Information Sciences 285 (2014) 112–137

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

On the use of MapReduce for imbalanced big data using Random Forest

Sara del Río[*], Victoria López, José Manuel Benítez, Francisco Herrera

Dept. of Computer Science and Artificial Intelligence, CITIC-UGR (Research Center on Information and Communications Technology), University of Granada, Granada, Spain

A R T I C L E   I N F O

Article history:
Received 28 March 2013
Received in revised form 21 February 2014
Accepted 11 March 2014
Available online 19 March 2014

Keywords:
Big data
MapReduce
Imbalanced dataset
Sampling
Cost-sensitive learning
Random Forest

A B S T R A C T

In this age, big data applications are increasingly becoming the main focus of attention because of the enormous increment of data generation and storage that has taken place in the last years. This situation becomes a challenge when huge amounts of data are processed to extract knowledge because the data mining techniques are not adapted to the new space and time requirements. Furthermore, real-world data applications usually present a class distribution where the samples that belong to one class, which is precisely the main interest, are hugely outnumbered by the samples of the other classes. This circumstance, known as the class imbalance problem, complicates the learning process as the standard learning techniques do not correctly address this situation.

In this work, we analyse the performance of several techniques used to deal with imbalanced datasets in the big data scenario using the Random Forest classifier. Specifically, oversampling, undersampling and cost-sensitive learning have been adapted to big data using MapReduce so that these techniques are able to manage datasets as large as needed providing the necessary support to correctly identify the underrepresented class. The Random Forest classifier provides a solid basis for the comparison because of its performance, robustness and versatility.

An experimental study is carried out to evaluate the performance of the diverse algorithms considered. The results obtained show that there is not an approach to imbalanced big data classification that outperforms the others for all the data considered when using Random Forest. Moreover, even for the same type of problem, the best performing method is dependent on the number of mappers selected to run the experiments. In most of the cases, when the number of splits is increased, an improvement in the running times can be observed, however, this progress in times is obtained at the expense of a slight drop in the accuracy performance obtained. This decrement in the performance is related to the lack of density problem, which is evaluated in this work from the imbalanced data point of view, as this issue degrades the performance of classifiers in the imbalanced scenario more severely than in standard learning.

[*] Corresponding author. Tel.: +34 958 240598; fax: +34 958 243317.
E-mail addresses: srio@decsai.ugr.es (S. del Río), vlopez@decsai.ugr.es (V. López), J.M.Benitez@decsai.ugr.es (J.M. Benítez), herrera@decsai.ugr.es (F. Herrera).

http://dx.doi.org/10.1016/j.ins.2014.03.043
0020-0255/© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Many real-world areas such as telecommunications, health care, pharmaceutical or financial businesses generate massive amounts of data. Gaining critical business insights by querying and analysing such massive amounts of data is becoming the need of the hour [24] and has become a challenge to the standard data mining approaches.

Traditionally, data warehouses have been used to manage large amounts of data. However, for the management of massive data that grow day after day, these are not able to provide reasonable response times. Big data can be defined as data that exceeds the processing capacity of conventional systems [36]. The traditional techniques for machine learning and data mining cannot cope with such amounts of data when there is a growing need to run machine learning and data mining algorithms on very large datasets [31].

Furthermore, real-world applications also present classes which are represented by a negligible number of instances with respect to other classes that are considered. This situation is known as classification with imbalanced datasets and has gained lots of attention in the last years [27,33,50]. Moreover, the classes which are underrepresented are usually the cases under consideration on the study; therefore, its correct identification becomes even more important.

Classifying imbalanced datasets is not usually a trivial task. Standard learning techniques are often guided by global search measures which do not take into consideration this contingency. In this manner, it is necessary to consider the characteristics of the problem and then solve it accordingly.

A large number of approaches have been proposed to address classification with imbalanced datasets. These approaches fall largely in two groups: data sampling solutions [6,12], which modify the original training set, and algorithmic modifications [32] which modify existing algorithms trying to benefit the classification of the minority class. Cost-sensitive solutions [18,60] combine the two previous options trying to minimise the misclassification costs, which are higher for the instances of the minority class.

The techniques used to deal with big data are focused on obtaining fast, scalable and parallel implementations. To achieve this goal, one of the more popular solutions nowadays is to follow a MapReduce procedure [37], dividing the original set into subsets which are more easily addressed, and then combining the partial solutions that are obtained. However, this data distribution operation may have a special negative effect in imbalanced datasets. Among the difficulties that can degrade the performance when classifying imbalanced datasets we can encounter the problem of small sample size [45] which is amplified as the original data is distributed in different machines. Furthermore, we may also confront the dataset shift problem [38] which occurs when the training and test set partitions are quite different between them.

In this work, we present an analysis of several techniques to deal with imbalanced big data. Specifically, the techniques evaluated in this study are techniques that have been proved as useful for imbalanced datasets and which we have adapted for big data following a MapReduce scheme. Concretely, we analyse the performance of several data sampling techniques such as random oversampling [6], random undersampling [6], the "Synthetic Minority Oversampling TEchnique" (SMOTE) algorithm [12] and cost-sensitive learning [14,34]. For each one of them, we will present an approach based on the MapReduce framework. These approaches are examined considering the effectiveness in the correct classification of the instances of each class and the runtime spent in the building of the model and classification of samples. The presence of small sample size problems [54] which are related to the lack of density is also analysed.

In order to do so, we use as base classifier the Random Forest algorithm [9], which is a well-known decision tree ensemble famous for its robustness and good performance. Ensembles have demonstrated a good behavior when confronted with imbalanced datasets [20] and therefore, using one of them as basis of the comparison should not bias the results neglecting the minority class.

For the experimental study, we will focus on three imbalanced big data problems that have been subdivided in several binary cases of study with different degrees of imbalance and size. The experiments performed provide, at first, a study about the limitations of the sequential versions when the size of the data available is increased, and then, an analysis of the performance obtained by the different methods together with the runtime needed to get each result. The effectiveness in classification for the approaches considered will be evaluated using two measures that are able to efficaciously rate the success in imbalanced classification: the Geometric Mean of the true rates [5] and the β-f-measure [27], so that the results are not biased by a specific metric. Additionally, we also provide some insight into the small sample size problem related to the minority class associated to the splitting of data that is performed in the MapReduce approaches and its relationship to the lack of density [28].

The paper is organized as follows. First, in Section 2, an introduction to classification with big data and imbalanced datasets is presented. Then, in Section 3, the Random Forest algorithm is presented together with some of its versions that are able to deal with big data and imbalanced datasets separately. Later, Section 4 provides the description of the imbalanced approaches that we have adapted for big data using the Random Forest algorithm. Next, Section 5 presents the experimental study performed, detailing information about the experiments configuration, the results obtained and providing a thorough analysis of them. Section 6 summarizes and concludes the work. Finally, we must point out that the paper has an associated website http://sci2s.ugr.es/rf_big_imb which collects additional information like detailed experimental results.

2. Classification with big data and imbalanced datasets

In this section we present the context in which this work is included. We first provide an introduction to big data and the MapReduce framework (Section 2.1) and and then, the problem of classification with imbalanced datasets is described (Section 2.2).

2.1. Big data and the MapReduce framework

The constantly increasing speed of data generation is heading us to handling large amounts of data using the traditional machine learning algorithms. This data is obtained from diversified sources and domains, building massive datasets that need to be inspected within reasonable response times. For instance, Facebook and Twitter are examples of applications that generate such amounts of data. In 2010, Facebook had 21 Peta Bytes of internal warehouse data with 12 TB new data added every day and 800 TB compressed data scanned daily [51].

This situation tends to be a problem as the knowledge extraction algorithms are not adapted to deal with such vast amounts of data. Nowadays, it is necessary to process those huge data collections quickly and efficiently in order to make an appropriate use of the available resources as well as to scale the traditional machine learning algorithms.

Big data is the popular term used to describe amounts of data so large and complex that it becomes difficult to process or analyse using traditional techniques. As far back as 2001, Gartner analyst Doug Laney introduced the 3Vs concept defining big data as high volume, velocity and variety information that require new forms of data processing [8]:

- *Volume:* Refers to the huge amount of data that are being produced daily and that have gone from MB and GB to PB.
- *Velocity:* Refers to how fast the data is coming in and how fast it needs to be analysed.
- *Variety:* Deals with the many number of types of data, both structured and unstructured.

More recently, additional big data Vs are getting attention and are added to the model like veracity, validity, volatility, variability or value.

In 2004, the MapReduce programming framework [15] was proposed. It is a platform designed for processing massive amounts of data in an extremely parallel manner, while providing an environment to easily develop scalable and fault tolerant applications. The MapReduce programming model abstracts the calculation process in two phases: **Map** and **Reduce**.

In the Map phase, the master node splits the input dataset into independent sub-problems and distributes them to worker nodes. Then, the worker nodes process in a parallel way the smaller problems and pass the answer back to its master node. Finally, in the Reduce phase, the master node takes the answers to all the sub-problems and combines them in a way to form the output. The users in this paradigm only have to define what should be computed in the Map and Reduce functions while the system automatically distributes the data processing over a highly distributed cluster of machines.

In the MapReduce model all the computation is organized around ⟨key, value⟩ pairs. In the first stage, the Map function, takes a single ⟨key, value⟩ pair as input and produces a list of intermediate ⟨key, value⟩ pairs as output. It could be represented as:

$$map(key1, value1) \longrightarrow list(key2, value2) \tag{1}$$

Then, the system merges and groups by keys these intermediate pairs and passes them to the Reduce function. Finally, the Reduce function, takes a key and an associated value list as input and generates a new list of values as output, which can be represented as follows:

$$reduce(key2, list(value2)) \longrightarrow (key2, value3) \tag{2}$$

Fig. 1 shows the data flow of a MapReduce operation.

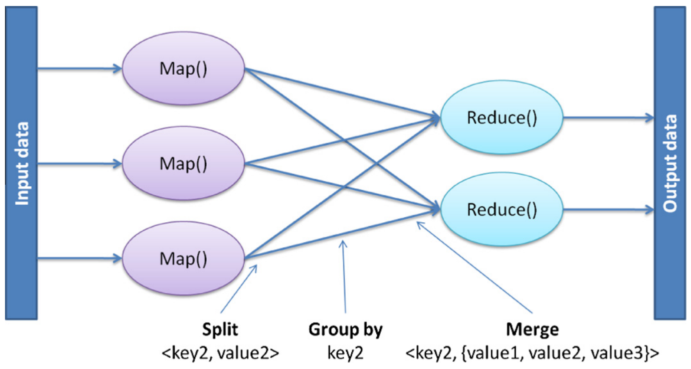


Fig. 1. Data flow overview of MapReduce.

Apache Hadoop [2,57], is the most popular open source implementation of MapReduce. Mahout [3,43] is a machine learning library that runs on top of a Hadoop system. It features a set of algorithms for clustering, recommendation systems and classification problems. Like Hadoop, Mahout is an open source project. Mahout contains various implementations of classification models such as Logistic Regression, Bayesian models, Support Vector Machines, and Random Forest among others.

More recently, other projects are emerging to deal big data. Some of them are Spark [47], a cluster computing system that aims to make data analytics fast; Storm [48], a distributed and fault-tolerant real-time computation system that makes it easy to reliably process unbounded streams of data; Dremel [35], a scalable, interactive ad hoc query system for analysis of read-only nested data; and Apache Drill [1], a distributed system that supports data-intensive distributed applications for interactive analysis of large-scale datasets.

2.2. Classification with imbalanced datasets

In this section we present the related aspects to classification with imbalanced datasets. First, an introduction to the problem is given in Section 2.2.1. Then, the techniques to address imbalanced classification are presented in Section 2.2.2. Finally, Section 2.2.3 describes specific evaluation metrics for imbalanced datasets.

2.2.1. The problem of imbalanced datasets

The problem of classification with imbalanced datasets occurs when there is a notable difference between the number of samples belonging to different classes [27,50], having one class with an abundant number of instances (known as the majority or negative class) and the other class with few number of samples (known as the minority or positive class). This problem has gained much importance in the last years because of its presence in lots of real-world applications such as medical diagnosis [39], software defects detection [44], finances [61], drug discovery [41] or bioinformatics [58]. In these problems, the class with the lower number of instances is usually the most important from the learning point of view and it entails high costs when its identification is not properly performed [18,60].

The difference in the class distribution poses a major challenge to standard machine learning algorithms because the search process that is embedded in most of the techniques is guided by a global search measure that does not consider the differences in the number of instances that belong to each class. In this manner, the instances of the minority class are usually neglected during the model construction as its identification is performed using specific learning rules. These specific rules are usually ignored in favor of more general rules, which are precisely the rules that cover the majority class.

Traditionally, the imbalance ratio (IR) [42], which is defined as the ratio of the number of instances from the majority class and the minority class, is the main identification to determine the difficulty level associated to a specific dataset. However, there are some additional factors that negatively influence the classification with imbalanced datasets, the data intrinsic characteristics [33]. These data intrinsic characteristics include the presence of small disjuncts [55,56], the lack of density or small sample size associated to the problem [54], the overlapping between the classes [16,23], the presence of noisy [46] and borderline [40] samples, and the differences in the data distribution for the training and test data, known as dataset shift [38].

2.2.2. Addressing imbalanced datasets

Numerous techniques are used to deal with imbalanced datasets in classification [19,33,34]. These techniques are usually divided into several groups: data level approaches [6,12,21], which modify the original training set to obtain a nearly balanced class distribution that can be used with standard learning algorithms, and algorithm level approaches [32,59] that modify current existing algorithms modifying inner operations to introduce mechanisms that are able to deal with the imbalance. Cost-sensitive learning solutions combine the ideas from both the data level and algorithm level approaches adopting higher misclassification costs for instances that belong to the minority class and minimizing the overall cost [18,60]. Ensemble methods are also frequently adapted to imbalanced domains where they have demonstrated a good performance [20].

The data level approaches are divided usually in several groups: oversampling methods, undersampling methods and hybrid methods. Oversampling methods [11,12,26] aim to balance the class distribution adding to the new dataset instances from the minority class; undersampling methods [22,30,52] try to adjust the class proportion deleting instances from the majority class; and hybrid methods [6] combine the two previous approaches, usually starting with an oversampling step that creates new samples for the minority class and then applying an undersampling step that can delete samples from the majority class or from both classes to ease the creation of accurate classification models.

