Evolutionary Undersampling for Imbalanced Big Data Classification

I. Triguero, M. Galar, S. Vluymans, C. Cornelis, H. Bustince, F. Herrera and Y. Saeys

Abstract— **Classification techniques in the big data scenario are in high demand in a wide variety of applications. The huge increment of available data may limit the applicability of most of the standard techniques. This problem becomes even more difficult when the class distribution is skewed, the topic known as imbalanced big data classification. Evolutionary undersampling techniques have shown to be a very promising solution to deal with the class imbalance problem. However, their practical application is limited to problems with no more than tens of thousands of instances.**

In this contribution we design a parallel model to enable evolutionary undersampling methods to deal with large-scale problems. To do this, we rely on a MapReduce scheme that distributes the functioning of these kinds of algorithms in a cluster of computing elements. Moreover, we develop a windowing approach for class imbalance data in order to speed up the undersampling process without losing accuracy. In our experiments we test the capabilities of the proposed scheme with several data sets with up to 4 million instances. The results show promising scalability abilities for evolutionary undersampling within the proposed framework.

I. Introduction

Big data is a hot topic in the data mining community because of the inexorable demand of a broad number of fields such as bioinformatics, marketing, medicine, etc. It could be defined as the data whose volume, diversity and complexity require new techniques, algorithms and analyses to extract valuable hidden knowledge [1]. Standard data mining tools may experience difficulties to appropriately analyze such enormous amount of data in a reasonable time. Nevertheless, new cloud platforms and parallelization technologies [2] offer a perfect environment to address this issue.

Among the different alternatives, the MapReduce framework [3] provides a simple and robust environment to tackle large-scale data sets. Its usage is highly recommended for data mining, due to its fault-tolerant mechanism (recommendable for time-consuming tasks) and ease of use [4] as opposed to other parallelization schemes such as Message Passing Interface [5]. Several data mining techniques have been successfully implemented within this paradigm, such as [6], [7].

Data skewness is an often encountered challenge in real-world applications [8], where positive data samples, which is usually the class of interest [9], are highly outnumbered by the negative ones. Classification in the presence of class imbalance [8], [10] has gained a considerable amount of attention in the last years. One aims at improving the correct identification of positive examples, without drastically deteriorating the performance on the negative class. A wide variety of solutions has been proposed to address this problem. They can largely be divided into two groups: data sampling [11] or algorithmic modifications. Methods from the former category modify the original training data to alleviate the imbalance between classes. The latter techniques are based on existing classifiers, which are modified to make them capable of dealing with the class imbalance in the learning phase. Combinations of both approaches via ensemble learning algorithms have also been proposed [12].

Among data sampling strategies, evolutionary undersampling (EUS, [13]) aims at selecting the best subset of instances from the original training set to reduce the effect of the class imbalance. EUS not only intends to balance the training set, but also to increase the overall performance over both classes of the problem. The balancing procedure is therefore performed in a guided manner by using a genetic algorithm to search for an optimal subset of instances. This set can thereafter be used by any standard classification algorithm to build a model that should be capable of equally distinguishing both the positive and negative classes. These techniques have been demonstrated to be very powerful, but, despite their capabilities, the execution of these models in large-scale problems becomes unfeasible due to the time needed by the evolutionary search. The increasing number of instances lead to obtain an excessive chromosome size that can limit their practicality.

In this work, our aim is to take advantage of the EUS model to address class imbalance in big data problems, obtaining a final classifier able to discriminate both classes. To do so, we consider the MapReduce algorithm, which splits the data in different chunks that are processed in different computing nodes (mappers), such that EUS can be applied concurrently. Moreover, the time required by EUS is further reduced by considering a windowing scheme inside the algorithm [14], [15]. This is specifically designed for the class imbalance problem. As a result, instead of obtaining only one reduced set as in the classical approach, as many sets as there are splits are obtained, namely one in each mapper. A classifier is learned with each reduced set, such

This work was supported by the Research Projects TIN2011-28488, TIN2013-40765-P, P10-TIC-6858 and P11-TIC-7765. I. Triguero holds a BOF postdoctoral fellowship from the Ghent University. S. Vluymans holds a BOF scholarship from Ghent University.

