A Preliminary Study of the Feasibility of Global Evolutionary Feature Selection for Big Datasets under Apache Spark

M. Galar, I. Triguero, H. Bustince, F. Herrera

Abstract— **Designing efficient learning models capable of dealing with tons of data has become a reality in the era of big data. However, the amount of available data is too much for traditional data mining techniques to be applicable. This issue is even more serious when evolutionary algorithms are a key part of the learning algorithm. In this scenario, one typical approach is to follow a divide-and-conquer strategy, where data is divided into different chunks that are individually and independently addressed. Afterwards, the partial knowledge obtained from each chunk of data is combined in order to give a solution to the problem. Nevertheless, these kinds of local approaches do not look at data as a whole, missing a global view of the problem, which may result in less accurate models that also depend on how data is split.**

In this work, we focus on evolutionary feature selection algorithms. A divide-and-conquer approach to handle evolutionary feature selection in big data was already developed. We aim at designing its global counterpart, which looks at the feature selection problem from a global perspective, making use of the data as a whole to select the most appropriate features. In order to do so, we consider Apache Spark as a big data technology where our algorithm is implemented. We design a genetic algorithm capable of dealing with big datasets by selecting the proper parameters for our base algorithm (the well-known CHC) and adapting the evaluation procedure to take all the distributed data into account. Several preliminary results are discussed to study the feasibility of global evolutionary feature selection methods for big datasets.

I. INTRODUCTION

Machine learning algorithms are supposed to improve their performance as long as more data is considered. However, in practice, several challenges are found when learning algorithms are applied to big datasets due to memory and time limitations [1]. In order to tackle these issues, parallelization technologies have spread out providing us with powerful frameworks easing the processing of large distributed datasets [2]. In this scenario, the problem is how to design effective machine learning algorithms using these kinds of technologies and taking into account the problems arising when working with distributed data such as the communication overhead.

Hadoop [3] was the first alternative to make working with big datasets accessible. Data-intensive applications were translated to the MapReduce paradigm [4] in order to be run in Hadoop. This way, one was able to work with large datasets distributed across a cluster of computing nodes via Hadoop Distributed File System (HDFS) in a transparent and fault-tolerant way [5], [6]. However, the MapReduce programming paradigm implemented in Hadoop had several limitations to make machine learning algorithms capable of working on top of it [7]. The main drawback was the fact that the batch-oriented data processing was unable to quickly handle iterative algorithms while most of the machine learning algorithms require a number of iterations to learn a model. For this reason, new frameworks, such as Spark [2] or Flink [8], arose to further improve the characteristics of Hadoop. These frameworks are built upon MapReduce paradigm and extend its capabilities providing one with new kind of high throughput in-memory distributed datasets, making the implementation of iterative processes easier and its running times much faster.

When designing machine learning algorithms for big datasets, there are two main approaches that can be followed. Local approaches [9] are based on taking advantage of how data is distributed in MapReduce to learn a model for each chunk of data. Obviously, these models are partial and approximate because they are learned only considering a part of the data and without interacting with the rest. Then, all the models learned are combined, which can be simply done by considering all of them together to form an ensemble or other specific aggregations can be developed. Hence, local approaches are dependent on the number of partitions considered and may degrade their performance when the number of partitions is too high with respect to the number of examples. Otherwise, global approaches [10], [11] consider the data as a whole. Therefore, the learning model should be capable of working with all the distributed data. Global solutions achieve the same performance and model regardless of the number of partitions considered and hence, they obtain the same model as the one that would be obtained if the method could be run in a single machine. The main disadvantage of global approaches is that their design is much more complex. All the machine learning algorithms developed in Spark's MLlib library [12] are global models.

Data preprocessing is a key part of data mining, required for machine learning algorithms to obtain meaningful models. Among data preprocessing techniques, feature selection methods [13] try to reduce the original number of features of the problem by discarding those irrelevant or redundant ones. Evolutionary algorithms have shown to be a great alternative to address feature selection [14]. However, they suffer from

This work was supported by the Research Projects TIN2017-89517-P and TIN2016-77356-P (AEI/FEDER, UE)

M. Galar and H. Bustince are with the Department of Automatic and Computation, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006 Pamplona, Spain. E-mails: {mikel.galar, bustince}@unavarra.es

I. Triguero is with the School of Computer Science, University of Nottingham, United Kingdom. E-mail: isaac.triguero@nottingham.ac.uk

F. Herrera is with the Department of Computer Science and Artificial Intelligence of the University of Granada, CITIC-UGR, Granada, Spain, 18071. E-mail: herrera@decsai.ugr.es

scalability issues when dealing with big datasets. For this reason, previous works have overcome this problem by developing a local evolutionary feature selection model [15]. In this model, the CHC evolutionary algorithm [16] is used for feature selection with each chunk of data. As a result, several sets of selected features are obtained, which needs to be aggregated. To do so, the number of times a feature appears as selected in each selected subset is counted and a user predefined threshold is applied to finally select those features with more counts. Nevertheless, this approach inherits all the drawbacks of local models such as the dependence on the number of partitions and the possible loss of performance with greater degrees of parallelism.