The easiest oversampling method, random oversampling (ROS) [6], randomly replicates minority class instances from the original dataset until the number of instances from the minority and majority classes is the same. The simplest undersampling method, random undersampling (RUS) [6], randomly deletes majority class examples from the original dataset until the balance with the minority class is achieved.

The SMOTE algorithm [12] is an oversampling method that adds synthetic minority class examples to the original dataset until the class distribution becomes balanced. In order to do so, the SMOTE algorithm generates the synthetic minority class examples using the original minority class examples in the following way: the SMOTE algorithm searches the k nearest neighbors of the minority class sample that is going to be used as base for the new synthetic sample. Then, in the segment that unites the minority class sample with one or all of its neighbors, a synthetic sample is randomly taken and is added to the new oversampled dataset. This process is depicted in Fig. 2, where x_i is the selected minority class instance, x_{i1} to x_{i4} are

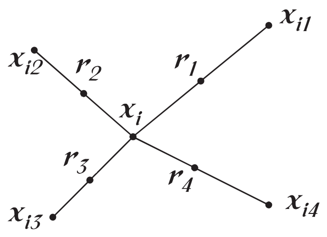


Fig. 2. An illustration of how to create the synthetic data points in the SMOTE algorithm.

some of its k nearest neighbors and r_1 to r_4 are the synthetic data points created by the randomized interpolation that will be added to the new training set.

Cost-sensitive learning methods consider the cost of a misclassification with respect to the classes available in the problem [17,60]. The misclassification costs ($C(i,j)$ is the penalty of classifying examples of the class i as class j) are usually represented in a cost matrix.

The obtaining of the cost matrix is a key issue when using cost-sensitive learning and should be provided by domain experts, however, when it is not available an estimation can be used [49,50]. When we are facing imbalanced datasets, it is much more important to correctly identify the positive class samples than the negative class ones, and therefore, in imbalanced datasets we find that the misclassification cost of a positive class sample is higher than the misclassification cost of a negative class sample $C(+,-) > C(-,+)$. Furthermore, it is usually assumed that the correct classification of the examples in imbalanced datasets should entail no penalization costs $C(+,+) = C(-,-) = 0$.

The usage of these costs in methods for imbalanced learning can be done in several ways. Usually, these costs are added to specific parts of a specific algorithm giving more importance to the instances of the minority class following a weighted scheme. In this case, the resulting model is only using costs at a data level and can be considered equivalent to preprocessing methods. However, these costs can be also used to modify specific operations within the algorithms in a manner that differs from the direct weighted scheme and changing the inner way of working of the methods. In this situation, we encounter that cost-sensitive learning is used as an algorithm level approach. That is why cost-sensitive learning is considered a combination of the data level and algorithm level solutions for classification with imbalanced datasets.

2.2.3. Evaluation in imbalanced domains

The measures of the quality of classification are built from a confusion matrix (shown in Table 1) which organizes the samples of each class according to their correct or incorrect identification.

The most frequently used empirical measure, accuracy, is not able to discriminate the correct classification of instances belonging to different classes and their importance towards the final performance, neither the misclassification cost is included. For this reason, in this work we will utilize two metrics that are widely known for classification with imbalanced datasets.

The Geometric Mean (GM) [5,29] is one measure that is able to avoid the problems related to the traditional accuracy metric and is defined as:

$$\mathrm{GM} = \sqrt{sensitivity \cdot specificity} \tag{3}$$

where $sensitivity = \frac{TP}{TP+FN}$ and $specificity = \frac{TN}{FP+TN}$. This metric attempts to maximize the accuracy of each one of the two classes with a good balance. It is a performance metric that links both objectives.

Another popular metric that is used in an imbalanced class scenario is the β-f-measure (β-f-m) [27], which is defined as follows:

$$\beta\text{-f-measure} = \frac{(1+\beta^2)(positive\_predictive\_value \cdot sensitivity)}{(\beta^2 \cdot positive\_predictive\_value) + sensitivity} \tag{4}$$

where $positive\_predictive\_value = \frac{TP}{TP+FP}$. The *positive_predictive_value* and *sensitivity* values are usually known as *precision* and *recall* respectively. The f-measure metric comes from information retrieval domains and in these problems the β parameter is equal to 1 giving the same importance to the *positive_predictive_value* and the *sensitivity*.

However, giving β a value of 1 is not suitable in an imbalanced scenario when we have this class imbalance in both the training and test datasets [7]. In this manner, larger values of β need to be used as we are more interested in improving the *sensitivity* than in increasing the *positive_predictive_value*. In this article, we follow the estimation given in [13], with $\beta = \frac{C(+|-)}{C(-|+)}$ where $C(+ \mid -)$ is the misclassification cost of a positive instance as negative and $C(- \mid +)$ is the misclassification cost of the contrary case, classifying a negative instance as positive.

Table 1
Confusion matrix for a two-class problem.

	Positive prediction	Negative prediction
Positive class	True Positive (TP)	False Negative (FN)
Negative class	False Positive (FP)	True Negative (TN)

3. Random Forest

In this section, we present several previous results about Random Forest. We briefly describe the original Random Forest algorithm (Section 3.1) and and two of its variations that were proposed to solve separately classification with big data (Section 3.2) and and the challenge of classification with imbalanced datasets (Section 3.3).

3.1. Random Forest for classification

Random Forest (RF) [9] is a well-known decision tree ensemble that is commonly used in classification. RF popularity comes from its good performance in relation with other classification algorithms [53].

As a decision tree ensemble, RF needs to build several different decision trees. In order to do so, each tree is built considering a bootstrap sample set of the original training data, that is, obtaining a new set sampling with replacement instances from the original set until we get the size of that original training data. Using one of the bootstrap sample sets of the training data, a *random tree* is created.

Each *random tree* follows a traditional top-down induction procedure with several modifications performed to favor the diversity of the ensemble. At each step, when the *best attribute* is selected, only a small subset of attributes from the dataset is considered, $m \ll M$, where m is the number of attributes selected for a node building decision and M is the total number of attributes of the dataset. Considering only the subset of attributes selected, then the *best attribute* is computed as in CART [10].

Each tree is built to its maximum depth and no pruning procedure is applied after the tree has been fully built. The predicted class for a sample is computed by aggregating the predictions of the ensemble of decision trees through majority voting.

3.2. Random Forest for classification with big data

To deal with big data experiments the original RF algorithm needs to be modified so it can effectively process all the data available. The Mahout Partial implementation (RF-BigData) [25] is an algorithm that builds multiple trees for different portions of the data.

This algorithm is divided into two different phases: the first phase is devoted to the creation of the model and the second phase is dedicated to the estimation of the classes associated with the dataset using the previous learned model.

In the first phase, the Random Forest is built from the original training set following a MapReduce procedure. This process is illustrated in Fig. 3 and consists of three steps: **Initial**, **Map** and **Final**. The **Initial** step performs a segmentation of the training dataset into independent data blocks; then, these blocks are replicated and transferred between the different processing nodes. Next, in the **Map** step, each Map task builds a subset of the forest (several random trees of the forest) with the data block of its partition and generates a file containing the built trees. Finally, the **Final** step parses the output files generated by the all mappers to extract the trees. The collection of all trees forms the forest.

When the building of the forest is finished, the classification phase is initiated to estimate the class associated to a data sample set. This process is illustrated in Fig. 4 and consists of three steps: **Initial**, **Map** and **Final**. The **Initial** step performs a segmentation of the available data sample set (it may be a large training or test set) into independent data blocks; replicates and transfers them to other machines to be finally processed independently by each map task in parallel. Next, in the **Map** step, each mapper estimates the class for the examples available in it using a majority vote of the predicted class by the trees in the RF model built in the previous phase. Finally, in the **Final** step, the predictions generated by each mapper are concatenated to form the final predictions file.

Algorithms 1 and 2 show the pseudocode of the Map function of the MapReduce job for the building of the model phase. Algorithm 1 is devoted to obtain all the instances in a mapper's partition in a ⟨key, value⟩ pair structure. When the previous process is finished, Algorithm 2 is called to build a subset of the forest (some random trees) using only instances previously obtained.

Algorithm 1. Map phase for the RF-BigData algorithm for the building of the model phase MAP (key, value):

```
Input: ⟨key, value⟩ pair, where key is the offset in bytes and value is the content of an instance.
Output: ⟨key′, value′⟩ pair, where key' indicates both the tree id and the data partition id used to grow the tree and
    value' contains a grow tree and its predictions.
1: $instance \leftarrow INSTANCE\_REPRESENTATION(value)$ {instances will contain all instances in this mapper's split}
2: $instances \leftarrow instances.add(instance)$
```

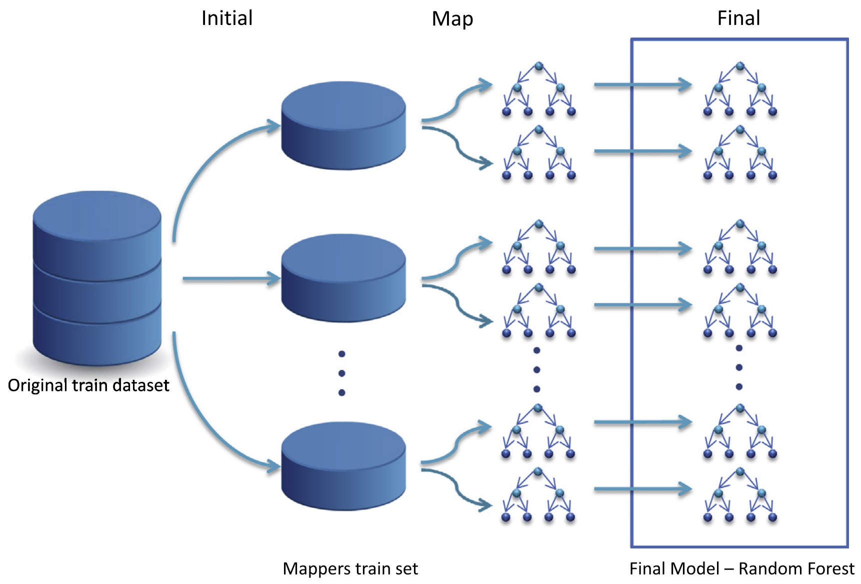


Fig. 3. A flowchart of how the building of the Random Forest is organized in Mahout.

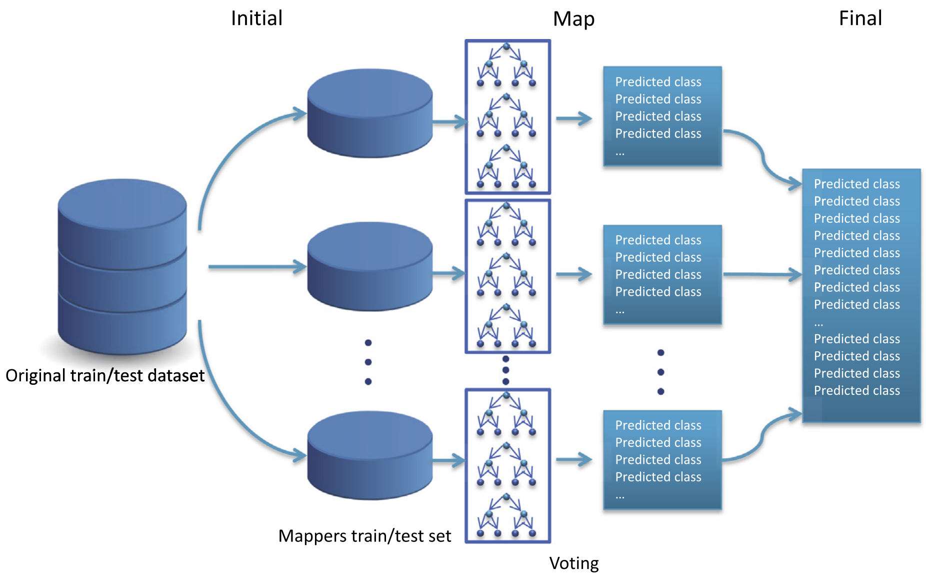


Fig. 4. A flowchart of how the classifying step of the Random Forest algorithm is organized in Mahout.

Algorithm 2. Map phase for the RF-BigData algorithm for the building of the model phase CLEANUP ():

```
1: $bagging \leftarrow BAGGING(instances)$
2: for $i = 0$ $to\ number\ of\ trees\ to\ be\ built\ by\ this\ mapper - 1$ do
3:   $tree \leftarrow bagging.build()$
4:   $key \leftarrow key.set(partitionId,\ treeId)$
5:   EMIT (key, tree)
6: end for
```

Algorithm 3 provides the pseudocode of the Map function of the MapReduce job for the estimation of the classes. In this algorithm, Step (2) estimates the class for an instance and Step (5) saves the previously generated predictions.

Algorithm 3. Map phase for the RF-BigData algorithm for classifying phase MAP (key, value):

```
Input: ⟨key, value⟩ pair, where key is the offset in bytes and value is the content of an instance.
Output: ⟨key′, value′⟩ pair, where key′ indicates the class of an instance and value' contains its prediction.
1: $instance \leftarrow INSTANCE\_REPRESENTATION(value)$
2: $prediction \leftarrow CLASSIFY(instance)$
3: $lkey \leftarrow lkey.set(instance.getClass())$
4: $lvalue \leftarrow lvalue.set(prediction)$
5: EMIT (lkey, lvalue)
```

This MapReduce design does not use a Reduce stage in any of the two phases, building of the model and classification of new instances, since the outputs of each mapper are combined in a straightforward way: in the first phase all the trees generated are combined with the same importance level and contribute equally to the final class prediction. The same situation occurs with the predicted class in the second phase.

3.3. Random Forest for classification with imbalanced datasets

Classification with imbalanced datasets poses a challenge to most of the standard learning techniques. The original RF algorithm is also negatively influenced by an imbalanced class distribution and therefore, it is needed to address this situation effectively for this method.

Weighted Random Forest [14] is a cost-sensitive learning based version of RF that has been prepared to deal with imbalanced datasets (RF-CS). As a cost-sensitive learning method, it depends on the misclassification costs associated to each class and incorporates these costs into its inner way of running. This cost-sensitive RF approach modifies the original RF in two key steps: during the building of the tree and during the classification step, modifying the voting scheme that is used.