I. Triguero, S. Vluymans, and Y. Saeys are with the Inflammation Research Center of the Flemish Institute of Biotechnology (VIB), 9052 Zwijnaarde, Belgium. E-mails: {isaac.triguero, sarah.vluymans, yvan.saeys}@irc.vib-ugent.be

S. Vluymans and C. Cornelis are with the Department of Applied Mathematics, Computer Science and Statistics, Ghent University, 9000 Gent, Belgium. E-mails: {sarah.vluymans, chris.cornelis}@ugent.be

C. Cornelis and F. Herrera are with the Department of Computer Science and Artificial Intelligence of the University of Granada, CITIC-UGR, Granada, Spain, 18071. E-mails: {chris.cornelis, herrera}@decsai.ugr.es

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

that their combination in the reduce phase of the MapReduce algorithm forms an ensemble of classifiers, which is the final model obtained. After that, a new MapReduce process will split the test data set and classify it by using the previously constructed models. In this work, we consider decision trees as classifiers, and more specifically C4.5 [16], a popular choice in imbalanced classification [8].

To test the performance of our proposal, we have considered different versions of the KddCup99 problem, where varying ratios of imbalance can be studied. Classification performance is measured by the appropriate metrics in this scenario such as the Area Under the ROC Curve and the geometric mean (g-mean). Moreover, both the building and classification times are analyzed. The effect of both levels of parallelization presented are studied considering a different number of mappers for the MapReduce algorithm and analyzing the usage of windowing compared to the original model.

The paper is structured as follows. Section II provides background information about imbalanced classification, EUS and MapReduce. Section III describes the proposal. Section IV analyzes the empirical results. Finally, Section V summarizes the conclusions.

II. Background

In this section we briefly describe the topics used in this paper. Section II-A presents the current state-of-the-art on imbalanced big data classification, whereas Section II-B recalls the EUS algorithm. Section II-C introduces the MapReduce paradigm.

A. Imbalanced classification in the Big Data context

As noted in the introduction, an imbalanced class distribution in a two-class dataset involves a large presence of negative instances compared to a small amount of positive ones. An important consideration in this framework is the measure used to evaluate the classification performance. In traditional classification applications, the performance of a classifier is commonly assessed by the classification accuracy (percentage of correctly classified examples). However, when dealing with class imbalance, this measure can lead to misleading conclusions, as the negative class gets a greater weight due to its size, combined with the fact that its examples are comparatively easier to classify. Popular alternatives are the Area Under the ROC Curve (AUC) and the g-mean. The first measure originates from the domain of signal detection, where it corresponds to the probability of correctly identifying which of two stimuli is noise (negative) and which is signal plus noise (positive). The AUC is the area under the ROC-curve [17], providing a single-number summary of how well a classifier is able to trade off its true positive (TP_{rate}) and false positive rates (FP_{rate}). A popular approximation [8] of this measure is given by

$$AUC = \frac{1 + \mathrm{TP}_{rate} - \mathrm{FP}_{rate}}{2}. \quad (1)$$

The g-mean is the geometric mean of the true positive and true negative rates (TN_{rate}) obtained by the classifier and is given by

$$\text{g-mean} = \sqrt{\mathrm{TP}_{rate} \cdot \mathrm{TN}_{rate}} \quad (2)$$

It assigns equal weights to the class-wise accuracies and therefore does not allow the performance on the negative class to be overshadowed by that on the positive class. These measures have been used in experimental studies on imbalanced classification in e.g. [8], [13].

The solutions to dealing with big data can themselves be affected by the presence of class imbalance and should therefore be further modified accordingly. Several of them, like the MapReduce scheme described in Section II-C, divide the data set into subsets to be processed separately. In such a procedure, one needs to carefully consider the construction of these subsets, as a smaller representation of the minority class may be detrimental to the general performance.

In [18], the authors developed a set of algorithms for the classification of imbalanced big data, evaluating how well traditional solutions to the class imbalance problem transfer to big data. They extended several existing data level algorithms handling class imbalance to be used in this context. In particular, they considered the random undersampling and oversampling methods [19] and the SMOTE oversampling technique [11]. These preprocessing methods were used in conjunction with the Random Forest classifier [20]. Additionally, they also developed a cost-sensitive version of this classifier able to handle imbalanced big data problems. Their experiments showed that sequential versions of methods dealing with class imbalance are indeed not equipped to be used on big data. They could not identify a single approach to be the overall best-performing one, but did observe that some methods are more suited for the big data setting than others. Alternatively, in [21], the authors develop a fuzzy rule based classification system [22] using a cost-sensitive approach to deal with class imbalance considering the MapReduce setup to handle big data.

