets. As there are no time series

Figure 3: Sequential FS and DFST Running Time vs Number of Time Series Graph Problem 2



classification algorithms in Big Data environments, the comparison of accuracy is not feasible. Instead, the accuracy of our proposal has been compared with that obtained by the FS algorithm, which has been the basis of the shapelet search process implemented in our proposal. This comparison has only been possible up to the processing limit of this algorithm on current computers.

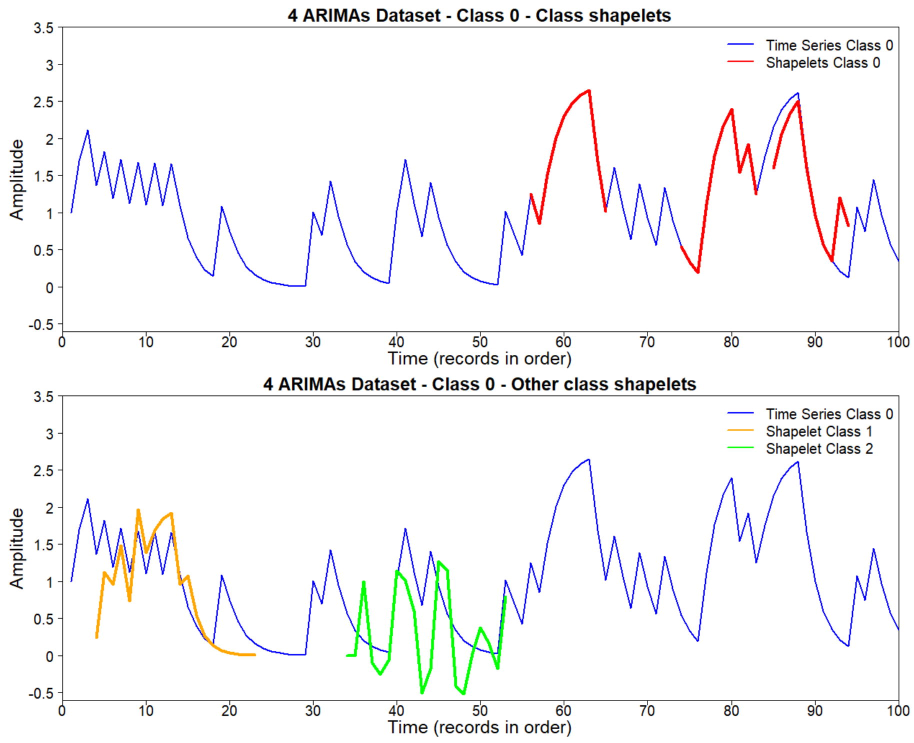
Accuracy is measured in terms of the rate of correctly classified time series, as a percentage. In the ARIMAs problem, the sequential FS algorithm has obtained an average accuracy of 93.11% for the problems that has been able to process. For these problems DFST has obtained an average accuracy of 99.32%, being higher than average accuracy obtained by the sequential algorithm. Over all processed datasets, 100,000 to 16,000,000,000 time series, DFST has achieved an average accuracy of 99.40%. Finally, DFST has obtained 99.66% accuracy on the largest dataset processed.

For the second problem, the sequential FS algorithm has obtained an average accuracy of 80.07% for the problems that has been able to process. For these problems, DFST has obtained an average accuracy of 82.60%, again higher than average accuracy obtained by the sequential algorithm. Over all processed datasets, 100,000 to 8,000,000,000 time series, DFST has achieved an average accuracy of 81.92%. Finally, DFST has obtained 74.17% accuracy on the largest processed dataset.

4.4. Interpretability of models

The learning model used in our proposal is a Random Forest that uses as input a distance matrix created from the extracted shapelets. Each row refers

Figure 4: Example of shapelets interpretability.



to an input time series and each column contains the distance from that time series to a shapelet. It is logical that the distance between a shapelet of a class and a time series of the same class is close to 0. On the other hand, the distance between a shapelet and a time series of different classes will be distant to 0. Since we have a large number of shapelets of different classes, there are a large number of combinations that allow a machine learning algorithm to find the relationships between the different shapelets that define the different classes. Figure 4 shows an example of the distances from different shapelets to a time series.

At the top of Figure 4, you can see how different class 0 shapelets are similar to time series of the same class. In this example we obtain two Euclidean distances equal to 0 and one of 1.39. In the lower part of Figure 4, you can see how the shapelets of classes 1 and 2 are quite dissimilar to the time series of class 0, obtaining distances equal to 2.21 and 2.77, respectively. In both examples, the shapelets have been placed at points where the distance between the shapelet and the time series is minimized.

To summarize, shapelets are graphical features that adapt to time series shapes. They are the building blocks (input features) to the classification models built with DFST.

The online resource[4] includes some examples for the different classes of the 2 problems proposed in this paper.