During the building of the tree, costs are incorporated into all the inner computations performed. For instance, the splitting criterion used is influenced by the costs associated to the instances where instead of counting all instances equally, the instances influence the information measure considering its associated cost. Additionally, these costs are also used when a leaf computes its label being this calculation also influenced asymmetrically.

Additionally to the previous modifications, it is also necessary to change the majority voting scheme into a weighted voting scheme. For each leaf of a tree, a weight associated to it is computed. This weight is calculated as a proportion of the cost associated to the instances of the leaf in relation to how many instances are there in the leaf. This weight will be the contribution to the ensemble when we need to predict the class for a sample: instead of the equal contribution of the original RF, an aggregated weighted vote that uses the weights of the leaf nodes determines the final classification.

The addition of costs to RF is not done in a straight-forward way that will be equivalent to the use of a random oversampling method. For the construction of the tree, the imbalance procedure of RF-CS does not add specific operations that imply a different behavior from a general weighted method. However, when a class is predicted for a new sample, the mechanism that is used has been specifically developed to enhance the performance of RF and is not a direct usage of the costs.

4. Random Forest for imbalanced big data

In this section, we will present the different alternatives that we have developed following the MapReduce paradigm to deal with imbalanced big data. First, in Section 4.1, we will describe a RF MapReduce version that uses cost-sensitive learning to enhance the learning of the minority class, named as RF-BigDataCS. Then, several data level solutions for imbalanced classification which have been adapted to the big data scenario are presented. These adaptations also follow the MapReduce scheme and have been designed to modify the initial input dataset that will be then fed to the Random Forest version for big data (RF-BigData) described in Section 3.2. Specifically, we have selected as preprocessing algorithms the ROS algorithm, the RUS algorithm and the SMOTE algorithm, whose MapReduce versions are detailed in Sections 4.2, 4.3 and 4.4 respectively.

4.1. RF-BigDataCS: A cost-sensitive approach for Random Forest to deal with Imbalanced Big Data using MapReduce

Inspired by the Mahout RF Partial implementation we build a new RF version that can be used to classify imbalanced big data. To adapt RF-CS to the Mahout environment, we need to include the cost-sensitive operations into the basic RF implementation. First of all (before the dataset is distributed among the mappers), we need to estimate the costs for each class (Fig. 5).

We modify the criteria used to select the best split to build the tree weighting the instances with respect to the misclassification costs. Furthermore, we also consider the costs to calculate which class is associated with the leaf (Fig. 5).

Algorithms 4 and 5 show the pseudocode of the Map function of the MapReduce job. Algorithm 4 is devoted to obtaining all instances in a mapper's partition and the Hadoop framework calls it for each ⟨key, value⟩ pair in this partition. When the previous process is finished, Algorithm 5 is called for each mapper to build a subset of the forest using the instances previously obtained.

Algorithm 4. Map phase for the RF-BigDataCS algorithm for the building of the model phase MAP (key, value):

```
Input: ⟨key, value⟩ pair, where key is the offset in bytes and value is the content of an instance.
Output: ⟨key′, value′⟩ pair, where key′ indicates both the tree id and the data partition id used to grow the tree and
   value′ contains a grow tree and its predictions.
 1: $instance \leftarrow INSTANCE\_REPRESENTATION(value)$ {instances will contain all instances in this mapper's split}
 2: $instances \leftarrow instances.add(instance)$
```

Algorithm 5. Map phase for the RF-BigDataCS algorithm for the building of the model phase CLEANUP ():

```
1: $bagging \leftarrow BAGGING(instances)$
2: for $i = 0$ to number of trees to be built by this mapper $- 1$ do
3:   $tree \leftarrow bagging.build(class\_weights\_estimation)$
4:   $key \leftarrow key.set(partitionId, treeId)$
5:   EMIT (key, tree)
6: end for
```

When each mapper has finished the building of its subset of the forest, we modify the method to classify new examples:

- As a finishing step of the building of the model, the algorithm calculates the leaves weights for each tree. In this manner, for each instance of the dataset the algorithm accumulates in each leaf the number of instances classified and the class weight. Finally, the leaf weight is the accumulated weight divided by the number of instances classified. After that, the

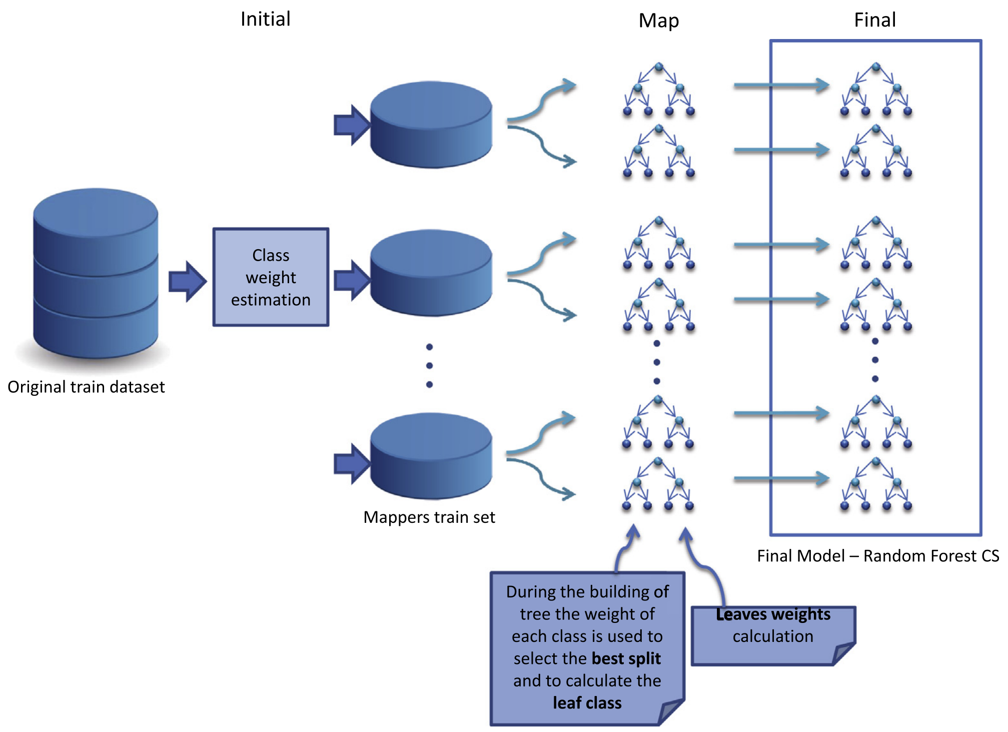

Initial
Map
Final
Class weight estimation
Original train dataset
Mappers train set
Final Model – Random Forest CS
During the building of tree the weight of each class is used to select the best split and to calculate the leaf class
Leaves weights calculation

Fig. 5. A flowchart of how the building of the Random Forest is organized in the RF-BigDataCS algorithm.

algorithm combines the outputs from each mapper. Even if the weights are computed similarly to RF-CS, the trees generated are different from the ones that would be built using the previous algorithm on the same training set. Not only the splits are modified according to the data included in each mapper partition but also the estimated weights for each tree leaf are also modified, influencing directly the final classification model (Fig. 5).

- When the final classification step is performed, for each tree involved in the prediction of the example, for each predicted class, the algorithm accumulates the leaf weight which classifies the example. For each instance in all classes, the accumulated weight is divided by the number of trees involved in the classification. Finally, the selected class is the one which obtains the highest value after this process (Fig. 6).

Algorithm 6 gives the pseudocode of the Map function of the MapReduce job for classifying phase. In this algorithm, Step (2) estimates the class for an instance and Step (5) saves the predictions generated.

Algorithm 6. Map phase for the RF-BigDataCS algorithm for classifying phase MAP (key, value):

```
Input: ⟨key, value⟩ pair, where key is the offset in bytes and value is the content of an instance.
Output: ⟨key′, value′⟩ pair, where key′ indicates the class of an instance and value′ contains its prediction.
1: $instance \leftarrow INSTANCE\_REPRESENTATION(value)$
2: $prediction \leftarrow CLASSIFY(instance)$
3: $lkey \leftarrow lkey.set(instance.getClass())$
4: $lvalue \leftarrow lvalue.set(prediction)$
5: EMIT (lkey, lvalue)
```

4.2. Random oversampling for big data: An approximation with MapReduce

The ROS algorithm has been adapted to deal with big data following a MapReduce design where each Map process is responsible for adjusting the class distribution in a mapper's partition through the random replication of minority class instances and the Reduce process is responsible for collecting the outputs generated by each mapper to form the balanced dataset.

This process is illustrated in Fig. 7 and consists of four steps: Initial, Map, Reduce and Final. At the beginning, in the Initial step, the algorithm performs a segmentation of the input dataset into independent data blocks; replicates and transfers them to other machines. Next, in the Map step, each map task balances the class distribution through the random replication of minority class examples. Then, the Reduce step collects the output generated by each mapper and randomizes the instances in the balanced dataset. In the Final step, the balanced dataset that is generated in the Reduce process forms the final dataset that will be the entry data for the RF-BigData algorithm.

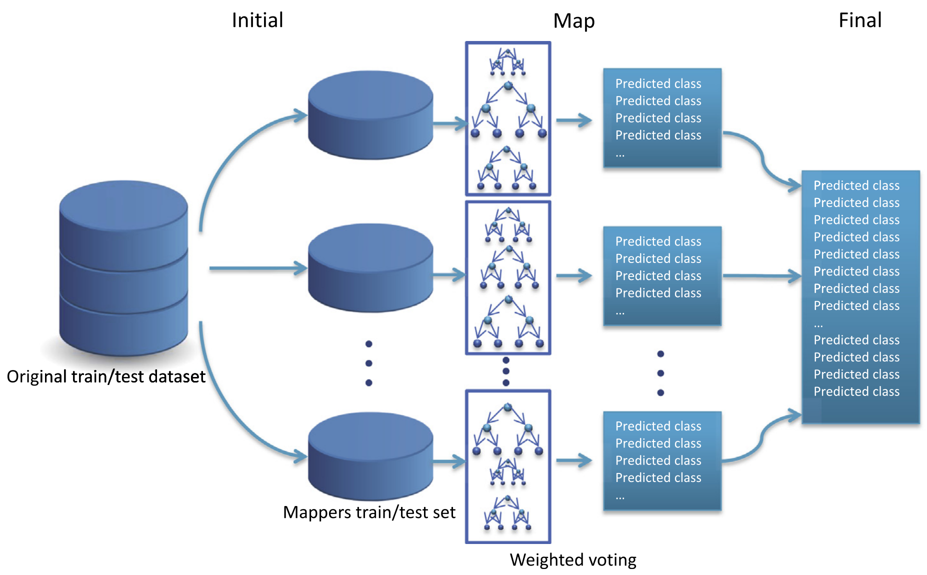


Fig. 6. A flowchart of how the classifying step is organized in the RF-BigDataCS algorithm.

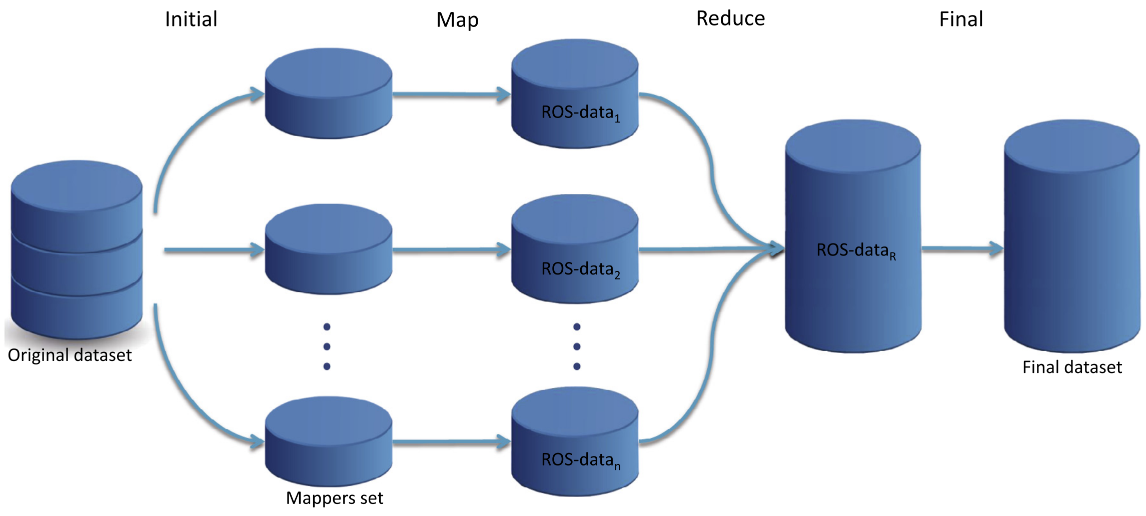


Fig. 7. A flowchart of how the ROS MapReduce design works.

Algorithm 7. Map phase for the ROS algorithm MAP (key, value):

```
Input: ⟨key, value⟩ pair, where key is the offset in bytes and value is the content of an instance.
Output: ⟨key′, value′⟩ pair, where key′ is any Long value and value′ is the content of an instance.
1: $instance \leftarrow INSTANCE\_REPRESENTATION(value)$
2: $class \leftarrow instance.getClass()$
3: $replication\_factor \leftarrow COMPUTE\_REPLICATION\_FACTOR(class)$
4: for $i = 0$ to $replication\_factor - 1$ do
5:    EMIT (key, instance)
6: end for
```

Algorithms 7 and 8 give the pseudocode of the Map and Reduce functions of the MapReduce job respectively. In Algorithm 7, Step (3) calculates the total number of replicas of each instance and is referred as the replication factor. For example, a replication factor of 1 means that there is only one copy of each instance in a mapper's partition, a replication factor of 2 means two copies of each instance and so on. This replication factor is calculated with the total majority class instances and the total instances of the class of the instance that we want to replicate. Step (5) outputs the intermediate $\langle key', value' \rangle$ pair, $\langle key, instance \rangle$. When each mapper has finished, Algorithm 8 is called to randomize (Step 5) the final instances obtained previously and write them as final output (Step 7).