B. Evolutionary Undersampling

EUS [13] is a preprocessing algorithm aimed at balancing the training set distribution, such that the models created from the new training set are no longer affected by the imbalanced distribution of the instances. It is an extension of evolutionary prototype selection algorithms for use in imbalanced data sets. In classical prototype selection methods [23], the objective is to reduce the training set in such a way that the storage necessity and testing times of the k-Nearest Neighbors (kNN) classifier are equally reduced, without losing performance or by even increasing it. In the class imbalance framework, the main goal of these algorithms changes, focusing on the balancing of the training set and the increasing of the performance on the minority class of the posterior classifier. Therefore, EUS obtains a balanced subset of the original data set, which is formed of those instances that allow one to reach an accurate classification in both classes of the problem. This set is evolved from randomly

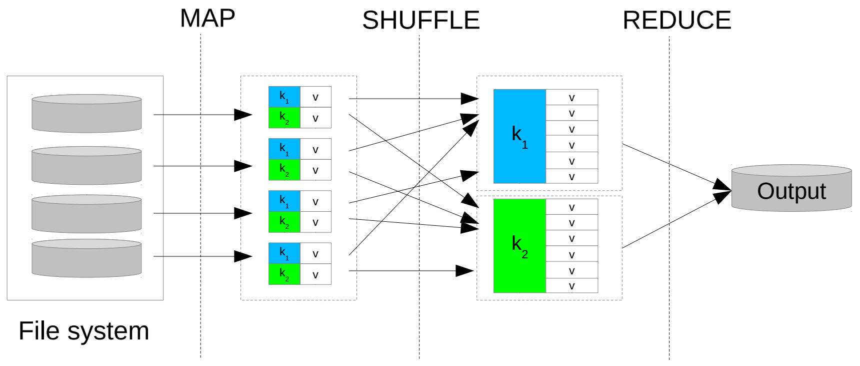

Fig. 1: The MapReduce framework

undersampled data sets until the best solution found cannot be further improved (in terms of the fitness function).

In EUS, each possible solution is codified in a binary chromosome, in which each bit represents the presence or absence of an instance in the final training set. In order to reduce the search space, only majority class instances can be considered for removal, while those of the minority class are automatically introduced in the final data set. Hence, a chromosome in EUS has the following form:

$$V = (v_{x_1}, v_{x_2}, v_{x_3}, v_{x_4}, \ldots, v_{x_{n^-}}), \tag{3}$$

where $v_{x_i} \in \{0,1\}$, indicating whether example x_i is included or not in the reduced data set and n^- is the number of negative class instances.

Chromosomes are ranked according to a fitness function that considers the balancing of the instances of both classes and the expected performance of the selected data subset at the same time. The performance is estimated by the leave-one-out technique using the 1NN classifier and is measured by the g-mean defined in Eq. (2). The complete fitness function of EUS is the following:

$$\text{fitness}_{\text{EUS}} = \begin{cases} \text{g-mean} - \left|1 - \frac{n^+}{N^-} \cdot P\right| & \text{if } N^- > 0 \\ \text{g-mean} - P & \text{if } N^- = 0, \end{cases} \tag{4}$$

where n^+ is the number of positive instances, N^- is the number of selected negative instances and P is a penalization factor that focuses on the balance between both classes. P is set to 0.2 as recommended by the authors, since it provides a good trade-off between both objectives.

As prototype selection method, the evolutionary algorithm CHC [24] is chosen because of its good balance between exploration and exploitation. CHC is an elitist genetic algorithm that uses the heterogeneous uniform cross-over (HUX) for the combination of two chromosomes, considers an incest prevention mechanism and reinitializes the population when the evolution does not progress instead of applying mutation.

C. MapReduce

MapReduce is a paradigm of parallel programming [3] developed to tackle large amounts of data over a computer cluster independently of the underlying hardware. It is mainly characterized by its great transparency for programmers, which allows one to parallelize applications in an easy and comfortable way.

This algorithm is composed of two main phases: map and reduce. Each step has key-value ($< k, v >$) pairs as input and output. The map phase collects the information from disk as a $< k, v >$ pair and generates a set of intermediate $< k, v >$ pairs. Then, all the values associated with the same intermediate key are merged as a list (shuffle step). The reduce stage takes this resulting list as its input to perform some operation and returns the final response of the system. Figure 1 shows a flowchart of this framework.

Both map and reduce processes are run in parallel. Firstly, all map functions are independently run. Meanwhile, reduce operations wait until their respective maps have ended. Then, they process different keys concurrently and independently. Note that inputs and outputs of a MapReduce job are stored in an associated distributed file system that is accessible from any computer of the used cluster.

In this contribution we will focus on the Hadoop implementation [25] of the MapReduce framework because of its performance, open source nature and distributed file system (Hadoop Distributed File System, HDFS). A Hadoop cluster is formed by a master-slave architecture, where one master node manages an arbitrary number of slave nodes. The HDFS replicates file data in multiple storage nodes that can concurrently access the data. In such a cluster, a certain percentage of these slave nodes may be out of order temporarily. For this reason, Hadoop provides a fault-tolerant mechanism: when one node fails, it automatically restarts the tasks that were running on that node on another one. Thus, Hadoop offers an advantageous environment to successfully speed up data mining techniques.