In order to avoid these drawbacks, in this work we focus on the design and study of feasibility of a global evolutionary feature selection (global EFS) model. This is possible thanks to technologies such as Apache Spark, which allows us to take multiple iterations over the same data without much overhead. However, there are some key points that need to be cautiously designed. Among them, the evaluation function is the most critical part, because it is the one with the greatest effect in the final runtime of the algorithm. For this reason, we will consider a fast alternative to evaluate each chromosome in the fitness function. The rest of the evolutionary algorithm is based on CHC [16] and on the optimizations introduced in [17] for big data problems. Our aim is to analyze whether this kind of global EFS is feasible to work with big datasets or not. To do so, we will carry out a preliminary experimental study with two big datasets with 631 and 2000 features, respectively. Our first results indicate that global EFS is viable if the appropriate characteristics are considered.

The remainder of this paper is as follows. Section II provides background information about big data frameworks and evolutionary feature selection for big data classification. Section III introduces our proposal for a global EFS model. Section IV empirically analyzes the feasibility of our proposal. Finally, Section V concludes this work.

II. Background

In this section, we describe the big data technologies used for the development of our global EFS model (Section II-A) and the feature selection problem together with the existing solutions for evolutionary big data feature selection (Section II-B).

A. Frameworks for Big Data processing

The MapReduce programming paradigm [4] was initially designed by Google in 2003 aiming at making data processing scalable to an arbitrary number of computing nodes. Even though it was designed to be a part of the most popular search engine, it rapidly became one of the most commonly used paradigm for data intensive applications due to its usefulness.

Apache Hadoop [18] is the most popular open-source implementation of MapReduce. Hadoop is composed of two main parts. The first one is the distributed file system, HDFS, which allows us to easily distribute the storage of data in a cluster of computing nodes. The second one is the implementation of MapReduce algorithm, the processing part, which allows one to take advantage of the already distributed data and process it following a data locality approach. This means that data is processed as near as possible to the node in which it is stored. Nonetheless, MapReduce is not the solution for every problem because of the overhead existing in the execution of each job. For this reason, reusing data in iterative algorithms is too costly and several frameworks aiming at improving this issue have been developed.

Among them, Apache Spark [2] is one of the most established frameworks. Spark aims at overcoming the main drawbacks of Hadoop, but it does not completely substitute Hadoop. With Spark, we only change the data processing engine, but one usually continues taking advantage of HDFS storage or uses any other kind of distributed storage system. Spark improves Hadoop MapReduce by extending the number of operations that can be performed over data. Moreover, its in-memory computation ability allows us to reuse cached distributed data in such a way that iterative computations become feasible. The main data abstraction of Spark is named as Resilient Distributed Datasets (RDDs). This distributed data structure is fault tolerant and allows the developer to easily and transparently implement a number of operations over data. Since they can be cached, they can be reused without added cost. Another key characteristic of Spark is that it follows a lazy evaluation strategy in such a way that consecutive transformations can be chained up together to make the computation faster without any intervention of the user. Recently, Spark is developing even more efficient APIs such as DataFrames and Datasets, which are always based on RDDs. These new APIs make use of the capabilities offered by Catalyst optimizer and Tungsten memory management, which allows data to be stored outside the Java Virtual Machine (JVM) making both the storage and computation more efficient.

B. Feature Selection in the Big Data context

Feature selection is a preprocessing technique used to reduce the dataset size by eliminating irrelevant or redundant features, which also usually seeks to improve the performance of the learning algorithms [13]. Hence, its main objective is to obtain a minimum set of features from the original dataset that allows one to obtain the same or even a better result with the posterior learning algorithm than considering the whole set of features. Moreover, having less features the knowledge represented in the model becomes easier to interpret and understand. Likewise, with less features, less data need to be stored and algorithms are able to run faster.

There are three clearly established categories of feature selection methods in the literature [13]:

- *Filtering methods*: The feature selection is based on measures that relate each feature or subset of features with the target variable so as to measure their degree of usefulness.
- *Embedded methods*: The learning algorithm itself has

the ability of selecting the optimal subset of features during its training phase.
- *Wrapper methods*: The features selected are optimized via a search algorithm whose fitness function is based on using a learning algorithm to evaluate the quality of each solution.

Working with big datasets, feature selection methods are not free of challenges. A proposal for feature selection in big datasets was presented in [19]. The authors developed a algorithm capable of efficiently dealing with ultra-high dimensional datasets. The main drawback of this approach is that it is designed to be executed in a single machine, which makes it not scalable to arbitrarily large datasets. To overcome this problem, in [20] a Spark-based information theoretic feature selection was developed.