Algorithm 8. Reduce phase for the ROS algorithm REDUCE (key, values):

```
Input: ⟨key, value⟩ pair, where key is any Long value and values is the content of the instances.
Output: ⟨key′, value′⟩ pair, where key′ is a null value and value′ is the content of an instance.
1: while $values.hasNext()$ do
2:    $instance \leftarrow INSTANCE\_REPRESENTATION(values.getValue())$
3:    $instances \leftarrow instances.add(instance)$
4: end while
5: $final\_instances \leftarrow RANDOMIZE(instances)$
6: for $i = 0$ to $final\_instances.length - 1$ do
7:    EMIT (null, final_instances.get (i))
8: end for
```

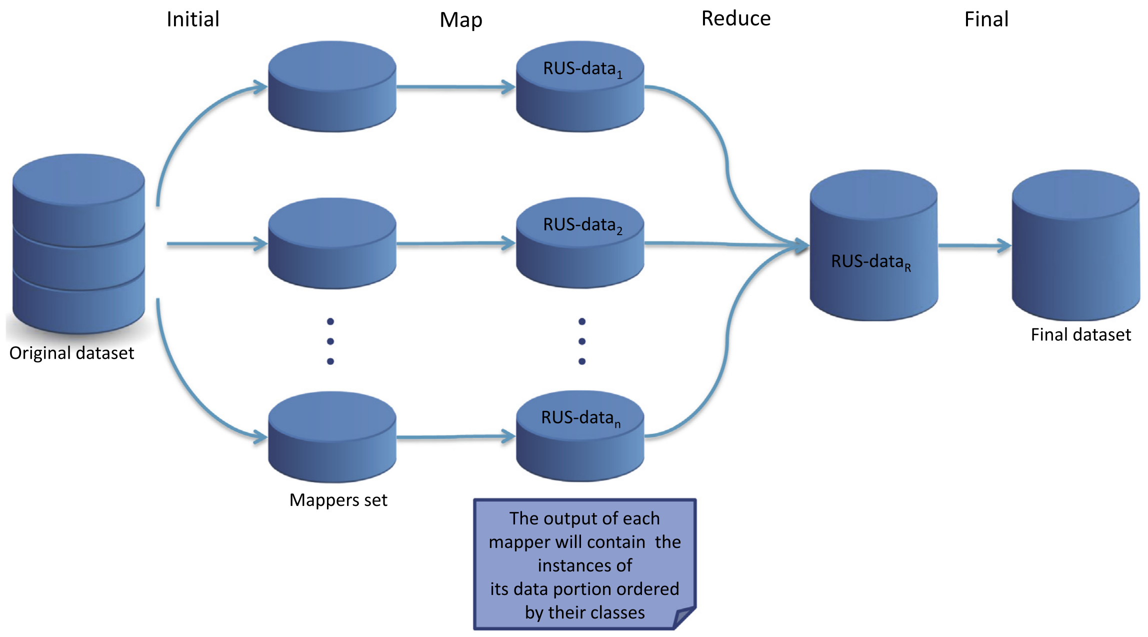

Initial
Map
Reduce
Final
RUS-data1
RUS-data2
RUS-datan
RUS-dataR
Original dataset
Mappers set
Final dataset
The output of each mapper will contain the instances of its data portion ordered by their classes

Fig. 8. A flowchart of how the RUS MapReduce design works.

4.3. Random undersampling for big data: Selecting samples following a MapReduce procedure

The RUS version adapted to deal with big data follows a MapReduce design where each Map process is responsible for grouping by classes all the instances in its data partition and the Reduce process is responsible for collecting the output by each mapper and equilibrating the class distribution through the random elimination of majority class instances to form the balanced dataset.

This process is illustrated in Fig. 8 and consists of four steps: Initial, Map, Reduce and Final. First of all, in the Initial step, the algorithm splits the input dataset into independent data blocks; these blocks are then automatically replicated and transferred between the distinct cluster nodes. Then, in the Map step each map task processes and groups by classes all the instances of its data portion. Next, the Reduce step collects the output generated by each mapper and balances the class distribution randomly eliminating majority class examples. Finally, the balanced dataset that is generated in the reduce process is the final dataset that will be the entry data for the RF-BigData algorithm.

Algorithm 9. Map phase for the RUS algorithm MAP (key, value):

```
Input: ⟨key, value⟩ pair, where key is the offset in bytes and value is the content of an instance.
Output: ⟨key′, value′⟩ pair, where key′ is the class name associated with the instance and value′ is the content of an
  instance.
1: $instance \leftarrow INSTANCE\_REPRESENTATION(value)$
2: $class \leftarrow instance.getClass()$
3: EMIT (class, value)
```

Algorithm 9 shows the operations of the Map function which generates a new ⟨key, value⟩ pair for each input ⟨key, value⟩ pair. The new key is the class name of the input instance and the new value is directly the instance. In this way, the Reduce function will receive all the instances grouped by class. Algorithms 10 and 11 display the pseudocode of the Reduce function. In Algorithm 10, the Steps (2–8) are executed when the input key corresponds to the majority class. In this case, the input values are shuffled (Step 6) to select (Step 7) a number of majority class instances equal to the number of minority class instances. On the other hand, Steps (10–13) are initiated when the input key corresponds to the minority class and select the input values as final instances. When the above process is finished, Algorithm 11 is called to randomize the instances obtained that later will be fed to the RF-BigData algorithm.

Algorithm 10. Reduce phase for the RUS algorithm REDUCE (key, values):

```
Input: ⟨key, value⟩ pair, where key represents a class name and values is the content of the instances.
Output: ⟨key′, value′⟩ pair, where key′ is a null value and value′ is the content of an instance.
 1: if $key == majorityClass$ then
 2:   while $values.hasNext()$
 3:     $instance \leftarrow INSTANCE\_REPRESENTATION(values.getValue())$
 4:     $instances \leftarrow instances.add(instance)$
 5:   end while
 6:   $instances \leftarrow SHUFFLE(instances)$
 7:   $instances \leftarrow instances.subList(0, number\ of\ ocurrences\ of\ minority\ class - 1)$
 8:   $final\_instances.add(instances)$
 9: else
10:   while $values.hasNext()$
11:     $instance \leftarrow INSTANCE\_REPRESENTATION(values.getValue())$
12:     $final\_instances.add(instance)$
13:   end while
14: end if
```

Algorithm 11. Reduce phase for the RUS algorithm CLEANUP ():

```
1: $final\_instances \leftarrow RANDOMIZE(final\_instances)$
2: for $i = 0$ to $final\_instances.length - 1$ do
3:   EMIT (null, final_instances.get (i))
4: end for
```

4.4. SMOTE for big data: Adapting the generation of synthetic minority samples using MapReduce

The SMOTE algorithm has been adapted to deal with big data following a MapReduce design where each Map process oversamples the minority class and the Reduce process randomizes the output generated by each mapper to form the balanced dataset.

This process is illustrated in Fig. 9 and consists of four steps: Initial, Map, Reduce and Final. In the Initial step, the algorithm performs a segmentation of the input dataset into independent data blocks; replicates and transfers them to other machines. Next, in the Map phase, each map process balances the class distribution in a mapper's partition using the basic SMOTE algorithm over the available data. Then, the Reduce step collects the output generated by each mapper and randomizes the final data. In the Final step, the balanced dataset that is generated in the Reduce process forms the final output that will be the entry data for the RF-BigData algorithm.

Algorithm 12. Map phase for the SMOTE algorithm MAP (key, value):

```
Input: ⟨key, value⟩ pair, where key is the offset in bytes and value is the content of an instance.
Output: ⟨key′, value′⟩ pair, where key′ is any Long value and value′ is the content of an instance.
1: $instance \leftarrow INSTANCE\_REPRESENTATION(value)$
2: $instances \leftarrow instances.add(instance)$
```

Algorithms 12 and 13 give the pseudocode of the Map function and Algorithm 14 shows the pseudocode of the Reduce function of the MapReduce job. In Algorithm 12, Steps (1 and 2) are called for each ⟨key, value⟩ pair in this partition and obtain all instances in a mapper's partition. Then, Algorithm 13 is called for each mapper to execute the SMOTE algorithm (Step 2). When each mapper has completed its operations, Algorithm 14 is run to randomize (Step 5) the SMOTE instances obtained previously.

Algorithm 13. Map phase for the SMOTE algorithm CLEANUP ():

```
1: $smote \leftarrow SMOTE(instances)$
2: $smote\_instances \leftarrow smote.run()$
3: for $i = 0$ to $smote\_instances.length - 1$ do
4:   EMIT (key, smote_instances.get (i))
5: end for
```

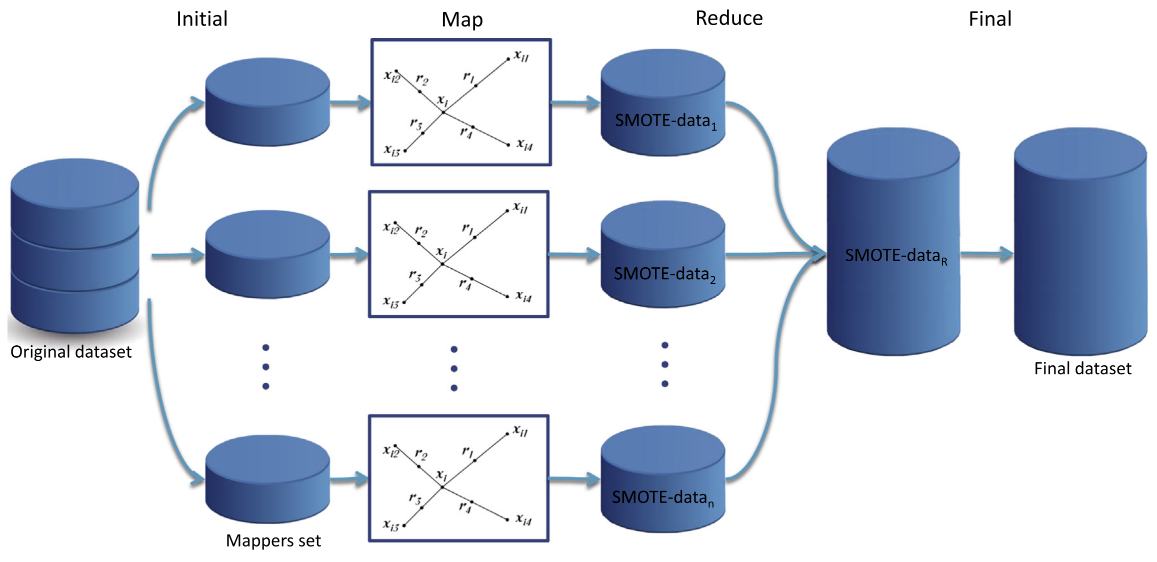


Fig. 9. A flowchart of how the SMOTE MapReduce design works.

Algorithm 14. Reduce phase for the SMOTE algorithm REDUCE (key, values):

```
Input: ⟨key, value⟩ pair, where key is any Long value and values is the content of the instances.
Output: ⟨key′, value′⟩ pair, where key’ is a null value and value’ is the content of an instance.
1: while values.hasNext()
2:    instance ← INSTANCE_REPRESENTATION(values.getValue())
3:    smote_instances ← smote_instances.add(instance)
4: end while
5: final_instances ← RANDOMIZE(smote_instances)
6: for i = 0 to final_instances.length − 1 do
7:    EMIT (null, final_instances.get (i))
8: end for
```

5. Experimental study

This section describes the experimental study that is carried out to compare the performance of the different approaches to deal with imbalanced big data using the Random Forest algorithm. First, Section 5.1 presents the classification problems that are utilized in the experiments together with the algorithms that are used in the comparison, as well as the parameters selected for them. Later, Section 5.2 provides a study about the limitations that the sequential versions encounter when dealing with datasets with an increased size. Next, in Section 5.3, the performance results for the approaches using the imbalanced classification measures can be found. Finally, Section 5.4 shows an analysis that evaluates the runtime spent by all the imbalanced big data techniques used in the comparison.

5.1. Experimental framework: Datasets, algorithms selected for the study and parameter settings

In order to analyse the quality of the solutions provided by the different algorithms for imbalanced big data, we have selected three big data problems that are available in the UCI Machine Learning Repository [4]. Specifically, we have selected the KDD Cup 1999 dataset, the Record Linkage Comparison Patterns (RLCP) dataset and the Poker Hand dataset. In the case where missing values were present, the attributes that contained them have been discarded.

As the selected problems contain multiple classes, we have derived several cases of study from them to address each issue separately. Concretely, we have created new datasets using the classes that contained a notable number of samples in comparison with the rest as majority classes, while the classes less represented have been considered as minority classes. The data are summarized in Table 2, where we include the number of examples (#Ex.), number of attributes (#Atts.), class name of each class (minority and majority), number of instances for each class, class distribution and IR.

To develop the different experiments we consider a 5-fold stratified cross-validation partitioning scheme, i.e., five random partitions of data with a 20% and the combination of four of them (80%) as training set and the remaining one as test set. The results provided for each dataset are the average results obtained by computing the mean of all the partitions.

Table 2
Summary of imbalanced datasets.