III. Evolutionary undersampling for imbalanced big data

In this section we describe the proposed MapReduce and windowing approach for EUS. We motivate our proposal in Section III-A, while an in-depth explanation of the proposed model is presented in Section III-B. Our aim is to build a model for imbalanced big data problems by extending the EUS algorithm to work within this scenario. Since we are

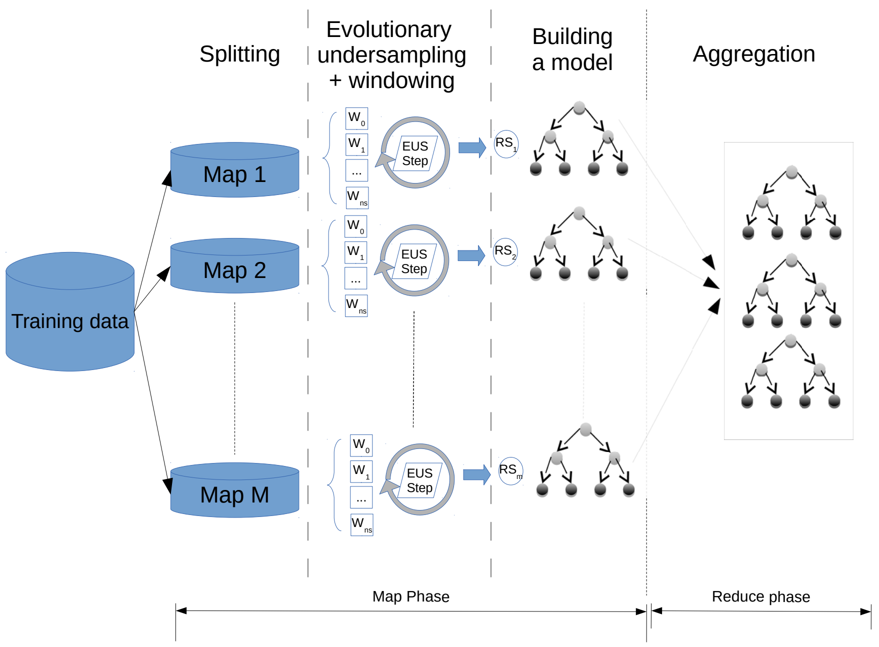

Fig. 2: Flowchart of the learning phase of EUS-based decision tree classifier ensemble.

considering a MapReduce framework, we will run several EUS processes over different chunks of the training set, resulting in as many reduced sets as the number of mappers considered. A model will be built with each one of these sets (in this case, a decision tree), which will then be aggregated in a classifier ensemble in the reduce phase. We obtain a decision tree based ensemble for imbalanced big data classification using EUS based on MapReduce. Furthermore, we will speed up each map phase by introducing a windowing scheme into EUS as a second level of parallelization. A flowchart presenting our proposal is shown in Fig. 2.

A. *Motivation*

The application of EUS in big data problems is interesting because it does not generate more data, as opposed to oversampling methods [18]. The latter can produce even bigger problems than the initial ones. As a consequence of the reduction process, the number of examples used to build the classification models becomes lower, speeding-up this process. Evidently, this fact is also achieved when random undersampling is applied, but due to its random nature it may discard important data of the majority class. In EUS on the other hand, the undersampling process is guided not only by the balancing, but also by the accuracy over both classes, which makes it more accurate [13]. However, the application of this model in big data problems is more complex and needs greater attention than random approaches due to its computational cost and memory consumption. This is why we propose a two-level parallelization model that allow us to run it over very large problems.

The computational cost of EUS is one of its drawbacks. The required execution time increases not only with the number of examples but also with the imbalance ratio (IR), that is, the number of majority class instances divided by the number of minority class ones. This is due to the fact that the greater the IR is, the greater the possibilities to form subsets of the majority class examples are (since the size depends on the number of minority class examples).

With these issues in mind, we propose a two-level parallelization scheme. MapReduce will allow us to divide the computational effort over different machines, running several EUS processes over different chunks of data. Then, for each data chunk, a windowing scheme will be applied to reduce the computational time required by EUS. To do so, each subset of the data will again be split in different strata that will be used to evaluate the 1NN algorithm in the successive iterations. Moreover, we will take the class imbalance problem into account in both phases of the method. In each map a decision tree will be built, which will thereafter be merged in a classifier ensemble in the reduce phase, obtaining the final model.