With respect to evolutionary algorithms for feature selection, they have already shown their usefulness both in standard problems [14] and big data ones [15]. These algorithms fall in the Wrapper category of feature selection methods, where an evolutionary algorithm is used for the search, while the fitness evaluation is performed via a learning algorithm. As we have already stated, evolutionary algorithms for big data problems need to be carefully designed to make their execution feasible. In [15], the authors developed a first approach for EFS in big data problems. To do so, they designed a local model, where a EFS model was executed in each chunk of data (that was already distributed by HDFS). Afterwards, the features selected in all chunks where aggregated, and those being selected more times than a threshold were finally considered for the subsequent learning process. From our point of view, this local view of the problem has several disadvantages such as the fact that the solution depends on the degree of parallelism and that the data is not considered as a whole. This is why in this work we develop a first attempt on a global EFS for big datasets. Our main focus is to study its feasibility with the current frameworks for big data processing.

III. A GLOBAL EVOLUTIONARY FEATURE SELECTION WITH APACHE SPARK

In this section we describe our proposal for a global EFS for big datasets based on Apache Spark. To develop a sensible global EFS using Spark, we have made several decisions that differ from the standard EFS models. Moreover, our model also has some key differences with respect to the local approach presented in [15].

In order to explain how our distributed EFS works, we need to detail every component of the evolutionary algorithm, including its distributed implementation for big data processing. As an evolutionary model for feature selection we have selected the CHC algorithm [16]. This algorithm has been previously applied for this purpose with success both in standard classification problems and also in Big data classification [15] but with a local implementation (with all the drawbacks mentioned in previous sections). Moreover, our previous experience on evolutionary algorithms for Big Data preprocessing [21], [22] encourages us to think of CHC as good candidate for this purpose.

CHC is a binary-coded elitist genetic algorithm with some specific properties:

- *Elitist Selection*: In each generation, the best chromosomes among the current population and the offspring are selected for the new population. In the case of equal fitness, chromosomes from the current population have preference over the offspring. This kind of selection puts a lot of selective pressure into the search, since only best chromosomes survive in each generation.
- *Incest prevention*: This is a mechanism for maintaining diversity in the population. Parents are not allowed to be crossed unless they are sufficiently different. How different two parents are is computed by their Hamming distance. Two parents are only crossed if their Hamming distance divided by two is greater than a threshold d, which is commonly initialized to $d = L/4$, L being the length of the chromosome. The threshold is decreased by one when none of the offspring becomes a member of the population (that is, either because no parents are crossed or none of the offspring improves the fitness of the parents).
- *Restarting mechanism*: When the crossover threshold d is equal to 0, all the chromosomes in the population except for the best are reinitialized by taking the best chromosome as a template and randomly flipping a certain percentage of its genes. This mechanism is required due to the high selective pressure introduced by the elitist selection and it will be applied when the population becomes stagnated.
- *Half Uniform Crossover (HUX)*: This operator combined with the incest prevention mechanism aims at enforcing a high diversity, avoiding the premature convergence of the algorithm. In HUX, exactly half of the different genes between two chromosomes are exchanged. These genes are randomly selected and allow the algorithm to assure that the offspring are maximally different from the parents.

Therefore, CHC usually has a fast convergence. In our scenario, this is a desirable characteristic because each evaluation requires a distributed algorithm to be run, which is expensive and the main problem when aiming at developing a global EFS method.

Regarding the fitness function, we have considered a combination between classification performance and feature reduction. We aim at maximizing both objectives. To do so, we have considered the following fitness function:

$$\text{fitness} = \text{g-mean} + \alpha \left(1.0 - \frac{f}{F} \right), \quad (1)$$

where f is the number of selected features, F is the total number of features, α is a weight measuring the importance of feature reduction over classification performance (in this work $\alpha = 0.05$) and g-mean is the geometric mean of the True Positive Rates (TPR_i) for $i = 1, \ldots, M$ (M being the

number of classes in the problem) computed as follows.

$$\text{g-mean} = \sqrt[n]{\text{TPR}_1 \cdot \text{TPR}_2 \cdot \ldots \cdot \text{TPR}_M}. \quad (2)$$

Recall that the TPR_i stands for the percentage of correctly classified examples from class i. We are not considering the accuracy rate as a performance measure but the g-mean of the TPRs over each class to properly guide the feature selection in such a way that all the classes in the problem are recognized. This fact is important when dealing with big datasets because there are many problems where classes are imbalanced and accuracy rate is no longer meaningful in that case.

Anyway, these are the standard components of any genetic algorithm and of EFS in general. In fact, we could consider these components to run a genetic algorithm for selecting the features in different chunks of data, which can be then fused as in the case of the local approach [15]. However, in our model we want to evaluate the quality of each chromosome considering the entire dataset to obtain a global feature selection model that does not need to deal with approximations. Hence, we need to learn and classify from distributed data. Distributed learning algorithms for Big data exist and some of them are included in Apache Spark MLlib library [12]. In this work we aim to analyze whether it is feasible or not to develop a genetic algorithm using this kind of models in the feature selection evaluation procedure.