5. Conclusions

In this work, we have proposed the Distributed FastShapelet Transform algorithm (DFST), based on the MapReduce paradigm. It is the first proposal of a completely scalable algorithm for time series classification. The state of the art of time series classification problems is dominated by algorithms such as COTE, very effective but also with a very high computational complexity, i.e. $O(n^2m^4)$. However, complexities this high prevent their application in a Big Data environment. Alternative proposals, like FS, reduce the complexity to the levels of $O(nm^2)$, at the price of a lower accuracy. DFST addresses both issues providing a linear complexity (with respect to the number of time series) approach with comparable accuracy results. Inspired on the shapelets search procedure of the FS algorithm, a number of decisive steps have been redesigned like the use of gain of information measures, the selection from multiple shapelets, or the generalization to allow the use of different classifiers, not being tied to classification trees.

The proposed algorithm has been implemented in the Apache Spark framework and is now available as an open-source contribution to the MLlib, making it available for any practitioner o researcher to use. We have carried out a thorough empirical study focused on scalability and accuracy. The accuracy of DFST is higher than that obtained by FS in all the cases were both could be applied. The linear complexity of the final algorithm allows for the application of the algorithm on datasets of any size.

6. Acknowledgements

This research has been partially funded by the following grants: TIN2013-47210-P and TIN2016-81113-R both from the Spanish Ministry of Economy and Competitivity, and P12-TIC-2958 from Andalusian Regional Government, Spain. Francisco J. Baldán holds the FPI grant BES-2017-080137 from the Spanish Ministry of Economy and Competitivity.

7. References

[1] A. Bagnall, J. Lines, A. Bostrom, J. Large, E. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Mining and Knowledge Discovery 31 (3) (2017) 606–660.

[4]Additional material on the Distributed FastShapelet Transform (DFST) proposal. `http://dicits.ugr.es/papers/DFST/`

[2] A. Bagnall, J. Lines, J. Hills, A. Bostrom, Time-series classification with cote: the collective of transformation-based ensembles, IEEE Transactions on Knowledge and Data Engineering 27 (9) (2015) 2522–2535.

[3] A. Bostrom, A. Bagnall, Binary shapelet transform for multiclass time series classification, in: Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXII, Springer, 2017, pp. 24–46.

[4] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.

[5] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[6] J. Buhler, M. Tompa, Finding motifs using random projections, Journal of computational biology 9 (2) (2002) 225–242.

[7] A. CCAFS Climate Change, F. Security, Ccafs-climate data weather stations, `http://www.ccafs-climate.org/weather_stations/`, accessed: 2018-01-31.

[8] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[9] T. A. S. Foundation, Apache hadoop, `http://hadoop.apache.org/`, accessed: 2018-01-31.

[10] T. A. S. Foundation, Apache spark: Lightning-fast cluster computing, `https://spark.apache.org/`, accessed: 2018-01-31.

[11] Y.-S. Jeong, R. Jayaraman, Support vector-based algorithms with weighted dynamic time warping kernel function for time series classification, Knowledge-based systems 75 (2015) 184–191.

[12] J. Lin, E. Keogh, W. Li, S. Lonardi, Experiencing sax: a novel symbolic representation of time series, Data Mining and knowledge discovery 15 (2) (2007) 107.

[13] J. Lines, L. M. Davis, J. Hills, A. Bagnall, A shapelet transform for time series classification, in: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2012, pp. 289–297.

[14] G. Louppe, Understanding random forests: From theory to practice, arXiv preprint arXiv:1407.7502.

[15] A. Mueen, E. Keogh, N. Young, Logical-shapelets: an expressive primitive for time series classification, in: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2011, pp. 1154–1162.

[16] F. R. B. of ST. Louis, Federal reserve bank of st. louis, `https://fred.stlouisfed.org/`, accessed: 2018-01-31.

[17] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria (2015). URL http://www.R-project.org/

[18] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh, Data mining a trillion time series subsequences under dynamic time warping, in: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, AAAI Press, 2013, pp. 3047–3051.

[19] T. Rakthanmanon, E. Keogh, Fast shapelets: A scalable algorithm for discovering time series shapelets, in: Proceedings of the 2013 SIAM International Conference on Data Mining, SIAM, 2013, pp. 668–676.

[20] A. W. Services, Ccafs-climate data, https://aws.amazon.com/es/datasets/ccafs-climate-data/?tag=datasets%23keywords%23climate, accessed: 2018-01-31.

[21] A. W. Services, Federal reserve economic data - fred, https://aws.amazon.com/es/datasets/federal-reserve-economic-data-fred/?tag=datasets%23keywords%23economics, accessed: 2018-01-31.

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

[23] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, 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, USENIX Association, 2012, pp. 2–2.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets., HotCloud 10 (10-10) (2010) 95.