B. *A two-level parallelization model for EUS*

In order to apply EUS in big data problems, we propose the usage of two levels of parallelism. In this section, we describe each level. Section III-B.1 introduces the distributed partitioning process based on the MapReduce algorithm, which uses the windowing scheme adapted to the imbalanced framework in each partition, as explained in Section III-B.2.

1) Parallelizing with MapReduce: We rely on a MapReduce process to split the training data into several subsets that will be processed by different computing nodes, allowing us to perform EUS in each subset, which would not be possible when considering the whole data set. The benefits of using a MapReduce technique to speed up data mining models regarding other parallelization approaches are multiple, as discussed in [7]. First, each data subset is individually tackled, avoiding an excessive data exchange between nodes [26] to proceed. Second, the computational cost associated to the EUS process in each chunk may be so high that the fault-tolerant mechanism provided by MapReduce becomes essential.

```
Require: Number of split $j$
1: Constitute $TR_j$ with the instances of split $j$.
2: $RS_j$=EUS_windowing($TR_j$)
3: $M_j$=BuildModel($RS_j$)
4: return $M_j$
```

Fig. 3: Map function.

```
Require: $M_j$, {Initially $M = \emptyset$}
1: $M$ =$M \cup M_j$
2: return $M$
```

Fig. 4: Reduce function.

In order to integrate EUS into the MapReduce model, the two key operations should be developed: map and reduce, which are designed as follows. The map operation is composed of three different stages:

1) *Splitting of the data*: Let TR be the training set stored in the HDFS as a single file. In Hadoop, this file is formed by h HDFS blocks that can be accessed from any computer. The map step firstly divides TR into m disjoint subsets that correspond to the number of map tasks specified by the user as a parameter. Each map task ($Map_1, Map_2, ..., Map_m$) will create an associated TR_j, where $1 \leq j \leq m$, with the instances of each chunk in which the training set file is divided. This partitioning process is performed sequentially, so that Map_j corresponds to the jth data chunk of h/m HDFS blocks. Therefore, each map task will approximately process the same number of instances.
2) *EUS preprocessing*: When a mapper has formed its corresponding set TR_j, the EUS method is performed using TR_j as input data. This method can be either applied with or without the windowing scheme (presented in Subsection III-B.2) as a second level parallelization. As a result of this stage, a reduced and balanced set of instances (RS_j) is obtained.
3) *Building a model*: Once each mapper obtains the reduced set, the learning phase is carried out. It consists of building a decision tree. More specifically, we consider the well-known C4.5 algorithm [16] that learns a model M_j from the preprocessed set RS_j as its input training set. In this way, we avoid storing the resulting subsets to disk and applying a second MapReduce process to build the final classification model.

As a result of the map phase, we obtain m decision trees. Figure 3 summarizes the pseudo-code of the map function.

When each map finishes its processing, the results are forwarded to a single reduce task. This task is devoted to aggregate all the decision trees in a classifier ensemble. Hence, a set of trees $M = \{M_1, M_2, ..., M_m\}$ is obtained as a result and stored as a binary file in the HDFS system so that it can be then used to predict the class for new examples. Figure 4 shows the pseudo-code of the reduce function.

Once the building phase is finished, the classification step is started to estimate the class associated to each test example. Given that in big data problems the test set can also be very large, another MapReduce process solely applying the map operation is designed. This process splits the available test data into independent data blocks, whose predictions are estimated by the majority vote of the m decision trees built in the previous phase.

2) Parallelizing EUS with windowing: The MapReduce algorithm itself allows the usage of EUS with big data problems. However, the execution time required by EUS in each mapper can still be long and highly depending on the ratio of imbalance. For this reason, we design a second level parallelization based on a windowing scheme.

The use of this scheme for evolutionary models was proposed in [14] aimed at speeding up genetic-based machine learning algorithms [2]. In order to decrease the required time for the computation of the fitness function, the training set is divided into ns disjointed stratified strata ($W_1, W_2, \ldots, W_{ns}$), see Figure 2. In each iteration, only one stratum is used to evaluate the population, which changes in the subsequent iterations following a round-robin policy. This approach is different from the partitioning of the data considered in MapReduce because all the information of the training set is available, but only a part of it is used in each iteration. This technique has already been extended to other evolutionary algorithms used in data mining tasks [15].

Recall that in the case of EUS, the evaluation of a chromosome is carried out using leave-one-out validation with 1NN considering the whole training set. Hence, the computational complexity is directly influenced by the size of this set. For this reason, by only considering a part of it in each evaluation, the preprocessing time can be reduced.