In the evaluation of the chromosome we can no longer consider k-Nearest Neighbors to measure the g-mean as it is done in the local approach. Doing so for millions of instances in a global manner is possible for evaluating one individual but not for a population throughout a number of generations. A much faster learning model is required for making global feature selection feasible. For this reason, our proposal consists of evaluating each chromosome quality (g-mean) by learning a Decision Tree from Apache Spark MLlib library. This algorithm can be fast enough if it is adequately parameterized. Thus, we need to establish its parameters to be as fast as possible. For this reason, we will set the depth of the tree as small as possible so that each evaluation becomes tractable to be repeated hundreds of times (more details are given in Section IV). Moreover, in order to make the convergence of the CHC algorithm even faster, we propose to modify HUX as it is done in instance selection methods [17]. We will introduce a probability to set a gene to 0 each time the crossover has caused it to become 1. This way, we force the search to reduce the number of features quickly, which also makes the decision tree construction, and therefore the evaluation, faster as long as the number of features selected are reduced.

In summary, we mainly consider Spark for the evaluation of each chromosome by learning a decision tree with the selected features. Then, the g-mean over the training set is considered to compute the final value of the fitness function. Notice that for the evaluation of each chromosome a distributed decision tree is learned and the instances used to learn the decision tree are classified in a distributed manner.

Algorithm 1 Global EFS based on CHC algorithm

```
Require: trainFile; PopSize, #Maps; #Windows
 1: trainRDD ← textFile(trainFile, #Maps).toLabeledPoint().cache()
 2: nFeatures = trainRDD.first().getNumFetures()
 3: d = nFeatures / 4
 4: nReInit = 0; bestFitR = 0; actualWindow = 0
 5: nFeaturesInit = nFeatures · 0.5
 6: {Initialisation}
 7: for i = 1 to PopSize do
 8:     {population(i) is a HashSet of selected features (without repeated values)}
 9:     population(i) = Randomly take nFeaturesInit indexes in the range [0,nFeatures-
        1].
10: end for
11: windowsRDDs = generateWindows(trainRDD, #Windows)
12: evaluate(population, trainRDD, windowsRDDs(actualWindow), nFeatures)
13: population.sorted { Sorted in descending order by fitness }
14: while eval < MAX_EVAL and nReInit < MAX_REINITIALISES do
15:     offspring = crossover(population)
16:     if offspring.size > 0 then
17:         evaluate(offspring, trainRDD, , windowsRDDs(actualWindow), nFeatures)
18:         offspring.sorted
19:     end if
20:     if offspring.size == 0 or offspring(0).fitness < population(0).fitness then
21:         d = d - 1
22:     else
23:         population = (offspring ++ population).sorted.take(PopSize)
24:     end if
25:     if d <= 0 then
26:         re-initialize(population) # All except the best are reinitialized
27:         evaluate(population.tail, trainRDD, , windowsRDDs(actualWindow), nFea-
            tures)
28:         population.sorted
29:         d = nFeatures / 4
30:         if population(0).fitness == bestFitR then
31:             {Without improvement from last reinitialization}
32:             nReInit += 1
33:         else {Improved after reinitialization}
34:             bestFitR = population(0).fitness
35:             nReInit = 0
36:         end if
37:     end if
38:     actualWindow = (actualWindow + 1) mod #Windows
39: end while
```

Algorithm 2 Global EFS: Parallel fitness function evaluation using Spark

```
Require: population; trainRDD; nFeatures {trainRDD can be the whole RDD or a
    window}
 1: for i = 1 to population.size do
 2:     newTrainRDD = trainRDD.map{e → selectedFeatures(e, population(i)) }
 3:     model = LearnGlobalDecisionTree(newTrainRDD)
 4:     predictions = model.predict(newTrainRDD)
 5:     gm = ComputeGmean(newTrainRDD.map(e → e.label), predictions)
 6:     population(i).fitness = gm + α · (1.0 − population(i).size / nFeatures) {with α = 0.05}
 7: end for
```

The pseudo-code for the whole CHC algorithm proposed together with the evaluation of the population are presented in Algorithm 1 and 2, respectively.

With respect to the actual implementation of our solution, we have followed our previous implementation of CHC for evolutionary undersampling [17], which was specifically designed for very large chromosomes. This way, we are able to easily cope with a large number of features in terms of the codification without added overhead in terms memory and communication. In feature selection, the most commonly employed codification is to consider a binary array where each position indicates whether the corresponding feature is selected or not. In our implementation, instead of codifying each chromosome as an array, we take advantage of HashSet structure to develop a sparse representation of the chromosome where the indexes of the selected features are stored.

This characteristic slightly affects the HUX operator as well as the computation of the Hamming distance but the final result is totally equivalent. We refer the reader to [17] for more details.

Regarding the initial population, all the chromosomes except one are randomly initialized by randomly selecting (with replacement) half of the features. Notice that the fact that we are using a HashSet will avoid the same feature to be considered twice. The last chromosome is initialized with all the features as a reference in terms of performance.