Datasets	#Ex.	#Atts.	Class (maj; min)	#Class (maj; min)	%Class (maj; min)	IR
kddcup_DOS_versus_normal	4,856,151	41	(DOS; normal)	(3,883,370; 972,781)	(79.968; 20.032)	3.992
kddcup_DOS_versus_PRB	3,924,472	41	(DOS; PRB)	(3,883,370; 41,102)	(98.953; 1.047)	94.481
kddcup_DOS_versus_R2L	3,884,496	41	(DOS; R2L)	(3,883,370; 1126)	(99.971; 0.029)	3448.819
kddcup_DOS_versus_U2R	3,883,422	41	(DOS; U2R)	(3,883,370; 52)	(99.999; 0.001)	74680.192
kddcup_normal_versus_PRB	1,013,883	41	(normal; PRB)	(972,781; 41,102)	(95.946; 4.054)	23.667
kddcup_normal_versus_R2L	973,907	41	(normal; R2L)	(972,781; 1126)	(99.884; 0.116)	863.926
kddcup_normal_versus_U2R	972,833	41	(normal; U2R)	(972,781; 52)	(99.995; 0.005)	18707.327
poker_0_vs_2	562,530	10	(0; 2)	(513,702; 48,828)	(91.32; 8.68)	10.521
poker_0_vs_3	535,336	10	(0; 3)	(513,702; 21,634)	(95.959; 4.041)	23.745
poker_0_vs_4	517,680	10	(0; 4)	(513,702; 3978)	(99.232; 0.768)	129.136
poker_0_vs_5	515,752	10	(0; 5)	(513,702; 2050)	(99.603; 0.397)	250.586
poker_0_vs_6	515,162	10	(0; 6)	(513,702; 1460)	(99.717; 0.283)	351.851
poker_1_vs_2	481,925	10	(1; 2)	(433,097; 48,828)	(89.868; 10.132)	8.870
poker_1_vs_3	454,731	10	(1; 3)	(433,097; 21,634)	(95.242; 4.758)	20.019
poker_1_vs_4	437,075	10	(1; 4)	(433,097; 3978)	(99.09; 0.91)	108.873
poker_1_vs_5	435,147	10	(1; 5)	(433,097; 2050)	(99.529; 0.471)	211.267
poker_1_vs_6	434,557	10	(1; 6)	(433,097; 1460)	(99.664; 0.336)	296.642
RLCP	5,749,132	2	(FALSE; TRUE)	(5,728,201; 20,931)	(99.636; 0.364)	273.671

In our comparisons we test all the algorithms that have been described in this work. When the sequential versions are examined, we check the behavior of the original Random Forest (RF) and its adaptation to the imbalanced scenario (RF-CS) described in Sections 3.1 and 3.3 respectively. The basic RF is also combined with the preprocessing approaches described in Section 2.2.2, namely ROS, RUS and SMOTE, to deal with imbalanced datasets.

When our focus is directed towards the algorithms that are able to deal with imbalanced big data, we test the behavior of the Random Forest Partial Implementation (RF-BigData) described in Section 3.2 and the modifications that we have added to it, to adapt its behavior to properly deal with imbalanced data (RF-BigDataCS), detailed in Section 4.1. The RF-BigData algorithm is again associated with the ROS, RUS and SMOTE algorithms that have been updated using MapReduce to deal with big data and that were respectively described in Sections 4.2, 4.3 and 4.4.

Please note that we use the "BigData" notation to differentiate the sequential versions in contrast with the MapReduce designs in their description, however, we will not add that suffix to the names of the preprocessing algorithms to save space. The sequential preprocessing algorithms are run with the sequential RF versions while the MapReduce preprocessing approaches are run with the MapReduce RF.

The RF variations are run using the following parameters: $maxDepth\ = unlimited$, $numFeatures\ =\ log_2(N_{atts}) + 1$ and $numTrees\ = 100$. The *maxDepth*, *numFeatures* and *numTrees* parameters represent how the forest is built: the *numTrees* parameter indicates how many trees compose the forest, *maxDepth* indicates the depth of each of the trees generated and the *numFeatures* parameter shows how many attributes are selected to build one of those trees.

For the RF-BigData variants we also include the parameter $numMappers\ = 8/16/32/64$ which is related to the MapReduce procedure that is included to deal with big data: this parameter represents the number of subsets of the original data that are created and are provided for the map tasks. In this manner, the number of trees associated to each map task is also modified according to the number of partitions of the data, as the final forest is composed of the trees generated by each map task.

The preprocessing algorithms (ROS, RUS and SMOTE) modify the dataset until the class distribution becomes completely balanced. The SMOTE algorithm uses the 5 nearest neighbors of minority class samples to generate new instances and it computes the distances using the Euclidean function.

The misclassification costs used for the cost-sensitive learning approaches and the computation of the β value of the f-measure are $C(+ \mid -) = IR$ and $C(- \mid +) = 1$.

Considering the infrastructure used, all the experiments were run at the atlas research group cluster. This cluster is composed of 12 nodes, each with two Intel E5-2620 microprocessors (at 2 GHz, 15 MB cache) and 64 MB of main memory, connected with 1 Gb/s ethernet. All of them work under Linux CentOS 6.3. The cluster is configured with Hadoop and Mahout. One of the nodes is configured as name-node and job-tracker, and the remaining nodes are both datanodes and trask-trackers. The Hadoop version used is 1.0 (Cloudera CDH4) and the Mahout version is 0.8.

5.2. Analysis of the sequential versions of the Random Forest algorithm when the size of the data available is increased

To analyse the behavior of the sequential versions when they are faced with data that grows in size, we select three of the cases of study that were derived from the KDD Cup 1999 dataset. For each one of these cases of study, we create smaller versions of them selecting a 10%, 20%, 30%, 40% and 50% of the samples of the original problem maintaining the proportion between the classes. A 5-fold stratified cross-validation model is also used in the smaller versions considered in this section. Table 3 shows the details of the selected cases of study and their generated reduced versions.

Table 3
Summary of imbalanced datasets with reduced size.

Datasets	#Ex.	#Atts.	Class (maj; min)	#Class (maj; min)	%Class (maj; min)	IR
kddcup_10_DOS_versus_normal	485,615	41	(DOS; normal)	(388,337; 97,278)	(79.968; 20.032)	3.992
kddcup_20_DOS_versus_normal	971,230	41	(DOS; normal)	(776,674; 194,556)	(79.968; 20.032)	3.992
kddcup_30_DOS_versus_normal	1,456,845	41	(DOS; normal)	(1,165,011; 291,834)	(79.968; 20.032)	3.992
kddcup_40_DOS_versus_normal	1,942,460	41	(DOS; normal)	(1,553,348; 389,112)	(79.968; 20.032)	3.992
kddcup_50_DOS_versus_normal	2,428,075	41	(DOS; normal)	(1,941,685; 486,390)	(79.968; 20.032)	3.992
kddcup_DOS_versus_normal	4,856,151	41	(DOS; normal)	(3,883,370; 972,781)	(79.968; 20.032)	3.992
kddcup_10_DOS_versus_U2R	388,342	41	(DOS; U2R)	(388,337; 5)	(99.999; 0.001)	77667.400
kddcup_20_DOS_versus_U2R	776,684	41	(DOS; U2R)	(776,674; 10)	(99.999; 0.001)	77667.400
kddcup_30_DOS_versus_U2R	1,165,026	41	(DOS; U2R)	(1,165,011; 15)	(99.999; 0.001)	77667.400
kddcup_40_DOS_versus_U2R	1,553,368	41	(DOS; U2R)	(1,553,348; 20)	(99.999; 0.001)	77667.400
kddcup_50_DOS_versus_U2R	1,941,711	41	(DOS; U2R)	(1,941,685; 26)	(99.999; 0.001)	74680.192
kddcup_DOS_versus_U2R	3,883,422	41	(DOS; U2R)	(3,883,370; 52)	(99.999; 0.001)	74680.192
kddcup_10_normal_versus_R2L	97,390	41	(normal; R2L)	(97,278; 112)	(99.885; 0.115)	868.554
kddcup_20_normal_versus_R2L	194,781	41	(normal; R2L)	(194,556; 225)	(99.884; 0.116)	864.693
kddcup_30_normal_versus_R2L	292,171	41	(normal; R2L)	(291,834; 337)	(99.885; 0.115)	865.976
kddcup_40_normal_versus_R2L	389,562	41	(normal; R2L)	(389,112; 450)	(99.884; 0.116)	864.693
kddcup_50_normal_versus_R2L	486,953	41	(normal; R2L)	(486,390; 563)	(99.884; 0.116)	863.925
kddcup_normal_versus_R2L	973,907	41	(normal; R2L)	(972,781; 1126)	(99.884; 0.116)	863.926

In Tables 4 and 5 we can see the performance results for all the sequential RF versions considered in the study over the three selected cases of study using increasing sized versions and considering the GM and β-f-measure respectively as effectiveness measures. The *N.D.* (Not Determinable) symbol is included for some of the results and it indicates that the corresponding algorithm was not able to complete the associated experiment. All the sequential implementations tested have not been especially enhanced to deal with large datasets and have been taken or implemented as they are described in the original references. In this manner, the presence of the *N.D.* symbol is not biased and the assumption that the subsequent algorithm is not able to manage datasets this size is reliable.

In terms of the precision achieved by the different alternatives considered in the study we can first observe that there are no differences about the increase or descent of the results when comparing the performance measures used, namely the GM and β-f-measure. The approaches show a more or less stable performance when the training set is used, however, the results vary depending on the case of study when we look at the test results. For the *kddcup_DOS_versus_normal* case we can see that the RF algorithm is able to provide very good results that are not affected by the size of the dataset; for the *kddcup_DOS_versus_U2R* case of study we can see that the results in test increase when larger datasets are used while the *kddcup_normal_versus_R2L* case is not able to show a tendency to improvement or its opposite for its test results.

Considering the algorithms that were able to provide results we can see that the results are what were expected: for the largest case of study, the algorithms stopped working in the smaller versions of the problem, for the medium sized, the algorithms were not able to provide results when the data variants were of medium size and for the smallest one, most of the versions were able to deal with the full sized datasets and only two algorithms were not able to provide results.

When looking at the specific type of algorithm that were able to provide an answer, we can find three groups of behavior, one corresponding to the direct RF usage either in its basic version or in its cost-sensitive learning variant; another one that incorporates the two oversampling approaches, ROS and SMOTE; and the third one containing only the undersampling procedure RUS. Since the oversampling approaches increase the size of the input data for the RF algorithm, it is expected that they are the methods that are able to deal with less data. In the opposite case, we find the undersampling procedure, as the one that can manage larger datasets because it reduces the size of the input data for RF. The direct application of RF provides results for data with sizes larger than the ones managed by the oversampling approaches but smaller than RUS.

5.3. Analysis of the effectiveness in classification of the diverse approaches for imbalanced big data

At this point, we present the results obtained by the MapReduce RF versions that are able to manage imbalanced big data over the whole datasets considered in the study according to the precision measures tested in this work: the GM and the β-f-measure. Instead of using all the datasets together to get an idea of which approaches are the best for dealing with this type of data, we have grouped the data available in three groups so that we can get a better idea of what is happening in the correct identification of instances of each class (from bigger datasets to smaller): the cases of study derived from the kddcup dataset define the first group; the second group involves the RLCP dataset; and the last group is composed of the cases of study derived from the poker dataset.

Table 6 shows the average results in training and test for all the MapReduce approaches studied in this work using 8, 16, 32 and 64 mappers respectively over the cases of study derived from the kddcup dataset and considering the GM performance measure. The bold values highlight the best performing algorithm related to the number of mappers considered. The detailed results per case of study (for all the tables in this section) are available in an associated website.[1]

[1] <http://sci2s.ugr.es/rf_big_imb/>.

Table 4
Average results for the sequential RF versions for the imbalanced big data cases of study using the GM measure.

Datasets	RF		RF-CS		ROS + RF		RUS + RF		SMOTE + RF	
	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}
kddcup_10_DOS_versus_normal	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
kddcup_20_DOS_versus_normal	1.000	1.000	1.000	1.000	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_30_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_40_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_50_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_10_DOS_versus_U2R	1.000	0.000	1.000	1.000	1.000	0.500	0.600	0.000	1.000	0.500
kddcup_20_DOS_versus_U2R	1.000	0.741	1.000	1.000	1.000	0.741	0.981	0.981	1.000	1.000
kddcup_30_DOS_versus_U2R	1.000	0.918	1.000	1.000	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_40_DOS_versus_U2R	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_50_DOS_versus_U2R	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_DOS_versus_U2R	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_10_normal_versus_R2L	1.000	0.981	0.999	0.996	1.000	0.985	0.997	0.996	1.000	0.988
kddcup_20_normal_versus_R2L	1.000	0.922	1.000	0.998	1.000	0.927	0.991	0.990	1.000	0.943
kddcup_30_normal_versus_R2L	1.000	0.960	1.000	0.998	1.000	0.961	0.993	0.992	1.000	0.967
kddcup_40_normal_versus_R2L	1.000	0.950	1.000	0.989	1.000	0.953	0.994	0.979	1.000	0.956
kddcup_50_normal_versus_R2L	1.000	0.976	1.000	0.998	1.000	0.968	0.994	0.993	1.000	0.978
kddcup_normal_versus_R2L	1.000	0.947	1.000	0.999	N.D.	N.D.	0.998	0.992	N.D.	N.D.

Table 5
Average results for the sequential RF versions for the imbalanced big data cases of study using the β-f-measure.

Datasets	RF		RF-CS		ROS + RF		RUS + RF		SMOTE + RF	
	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}
kddcup_10_DOS_versus_normal	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
kddcup_20_DOS_versus_normal	1.000	1.000	1.000	1.000	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_30_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_40_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_50_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_DOS_versus_normal	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_10_DOS_versus_U2R	1.000	0.000	1.000	1.000	1.000	1.000	1.000	0.000	1.000	1.000
kddcup_20_DOS_versus_U2R	1.000	0.700	1.000	1.000	1.000	0.875	1.000	1.000	1.000	1.000
kddcup_30_DOS_versus_U2R	1.000	0.853	1.000	1.000	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_40_DOS_versus_U2R	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_50_DOS_versus_U2R	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	1.000	1.000	N.D.	N.D.
kddcup_DOS_versus_U2R	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.	N.D.
kddcup_10_normal_versus_R2L	1.000	0.963	1.000	1.000	1.000	0.971	1.000	0.998	1.000	0.976
kddcup_20_normal_versus_R2L	1.000	0.852	1.000	1.000	1.000	0.860	1.000	0.996	1.000	0.890
kddcup_30_normal_versus_R2L	1.000	0.923	1.000	1.000	1.000	0.924	1.000	0.997	1.000	0.936
kddcup_40_normal_versus_R2L	1.000	0.903	1.000	0.981	1.000	0.909	1.000	0.969	1.000	0.915
kddcup_50_normal_versus_R2L	1.000	0.952	1.000	0.998	1.000	0.937	1.000	0.998	1.000	0.956
kddcup_normal_versus_R2L	1.000	0.896	1.000	1.000	N.D.	N.D.	1.000	0.989	N.D.	N.D.

In a first glance, we can see that the results in training indicate that there is not a strong overfitting in most of the versions (except for oversampling methods) tested as there is not a huge gap between the training and test results. Looking at that table we can also observe that when 8 mappers are used the best performing algorithm in average is the RUS approach, closely followed by the ROS algorithm. However, when the number of mappers is increased we can see that the RUS method has a drop in its performance greater than the loss of performance for the other methods, leaving the ROS algorithm as the best performing one with a large number of mappers. The other algorithms also show a descent in the correct identification of samples when the number of mappers is incremented, being that descent in a close proportion among them.

In Table 7, we can observe the average results in training and test of all the MapReduce approaches studied in this work using 8, 16, 32 and 64 mappers respectively but in this case, considering the β-f-measure to test the performance over the kddcup dataset cases of study. The best performing algorithm is highlighted in bold for each number of mappers considered.

When the β-f-measure is used we can see that the tendencies observed for the GM have not changed. There is no overfitting (except for oversampling techniques) as the performance in training and test results is not quite different. For this measure, we can see again that the RUS algorithm is the one that obtains better results when using a smaller number of mappers while it is the ROS algorithm the one that performs better when a larger number of mappers are used. This behavior appears because the RUS algorithm is more degraded than the ROS algorithm when the number of mappers grows due to

Table 6
Average GM results for the MapReduce RF versions using 8, 16, 32 and 64 mappers on the imbalanced big data kddcup cases.

Dataset	Average (kddcup)							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}
Big data versions								
RF-BigData	0.7620	0.7505	0.6985	0.6976	0.6852	0.6836	0.6626	0.6598
RF-BigDataCS	0.9404	0.9305	0.9480	0.9651	0.9173	0.9328	0.9372	0.9286
ROS + RF-BigData	1.0000	0.9661	0.9999	**0.9696**	0.9999	**0.9773**	0.9999	**0.9857**
RUS + RF-BigData	0.9869	**0.9843**	0.9490	0.9336	0.7103	0.7104	0.7049	0.7048
SMOTE + RF-BigData	0.9477	0.9140	0.9381	0.9191	0.9445	0.9091	0.8994	0.8722

Table 7
Average β-f-measure results for the MapReduce RF versions using 8, 16, 32 and 64 mappers on the imbalanced big data kddcup cases.

Dataset	Average (kddcup)							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}
Big data versions								
RF-BigData	0.7169	0.7111	0.6842	0.6825	0.6605	0.6576	0.6252	0.6208
RF-BigDataCS	0.9008	0.9038	0.9132	0.9406	0.8784	0.8968	0.8969	0.8836
ROS + RF-BigData	1.0000	0.9406	0.9999	**0.9473**	0.9999	**0.9608**	0.9999	**0.9735**
RUS + RF-BigData	0.9864	**0.9816**	0.9642	0.9444	0.8839	0.8841	0.8527	0.8525
SMOTE + RF-BigData	0.9111	0.8707	0.9250	0.8985	0.9088	0.8522	0.8602	0.8173

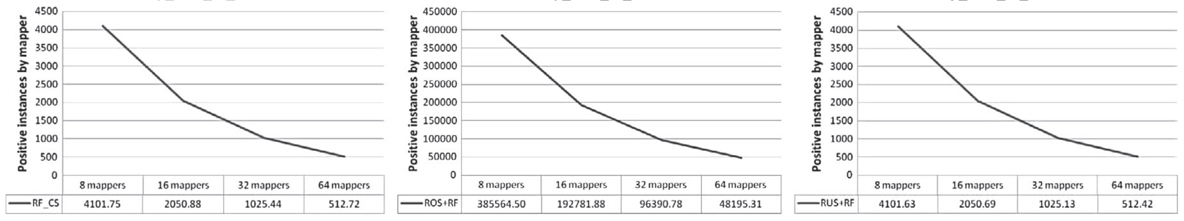
the small sample size problem. The rest of the algorithms have a similar degradation development when the number of mappers is decreased.

These results in performance are directly related to the number of instances present in the mapper to build the final model. In Fig. 10 we can see for the first partition of the kddcup_DOS_vs_PRB case of study the average number of minority class instances per mapper that are made available for the RF-BigDataCS approach (Fig. 10a), the ROS approach (Fig. 10b), and the RUS method (Fig. 10c).

Fig. 10 shows three curves with a same tendency in the number of examples available, however, the magnitude of the minority class samples is notably higher for the ROS algorithm than for the other two approaches. When we consider the most difficult case, using 64 mappers, we can see that the ROS methodology has approximately 90 more times the number of instances than the other methods.

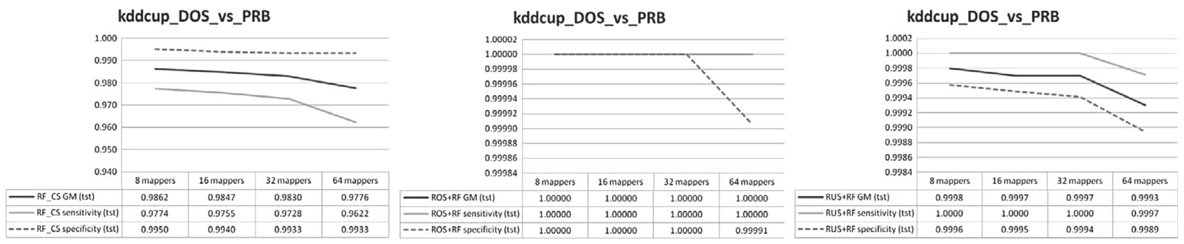
In this case, we can observe a small sample size problem for the minority class when we are using the RF-BigDataCS approach or the RUS method. The number of samples from the minority class that are used when a higher number of mappers are considered is considerably smaller than the original minority class and amplifies the lack of density that is inherent to some imbalanced distributions. In fact, this reduction of available samples can even be detected with a quantity of mappers not that high.

The small sample size problem has even a greater impact when the RF algorithm is used. The original RF algorithm (and all its versions) first perform a bootstrap sample of the dataset given to the algorithm. This bootstrap sample is said to preserve about two-thirds of the original population samples while it leaves one-third of the samples out of it. This means, that each random tree that is learned does not consider one-third of the minority class instances available in the problem, an event which causes a bias when we do not have enough samples to properly represent the class.



(a) RF-BigDataCS (b) ROS+RF-BigData (c) RUS+RF-BigData

Fig. 10. Average number of positive instances by mapper for kddcup_DOS_vs_PRB.



(a) RF-BigDataCS

	8 mappers	16 mappers	32 mappers	64 mappers
RF_CS GM (tst)	0.9862	0.9847	0.9830	0.9776
RF_CS sensitivity (tst)	0.9774	0.9755	0.9728	0.9622
RF_CS specificity (tst)	0.9950	0.9940	0.9933	0.9933

(b) ROS+RF-BigData

	8 mappers	16 mappers	32 mappers	64 mappers
ROS+RF GM (tst)	1.00000	1.00000	1.00000	1.00000
ROS+RF sensitivity (tst)	1.00000	1.00000	1.00000	1.00000
ROS+RF specificity (tst)	1.00000	1.00000	1.00000	0.99991

(c) RUS+RF-BigData

	8 mappers	16 mappers	32 mappers	64 mappers
RUS+RF GM (tst)	0.9998	0.9997	0.9997	0.9993
RUS+RF sensitivity (tst)	1.0000	1.0000	1.0000	0.9997
RUS+RF specificity (tst)	0.9996	0.9995	0.9994	0.9989

Fig. 11. Average results for kddcup_DOS_vs_PRB using the GM measure.

In this manner, the ROS algorithm is less affected by the small sample size problem because it provides a high number of samples from the positive class in comparison with the other algorithms. However, it is not only because it provides a higher number of examples but, because it replicates the original number of minority class samples so that the examples available can be represented in the bootstrap thus alleviating the lack of density.

Fig. 11 shows the average results for the first partition of the kddcup_DOS_vs_PRB case of study for the algorithms considered for the previous graph: the RF-BigDataCS approach (Fig. 11a), the ROS approach (Fig. 11a), and the RUS method (Fig. 11c). This figure shows the evolution of the GM, the sensitivity and specificity measures when the number of mappers is varied.

When we analyse each of the graphs separately we can find that the results shown match the extracted conclusions from the previous tables.

- For the RF-BigDataCS algorithm we can see that the diminishment in the GM when using a larger number of mappers comes from a loss of performance for the minority class as we do not have enough samples to create a reliable RF model when the number of mappers is large. These are clear signs of the small sample size problem on the minority class.
- When we look at the results for the ROS algorithm used together with RF-BigData we can see that results are very good for both the GM and the sensitivity measures, with a slight drop in performance for the correct identification of the majority class. In this case, we avoid the small sample size problem because we have extra samples to identify the minority class.
- The RUS method combined with RF decreases its GM because the accuracy in its classification for both classes decreases when the number of mappers is incremented. This method is a example of how the number of examples can degrade a classifier performance as we observe the small sample size problem for both classes when the number of mappers is incremented.

In Tables 8 and 9 the average results in training and test of all the MapReduce approaches studied in this work using 8, 16, 32 and 64 mappers respectively and considering the GM and β-f-measure performance measures respectively are shown for the RLCP dataset. The bold values highlight the best performing method for each number of mappers considered.

Considering the results presented in both tables we can see, first of all, that there are no differences in behavior depending on the performance measure used in each dataset, so the information that can be extracted from the two tables is the same. Considering the practically lack of differences between the training and test results for each algorithm we can also say that the approaches tested do not suffer from the overfitting problem. When we compare the approaches tested, we can see that there are two groups of behavior with respect to the performance obtained: the basic RF-BigData algorithm and its usage together with the SMOTE algorithm, which do not provide good results for the classification; and the RF-BigDataCS, the ROS and RUS methods used together with the RF-BigData method which obtain a good performance for all the cases considered. This second group of algorithms with good behavior does not show a drop in the performance when the number of mappers used is incremented.

Table 10 presents the GM average results in training and test of all the MapReduce approaches studied in this work using 8, 16, 32 and 64 mappers respectively over the poker dataset cases of study. The best performing algorithm is highlighted in bold for each number of mappers considered.

Table 10 allows us to divide again the algorithms in two groups according to their results: the basic RF-BigData algorithm and its usage with SMOTE, which obtain very poor results for the cases under consideration; and the RF-BigDataCS algorithm together with the ROS and RUS methods that are combined with the basic RF-BigData approach which are able to yield better results.

Comparing the performance obtained in training and test we can see that there is some overfitting for all the algorithms considered, however, it is especially noticeable for the ROS algorithm with a smaller number of mappers. Considering the performance of the RF-BigDataCS method, we can see that it obtains the best performance with a smaller number of mappers and it is only exceeded by the ROS method when using 64 mappers. The RF-BigDataCS method reduces its performance when the number of mappers is increased while the ROS algorithm is able to increase its efficacy when that situation arises, due to the small sample size problem. The RUS approach starts with a better behavior than ROS when it is run using 8 mappers, however, RUS also lowers its performance when the number of mappers grows leaving this algorithm at disadvantage

Table 8
Average GM results for the MapReduce RF versions using 8, 16, 32 and 64 mappers on the RLCP dataset.

Dataset	RLCP							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}
Big data versions								
RF-BigData	0.116	0.116	0.116	0.116	0.116	0.116	0.116	0.116
RF-BigDataCS	0.932	**0.932**	0.932	**0.932**	0.932	**0.932**	0.932	**0.932**
ROS + RF-BigData	0.932	**0.932**	0.932	**0.932**	0.932	**0.932**	0.932	**0.932**
RUS + RF-BigData	0.932	**0.932**	0.932	**0.932**	0.931	0.931	0.931	0.931
SMOTE + RF-BigData	0.117	0.116	0.118	0.118	0.118	0.118	0.118	0.118

Table 9
Average β-f-measure results for the MapReduce RF versions using 8, 16, 32 and 64 mappers on the RLCP dataset.

Dataset	RLCP							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}	β-f-m_{tr}	β-f-m_{tst}
Big data versions								
RF-BigData	0.014	0.014	0.014	0.014	0.014	0.014	0.014	0.014
RF-BigDataCS	0.992	**0.991**	0.992	0.991	0.992	0.991	0.991	0.991
ROS + RF-BigData	0.992	**0.991**	0.992	**0.992**	0.992	**0.992**	0.992	**0.992**
RUS + RF-BigData	0.991	**0.991**	0.991	0.991	0.991	0.991	0.990	0.990
SMOTE + RF-BigData	0.014	0.014	0.014	0.014	0.014	0.014	0.014	0.014

Table 10
Average GM results for the MapReduce RF versions using 8, 16, 32 and 64 mappers on the imbalanced big data poker cases.

Dataset	Average (poker)							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}
Big data versions								
RF-BigData	0.191	0.108	0.066	0.053	0.038	0.035	0.023	0.021
RF-BigDataCS	0.946	**0.878**	0.926	**0.853**	0.891	**0.818**	0.828	0.771
ROS + RF-BigData	0.974	0.748	0.972	0.810	0.943	0.816	0.948	**0.863**
RUS + RF-BigData	0.885	0.843	0.838	0.811	0.793	0.772	0.705	0.696
SMOTE + RF-BigData	0.231	0.206	0.205	0.201	0.198	0.195	0.091	0.076

against ROS which has a different attitude in that case. We find again in this case the results of the small sample size problem associated to the minority class as we increase the number of splits performed in the data.

In Table 11, the average results in training and test of all the MapReduce approaches studied in this work using 8, 16, 32 and 64 mappers are presented for the poker dataset cases of study and using to test the effectiveness of the approaches the β-f-measure. The bold values highlight the best performing method for each number of mappers considered.

The results obtained using the β-f-measure are equivalent to the ones discussed for the GM: there is overfitting for all the algorithms and being more obvious for the ROS method. The basic RF-BigData algorithm and its usage together with SMOTE provide poor results for the poker cases of study. The RF-BigDataCS is the best performing algorithm when a smaller number of mappers is used but it is outperformed by ROS when the number of mappers is 64. RF-BigDataCS and RUS offer worse results when the number of mappers is increased in contrast with ROS that obtain better performance results in that case.

To sum up, in this study we have tested the different approaches developed in this work over a wide range of cases of study that have helped us to have an insight into imbalanced big data. There is not a clear winner, however, we would like to highlight three of the methods tested: the RF-BigDataCS approach, ROS and RUS. The RUS method is very effective when dealing with a small number of mappers while ROS is very useful when larger values of mappers are used because it is able to come up against the small sample size problem. The cost-sensitive learning is not as good as ROS when a high number of mappers is used but it provides a good trade-off when smaller number of mappers are considered.

In the contrary case, the SMOTE algorithm, which is one of the best performing in the imbalanced scenario, has provided poor results in several of the cases considered. These results suggest that another view should be taken to generate synthetic minority samples in the imbalanced big data scenario.

Table 11
Average β-f-measure results for the MapReduce RF versions using 8, 16, 32 and 64 mappers on the imbalanced big data poker cases.

Dataset	Average (poker)							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	β-f-m$_{tr}$	β-f-m$_{tst}$	β-f-m$_{tr}$	β-f-m$_{tst}$	β-f-m$_{tr}$	β-f-m$_{tst}$	β-f-m$_{tr}$	β-f-m$_{tst}$
Big data versions								
RF-BigData	0.068	0.031	0.019	0.014	0.008	0.006	0.003	0.002
RF-BigDataCS	0.990	**0.881**	0.965	**0.842**	0.911	**0.787**	0.803	0.709
ROS + RF-BigData	0.965	0.624	0.976	0.721	0.946	0.739	0.957	**0.808**
RUS + RF-BigData	0.919	0.840	0.867	0.816	0.821	0.781	0.771	0.753
SMOTE + RF-BigData	0.203	0.200	0.200	0.200	0.194	0.191	0.043	0.034

5.4. Analysis of the runtime of the diverse approaches for imbalanced big data

One of the main issues about using distributed and parallel implementations is the obtaining of better runtime results. In this section we compare the runtime between all the approaches for imbalanced big data for the different problems selected and the diverse number of mappers used in the experiments. We present the average results for the runtime in a similar manner to the analysis of the precision given in Section 5.3, presenting in a first group the results for the cases of study based on the kddcup dataset; then, the runtime associated to the RLCP dataset; and finally, the cases of study based on the poker dataset are shown.

Table 12 shows the average time elapsed in seconds and in the hh:mm:ss.SS format for the cases of study derived from the kddcup dataset on the MapReduce algorithms selected for imbalanced big data with 8, 16, 32 and 64 mappers respectively. Fig. 12 represents this information using a logarithmic scale. For this and future tables we use the hh:mm:ss.SS format: we show the hours spent using the *hh* symbol, the minutes needed with the *mm* symbol, the seconds are represented through the *ss* digits, and the *SS* token represents the centiseconds elapsed. The bold values highlight the quickest algorithm for each number of mappers considered.

Considering these results, we can see that the runtime spent is directly related to the operations that need to be performed by the different approaches. The quickest method is the basic RF-BigData approach, followed in speed by the RF-BigDataCS method. Then, RUS is the next method considering the execution times, pursued by ROS and SMOTE.

When we observe the performance of the methods considering the different number of mappers, we can detect three different groups of behavior: one for the RUS method; another for RF-BigData, RF-BigDataCS and ROS; and a third one for SMOTE.

The RUS method seems to have stable runtimes with respect to the number of mappers in the experiments. In this case, it is very difficult to find a speed gain when larger values for the number of mappers are used.

RF-BigData, RF-BigDataCS and ROS runtime slowly descends when we increase the number of mappers. However, this speed gain is not equally distributed. When 16 mappers are tested against 8 mappers, the advance in the runtime can be appreciated, however, when 32 and 64 mappers are compared we cannot find differences in the time needed for the algorithms to run.

The SMOTE algorithm is the slowest algorithm considered for the kddcup cases of study. Even though it starts with slow runtimes in comparison with the other algorithms, the improvement that can be seen for the method when the number of mappers is augmented is notable for all the different number of mappers. Furthermore, we can also observe that the speed gain is less powerful when higher values of mappers are considered.

Table 13 displays in seconds and in the hh:mm:ss.SS format the runtime for the RLCP dataset using the MapReduce algorithms developed for imbalanced big data with 8, 16, 32 and 64 mappers respectively. Fig. 13 represents this information using a logarithmic scale.

When ordering the alternatives with respect to the runtime spent, we can see that the order obtained in the previous cases is maintained: the fastest method is RF-BigData, followed by RF-BigDataCS and RUS, which are pursued by ROS and SMOTE.

Table 12
Average runtime elapsed in seconds and in the hh:mm:ss.SS format for the MapReduce RF versions on the imbalanced big data kddcup cases.

Dataset	Average (kddcup)							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS
Big data versions								
RF-BigData	**169.03**	**0:02:49.03**	**122.91**	**0:02:02.91**	**110.29**	**0:01:50.29**	**110.37**	**0:01:50.37**
RF-BigDataCS	238.15	0:03:58.15	190.86	0:03:10.86	179.44	0:02:59.44	178.17	0:02:58.17
ROS + RF-BigData	440.86	0:07:20.86	336.26	0:05:36.26	318.17	0:05:18.17	300.97	0:05:00.97
RUS + RF-BigData	205.20	0:03:25.20	200.11	0:03:20.11	196.63	0:03:16.63	194.87	0:03:14.87
SMOTE + RF-BigData	6694.61	1:51:34.61	1863.29	0:31:03.29	679.38	0:11:19.38	445.00	0:07:25.00

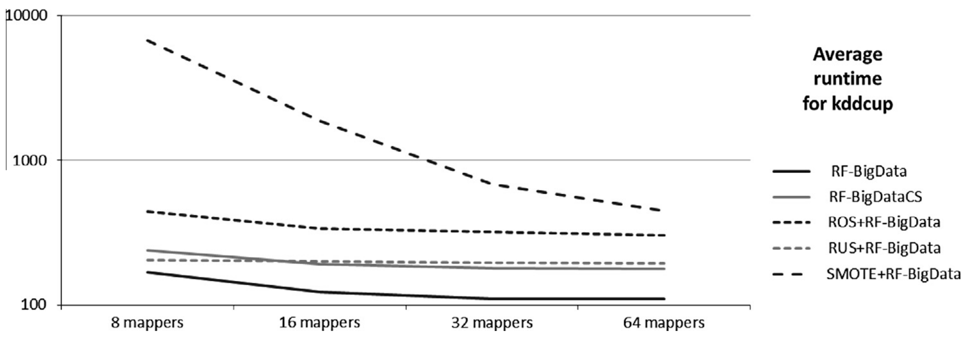


Fig. 12. Average runtime over the kddcup cases (logarithmic scale).

Table 13
Runtime elapsed in seconds and in the hh:mm:ss.SS format for the MapReduce RF versions on the RLCP dataset.

Dataset	RLCP							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS
Big data versions								
RF-BigData	**131.55**	**0:02:11.55**	**99.12**	**0:01:39.12**	**89.71**	**0:01:29.71**	**87.80**	**0:01:27.80**
RF-BigDataCS	179.04	0:02:59.04	140.18	0:02:20.18	128.34	0:02:08.34	127.45	0:02:07.45
ROS + RF-BigData	454.57	0:07:34.57	361.70	0:06:01.70	348.70	0:05:48.70	208.75	0:03:28.75
RUS + RF-BigData	140.50	0:02:20.50	136.40	0:02:16.40	136.34	0:02:16.34	130.48	0:02:10.48
SMOTE + RF-BigData	9684.96	2:41:24.96	1675.56	0:27:55.56	428.06	0:07:08.06	487.83	0:08:07.83

Looking at groups of behavior, we can find in this case four different attitudes: one for the RUS method; another for RF-BigData and RF-BigDataCS; a third one for ROS and a fourth for SMOTE.

The RUS method maintains an equivalent behavior to the one observed for kddcup: it has stable runtimes with respect to the number of mappers in the experiments. RF-BigData and RF-BigDataCS also preserve their behavior as their runtime improves its speed when the number of mappers is increased, however, that improvement is not much appreciated when 32 and 64 mappers are used with respect to the speed gain obtained when 16 mappers are considered.

The ROS method is the one that shows a continuous speed gain, however, it is more notable when higher values for the number of mappers are employed in contrast with RF-BigData and RF-BigDataCS.

The SMOTE algorithm again starts with slow runtimes in comparison with the other algorithms, the improvement that can be seen for the method when the number of mappers is enlarged is notable when 16 and 32 mappers are used, however, when 64 mappers are used the runtime suffers from a negative effect that increases the execution times.

Table 14 presents the average runtime for the poker cases of study in seconds and in the hh:mm:ss.SS format. This runtime is referred to the MapReduce algorithms developed for imbalanced big data and has been tested with 8, 16, 32 and 64 mappers respectively. Fig. 14 represents this information using a logarithmic scale.

The different approaches are positioned according to their runtime in a similar manner than the one observed in the previous groups: the fastest method is RF-BigData closely followed by RF-BigDataCS, then we find the RUS method and finally we encounter the ROS and SMOTE algorithms.

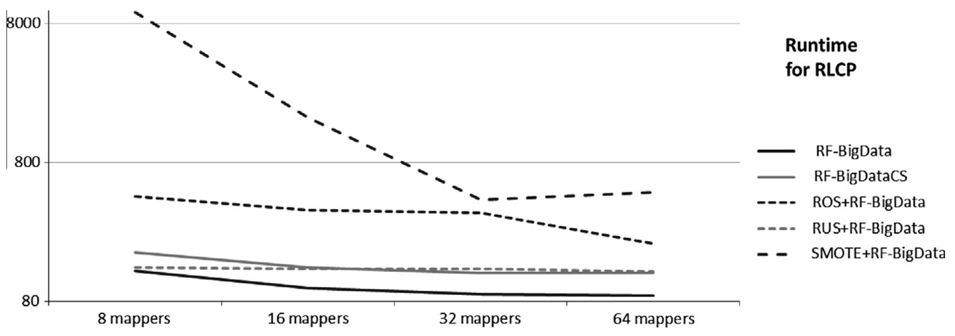


Fig. 13. Average runtime over the RLCP dataset (logarithmic scale).

Table 14
Average runtime elapsed in seconds and in the hh:mm:ss.SS format for the MapReduce RF versions on the imbalanced big data poker cases.

Dataset	Average (poker)							
	8 Mappers		16 Mappers		32 Mappers		64 Mappers	
	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS	seconds	hh:mm:ss.SS
Big data versions								
RF-BigData	**67.51**	**0:01:07.51**	66.12	0:01:06.12	**59.92**	**0:00:59.92**	**58.85**	**0:00:58.85**
RF-BigDataCS	70.41	0:01:10.41	**63.35**	**0:01:03.35**	60.23	0:01:00.23	59.10	0:00:59.10
ROS + RF-BigData	113.44	0:01:53.44	98.58	0:01:38.58	93.67	0:01:33.67	87.19	0:01:27.19
RUS + RF-BigData	80.70	0:01:20.70	78.36	0:01:18.36	76.17	0:01:16.17	72.81	0:01:12.81
SMOTE + RF-BigData	338.35	0:05:38.35	141.54	0:02:21.54	105.43	0:01:45.43	102.53	0:01:42.53

When we try to establish different patterns of behavior for the approaches, we are able to establish three groups as in the first case: one for the RUS method; another for RF-BigData, RF-BigDataCS and ROS; and a third one for SMOTE. Looking at groups of behavior, we can find in this case four different attitudes: one for the RUS method; another for RF-BigData and RF-BigDataCS; a third one for ROS and a fourth for SMOTE.

The RUS method maintains an equivalent behavior to the one previously observed: it has stable runtimes with respect to the number of mappers in the experiments. RF-BigData, RF-BigDataCS and ROS slightly improve their runtime as the number of mappers are enlarged, however, that improvement is not clearly shown when 32 and 64 mappers are used in contrast with the improvement observed when 16 mappers are considered. The SMOTE algorithm also starts with slow runtimes which enable it to obtain a higher speed gain for all the different number of mappers. However, this speed gain is less impressive as the used number of mappers is higher.

To summarize the findings of this runtime study we can say that the runtimes obtained by each algorithm are what are expected from them considering the operations that have to be computed in order to obtain a final result. The simplest approach, RF-BigData, is usually the fastest one, closely followed by RF-BigDataCS that only adds extra computation in its internal operations. The RUS method is also relatively quick and can be considered as fast as these simpler methods in general.

On the other hand, the oversampling approaches are the ones that present higher runtimes with respect to the other methods considered for handling imbalanced big data. ROS is a rather straightforward algorithm that does not spend much time preparing the input dataset for RF while the SMOTE algorithm needs to spend some time preparing the input data with some non-trivial operations, something that notably increases the runtime needed to complete an experiment.

We have also seen that when the number of mappers is increased it generally implies a reduction on the runtime. However, this reduction is not uniform for all the number of mappers, datasets and algorithms. The greatest diminution in the runtime is found for the slowest algorithms, the preprocessing methods, and for the larger datasets such as the cases of study based on kddcup. When we use fast methods over smaller problems the gain when using a larger number of mappers is usually imperceptible. That is clearly observed for the RUS algorithm, as speed gains are not observed in any cases. Furthermore, using larger values of the number of mappers does not usually imply great speed gains in most of the cases and smaller values for this parameter are preferred.

This situation arises due to the overhead introduced by the Hadoop environment when running the algorithms: when the total runtime is small, the time spent distributing the data and collecting the results around the processing nodes is quite noticeable with respect to the processing time spent in each node. That is why the speed gain is higher in the methods that need to perform more complex operations and in bigger datasets.

Finally, we selected as good approaches the RF-BigDataCS method together with the RUS and ROS algorithms, but we have found here that they have different behavior with respect to the runtime. In this manner, the RF-BigDataCS and RUS approaches demonstrated a good performance for a small or medium sized number of mappers because they suffer from the small sample size problem when the number of mappers is elevated. As these methods do not gain much in runtime by increasing the number of mappers it is best in these cases to use them with these number of mappers. On the other hand,

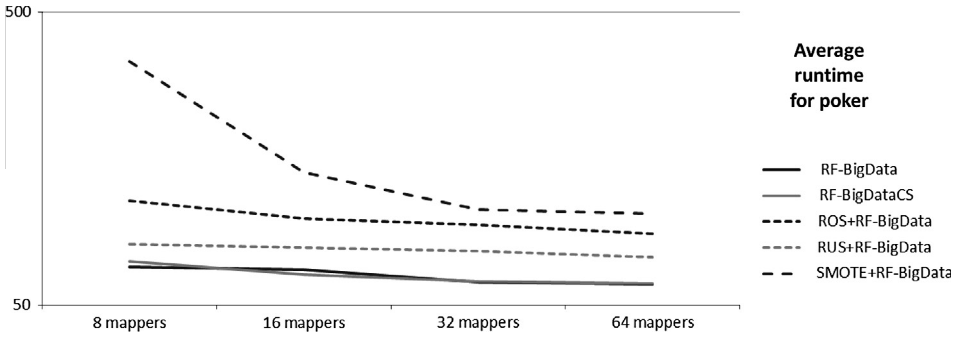


Fig. 14. Average runtime over the poker cases (logarithmic scale).

the ROS algorithm tends to obtain more precise results when a higher number of mappers are used. As it is a slower algorithm than the mentioned approaches and it is benefited by the runtime gain when larger values for the number of mappers are found, it would be a good idea to use it with a relatively high number of mappers.

6. Concluding remarks

In this work we have presented a comparison among several techniques for imbalanced big data. More specifically, we have contrasted different approaches for imbalanced classification, namely, ROS, RUS, the SMOTE algorithm and cost-sensitive learning which we have adapted to deal with big data. In order to do so, we used the RF algorithm as base, which is a well-known decision tree ensemble famous for its good performance.

Nowadays, big data is gaining recognition because of the large amounts of data that are currently generated. The traditional data mining approaches are not able to cope with the new requirements imposed by big data. In this manner, we use one of the most popular environment nowadays to deal with big data: the MapReduce framework. We use the Hadoop framework, which is the most popular open source implementation of MapReduce which facilitates the development of scalable and distributed solutions. Furthermore, one of the complications that difficult the extraction of useful information is the problem of classification with imbalanced datasets and the importance of this problem resides on its prevalence in numerous real-world applications. Big data is also affected by this kind of unequal distribution because of the variety and veracity of the collected information.

The results obtained show that the sequential versions are not an appropriate approach to deal with imbalanced big data and it is necessary to address those problems to provide appropriate solutions when the size of the data available is increased, such as the MapReduce approaches suggested in this work. The execution time is reduced typically when the number of mappers is increased, however, a too large number of mappers may cause a negative impact in the performance, due to the small sample size problem.

We have also found that there is not a best approach to deal with imbalanced big data when using the RF algorithm since it depends on the type of problem and the influence of the lack of density over the specific approach. We can highlight the poor classification performance of the SMOTE algorithm in the imbalanced big data problems considered when it is adapted for big data as it is considered one of the more competitive methods for imbalanced classification. Furthermore, we have also seen that the RUS version drops in classification performance to a greater extent than its competitors when increasing the number of mappers because it is affected in a greater degree by the small sample size problem.

On the other hand, the obtained results allow us to conclude that is needed to analyse the topic in more depth about the following challenges:

1. It is needed to determine the threshold for the minority class with respect to the number of mappers in order to find the best configuration that provides a better performance and lesser response time. Our results show that in the extremely imbalanced big data problems considered the average results are much lower and the small sample size is encountered.
2. Considering the results of the SMOTE algorithm, is necessary to design new techniques that are able to generate synthetic data that represent the minority class instances in the best way when a MapReduce framework is used.
3. It is necessary to analyse some data intrinsic characteristics that interact with this issue aggravating the problem in order to increase the performance with imbalanced big data. It is needed to pay special attention to the presence of noisy and borderline examples since these characteristics can difficult the imbalanced problem when the number of minority class examples included in a mapper's partition is too small; the induction of the small sample size problem over the selected datasets; the presence of small disjuncts in the data when the size of the original data is reduced; the overlapping between the classes which can be spread by the division of the original data in subsets and the dataset shift problem that is aggravated even more when increasing the number of splits.

In future studies, it is necessary to study in depth the described challenges so that the classification performance can be greatly improved with new approaches. Furthermore, it is also advisable to improve the current approaches considering the small sample size problem so that we can decrease the performance loss observed when using a higher number of mappers.

Acknowledgments

This work was partially supported by the Spanish Ministry of Science and Technology under project TIN2011-28488 and the Andalusian Research Plans P11-TIC-7765 and P10-TIC-6858. V. López holds a FPU scholarship from Spanish Ministry of Education.

References

[1] Apache Drill, 2013 <http://incubator.apache.org/drill/> (accessed December 2013).
[2] Apache Hadoop Project, Apache Hadoop, 2013 <http://hadoop.apache.org/> (accessed December 2013).
[3] Apache Mahout Project, Apache Mahout, 2013 <http://mahout.apache.org/> (accessed December 2013).

[4] K. Bache, M. Lichman, UCI Machine Learning Repository, 2013 <http://archive.ics.uci.edu/ml>.
[5] R. Barandela, J.S. Sánchez, V. García, E. Rangel, Strategies for learning in class imbalance problems, Pattern Recognit. 36 (3) (2003) 849–851.
[6] G.E.A.P.A. Batista, R.C. Prati, M.C. Monard, A study of the behaviour of several methods for balancing machine learning training data, SIGKDD Explor. 6 (1) (2004) 20–29.
[7] R. Batuwita, V. Palade, Adjusted geometric-mean: a novel performance measure for imbalanced bioinformatics datasets learning, J. Bioinform. Comput. Biol. 10 (4) (2012).
[8] M. Beyer, D. Laney, 3D Data Management: Controlling Data Volume, Velocity and Variety, 2001 <http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf> (accessed August 2013).
[9] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[10] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth and Brooks, 1984.
[11] C. Bunkhumpornpat, K. Sinapiromsaran, C. Lursinsap, Safe-level-SMOTE: safe-level-synthetic minority over-sampling TEchnique for handling the class imbalanced problem. in: Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining PAKDD'09, 2009, pp. 475–482.
[12] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[13] N.V. Chawla, D.A. Cieslak, L.O. Hall, A. Joshi, Automatically countering imbalance and its empirical relationship to cost, Data Min. Knowl. Discov. 17 (2) (2008) 225–252.
[14] C. Chen, A. Liaw, L. Breiman, Using Random Forest to Learn Imbalanced Data. Tech. Rep. 666, Statistics Department, University of California Berkeley, 2004.
[15] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[16] M. Denil, T. Trappenberg, Overlap versus imbalance, in: Proceedings of the 23rd Canadian conference on Advances in Artificial Intelligence (CCAI'10), vol. 6085 of Lecture Notes on Artificial Intelligence, 2010, pp. 220–231.
[17] P. Domingos, Metacost: a general method for making classifiers cost-sensitive, in: Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD'99), 1999, pp. 155–164.
[18] C. Elkan, The foundations of cost-sensitive learning, in: Proceedings of the 17th IEEE International Joint Conference on Artificial Intelligence (IJCAI'01), 2001, pp. 973–978.
[19] A. Fernández, V. López, M. Galar, M.J. del Jesus, F. Herrera, Analysing the classification of imbalanced data-sets with multiple classes: binarization techniques and ad-hoc approaches, Knowl.-Based Syst. 42 (2013) 97–110.
[20] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for class imbalance problem: bagging, boosting and hybrid based approaches, IEEE Trans. Syst., Man, Cyber. – Part C: Appl. Rev. 42 (4) (2012) 463–484.
[21] S. García, J. Derrac, I. Triguero, C.J. Carmona, F. Herrera, Evolutionary-based selection of generalized instances for imbalanced classification, Knowl. Based Syst. 25 (1) (2012) 3–12.
[22] S. García, F. Herrera, Evolutionary under-sampling for classification with imbalanced data sets: proposals and taxonomy, Evol. Comput. 17 (3) (2009) 275–306.
[23] V. García, R.A. Mollineda, J.S. Sánchez, On the k-NN performance in a challenging scenario of imbalance and overlapping, Pattern Anal. Appl. 11 (3–4) (2008) 269–280.
[24] R. Gupta, H. Gupta, M. Mohania, Cloud computing and big data analytics: what is new from databases perspective? in: Proceedings of the 1st International Conference on Big Data Analytics (BDA 2012), vol. 7678 of Lecture Notes on Computer Science, 2012, pp. 42–61.
[25] D.A. Hakim, PartialData MapReduce Random Forests, 2013 <http://cwiki.apache.org/confluence/display/MAHOUT/Partial+Implementation> (accessed December 2013).
[26] H. Han, W.Y. Wang, B.H. Mao, Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning, in: Proceedings of the 2005 International Conference on Intelligent Computing (ICIC'05), vol. 3644 of Lecture Notes in Computer Science, 2005, pp. 878–887.
[27] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[28] N. Japkowicz, S. Stephen, The class imbalance problem: a systematic study, Intell. Data Anal. J. 6 (5) (2002) 429–450.
[29] M. Kubat, S. Matwin, Addressing the curse of imbalanced training sets: one-sided selection, in: Proceedings of the 14th International Conference on Machine Learning (ICML'97), 1997, pp. 179–186.
[30] J. Laurikkala, Improving identification of difficult small classes by balancing class distribution, in: Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine (AIME'01), 2001, pp. 63–66.
[31] V. López, S. del Río, J. Benítez, F. Herrera, Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data, Fuzzy Sets Syst. (2014), http://dx.doi.org/10.1016/j.fss.2014.01.01 (in press).
[32] V. López, A. Fernández, M.J. del Jesus, F. Herrera, A hierarchical genetic fuzzy system based on genetic programming for addressing classification with highly imbalanced and borderline data-sets, Knowl.-Based Syst. 38 (2013) 85–104.
[33] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics, Inform. Sci. 250 (2013) 113–141.
[34] V. López, A. Fernández, J.G. Moreno-Torres, F. Herrera, Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. Open problems on intrinsic data characteristics, Exp. Syst. Appl. 39 (7) (2012) 6585–6608.
[35] S. Melnik, A. Gubarev, J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, Dremel: interactive analysis of web-scale datasets, in: Proceedings of the 36th International Conference on Very Large Data Bases, 2010, pp. 330–339.
[36] M. Minelli, M. Chambers, A. Dhiraj, Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses, John Wiley & Sons, 2013.
[37] D. Miner, A. Shook, MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems, O'Reilly Media, 2012.
[38] J.G. Moreno-Torres, T. Raeder, R. Aláiz-Rodríguez, N.V. Chawla, F. Herrera, A unifying view on dataset shift in classification, Pattern Recognit. 45 (1) (2012) 521–530.
[39] J. Nahar, T. Imam, K.S. Tickle, Y.-P.P. Chen, Computational intelligence for heart disease diagnosis: a medical knowledge driven approach, Exp. Syst. Appl. 40 (1) (2013) 96–104.
[40] K. Napierala, J. Stefanowski, S. Wilk, Learning from imbalanced data in presence of noisy and borderline examples, in: Proceedings of the 7th International Conference on Rough Sets and Current Trends in Computing (RSCTC'10), vol. 6086 of Lecture Notes on Artificial Intelligence, 2010, pp. 158–167.
[41] D. Newby, A.A. Freitas, T. Ghafourian, Coping with unbalanced class data sets in oral absorption models, J. Chem. Inform. Model. 53 (2) (2013) 461–474.
[42] A. Orriols-Puig, E. Bernadó-Mansilla, Evolutionary rule-based systems for imbalanced datasets, Soft Comput. 13 (3) (2009) 213–225.
[43] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, Manning Publications Co., 2012.
[44] B.-J. Park, S.-K. Oh, W. Pedrycz, The design of polynomial function-based neural network predictors for detection of software defects, Inform. Sci. 229 (2013) 40–57.
[45] S.J. Raudys, A.K. Jain, Small sample size effects in statistical pattern recognition: recommendations for practitioners, IEEE Trans. Pattern Anal. Mach. Intell. 13 (3) (1991) 252–264.
[46] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, A. Folleco, An empirical study of the classification performance of learners on imbalanced and noisy software quality data, Inform. Sci. 259 (2014) 571–595.
[47] Spark, 2013 <http://spark-project.org/> (accessed December 2013).
[48] Storm, 2013 <http://storm-project.net/> (accessed December 2013).
[49] Y. Sun, M.S. Kamel, A.K.C. Wong, Y. Wang, Cost-sensitive boosting for classification of imbalanced data, Pattern Recognit. 40 (12) (2007) 3358–3378.

[50] Y. Sun, A.K.C. Wong, M.S. Kamel, Classification of imbalanced data: a review, Int. J. Pattern Recognit. Artif. Intell. 23 (4) (2009) 687–719.
[51] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, H. Liu, Data warehousing and analytics infrastructure at facebook, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2010), 2010, pp. 1013–1020.
[52] I. Tomek, Two modifications of CNN, IEEE Trans. Syst. Man Commun. 6 (1976) 769–772.
[53] A. Verikas, A. Gelzinis, M. Bacauskiene, Mining data with random forests: a survey and results of new tests, Pattern Recognit. 44 (2) (2011) 330–349.
[54] M. Wasikowski, X.-W. Chen, Combating the small sample class imbalance problem using feature selection, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1388–1400.
[55] G.M. Weiss, Mining with rare cases, in: O. Maimon, L. Rokach (Eds.), The Data Mining and Knowledge Discovery Handbook, Springer, 2005, pp. 765–776.
[56] G.M. Weiss, The impact of small disjuncts on classifier learning, in: R. Stahlbock, S.F. Crone, S. Lessmann (Eds.), Data Mining, Annals of Information Systems, vol. 8, Springer, 2010, pp. 193–226.
[57] T. White, Hadoop, The Definitive Guide, O'Reilly Media, Inc., 2012.
[58] H. Yu, J. Ni, J. Zhao, ACOSampling: an ant colony optimization-based undersampling method for classifying imbalanced dna microarray data, Neurocomputing 101 (2013) 309–318.
[59] B. Zadrozny, C. Elkan, Learning and making decisions when costs and probabilities are both unknown, in: Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD'01), 2001, pp. 204–213.
[60] B. Zadrozny, J. Langford, N. Abe, Cost-sensitive learning by cost-proportionate example weighting, in: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03), 2003, pp. 435–442.
[61] L. Zhou, Performance of corporate bankruptcy prediction models on imbalanced dataset: the effect of sampling methods, Knowl.-Based Syst. 41 (2013) 16–25.