However, the partitioning process must take into consideration the class imbalance problem. Dividing the training set into several disjoint windows with equal class distribution may lead to an important loss of information of the positive class. It would be even more accentuated with larger IRs. For this reason, we propose to carry out the windowing scheme introducing some variations. Minority class instances will be always used to evaluate a chromosome. However, the set of majority class instances is divided into several disjoint strata. The size of each subset of majority examples will correspond to the number of minority class instances. A similar idea was proposed in [27]. Consequently, we will obtain as many strata as the value of the IR, which also allows us to avoid setting a fixed value for the number of strata. In each iteration, the population is evaluated by one stratum from the majority class and the whole set of minority class instances. Then, the majority class stratum is changed following the round-robin policy.

Summing up, in the j^{th} mapper TR_j data set is received. This set is divided into two sets: one with positive instances TR_j^+ and the other with negative instances TR_j^-. Then, TR_j^- is divided into $ns = \lfloor IR \rfloor$ windows $(W_1^-, W_2^-, \ldots, W_{ns}^-)$. Finally, the instances of TR_j^+ are added to every window obtaining the final set $(W_1, W_2, \ldots, W_{ns})$, which are used in the windowing scheme. As a consequence, these windows are balanced and the chromosomes are evaluated with the same number of positive and negative instances.

IV. Experimental study

This section establishes the experimental setup (Section IV-A) and discusses the obtained results (Section IV-B).

A. *Experimental Framework*

To assess the quality of the proposed method for imbalanced big data, we have focused on the KDD Cup 1999 data set, available in the UCI machine learning repository [28], which consists of more than 4 million instances and 41 attributes. Since it contains multiple classes, we have formed several case studies from them corresponding to two-class imbalanced problems. In order to do so, we have taken the majority class (i.e., DOS) in comparison with the rest of the minority classes (i.e., PRB, R2L and U2R) in order to investigate the influence of different IRs. The data characteristics are summarized in Table I.

TABLE I: Data sets

Data set	#negative	#positive	IR
kddcup DOS vs. normal	3883370	972781	3.99
kddcup DOS vs. PRB	3883370	41102	94.48
kddcup DOS vs. R2L	3883370	1126	3448.82
kddcup DOS vs. U2R	3883370	52	74680.25

It is important to note that the last two data sets of this table have so few positive examples that they should be appropriately treated. Splitting the positive instances in different mappers would imply very few positive instances to be present in each one. For this reason, in these cases only the negative set of instances is split by the map phase, whereas all positive instances are read by all mappers, and are therefore present in every chunk (that is, every TR_j for $j = 1, \ldots, m$). In the other two cases, both positive and negative instances are split in the different mappers.

In our experiments we consider a 5-fold stratified cross-validation model, meaning that we construct 5 random partitions of each dataset maintaining the prior probabilities of each class. Each fold, corresponding to 20% of the data is used once as test set, evaluated on a model trained on the combination of the 4 remaining folds. The reported results are taken as averages of the five partitions. To evaluate our model, we consider the AUC and g-mean measures recalled in Section II-A. Moreover, we evaluate the time requirements of the method in two ways:

- *Building time*: we will quantify the total time in seconds spent by our method to generate the resulting learned model, including all the computations performed by the MapReduce framework.
- *Classification time*: this refers to the time needed in seconds to classify all the instances of the test set with the given learned model. To do so, we use a fixed number of 128 mappers used in the classification stage.

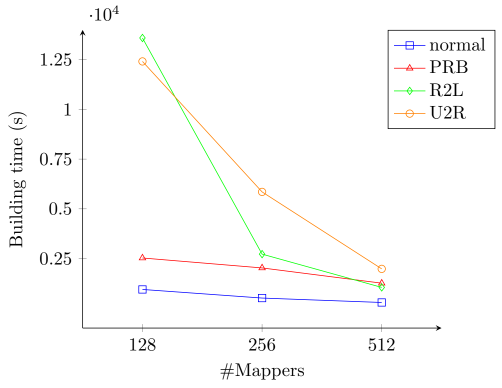

Fig. 5: Plot of the required building time against the number of mappers, without using the windowing scheme.

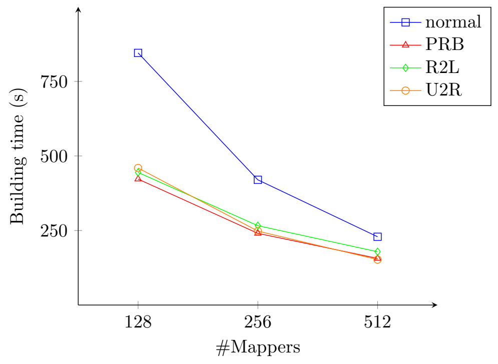

Fig. 6: Plot of the required building time against the number of mappers, using the windowing scheme.