Finally, aiming at accelerating the fitness function evaluation, we have also considered a windowing scheme [21]. Windowing consists of assessing the chromosomes with a subset of the training set (called window). The whole dataset is divided into a number of disjoint windows prior to the execution of the evolutionary algorithm and the window used to evaluate the chromosomes in each iteration is sequentially changed in each iteration of the evolution process. Therefore, in our global EFS, the windowing scheme affects both the decision tree used to evaluate each chromosome (because only the examples in the window are used for its construction) and the estimated performance (because only the examples in the window are taken into account for the g-mean computation). Obviously, if only one window is considered, all the training set is used for the evaluation of every chromosome.

IV. Experimental study

In this section we present a number of experiments to evaluate the proposed approach. Section IV-A will present the details of the experimental framework. Then, Section IV-B will show the obtained results and a discussion around them. Finally, in Section IV-C, we will analyze the behavior of the EFS approach in terms of convergence.

A. Experimental set-up

In these preliminary experiments, we have selected two binary classification datasets with a large number of features to assess the correctness of the proposed global EFS approach. The first dataset, named as epsilon, is composed of 500 000 instances and 2000 numerical features. This dataset was artificially generated for the Pascal Large Scale Learning Challenge[1]. The second dataset comes from the Evolutionary Computation for Big Data and Big Learning competition[2] (we refer to it as ECBDL14). This classification dataset contains 631 features (including both numerical and categorical attributes), and it is originally comprised of approximately 32 million instances. For this study, we will consider a subset of the 25% of the instances. Moreover, this dataset is highly imbalanced, containing 98% of the instances belonging to a single class (the majority class). To handle the imbalanced problem in this study, we have applied an undersampling approach that randomly removes instances for that majority class in order to obtain a balanced number of instances of both classes. We have focused on random undersampling (RUS) for simplicity and good results shown in previous experiments [22].

[1]http://largescale.ml.tu-berlin.de/instructions/

[2]http://cruncher.ncl.ac.uk/ bdcomp/

TABLE I: Parameters used for all the algorithms involved in the experiments

Algorithm	*Parameters*
Global EFS (CHC)	Population Size = 50, Number of Evaluations = 1000, Probability of 1 to 0 in HUX = 0.25, Evaluation Measure = g-mean + $\alpha\cdot$ reduction ($\alpha = 0.05$) Partitions = [64,192] Windows [1, 5] Max. Depth for internal Decision Tree: [5, 9]
Decision Tree	Max Depth = 9, maxBins = 100 Minimum number of item-sets per leaf = 2
SVM	Iterations: 100 StepSize: 1.0 miniBatchFraction: 1.0 Regularization parameter: 0.5
Logistic Regression	StepSize: 1.0 miniBatchFraction: 1.0 Regularization parameter: 0.01

In our experiments, we follow a 5-fold stratified cross-validation scheme, meaning that we build 5 random partitions of each dataset maintaining the prior probabilities of each class. Each fold is composed of 20% of the data, is used once as test set, evaluating a model trained with the union of the 4 remaining folds. The reported results are taken as averages of the five partitions. To evaluate our model, we will consider different performance measures. As we are dealing with an imbalanced classification problem, we will use AUC and g-mean measures to assess classification performance [23]. The g-mean was previously defined in Eq. (1). The AUC (Area Under the ROC-Curve) provides a scalar value measuring how well a classifier trades off true positive (TPR) and false positive rates (FPR). A popular approximation [23] of this measure is given by:

$$AUC = \frac{1 + \text{TPR} - \text{FPR}}{2}. \quad (3)$$

We will also account for the final number of features selected as an important aspect to determine the reduction factor of the proposed EFS approach. In addition, we also measure the total runtime spent to preprocess the dataset.

Table I collects the parameters of all the algorithms involved in this experiment. For the proposed EFS algorithm, we have used standard parameters for the CHC algorithm. Similarly to [15], we have limited the number evaluations to 1000. As explained before, the proposed EFS uses a distributed Decision Tree (from Apache Spark) to evaluate the fitness function. We will investigate two key parameters regarding the fitness function: (1) the influence of the maximum depth of the created tree, and (2) the use of a windowing scheme. Finally, the configuration for the distributed execution in Spark slightly varies from one dataset to the other due to their properties (number of instances). We have considered 64 and 192 partitions for epsilon and ECBDL14, respectively.

After applying the EFS phase, we measure the quality of the resulting preprocessed datasets using three classical clas-

sification algorithms that are available in the Mllib library of Spark [12] for large-scale processing, including the Decision Tree (DT) (used in the fitness function evaluation), as well as a linear Support Vector Machine (SVM), and Logistic Regression (LR). We have used the default parameters for these algorithms in Spark, except for SVM, in which we have explored a number of regularization parameters. However, for sake of clarity, we will only report the best results that were found with a regularization parameter of 0.5 for SVM. Note that for the correct application of SVM and LR, the resulting preprocessed datasets are properly normalized.

The experiments have been carried out in a cluster composed of 14 computing nodes managed by the master node. Each one of these nodes has 2 Intel Xeon CPU E5-2620 processor, 6 cores (12 threads) per processor, 2 GHz and 64 GB of RAM. The network is Infiniband 40Gb/s. This hardware was configured to provide a maximum number of current tasks to 256. In terms of software, every node runs on Cent OS 6.5, and uses Cloudera's open-source Apache Hadoop distribution (Hadoop 2.6.0-cdh5.8.0) and Spark 2.2.1.

B. Preliminary results and discussion

To evaluate the success of the feature selection process we have first evaluated the quality of the classifiers considered without feature selection. Table II presents the results of applying the three used classifiers on both datasets without any preprocessing. This table includes the average learning time, AUC and g-mean of each model. As such, these results should serve of baseline to compare the performance of the proposed EFS algorithm in these datasets.

Tables III and IV gather the results of applying our proposal without using windowing on ECBDL14 and epsilon, respectively. Note that for ECBDL14 we have used a total of 192 partitions, and for epsilon we limited it to 64. These tables focus on the comparison of the results using a different maximum depth for the fitness function evaluation. In both tables, we report learning times of the classifiers, AUC and g-mean results as well as the average preprocessing time (i.e. runtime of our EFS algorithm) together with the average number of resulting selected features.

TABLE II: Classification performance of base classifiers without applying Feature Selection

	Classifier	Learn. Time (s)	AUC	g-mean
RUS-ECBDL14 (25%)	DT	15.6231	0.7187	0.7186
	SVM	8.6990	0.6972	0.6970
	LR	10.1379	0.7108	0.7098
Epsilon	DT	50.9893	0.6895	0.6894
	SVM	11.5918	0.5353	0.5332
	LR	12.4540	0.7922	0.7907

Observing the results for the ECBDL14 dataset, the number of features have been drastically reduced from 631 to approximately 7 and 23 features with depth 5 and 9, respectively. Hence, the depth of the tree in the evolutionary process has an influence in the number of selected features. This can be expected as more features are required to construct a deeper decision tree. The processing time is also increased and almost doubled when depth 9 is considered. In terms of performance, both AUC and g-mean are positively affected by the increase in the depth, but the improvement is low in absolute terms. Compared with the performance over the original dataset, the AUC and g-mean of DT are slightly improved, whereas SVM and LR slightly decrease its performance (by less than 0.1). In any case, this points out the fact that the DT algorithm is being used as learning algorithm in the EFS process. Hence, it should be further studied whether considering the corresponding classifier for the EFS allows us to also improve the performance in the case of SVM and LR.

TABLE III: Results of applying EFS on RUS-ECBDL14 (25%) dataset without windowing

EFS-DT Depth	Prep. Time (s)	#Features	Classifier	Learn. Time (s)	AUC	g-mean
5	2217.85	7.6	DT	3.6642	0.7207	0.7207
			SVM	0.7402	0.6877	0.6858
			LR	7.3811	0.6922	0.6887
9	4926.69	23.4	DT	4.1633	0.7229	0.7229
			SVM	0.7849	0.6863	0.6850
			LR	7.3849	0.6961	0.6932

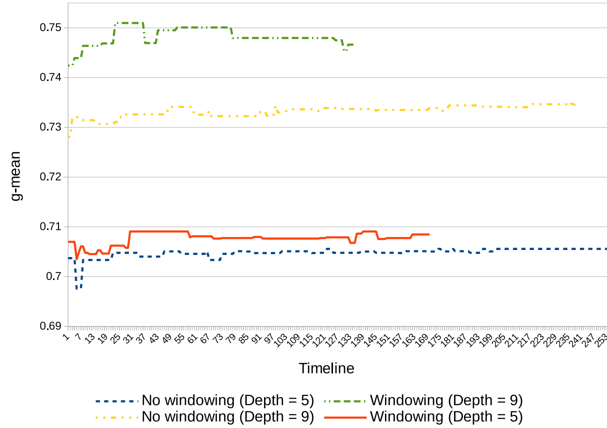
TABLE IV: Results of applying EFS on Epsilon dataset without windowing

EFS-DT Depth	Prep. Time (s)	#Features	Classifier	Learn. Time (s)	AUC	g-mean
5	4576.5	13.2	DT	3.1748	0.6829	0.6829
			SVM	0.5657	0.706	0.7060
			LR	5.7422	0.7099	0.7099
9	10901.77	46.8	DT	4.6289	0.6928	0.6927
			SVM	0.6414	0.7709	0.7709
			LR	5.7659	0.7791	0.7791

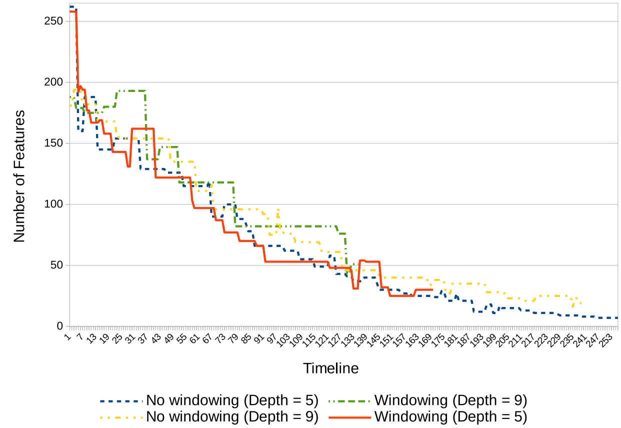
TABLE V: Results of applying EFS on RUS-ECBDL14 (25%) dataset using windowing

EFS-DT Depth	Prep. Time (s)	#Features	Classifier	Learn. Time (s)	AUC	g-mean
9	2240.60	26.6	DT	4.0205	0.7191	0.7190
			SVM	0.7606	0.6849	0.6836
			LR	7.3896	0.6945	0.6913
9	4814.36	62.4	DT	4.7427	0.7183	0.7180
			SVM	0.8276	0.6845	0.6836
			LR	7.4826	0.6968	0.6941

Looking at the epsilon dataset, the conclusions drawn with respect to the depth of the tree in terms of processing time and number of features remain the same. The processing time is more than the double and the number of features is three times bigger. However, in this case, this change results in a greater difference in terms of performance, mainly for SVM and LR, which increase both AUC and g-mean in 7 points, on average. In this case, SVM has largely improved the result over the original dataset. DT is also able to give better results with much less features, whereas LR slightly decreases its performance but using 44 times less features. Again, it remains to be studied whether considering, e.g.,



(a) G-mean convergence



(b) Num. of Features convergence

Fig. 1: Global EFS: Convergence of the algorithm with different settings in one partition of RUS-ECBDL14 (25%) dataset. These plots show the g-mean and number of selected feature for the best individual at each generation

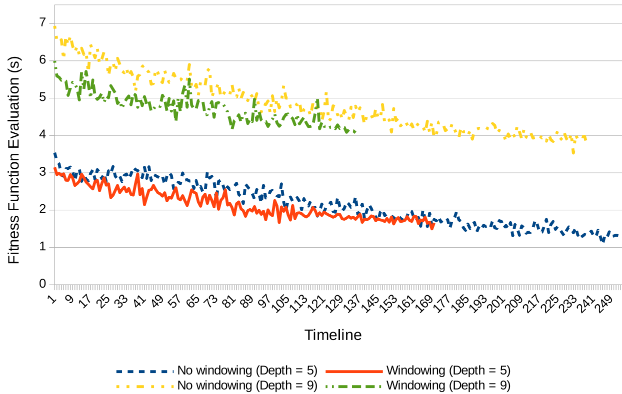


Fig. 2: Average fitness evaluation per generation

LR for the EFS procedure the final results would be better or not.

To analyze the behavior of the windowing mechanism, Table V shows the results obtained on the ECBDL14 dataset when this mechanism is activated. There are three main points that can be highlighted from this table. First, the windowing mechanism is supposed to ease the cost of fitness evaluations, so that, the total runtime should normally be smaller. However, this behavior is not really appreciated in this table. In addition, we can also observe that the average number of selected features using the windowing mechanism is significantly higher than the number obtained with the same depth of 5 for the DT. Having a higher number of selected features in the chromosomes will increase the cost of every fitness function evaluation, explaining why the total runtime is eventually very similar. In terms of precision, the results in this dataset are fairly similar to the ones obtained in Table III without windowing. We will deepen into the analysis of the windowing in the following subsection.

Although a more thorough comparison analysis is required, we can observe that a global EFS is not only possible, but it is actually able to perform well compared to the results obtain in [15] with the local approach (for the epsilon dataset that is the one is shared between the experimental studies).

C. Analysis of the behavior of global EFS

In this subsection, we aim to analyze the behavior of the proposed evolutionary model and check whether the method is properly evolving towards more promising solutions as the algorithm runs. Figure 1 depicts a convergence plot in which every time we evaluate a new bunch of chromosomes we provide the g-mean and number of selected features of the best chromosome obtained so far. This experiment has been carried out for a single partition of the ECBDL14 dataset to illustrate the convergence process. Both plots compare the algorithm using windowing or not, as well as using different maximum depths for the internal decision tree. Note that the number of the final number of generations of the CHC algorithm may differ depending on the number of chromosomes that are being evaluated (i.e. the Hamming distance may prevent some chromosomes from being tested). Using the same partition of the ECBDL14 dataset, Figure 2 plots the average fitness evaluation cost (in seconds) for each generation of the CHC algorithm.

From these figures, we can observe that the algorithm clearly evolves towards a better solution in all cases and therefore, the global selection seems to be working as expected. Note that Figure 1a shows ups and downs for all the settings because we are representing the actual g-mean value for evaluating the training data (and not the fitness value, which is progressively higher due to the elitist selection), but overall, the g-mean goes uptrend.

Nevertheless, there are differences in the behavior of the model with or without windowing, or using a different depth for the DT. When the windowing mechanism is applied, it is clear that the number of generations performed is much smaller than the number of generations that are carried out

without windowing. That shows that the chromosomes must be more different among themselves when using windowing. This might also indicate that with the windowing mechanism, the proposed approach could have used a higher number of evaluations to converge, as it does not use all the data at each evaluation. In Figure 2, we can appreciate that the overall fitness evaluation is certainly performed faster when the windowing is activated although it does not improve proportionally with the number of windows.

In Figure 1b, we can also see how the EFS tends to quickly decrease the number of selected features. This is mainly due to the parameter configuration we have followed in this preliminary study. In particular, this is related to probability in which the modified HUX operator sets a gene to 0 each time the crossover has caused it become 1.

V. Concluding remarks

In this contribution we have presented a first approach to perform global evolutionary feature selection in big datasets. To do so, we have made use of Apache Spark as a big data technology. The main advantage of the proposed scheme in comparison to existing local approaches is that it analyzes all the data as a whole, providing a more effective use of the existing data. Our preliminary results have shown that the proposed approach is able to handle very big datasets with a large number of instances and features. However, we still need to extend our experiments to get more insights. As future work, we consider different mechanisms to further accelerate the fitness function evaluation that becomes imperative to deal with big data problems from a global perspective.

References

[1] M. Minelli, M. Chambers, and A. Dhiraj, *Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses (Wiley CIO)*, 1st ed. Wiley Publishing, 2013.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and 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. 1–14.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP '03, 2003, pp. 29–43.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[5] A. Fernández, S. Río, V. López, A. Bawakid, M. del Jesus, J. Benítez, and F. Herrera, "Big data with cloud computing: An insight on the computing environment, mapreduce and programming frameworks," *WIREs Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 380–409, 2014.

[6] S. Ramírez-Gallego, A. Fernández, S. Garcá, M. Chen, and F. Herrera, "Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce," *Information Fusion*, vol. 42, pp. 51 – 61, 2018.

[7] K. Grolinger, M. Hayes, W. Higashino, A. L'Heureux, D. Allison, and M. Capretz, "Challenges for mapreduce in big data," in *Services (SERVICES), 2014 IEEE World Congress on*, June 2014, pp. 182–189.

[8] A. F. Project, "Apache flink," 2017. [Online]. Available: https://flink.apache.org/

[9] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera, "MRPR: A mapreduce solution for prototype reduction in big data classification," *Neurocomputing*, vol. 150, pp. 331–345, 2015.

[10] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera, "knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data," *Knowledge-Based Systems*, vol. 117, pp. 3 – 15, 2017, volume, Variety and Velocity in Data Science.

[11] "Chi-bd: A fuzzy rule-based classification system for big data classification problems," *Fuzzy Sets and Systems*, vol. in press, 2017.

[12] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

[13] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[14] B. de la Iglesia, "Evolutionary computation for feature selection in classification problems," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 3, no. 6, pp. 381–407, 2013.

[15] D. Peralta, S. Río, S. Ramírez, I. Triguero, J. M. Benítez, and F. Herrera, "Evolutionary feature selection for big data classification: A mapreduce approach," *Mathematical Problems in Engineering*, vol. 2015, 2015, article ID 246139.

[16] L. J. Eshelman, "The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Francisco, CA: Morgan Kaufmann, 1991, pp. 265–283.

[17] I. Triguero, M. Galar, H. Bustince, and F. Herrera, "A first attempt on global evolutionary undersampling for imbalanced big data," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2017, pp. 2054–2061.

[18] A. H. Project, "Apache hadoop," 2013. [Online]. Available: http://hadoop.apache.org/

[19] M. Tan, I. W. Tsang, and L. Wang, "Towards ultrahigh dimensional feature selection for big data," *Journal of Machine Learning Research*, vol. 15, pp. 1371–1429, 2014.

[20] S. Ramírez-Gallego, H. Mourio-Talín, D. Martínez-Rego, V. Bolón-Canedo, J. M. Benítez, A. Alonso-Betanzos, and F. Herrera, "An information theory-based feature selection framework for big data under apache spark," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. in press, pp. 1–13, 2017.

[21] I. Triguero, M. Galar, S. Vluymans, C. Cornelis, H. Bustince, F. Herrera, and Y. Saeys, "Evolutionary undersampling for imbalanced big data classification," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*, May 2015, pp. 715–722.

[22] I. Triguero, M. Galar, D. Merino, J. Maillo, H. Bustince, and F. Herrera, "Evolutionary undersampling for extremely imbalanced big data classification under apache spark," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, July 2016, pp. 640–647.

[23] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, no. 0, pp. 113 – 141, 2013.