We will also investigate how these measures are affected by modifying the number of mappers (128, 256 and 512) as well as the use of the proposed windowing scheme.

The experiments have been carried out on twelve nodes in a cluster: a master node and eleven compute nodes. Each one of these compute nodes has 2 Intel Xeon CPU E5-2620 processors, 6 cores per processor (12 threads), 2.0 GHz and 64GB of RAM. The network is Gigabit ethernet (1Gbps). In terms of software, we have used the Cloudera's open-source Apache Hadoop distribution (Hadoop 2.0.0-cdh4.4.0). A maximum of 128 map tasks are available and one reducer.

B. *Results and discussion*

This section presents and analyzes the results obtained in the experimental study.

Firstly, we only focus on the first level of parallelization, studying the results of the proposed MapReduce approach without using the windowing strategy. Table II shows the results obtained in the four cases of study considered. The

TABLE II: Results obtained without using the windowing mechanism

Data set	#Mappers	AUC	g-mean	Building Time	Classification Time
kddcup DOS vs. normal	128	0.99962397	0.99962395	942.2014	35.8988
	256	0.99924212	0.99924194	509.8122	38.4968
	512	0.99904700	0.99904674	287.2052	52.7572
kddcup DOS vs. PRB	128	0.99942829	0.99942822	2525.5332	33.4956
	256	0.99808006	0.99807901	2025.4140	41.9696
	512	0.99595677	0.99595641	1258.4924	48.9682
kddcup DOS vs. R2L	128	0.99817501	0.99817073	13595.0602	32.0616
	256	0.99817501	0.99817073	2720.0972	35.9038
	512	0.99817501	0.99817073	1045.7074	46.0528
kddcup DOS vs. U2R	128	0.97429702	0.97393535	12414.7948	31.2804
	256	0.98306267	0.98280252	5850.2702	35.2638
	512	0.98365571	0.98339482	1978.1212	46.0796

TABLE III: Results obtained using the windowing mechanism

Data set	Mappers	AUC	g-mean	Building Time	Classification Time
kddcup DOS vs. normal	128	0.99986345	0.99986345	845.5972	36.9734
	256	0.99979807	0.99979806	419.9624	31.3188
	512	0.99906136	0.99906110	228.9790	52.6386
kddcup DOS vs. PRB	128	0.99941760	0.99941754	422.4786	34.2640
	256	0.99778456	0.99778390	240.4662	36.7934
	512	0.99513122	0.99513099	156.4354	48.4240
kddcup DOS vs. R2L	128	0.99817501	0.99817073	444.7252	31.7255
	256	0.99817501	0.99817073	266.2424	36.1147
	512	0.99817501	0.99817073	178.8536	42.0057
kddcup DOS vs. U2R	128	0.98750466	0.98728379	459.6002	31.8436
	256	0.97617662	0.97583158	248.1038	35.5862
	512	0.97656950	0.97624880	152.3752	46.6194

averaged AUC, g-mean, building and test classification runtimes are presented, depending on the number of mappers used (#Mappers). Figure 5 plots the building time required according to the number of maps.

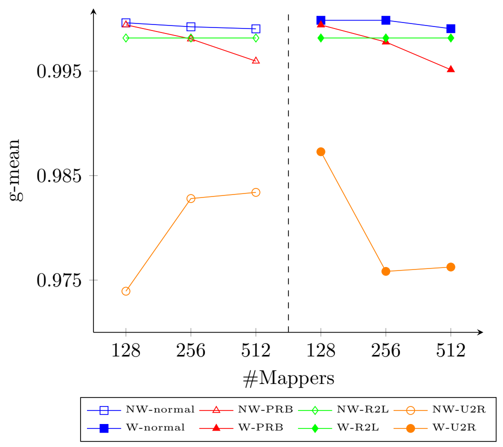

Fig. 7: The g-mean values for all datasets are plotted against the number of mappers. On the left, the results without using the windowing scheme (NW) are displayed. On the right, the results of the windowing (W) are presented.

From this table and figure we can highlight several factors:

- Since within the MapReduce framework the EUS algorithm does not have all the information of the problem tackled, it is expected that the performance (AUC and g-mean) will decrease as the number of available instances is reduced (i.e., the number of mappers is incremented). However, in this table, we can observe that this fact is only fulfilled for the first two data sets. For DOS vs. R2L and DOS vs. U2R we find a different behavior. For these two problems, we included all positive instances in every chunk, and hence more mappers could result in an improved performance because more models are used to classify new examples.
- Analyzing the building time, a great reduction is shown when the number of mappers is increased in all the problems. The time spent by the EUS algorithm is influenced by the IR because it reduces the negative class instances until its size is equal to the positive one. For that reason, in Fig. 5 we can observe that the building time on data sets with higher imbalance ratio is slower (DOS vs. R2L and DOS vs. U2R), although the others (DOS vs. normal and DOS vs. PRB) have a greater number of instances.
- The classification time tends to be a bit higher as the

number of mappers is incremented, which is due to the fact that more partial models are being aggregated.

Table III and Fig. 6 present the results obtained taking into account the proposed windowing model. Fig. 7 compares the g-mean results using either the windowing model or without using it. According to these results, the following points can be stressed:

- Looking at Fig. 7, we can observe that the use of the designed windowing scheme has resulted in a comparable performance, with respect to not using it, in most of the problems. As in the previous case, the increment in the number of mappers also implies a small drop of performance, since less instances are considered in each mapper.
- In terms of runtime, the windowing scheme has addressed the negative influence of the IR over the EUS model. Since the number of windows depends on the IR, the greater the IR is, the more effective the windowing scheme becomes in reducing the running time. As a consequence, the building times of the difference problems are more similar than without using the windowing scheme.
- Both the performance obtained and the running time reduction show the good synergy between the MapReduce and windowing approaches, which allow one to apply EUS model in big data problems with promising results.

V. Concluding remarks

In this contribution a parallelization scheme for EUS models is proposed. It is based on a two-stage MapReduce approach that first learns a decision tree in each map after performing EUS preprocessing and then classifies the test set. The EUS step is further accelerated by adding a windowing scheme adapted to the imbalanced scenario. The application of this parallelization strategy enables EUS to be applied on data sets of almost arbitrary size. Our experimental study has shown the benefits of applying the windowing scheme in combination with the MapReduce process, resulting in a very big reduction of the building time while maintaining a comparable performance. As future work, we plan more extensive experiments on alternative imbalanced big data sets. Additionally, it will be interesting to replace the EUS step by other preprocessing mechanisms such as hybrid oversampling/undersampling approaches.

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] J. Bacardit and X. Llorà, "Large-scale data mining using genetics-based machine learning," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 3, no. 1, pp. 37–61, 2013.

[3] J. Dean and S. Ghemawat, "Map reduce: A flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[4] 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.

[5] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[6] A. Srinivasan, T. Faruquie, and S. Joshi, "Data and task parallelism in ILP using mapreduce," *Machine Learning*, vol. 86, no. 1, pp. 141–168, 2012.

[7] 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.

[8] 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.

[9] G. Weiss, "Mining with rare cases," in *Data Mining and Knowledge Discovery Handbook*. Springer, 2005, pp. 765–776.

[10] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 4, pp. 463–484, 2012.

[11] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *arXiv preprint arXiv:1106.1813*, 2011.

[12] M. Galar, A. Fernández, E. Barrenechea, and F. Herrera, "Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling," *Pattern Recognition*, vol. 46, no. 12, pp. 3460–3471, 2013.

[13] S. García and F. Herrera, "Evolutionary under-sampling for classification with imbalanced data sets: Proposals and taxonomy," *Evolutionary Computation*, vol. 17, no. 3, pp. 275–306, 2009.

[14] J. Bacardit, D. E. Goldberg, M. V. Butz, X. Llorà, and J. M. Garrell, "Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. LNCS, vol. 3242, 2004, pp. 1021–1031.

[15] I. Triguero, D. Peralta, J. Bacardit, S. Garcia, and F. Herrera, "A combined mapreduce-windowing two-level parallel scheme for evolutionary prototype generation," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*, July 2014, pp. 3036–3043.

[16] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993.

[17] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[18] S. del Río, V. López, J. Benítez, and F. Herrera, "On the use of mapreduce for imbalanced big data using random forest," *Information Sciences*, vol. 285, pp. 112–137, 2014.

[19] G. Batista, R. Prati, and M. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[20] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[21] V. López, S. del Río, J. Benítez, and F. Herrera, "Cost-sensitive linguistic fuzzy rule based classification systems under the mapreduce framework for imbalanced big data," *Fuzzy Sets and Systems*, vol. 258, pp. 5–38, 2014.

[22] H. Ishibuchi, T. Nakashima, and M. Nii, *Classification and modeling with linguistic information granules*. Springer, 2005.

[23] S. García, J. Derrac, J. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 417–435, 2012.

[24] 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.

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

[26] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Chang, "Parallel spectral clustering in distributed systems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 3, pp. 568–586, 2011.

[27] J. R. Cano, S. García, and F. Herrera, "Subgroup discover in large size data sets preprocessed using stratified instance selection for increasing the presence of minority classes," *Pattern Recognition Letters*, vol. 29, no. 16, pp. 2156–2164, 2008.

[28] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml