Available online at www.sciencedirect.com

ScienceDirect

Fuzzy Sets and Systems 258 (2015) 5–38

FUZZY sets and systems

www.elsevier.com/locate/fss

Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data

Victoria López *, Sara del Río, 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

Available online 24 February 2014

Abstract

Classification with big data has become one of the latest trends when talking about learning from the available information. The data growth in the last years has rocketed the interest in effectively acquiring knowledge to analyze and predict trends. The variety and veracity that are related to big data introduce a degree of uncertainty that has to be handled in addition to the volume and velocity requirements. This data usually also presents what is known as the problem of classification with imbalanced datasets, a class distribution where the most important concepts to be learned are presented by a negligible number of examples in relation to the number of examples from the other classes. In order to adequately deal with imbalanced big data we propose the Chi-FRBCS-BigDataCS algorithm, a fuzzy rule based classification system that is able to deal with the uncertainly that is introduced in large volumes of data without disregarding the learning in the underrepresented class. The method uses the MapReduce framework to distribute the computational operations of the fuzzy model while it includes cost-sensitive learning techniques in its design to address the imbalance that is present in the data. The good performance of this approach is supported by the experimental analysis that is carried out over twenty-four imbalanced big data cases of study. The results obtained show that the proposal is able to handle these problems obtaining competitive results both in the classification performance of the model and the time needed for the computation.

Keywords: Fuzzy rule based classification systems; Big data; MapReduce; Hadoop; Imbalanced datasets; Cost-sensitive learning

1. Introduction

The development and maturity of the information technologies has enabled an exponential growth on the data that is produced, processed, stored, shared, analyzed and visualized. According to IBM [1], in 2012, every day 1.5 quintillion bytes of data are created, which means that the 90% of the data created in the world has been produced in the last two years. Big data [2] encompass a collection of datasets whose size and complexity challenges the standard database management systems and defies the application of knowledge extraction techniques. This data

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

http://dx.doi.org/10.1016/j.fss.2014.01.015
0165-0114/

comes from a wide range of sources such as sensors, digital pictures and videos, purchase transactions, social media posts, everywhere [3].

This generation and collection of large datasets has further encouraged the analysis and knowledge extraction process with the belief that with more data available, the information that could be derived from it will be more precise. However, the standard algorithms that are used in data mining are not usually able to deal with these huge datasets [4]. In this manner, classification algorithms must be redesigned and adapted considering the solutions that are being used in big data so that they are able to be used under these premises maintaining its predictive capacity.

One of the complications that make difficult the extraction of useful information from datasets is the problem of classification with imbalanced data [5,6]. This problem occurs when the number of instances of one class (positive or minority class) is substantially smaller than the number of instances that belong to the other classes (negative or majority classes). The importance of this problem resides on its prevalence in numerous real-world applications such as telecommunications, finances, medical diagnosis and so on. In this situation, the interest of the learning is focused towards the minority class as it is the class that needs to be correctly identified in these problems [7]. Big data is also affected by this uneven class distribution.

Standard classification algorithms do not usually work appropriately when dealing with imbalanced datasets. The usage of global performance measures for the construction of the model and the search for the maximum generalization capacity induce in algorithms a mechanism that tends to neglect the rules associated with instances of the minority class.

Fuzzy Rule Based Classification Systems (FRBCSs) [8] are effective and accepted tools for pattern recognition and classification. They are able to obtain a good precision while supplying an interpretable model for the end user through the usage of linguistic labels. Furthermore, the FRBCSs can manage uncertainty, ambiguity or vagueness in a very effective way. This trait is especially interesting when dealing with big data, as uncertainty is inherent to this situation. However, when dealing with big data, the information at disposal usually contains a high number of instances and/or features. In this scenario the inductive learning capacity of FRBCSs is affected by the exponential growth of the search space. This growth complicates the learning process and it can lead to scalability problems or complexity problems generating a rule set that is not interpretable [9].

To overcome this situation there have been several approaches that aim to build parallel fuzzy systems [10]. These approaches can distribute the creation of the rule base [11] or the post-processing of the built model, using a parallelization to perform a rule selection [12] or a lateral tuning of the fuzzy labels [13]. Moreover, a fuzzy learning model can be completely redesigned to obtain a parallel approach that decreases the computation time needed [14]. However, these models aim to reduce the wait for a final classification without damaging the performance and are not designed to handle huge volumes of data. In this manner, it is necessary to redesign the FRBCSs accordingly to be able to provide an accurate classification in a small lapse of time from big data.

Numerous solutions have been proposed to deal with imbalanced datasets [7,15]. These solutions are typically organized in two groups: data-level solutions [16,17], which modify the original training set to obtain a more or less balanced class distribution that can be used with any classifier, and algorithm-level solutions, which alter the operations of an algorithm so that the minority class instances have more relevance and are correctly classified. Cost-sensitive solutions [18,19] integrate both approaches as they are focused in reducing the misclassification costs, higher for the instances of the minority class.

The approaches used to tackle big data usually involve some kind of parallelization to efficiently process and analyze all the available data. One of the most popular frameworks for big data, MapReduce [20], organizes the processing in two key operations: a map process that is responsible for dividing the original dataset and processing each chunk of information, and a reduce process that collects the results provided in the previous step and combines those results accordingly including new treatment if necessary. This approach that divides the original dataset in parts can have a strong pernicious effect when dealing with imbalanced datasets as the data intrinsic characteristics impact is amplified. Specifically, the small sample size [21] is induced when the original dataset is shared out and the dataset shift problem [22] may also be encouraged in the process. The addition of these problems reinforce the necessity of properly dealing with imbalanced datasets, not only for the original imbalance that is present in the data but also for the occasioned problems that arise when the partitions are created.

In this paper, we present a FRBCS that is capable of classifying imbalanced big data which has been denoted as Chi-FRBCS-BigDataCS. The method is based on the Chi et al.'s approach [23], a classical FRBCS learning method, which has been modified to deal with imbalanced datasets and big data at the same time. The usage of a FRBCS

enables the treatment of the uncertainty that is inherent to real-world problems and especially, in big data problems, as the variety and veracity of the collected information pose a serious source of uncertainty and vagueness in the data. Fuzzy rules have demonstrated to adequately manage the uncertainty in a reasonable manner and therefore, FRBCSs seem to be a sensible choice to overcome this situation. Furthermore, FRBCSs [24,25], and specifically the Chi et al.'s method [26,27], have also been successfully applied to imbalanced domains where they do not only combat the problem of an uneven class distribution but they also face up to the challenge of the uncertainty in the class frontiers which comes up because of the borderline samples [28], the noise in the data [29] and the small disjuncts [30] among others.

Furthermore, using the Chi et al.'s method helps the classification in big data as it is a model that shows some characteristics that make it especially suitable to build a parallel approach instead of using a more state-of-the-art FRBCS method. The Chi et al.'s method is a simple approach that does not have complex operations and strong interactions between parts of the algorithm. This behavior allows a division of the processing operations without deeply degrading the performance of the algorithm. Moreover, all the rules generated by the Chi et al.'s method have the same structure: rules with as many antecedents as attributes in the dataset that only use one fuzzy label. Maintaining a common structure for the rules enormously benefits the combination and aggregation of rules that were created in different parallel operations and it greatly reduces the processing time. Other state-of-the-art methods may create more accurate rule bases, however, the associated rules do not have a common design and then, grouping them together substantially complicates the learning.

To deal with imbalanced big data, the proposed Chi-FRBCS-BigDataCS algorithm modifies the basic FRBCS approach combining two approaches:

- To deal with big data, the FRBCS method has been adapted following the MapReduce principles that direct a distribution of the work on several processing units.
- To address the imbalance that is present in the data, some modifications induced by cost-sensitive learning have been applied to the model. The use of a cost-sensitive approach is appropriate in this case as it does not introduce intensive computation operations and not adding thus extra runtime to the final model. For this, we propose a new rule weight computation, the Penalized Cost-Sensitive Certainty Factor (PCF-CS), an approach based on the original Penalized Certainty Factor that takes into consideration the misclassification costs.

In order to assess the performance of the suggested approach, we have used twenty-four imbalanced big data cases of study that provide information about how the proposal works, its strengths and its limitations. The experimental study is organized to analyze the performance related to two types of measures: an evaluation on the classification performance, which is measured by a well-known metric in imbalanced classification, the Area Under the ROC Curve [31], and an examination on the runtime of the approaches tested.

This paper is arranged as follows. In Section 2 some background information about classification with big data and imbalanced datasets is given. Next, Section 3 introduces some basic concepts about FRBCSs, describes the Chi et al.'s algorithm, and presents a scalability study to show the unfeasibility of this algorithm for big data. Section 4 shows how the basic Chi et al.'s algorithm is modified to address imbalanced datasets including the information about the new rule weight computation, and replays the scalability study to demonstrate that big data needs to be specifically addressed. Then, Section 5 characterizes the Chi-FRBCS-BigDataCS approach to deal with big data. Section 6 indicates the configuration of the experimental study, the results obtained and a discussion about them. Finally, the conclusions achieved in this work are shown in Section 7.

2. Classification with big data and imbalanced datasets

In this section we present some background information about the specific related data problems that we are trying to clarify. In Section 2.1 we provide information about big data, its characteristics and some solutions that have been proposed to overcome this challenge. Then, in Section 2.2, an overview about classification with imbalanced datasets is supplied featuring a description of its traits, given solutions, which are the main threats to properly solve this problem and how the performance of algorithms is measured in this scenario.

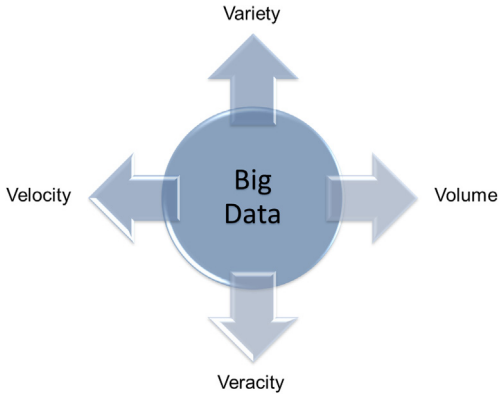

Fig. 1. The 4Vs of big data.

2.1. The difficulties of classification with big data

With the development of information technologies, organizations have had to face new challenges to analyze vast amounts of information. For this reason, the concept of "Big Data" is formulated, which is applied to all the information that cannot be processed or analyzed using traditional techniques or tools [32]. According to the definition given by the Gartner analyst Doug Laney in a 2001 MetaGroup research publication [33], we may describe big data as a 3Vs model (Volume, Velocity and Variety) [34,35]:

- **Volume:** It refers to the huge amount of data that needs to be processed, stored and analyzed.
- **Velocity:** It is an indication of how quickly the data needs to be analyzed so that it can provide an informed response.
- **Variety:** It is related to the different types of structured and unstructured data that organizations can accumulate such as tabular data (databases), hierarchical data, documents and e-mail, among others.

More recently, an additional V has been proposed by some organizations to describe the big data model [1] (Fig. 1): Veracity, which is an indication of data integrity and the trust on this information to make decisions. In this work we focus on effectively addressing the volume challenge, while trying to achieve reasonable results concerning the velocity model and also attempting to manage the uncertainty introduced by the variety and veracity.

These data volumes that we call big data are coming from different sources. For example, Facebook hosts approximately 10 billion photos, taking up one Petabyte of storage. The New York Stock Exchange generates about one Terabyte of new trade data per day, or the Internet Archive stores around 2 Petabytes of data, and is growing at a rate of 20 Terabytes per month [32].

Among the proposed solutions to the problem, one of the most popular approaches was proposed by Dean and Ghemawat, who worked at Google. They presented a parallel programming model, MapReduce, which is a framework for processing large volumes of data over a cluster of machines [20,36,37]. Generally, a MapReduce program contains two main phases: a map-function and a reduce-function. In the first phase, the input data is processed by the map-function, generating some intermediate results as the input of the reduce function in the second phase, which process the generated intermediate results to produce a final output.

Specifically, the MapReduce model is based on basic data structure which is the key-value pair, and all data processed in MapReduce is used in those key-value pair terms. In this manner, the map and reduce functions work as follows:

- **Map-function:** the master node performs a segmentation of the input dataset into independent blocks and distributes them to the worker nodes. Next, the worker node processes the smaller problem, and passes the answer back to its master node. In terms of key-value pairs, the map-function receives a key-value pair as input and emits a set of intermediate key-value pairs as output. Before the execution of a reduce function, the MapReduce library

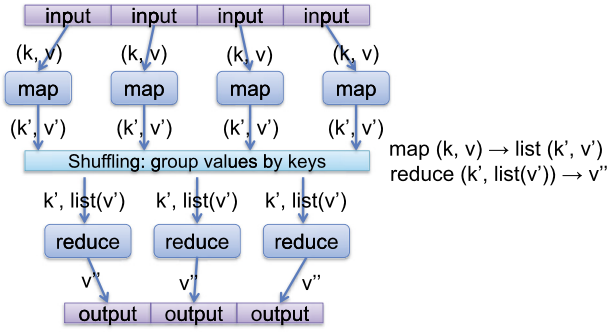

Fig. 2. The MapReduce programming model.

groups all intermediate values associated with the same intermediate key and transforms them to speed up the computation in the reduce function.

- **Reduce-function:** the master node collects the answers to all the sub-problems and combines them in some way to form the final output. Considering the key-value pairs, the reduce-function accepts the intermediate key provided by the MapReduce library and generates as final results the corresponding pair of key and value.

Fig. 2 depicts a typical MapReduce program with its map step and its reduce step. The terms k and v refer to the key and value pair respectively; k' and v' to the intermediate model and v'' to the output generated.

Apache Hadoop is the most popular implementation of the MapReduce programming model [32,38]. It is an open-source framework written in Java that supports the processing of large datasets in a distributed computing environment. Hadoop has a distributed file system, HDFS, that facilitates rapid data transfer rates among nodes and allows the system to continue operating uninterrupted in case of a node failure. The Apache Mahout project [39] is one of the most relevant tools that integrate machine learning algorithms in a Hadoop system.

However, following a MapReduce design is not always the best solution when dealing with big data [40]. Specifically, iterative algorithms are not able to obtain a good performance as they need to launch a MapReduce job for each iteration notably increasing the computation time due to the overhead. Therefore, there are some other open-source projects that are emerging to address big data as alternatives to MapReduce and Hadoop:

- **Spark [41]:** It is a cluster computing system that was developed in the UC Berkeley AMPLab and it is used to run large-scale applications such as spam filtering and traffic prediction. Spark provides primitives for in-memory cluster computing and APIs in Scala, Java and Python.
- **Apache Drill [42]:** It is a framework that supports data-intensive distributed applications for interactive analysis of large-scale datasets. Drill is a version of Google's Dremel system, which is a scalable, interactive ad-hoc query system for analysis of read-only nested data. Furthermore, its goal is to be able to scale to 10,000 servers or more and to be able to process Petabytes of data and trillions of records in seconds.

Some other incipient software projects are Twister [43], Ricardo [44], D3.js [45], HCatalog [46], Storm [47] or Impala [48], among others.

2.2. *Classification with imbalanced datasets*

Real-world classification problems typically present a class distribution where one or more classes have an insignificant number of examples in contrast with the number of examples from the other classes. This circumstance is known as the problem of classification with imbalanced datasets [5,6] and has been recognized as a challenge from the data mining community [49]. The main concern in this problem resides in the importance of the correct identification of the minority classes as they are the major focus of interest and their incorrect identification may entail high costs [18]. Imbalanced classification problems are found in diverse domains such as software defect prediction [50,51], finances [52], bioinformatics [53–55] and medical applications [56,57], just to mention some of them.

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)

Standard classification algorithms are usually unable to correctly deal with imbalanced datasets because they are built under the premise of obtaining the maximum generalization ability. In this manner, these algorithms try to obtain general rules that cover as many examples as possible, benefiting the majority class, while more specific rules that cover the minority class are discarded because of its small presence in the whole dataset. In this way, the minority class examples are treated like noise and therefore, these samples are finally neglected in the classification.

The imbalance ratio (IR) [58], which is the ratio of the number of instances in the majority class to the number of examples in the minority class $IR = \frac{\#maj\_class\_samples}{\#min\_class\_samples}$, is usually a clue to determine how difficult an imbalanced problem is. However, classification with imbalanced datasets is not only complicated by the dissimilar class distribution but also by some data intrinsic characteristics that interact with this issue aggravating the problem to a major extent than those difficulties in isolation [7]. Some of these data intrinsic characteristics include the presence of small disjuncts in the data [30], the small sample size for imbalanced classes [21], the overlapping between the classes [59], the presence of noisy [60] and borderline [61] examples and the dataset shift [22], which unites all the differences in the data distribution for the training and testing sets.

Big data techniques usually work in a parallel way dividing the original training set in subsets and distributing them along the processing units. This way of working is especially pernicious if the big data available is also imbalanced as it induces some of the aforementioned data problems: the small sample size problem and the dataset shift problem. In the first case, it is needed to establish a processing scheme that does not dramatically decrease the size of the new processed subsets. In the second case, a new subdivision of the dataset must be carefully done so that the subsets that are created for the training in each processing unit are as close as possible to the original training set. In this manner, we should avoid the prior probability shift [62], not changing the class distribution in the subsets, as well as the covariate shift [63], not changing the input attribute values distribution when the data portions are created.

Various approaches have been proposed to deal with imbalanced datasets [5–7,15]. These approaches are usually organized in two groups: *data-level* approaches and *algorithm-level* approaches. The data-level approaches [16,17] modify the original training set to obtain a more or less balanced distribution that is properly addressed by standard classification algorithms. This balancing process can be done adding examples to the minority class extending the dataset (over-sampling) or deleting examples from the majority class reducing the dataset (under-sampling). Algorithm-level approaches [25,64] adapt classification algorithms to guide the learning process towards the minority class. This adaptation can modify the inner way of working of an algorithm in favor of the minority class or it can even evidence the creation of new algorithms with this goal.

Additionally, cost-sensitive learning solutions include strategies at the data-level and the algorithm-level by considering variable misclassification costs for each class [19,65]. When dealing with imbalanced datasets it is more relevant to correctly classify minority instances than majority ones, and therefore, the cost associated to the misclassification of a minority instance should be higher than the cost associated to the contrary case: $Cost(min, maj) > Cost(maj, min)$. In this manner, cost-sensitive learning is either used as a direct approach that modifies how the algorithm works or is used as a meta-learning technique that modifies how the input or output information is processed [65,66]. Finally, another family of algorithms that has demonstrated a good behavior for imbalanced datasets is the ensembles of classifiers [67].

Selecting an appropriate performance measure is a vital decision when dealing with imbalanced datasets, not only to guide the construction of the model, but also to evaluate its achievement in comparison with other algorithms. The most used performance measure in classification, the overall classification accuracy, is not recommended when there is an uneven class distribution as it is biased towards the majority class: a classifier over a dataset with an IR of 9 that obtains a 90% of accuracy may not be a proper classifier as it may classify all the instances as belonging to the majority class, completely neglecting the minority class which is our interest in the problem.

In the imbalanced scenario, the evaluation of the classifiers performance should be computed considering specific metrics that observe the current class distribution. The confusion matrix (Table 1), which reports the results of correctly

or incorrectly classifying the examples of each class, leads to the obtaining of four metrics that describe both classes independently:

- **True positive rate** $TP_{rate} = \frac{TP}{TP+FN}$ is the percentage of positive instances correctly classified.
- **True negative rate** $TN_{rate} = \frac{TN}{FP+TN}$ is the percentage of negative instances correctly classified.
- **False positive rate** $FP_{rate} = \frac{FP}{FP+TN}$ is the percentage of negative instances misclassified.
- **False negative rate** $FN_{rate} = \frac{FN}{TP+FN}$ is the percentage of positive instances misclassified.

However, these measures are not satisfactory by themselves as we are seeking a good classification accuracy in both classes, and therefore, an approach to combine these measures is needed.

A graphical method that could be used to measure the performance of classification with imbalanced datasets is the Receiver Operating Characteristic (ROC) curve [68]. The ROC curve depicts the variation of the TP_{rate} against the FP_{rate} taking into account different decision threshold values. The Area Under the ROC Curve (AUC) metric [31] is able to provide a numerical performance measure that can be used to analyze the behavior of different learning algorithms. The AUC measure is computed obtaining the area of the ROC graphic. Specifically, we approximate this area following the next formula:

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} \tag{1}$$

3. Classification with fuzzy rule based classification systems: The Chi et al.'s algorithm and the scalability problem

This section purpose is to provide the information needed to explain the necessity of modifying traditional methods when building FRBCSs in imbalanced big data. As a basis for the approach, we will recall some elementary definitions about FRBCSs in Section 3.1. Then, we will present the FRBCS that has been used to construct our approach, the Chi et al.'s algorithm in Section 3.2. Finally, we will show a scalability study in Section 3.3 that demonstrates the requirement of effectively addressing big data.

3.1. Fuzzy rule based classification systems

Among the diverse techniques that are used to deal with classification problems in data mining, FRBCSs are widely used because they produce an interpretable model with a reasonable prediction rate.

A FRBCS is formed of two main components: the knowledge base (KB) and the inference system. In a linguistic FRBCS, the KB is built from the rule base (RB) and the data base (DB). The RB contains all the rules that compose the model and the DB encodes the membership functions associated to the fuzzy data partitions that are related to the input attribute values. The inference system directs the way in which new examples are classified considering the information stored in the KB. The most advantageous situation arises when expert information is available, however, this is very unusual and automatic learning methods to build the KB are needed.

Let m be the number of training patterns $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn})$ from C classes that form a classification problem, being x_{pi} is the ith attribute value ($i = 1, 2, \ldots, n$) of the p-th training pattern.

In this work, we use fuzzy rules of the following form to build our classifier:

$$\text{Rule } R_j: \text{ If } x_1 \text{ is } A_j^1 \text{ and } \ldots \text{ and } x_n \text{ is } A_j^n \text{ then Class} = C_j \text{ with } RW_j \tag{2}$$

where R_j is the label of the jth rule, $\mathbf{x} = (x_1, \ldots, x_n)$ is an n-dimensional pattern vector, A_j^i is an antecedent fuzzy set, C_j is a class label, and RW_j is the rule weight [69]. We use triangular membership functions as linguistic labels.

Numerous heuristics have been proposed to compute the rule weight [69]. A good choice for the computation of the rule weight is the Penalized Certainty Factor (PCF) [70], showed in Eq. (3):

$$RW_j = PCF_j = \frac{\sum_{x_p \in ClassC_j} \mu_{A_j}(x_p) - \sum_{x_p \notin ClassC_j} \mu_{A_j}(x_p)}{\sum_{p=1}^{m} \mu_{A_j}(x_p)} \tag{3}$$

where $\mu_{A_j}(x_p)$ is the matching degree of the pattern x_p with the antecedent part of the fuzzy rule R_j. We use the fuzzy reasoning method (FRM) of the wining rule [71], a classical approach, for the classification of new patterns by the RB. When a new pattern $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn})$ needs to be classified, the winner rule R_w is decided as the rule verifying:

$$\mu_w(x_p) \cdot RW_w = \max\left\{\mu_j(x_p) \cdot RW_j;\ j = 1 \ldots L\right\} \tag{4}$$

The pattern $\mathbf{x}_p$ is classified as class C_w which is the class indicated in the consequent of the winner rule R_w. In the case where several rules obtain the same maximum value in Eq. (4) for the example $\mathbf{x}_p$ but with different classes on the consequent, the classification of the pattern $\mathbf{x}_p$ is rejected and therefore, no class is assigned to it. Similarly, if the example $\mathbf{x}_p$ does not match any rule in the RB, the classification is also rejected and no class is given to the example.

3.2. The Chi et al.'s algorithm for classification

As a base for our FRBCS for imbalanced big data, we have used a simple learning procedure to generate the KB. Specifically, we have considered the method described in [23], that we have called the Chi et al.'s rule generation method or Chi-FRBCS, which is an extension for classification problems of the well-known Wang and Mendel algorithm [72].

To build the KB, this FRBCS method tries to find the relationship between the variables of the problem and constitute an association between the domain of features and the domain of classes following the next steps:

1. *Establishment of the linguistic partitions:* Using the range of values for each attribute A_i, the linguistic fuzzy partitions that form the DB are computed with the same number of linguistic terms for all input variables, composed of symmetrical triangular-shaped and uniformly distributed membership functions.
2. *Generation of a fuzzy rule for each example* $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn}, C_p)$*:* From each example present in the training set, a new fuzzy rule is created following the subsequent steps:
   (a) To compute the matching degree $\mu(\mathbf{x}_p)$ of the example with the different linguistic fuzzy labels for each attribute using a conjunction operator (represented with a T-norm operator).
   (b) To assign the example $\mathbf{x}_p$ to the different linguistic fuzzy labels that obtain the largest membership degree.
   (c) To generate a rule for the example $\mathbf{x}_p$. This rule will have as antecedent the linguistic fuzzy labels computed in the previous step and as consequent the class associated to the example C_p.
   (d) To compute the rule weight.

This procedure can generate several rules with the same antecedent. If the consequent of those rules belongs to the same class then, the replicated rules are deleted. However, if the consequent of those rules belongs to different classes then, only the rule with the highest weight is maintained in the RB.

3.3. Testing the scalability of the Chi-FRBCS algorithm

At this point, we want to test how the Chi-FRBCS algorithm is able to deal with huge amounts of data running a scalability test over the KDD Cup 1999 dataset from the UCI dataset repository [73]. The KDD Cup 1999 dataset features multiple classes while in our imbalanced scenario we are interested in problems with two classes. To test the Chi-FRBCS algorithm we have created several two-class big data cases of study derived from the KDD Cup 1999 dataset: specifically, the generated versions of the dataset use the *normal* and *DOS* connections as majority classes and the rest of attacks have been considered as minority classes. From these two-class datasets, we have created several imbalanced big data cases of study derived from it that differ in their size. From all the KDD Cup 1999 combinations we have selected three imbalanced big data cases of study that will be compared selecting only a percentage of samples from the original set maintaining the a priori probability between the classes. The percentage of the instances considered are the 10%, 25%, 40%, 50%, 60% and 75% and the experiments were run following a 5-fold stratified cross validation partitioning scheme. Further information about how the two-class sets are built can be found in Section 6.1.

Table 2 shows the information about the cases of study considered together with the average results in training and test for them. This table is divided by columns in four parts: the first three columns correspond to, for each case of

Table 2
Average results for the Chi-FRBCS algorithm for the imbalanced big data cases of study using the AUC measure, number of rules and time elapsed.

Datasets	#Atts.	#Ex.	#Class(maj; min)	Chi-FRBCS				
				AUC_{tr}	AUC_{tst}	numRules	Runtime (s)	Runtime (hh:mm:ss.SSS)
kddcup_10_normal_versus_R2L	41	97 390	(97 278; 112)	0.5000	0.5000	131.6	1578.991	00:26:18.991
kddcup_25_normal_versus_R2L	41	243 476	(243 195; 281)	0.5036	0.5000	178.4	10 327.567	02:52:07.567
kddcup_40_normal_versus_R2L	41	389 562	(389 112; 450)	0.5047	0.5000	200.2	28 329.681	07:52:09.681
kddcup_50_normal_versus_R2L	41	486 953	(486 390; 563)	0.5062	0.5044	213.4	40 170.131	11:09:30.131
kddcup_60_normal_versus_R2L	41	584 343	(583 668; 675)	0.5046	0.5007	226.4	57 060.828	15:51:00.828
kddcup_75_normal_versus_R2L	41	730 429	(729 585; 844)	0.5067	0.5047	240.0	85 336.009	23:42:16.009
kddcup_full_normal_versus_R2L	41	973 907	(972 781; 1126)	0.5083	0.5030	219.2	174 285.276	48:24:45.276
kddcup_10_DOS_versus_R2L	41	388 449	(388 337; 112)	1.0000	0.9897	70.0	25 498.727	07:04:58.727
kddcup_25_DOS_versus_R2L	41	971 123	(970 842; 281)	0.9697	0.9645	79.0	141 280.704	39:14:40.704
kddcup_40_DOS_versus_R2L	41	1 553 798	(1 553 348; 450)	ND	ND	ND	ND	ND
kddcup_50_DOS_versus_R2L	41	1 942 248	(1 941 685; 563)	ND	ND	ND	ND	ND
kddcup_60_DOS_versus_R2L	41	2 330 697	(2 330 022; 675)	ND	ND	ND	ND	ND
kddcup_75_DOS_versus_R2L	41	2 913 371	(2 912 527; 844)	ND	ND	ND	ND	ND
kddcup_full_DOS_versus_R2L	41	3 884 496	(3 883 370; 1126)	ND	ND	ND	ND	ND
kddcup_10_DOS_versus_normal	41	485 615	(388 337; 97 278)	0.9973	0.9972	162.2	32 892.936	09:08:12.936
kddcup_25_DOS_versus_normal	41	1 214 037	(970 842; 243 195)	0.9973	0.9973	218.8	267 496.363	74:18:16.363
kddcup_40_DOS_versus_normal	41	1 942 460	(1 553 348; 389 112)	ND	ND	ND	ND	ND
kddcup_50_DOS_versus_normal	41	2 428 075	(1 941 685; 486 390)	ND	ND	ND	ND	ND
kddcup_60_DOS_versus_normal	41	2 913 690	(2 330 022; 583 668)	ND	ND	ND	ND	ND
kddcup_75_DOS_versus_normal	41	3 642 112	(2 912 527; 729 585)	ND	ND	ND	ND	ND
kddcup_full_DOS_versus_normal	41	4 856 151	(3 883 370; 972 781)	ND	ND	ND	ND	ND

study, the number of attributes (#Atts.), number of examples (#Ex.) and number of instances for each class (minority and majority). The fourth column is devoted to the results of the Chi-FRBCS algorithm. The results for that algorithm are organized in the following way: the first two columns correspond to the AUC average results in training and test, the third column shows the average number of rules created by the FRBCS and the fourth and fifth columns present the average response times in seconds and in the hh:mm:ss.SSS format. Please note, that the hh:mm:ss.SSS format stands for the hours, minutes, seconds and milliseconds spent in the computation. For each dataset we consider the average results of the partitions.

Analyzing the results we can observe the *ND* (Not Determinable) symbol, which indicates that the algorithm was not able to complete the experiment. The implementation tested has not been especially prepared for huge datasets and the appearance of the *ND* symbol means that the current algorithm cannot be scaled for big data, as it is not able to deal with datasets this size.

For example, for the dataset *kddcup_normal_versus_R2L*, the smallest one considered in this test, we can see that the algorithm was able to provide results for all the versions of the problem. The results in training and test do not provide huge differences between the different reduced versions while we are able to observe an increment in the number of rules and in the processing time as more data is available.

For the larger datasets, *kddcup_DOS_versus_R2L* and *kddcup_DOS_versus_normal*, we can observe that the reduced versions of the datasets which were not able to finish have considerably increased from the previous case as their size is more than four times the size of the *kddcup_normal_versus_R2L* dataset. Specifically, the Chi-FRBCS algorithm was not able to complete the experiment starting from the 40% reduced version of the *kddcup_DOS_versus_R2L* and the *kddcup_DOS_versus_normal* cases of study, and for the 25% versions, the elapsed time is huge in relation with the elapsed time for the 10% versions.

Furthermore, we could be tempted to address big data just reducing the size of the original training set so that the current model is able to provide a result; moreover, when the results obtained by the 10% reduced version provide a reasonable performance. However, the reduction in the dataset is not only performed in the training set but also in the test set which alters the conclusions we can extract. In [74], we can observe a set of experiments that are related to the training of a FRBCS with different versions of the same dataset reducing its size. Their findings showed that the performance in test (which was maintained) was truly affected by the usage of different training sets.

In this manner, we can conclude that the basic Chi-FRBCS is not an appropriate approach to address imbalanced big data and it is necessary to specifically address those problems to provide a FRBCS that is able to provide proper classification results in a sensible time.

4. The Chi et al.'s algorithm for classification with imbalanced datasets and the scalability problem

In this section we provide some knowledge about how the basic Chi-FRBCS model can be modified to be able to address imbalanced problems. First, in Section 4.1, we will present a proposal to improve the classification in this arduous scenario presenting an approach that uses a new rule weight computation based on the PCF. Then, in Section 4.2, we perform again a scalability study to show that the modifications introduced are adequate to deal with imbalanced data but they are not enough to effectively address imbalanced big data.

4.1. The Chi et al.'s algorithm for classification with imbalanced datasets: using the penalized cost-sensitive certainty factor

As stated in the previous section, we have selected as basis for our FRBCS for imbalanced big data the Chi-FRBCS method [23]. This procedure creates a KB that is able to perform reasonably well in a more or less balanced situation; however, the Chi-FRBCS does not perform properly when classifying imbalanced datasets [26]. To accurately deal with imbalanced datasets we need to modify the previous proposal using cost-sensitive learning so that it considers during the building of the model the different misclassification costs associated to the various examples. In this manner, the learning will be biased to better identify the instances of the minority class. This proposal will be called Chi-FRBCS-CS.

Chi-FRBCS-CS follows the same set of steps as Chi-FRBCS changing how the rule weights are computed. Specifically, using the PCF heuristic, we have included the misclassification costs in the rule weight developing the Penalized Cost-Sensitive Certainty Factor (PCF-CS). In this way, the PCF-CS is computed as:

$$\mathrm{RW}_j = \mathrm{PCF\text{-}CS}_j = \frac{\sum_{x_p \in ClassC_j} \mu_{A_j}(x_p) \cdot Cs(C_p) - \sum_{x_p \notin ClassC_j} \mu_{A_j}(x_p) \cdot Cs(C_p)}{\sum_{p=1}^{m} \mu_{A_j}(x_p) \cdot Cs(C_p)} \quad (5)$$

where $Cs(C_p)$ is the misclassification cost associated to class C_p, the class of the example $\mathbf{x}_p$.

The misclassification costs associated to any class should be given by the experts if knowledgeable information about the problem is available. Unfortunately, this situation is very rare and therefore, we need to establish a procedure to estimate these costs. In our approach we have selected the costs in the following way: $Cs(min, maj) = IR$ and $Cs(maj, min) = 1$. As requested in imbalanced datasets, the misclassification cost for the minority class is much higher than the misclassification cost associated to the majority class. Additionally, as the cost is dependent on the proportion between the majority and minority instances, this estimation is valid for datasets that range from a low imbalance level to extremely imbalanced datasets.

4.2. Testing the scalability of the Chi-FRBCS-CS algorithm

At this point, we want to reproduce the scalability test performed for the Chi-FRBCS-CS algorithm in order to test how the proposal works in imbalanced big data problems considering their size. In this manner, we use the same cases of study as in Section 3.3, the two-class variants of the KDD Cup 1999 dataset that were sampled at the 10%, 25%, 40%, 50%, 60% and 75% of its size. Table 3 shows the average results in training and test for the three selected imbalanced datasets for the Chi-FRBCS and Chi-FRBCS-CS algorithms. We include both algorithms to check the differences in behavior between them.

When comparing both approaches we can see that there are not many differences between both Chi-FRBCS versions and that the conclusions extracted for Chi-FRBCS can also be applied to Chi-FRBCS-CS. Specifically, we can recognize the presence of the *ND* symbol also for the Chi-FRBCS-CS algorithm and that it appears for the same cases of study where Chi-FRBCS has it. For instance, the *kddcup_normal_versus_R2L* dataset is processed in all cases while the larger datasets, *kddcup_DOS_versus_R2L* and *kddcup_DOS_versus_normal*, are only able to produce results when the smallest versions of the datasets are considered. In this manner, it can be inferred that the new approach for imbalanced datasets does not improve its behavior with respect to the dataset size.

When considering the AUC results in training and test, it can be detected a much better performance for the Chi-FRBCS-CS algorithm. This better results can be examined in the *kddcup_normal_versus_R2L* dataset where the AUC values experiment a greater improvement, going from a situation where the minority class is not properly identified to a situation where this minority instances are generally considered. This behavior can be seen in the different cases of study considered and does not depend on the data size. In the case of the *kddcup_DOS_versus_R2L* and *kddcup_DOS_versus_normal* datasets, the improvement is not as noticeable, however, the tendency to slightly improve the results is clear.

Viewing the number of rules generated by both approaches, the Chi-FRBCS-CS is the one that creates a model with the lesser number of rules. Regarding the time elapsed to complete the experiments, we can see that there is not a clear tendency between the two Chi-FRBCS versions. Even when they are able to provide results in the same cases, the time needed to finish the computation does not always benefit one algorithm over the other, which means that the calculation of the PCF-CS does not clearly increase the computation time needed while benefiting the classification performance.

Finally, we can conclude that the Chi-FRBCS-CS method is a step forward to deal with imbalanced datasets however, it is necessary to specifically address big data using techniques that have been designed to manage huge datasets, as standard learning algorithms have not been adapted to learn in this arduous situation.

5. The Chi-FRBCS algorithm for imbalanced big data: A MapReduce design

In this section, we will describe our proposal of a FRBCS for imbalanced big data, denoted as Chi-FRBCS-BigDataCS. This proposal is introduced in the following way: Section 5.1 presents a general overview of how the Chi-FRBCS algorithm is adapted for big data. Next, in Section 5.2, the building of the model is detailed. Later, Section 5.3 describes how the instances of a big dataset are classified considering the learned model. Finally, Section 5.4 presents a case of study over one of the imbalanced big data problems considered.

Table 3
Average results for the sequential Chi-FRBCS and Chi-FRBCS-CS versions for the imbalanced big data cases of study using the AUC measure, number of rules and time elapsed.

Datasets	#Atts.	#Ex.	#Class(maj; min)	Chi-FRBCS					Chi-FRBCS-CS				
				AUC_{tr}	AUC_{tst}	numRules	Runtime (s)	Runtime (hh:mm:ss.SSS)	AUC_{tr}	AUC_{tst}	numRules	Runtime (s)	Runtime (hh:mm:ss.SSS)
kddcup_10_normal_versus_R2L	41	97 390	(97 278; 112)	0.5000	0.5000	131.6	**1578.991**	**00:26:18.991**	0.9729	**0.9499**	**119.0**	1599.831	00:26:39.831
kddcup_25_normal_versus_R2L	41	243 476	(243 195; 281)	0.5036	0.5000	178.4	10 327.567	02:52:07.567	0.9629	**0.9563**	**160.4**	**8426.257**	**02:20:26.257**
kddcup_40_normal_versus_R2L	41	389 562	(389 112; 450)	0.5047	0.5000	200.2	28 329.681	07:52:09.681	0.9637	**0.9587**	**180.4**	**21 274.452**	**05:54:34.452**
kddcup_50_normal_versus_R2L	41	486 953	(486 390; 563)	0.5062	0.5044	213.4	**40 170.131**	**11:09:30.131**	0.9649	**0.9625**	**195.0**	40 877.748	11:21:17.748
kddcup_60_normal_versus_R2L	41	584 343	(583 668; 675)	0.5046	0.5007	226.4	**57 060.828**	**15:51:00.828**	0.9634	**0.9597**	**205.6**	58 008.036	16:06:48.036
kddcup_75_normal_versus_R2L	41	730 429	(729 585; 844)	0.5067	0.5047	240.0	85 336.009	23:42:16.009	0.9657	**0.9638**	**218.8**	**84 191.977**	**23:23:11.977**
kddcup_full_normal_versus_R2L	41	973 907	(972 781; 1126)	0.5083	0.5030	219.2	**174 285.276**	**48:24:45.276**	0.9653	**0.9620**	**199.4**	176 795.885	49:06:35.885
kddcup_10_DOS_versus_R2L	41	388 449	(388 337; 112)	1.0000	**0.9897**	70.0	25 498.727	07:04:58.727	0.9999	0.9897	**64.6**	**25 448.700**	**07:04:08.700**
kddcup_25_DOS_versus_R2L	41	971 123	(970 842; 281)	0.9697	0.9645	79.0	141 280.704	39:14:40.704	0.9981	**0.9928**	**73.8**	**136 368.526**	**37:52:48.526**
kddcup_40_DOS_versus_R2L	41	1 553 798	(1 553 348; 450)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_50_DOS_versus_R2L	41	1 942 248	(1 941 685; 563)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_60_DOS_versus_R2L	41	2 330 697	(2 330 022; 675)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_75_DOS_versus_R2L	41	2 913 371	(2 912 527; 844)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_full_DOS_versus_R2L	41	3 884 496	(3 883 370; 1126)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_10_DOS_versus_normal	41	485 615	(388 337; 97 278)	0.9973	0.9972	162.2	**32 892.936**	**09:08:12.936**	0.9975	**0.9974**	**160.8**	33 670.214	09:21:10.214
kddcup_25_DOS_versus_normal	41	1 214 037	(970 842; 243 195)	0.9973	0.9973	218.8	**267 496.363**	**74:18:16.363**	0.9979	**0.9978**	**216.6**	273 740.590	76:02:20.590
kddcup_40_DOS_versus_normal	41	1 942 460	(1 553 348; 389 112)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_50_DOS_versus_normal	41	2 428 075	(1 941 685; 486 390)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_60_DOS_versus_normal	41	2 913 690	(2 330 022; 583 668)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_75_DOS_versus_normal	41	3 642 112	(2 912 527; 729 585)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
kddcup_full_DOS_versus_normal	41	4 856 151	(3 883 370; 972 781)	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND

5.1. General overview of the Chi-FRBCS algorithm for big data

The Chi-FRBCS-BigDataCS algorithm is an approach that can be used to classify imbalanced big data. It is a MapReduce design where each map process is responsible for building a RB using only the data included in its portion and where the reduce process is responsible for collecting and combining the RB generated by each mapper to form the final RB.

We will divide the description of the proposal in two parts: the first part is devoted to the description of the creation of the model, shown in Section 5.2, and the second part is dedicated to the explanation on how new instances are classified using the previous learned model, in Section 5.3. Both parts follow the MapReduce structure distributing all the computations needed along several processing units that manage different chunks of information, aggregating the results obtained in an appropriate manner.

In this description, we do not make a distinction between the steps that need to be followed to create a "normal" model that is able to process big data based on the Chi-FRBCS algorithm, Chi-FRBCS-BigData, and the steps needed to transform this model into our proposal, Chi-FRBCS-BigDataCS, based on the Chi-FRBCS-CS model. The differences in the computation of both models are related to the computation of the rule weight, as stated in Section 4.1, sharing most of the algorithm structure. In this manner, the transition to a big data model follows similar steps and only the variances associated to the cost-sensitive model will be stated when applicable.

The model presented is a FRBCS built on MapReduce using cost-sensitive learning for the following reasons:

- A FRBCS is able to deal with the uncertainty and imprecise information that emanates from big data, as those huge information sources become available from diverse sources that include a high variety while trying to cope with the veracity and trust on the data.
- The MapReduce framework is one of the most currently known alternatives to handle big data and has demonstrated that is capable to perform reasonably well in data mining problems producing even libraries like Mahout that include machine learning and data mining algorithms.
- In cost-sensitive learning, the addition of costs into the algorithm way of working does not heavily increase the time complexity while properly managing the imbalanced problem.

Finally, we have preferred the use of cost-sensitive learning instead of data preprocessing techniques to avoid an extra step in the building of the model following a MapReduce design. Over-sampling techniques would increase the size of the data to process, therefore increasing the computational needs, while under-sampling may disregard potentially useful examples which could be underestimated because of the subdivision induced by the MapReduce structure.

5.2. Building the knowledge base for the Chi-FRBCS-BigDataCS using a MapReduce design

In this section, we will describe how the KB is built from the original training set provided following a MapReduce procedure. This process is illustrated in Fig. 3 and consists of the following steps:

- *Initial:* In the CS version, the first step needs to estimate the costs for each class giving the minority class a greater cost than the majority class. This cost is estimated in the same way as described in Section 4.1, giving a misclassification cost of 1 for instances belonging to the majority class and a misclassification cost of IR for instances of the minority class.
  Next, in both versions of the algorithm, the domain of variation of each feature in the dataset is determined. Then, the different fuzzy membership labels that compose the DB are computed using these domains according to the number of labels considered.
  Finally, in order to comply with Hadoop way of working, the algorithm performs a segmentation of the training dataset into independent HDFS blocks. These blocks are then automatically replicated and transferred between the different cluster nodes thanks to the Hadoop environment that implements the MapReduce structure.
- *Map:* In this step each map task builds a RB with the data blocks of its data portion and generates a file containing the RB (called RB_i, see Fig. 3). More specifically, for each instance belonging to the mapper, a fuzzy rule is created in a similar way as described in Section 3.2: we first search for the linguistic fuzzy labels that match

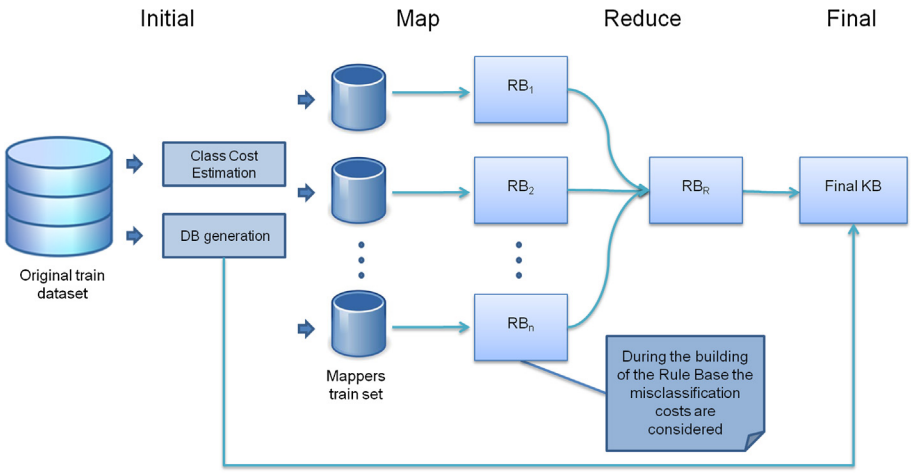

Fig. 3. A flowchart of how the building of the KB is organized in Chi-FRBCS-BigDataCS.

the attribute values of the current example, we select the among the matching fuzzy labels the ones that obtain the largest matching degree for each attribute, we build the rule using as antecedent the fuzzy labels previously selected and as consequent the class associated to the example and finally we compute the rule weight.
Please note that for computing the rule weight we use the PCF or PCF-CS for the Chi-FRBCS-BigData or Chi-FRBCS-BigDataCS methods, and that the set of examples used for the rule weight is the set of examples that belong to the current map process. In this manner, rules with the same antecedents and consequent can be generated by different mappers but they can have different rule weight values. Moreover, when a new rule is created in a mapper we check, as in the original Chi-FRBCS algorithm, if there is a rule with the same antecedents already in the mapper RB. In that case, if the consequent of the rule is also the same as the rule in the mapper RB, this rule is discarded while if the consequent of the new rule is different from the consequent of the previously created rule, then only the rule with the maximum weight is preserved.
In this manner, the Map step applies the original Chi-FRBCS classifier or the Chi-FRBCS-CS approach described in Section 4.1 to the data available in the data partition.

- *Reduce:* In this next step, the reduce process combines the RBs generated by each mapper (RB_i) to form the final RB (called RB_R, see Fig. 3). Specifically, the final RB is built from the RBs built from each mapper $RB_1, RB_2, \ldots, RB_n$ in a similar way as in the creation of new rules in each mapper (Fig. 3): we browse the rules that belong to the RB generated by each mapper, RB_i; if there is a rule in the final RB, RB_R, with the same antecedent as the rule we are trying to add we only maintain in the final RB, RB_R, the rule with the highest rule weight. In this case it is not necessary to check if the consequent is the same or not as we are maintaining the most powerful rules. Equivalent rules (rules with the same antecedent and consequent) can present different weights as they are computed in different mappers over different training sets.
  Please note that it is not needed to recompute the rules weights as we are selecting the most confident rules provided by each mapper. An alternative that would involve a new weight computation would have been the case where equivalent rules are combined to produce a new rule, for instance, computing an average weight between them. However, the direct selection of rules was preferred because of its simplicity which enables to speed up the algorithm in its reduce step.
- *Final:* In this last step, the RB that is generated in the reduce process (RB_R) and the DB that was calculated in the initial phase conform the KB that is provided as the output of the computation process. This output will be the entry data for the mechanism that classifies new examples.

Algorithms 1 and 2 show the pseudo-code of the Map function of the MapReduce job for the creation of the model phase. Algorithm 1 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 2 is called for each mapper to build a RB with the data blocks of its data portion. Furthermore, Algorithm 3 gives the pseudo-code of the Reduce

function and is called when all mappers have finished, to combine the RBs generated by each mapper to form the final RB.

Algorithm 1 Map phase for the Chi-FRBCS-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′ is any Long value and value′ contains a RB.

```
1: instance ← INSTANCE_REPRESENTATION(value) {instances will contain all instances in this mapper's split}
2: instances ← instances.add(instance)
```

Algorithm 2 Map phase for the Chi-FRBCS-BigDataCS algorithm for the building of the model phase CLEANUP():

```
1: fuzzy_ChiBuilder.build(instances, posClass, posclassCost, negClassCost)
2: ruleBase ← fuzzy_ChiBuilder.getRuleBase()
3: EMIT (key, ruleBase)
```

Algorithm 3 Reduce phase for the Chi-FRBCS-BigDataCS algorithm REDUCE(key, values):

Input: <key,values> pair, where key is any Long value and values are the RBs generated by each mapper.
Output: <key′,value′ > pair, where key′ is a null value and value′ is the final RB.

```
1: while values.hasNext() do
2:     ruleBase ← values.getValue()
3:     for i = 0 to ruleBase.size() − 1 do
4:         if finalRuleBase.size() == 0 then
5:             finalRuleBase ← finalRuleBase.add(ruleBase.get(i));
6:         else
7:             if !finalRuleBase.duplicated(ruleBase.get(i)) then
8:                 finalRuleBase ← finalRuleBase.add(ruleBase.get(i));
9:             else
10:                if The consequent of those rules belongs to different classes then
11:                    rule ← finalRuleBase.getRuleWithHighestRuleWeight(ruleBase.get(i))
12:                    finalRuleBase ← finalRuleBase.add(rule);
13:                end if
14:            end if
15:        end if
16:    end for
17: end while
18: EMIT (null, finalRuleBase)
```

5.3. Classification of new patterns

In this section, we will describe how new instances belonging to a dataset are classified considering the KB built previously. When the MapReduce process devoted to the building of the KB has finished, a new MapReduce process is initiated to estimate the class associated to a dataset. Specifically, this phase is also based on a MapReduce design where each map process is responsible for estimating the class for the examples included in its data segment using the final KB previously generated. The process follows the next steps:

- *Initial:* In the same way as in the first step of the building of the model, this step performs a segmentation of the input dataset into independent HDFS blocks; replicates and transfers them to other machines to be finally processed independently by each map task at the same time. This step is automatically performed by the Hadoop system, the MapReduce implementation we are using.
- *Map:* In this next step, each map task estimates the class for the examples included in the data block available for the mapper using the FRM of the winner rule. In particular, for each example, we compute for all the rules in the RB the product of the rule weight with the compatibility degree between the linguistic fuzzy labels that

compose the antecedent of the rule and the example attribute values. The rule that obtains the highest value in this computation determines the new class for the example which is the class consequent of that rule.
- *Final:* In this last step, the predictions generated by each mapper are aggregated to conform the final predictions file. This step is just a concatenation of the results provided by each mapper without any extra computation.

It is important to note that the classification routine does not include a reduce step as it does not need to perform any kind of calculation to combine the results obtained by each mapper. Algorithm 4 gives the pseudo-code of the Map function of the MapReduce job for the classification phase. In this algorithm, Line (2) estimates the class for an instance and Line (5) saves the previously generated predictions.

Algorithm 4 Map phase for the Chi-FRBCS-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* ← *INSTANCE_REPRESENTATION*(*value*)
2: *prediction* ← *CLASSIFY*(*finalRuleBase*, *instance*)
3: *lkey* ← *lkey.set*(*instance.getClass*())
4: *lvalue* ← *lvalue.set*(*prediction*)
5: EMIT (lkey, lvalue)

5.4. Sample procedure of the Chi-FRBCS-BigDataCS algorithm for Imbalanced Big Data: A Case of Study

In order to illustrate how the Chi-FRBCS-BigDataCS algorithm works we have selected an imbalanced big data problem, the *kddcup_full_DOS_versus_U2R* dataset, to describe how the proposal behaves over it. This dataset is an imbalanced big data example with 41 input attributes and 3 883 422 instances. For this specific run, we have chosen the 5th partition of the 5-fcv used in the experimental study developed in this paper. This partition uses 3 105 769 instances for training (38 from the minority class, 3 105 731 from the majority class) and 777 653 for test (10 from the minority class, 777 643 from the majority class). We use 8 mappers in the Hadoop environment. Further information about this dataset is available in Section 6.1.

First, a MapReduce process is initiated in the building of the KB of the Chi-FRBCS-BigDataCS algorithm. The process follows the next steps:

- *Initial:* The first step is to estimate the costs for each class according to the procedure described in Section 4.1: the misclassification cost for instances in the majority class is 1 and the misclassification cost for examples that are associated to the minority class is the IR, that is, 81 729.76. The range of the different features of the dataset and the DB are also computed in this stage. Then, a segmentation of the training dataset into independent HDFS blocks is automatically performed; these blocks are replicated and transferred to other machines in the cluster and are processed by the map tasks in parallel. Each of these data blocks contains approximately 4.75 minority class samples and 388 216.38 majority class samples. Table 4 shows the actual number of instances from both classes available for each map task. This table shows that the distribution of samples is not completely stratified, as it is performed automatically by the Hadoop environment which does not consider the classes distribution.
- *Map:* Next, each map task builds a RB with the data available in its partition and generates a file containing the RB.
- *Reduce:* Later, the final RB is built from the RBs provided by each mapper, selecting from rules that share the same antecedent the rules with the greatest weight. In this manner, the reduce phase is able to decrement the number of final rules and easing the complexity of the model. Table 5 shows the number of rules by map task created in our case of study and the number of final rules. We have created 8 RBs, the number of map process that was made available in the Hadoop environment. We can observe that the number of rules has dramatically decreased from the 446 rules that were created by all the mappers to the 70 rules that finally compose the rule base.
- *Final:* Finally, the RB generated in the previous step and the DB calculated in the initial phase form the final KB that is provided as the output of the computation process.

Table 4
Number of instances available for each map task for the Chi-FRBCS-BigDataCS version with 8 mappers.

kddcup_full_DOS_versus_U2R			
Mapper ID	Total instances	Minority class instances	Majority class instances
1	388 226	7	388 219
2	388 223	5	388 218
3	388 220	2	388 218
4	388 201	4	388 197
5	388 233	6	388 227
6	388 220	4	388 216
7	388 222	5	388 217
8	388 224	5	388 219

Table 5
Number of rules generated by map task and number of final rules for the Chi-FRBCS-BigDataCS version with 8 mappers.

kddcup_full_DOS_versus_U2R	
NumRules by mapper	Final numRules
RB_1 size: 60	RB_R size: 70
RB_2 size: 60	
RB_3 size: 55	
RB_4 size: 52	
RB_5 size: 49	
RB_6 size: 60	
RB_7 size: 52	
RB_8 size: 58	

Once we have finished the MapReduce process devoted to the building of the model, we generate a new MapReduce process to estimate the class for the examples of the training and test dataset:

- *Initial:* At the beginning, in the same way as in the building of the model, the algorithm performs a segmentation of the input dataset into independent HDFS blocks; replicates and transfers them to other machines to be finally processed independently by each map task concurrently.
- *Map:* Next, each map task estimates the classes of a subset of the dataset for every instance stored in it considering the final KB built previously, using the winning rule as FRM.
- *Final:* Finally, an aggregation of the predictions generated by each mapper compose the final predictions file.

6. Experimental study

In this section we show the experimental study carried out on the behavior of Chi-FRBCS-BigDataCS for imbalanced big data. First, in Section 6.1 we provide details of the classification problems chosen for the experimentation. Some of them have been used in previous sections for specific cases of study. Then, Section 6.2 introduces the algorithms selected for the comparison with the proposed approach and their configuration parameters. This section also details the infrastructure on which the experiments have been executed. Finally, Section 6.3 provides the performance results for the approaches using the AUC measure and shows the time elapsed for the datasets considered in the study.

6.1. Datasets used in the study

In order to analyze the quality of our approach, Chi-FRBCS-BigDataCS, we have run our experiments around three datasets from the UCI dataset repository [73]: the KDD Cup 1999 dataset, the Record Linkage Comparison Patterns (RLCP) dataset and the Poker Hand dataset. The KDD Cup 1999 dataset was used in the Third International Knowledge Discovery and Data Mining Tools Competition. It is a problem that represents a network intrusion detector,

and it aims to differentiate between *good* normal connections and *bad* connections that represent the different types of attacks. On the other hand, the underlying records in the Record Linkage Comparison Patterns Dataset stem from the epidemiological cancer registry of the German state of North Rhine-Westphalia. The Poker Hand dataset purpose is to predict poker hands.

Since the KDD Cup 1999 dataset and the Poker Hand dataset contain multiple classes, we have created several big data cases of study derived from them. More specifically, for the KDD Cup 1999 dataset we have generated new versions of the KDD Cup data using the *normal* and *DOS* connections as majority classes and the rest of attacks have been considered as minority classes. For the Poker Hand dataset we have obtained new versions using the 0 and 1 classes ("Nothing in hand" and "One pair" respectively) as majority classes and the rest of classes as minority classes. Moreover, we have also generated smaller versions of the original dataset selecting the 10% of the instances. For these reduced versions we have excluded the cases of study that contain less than 200 samples in their full versions, to make sure that in each mapper there is at least one sample of each class to learn the model.

Table 6 summarizes the data employed in this study and shows, for each dataset, the number of examples (#Ex.), number of attributes (#Atts.), class name of each class (minority and majority), number of instances for each class, class attribute distribution and IR.

In order to develop our study we use a 5-fold stratified cross validation partitioning scheme, that is, five random partitions of data with a 20% of the samples where the combination of 4 of them (80%) is considered as training set and the remaining one is treated as test set. For each dataset we consider the average results of the five partitions.

6.2. Algorithms and parameter settings

To verify the performance of the proposed model, we compare the results obtained by Chi-FRBCS-BigDataCS with the basic versions of the algorithm that solve the big data and imbalanced problems separately. Specifically, the algorithms considered in the study have been:

- **Chi-FRBCS** [23]: The classical fuzzy rule based classifier which was described in Section 3.2.
- **Chi-FRBCS-CS**: This is the proposed Chi-FRBCS version that introduces cost-sensitive learning modifying some of the Chi-FRBCS operations. This algorithm has been described in Section 4.1.
- **Chi-FRBCS-BigData**: This is the basic Chi-FRBCS version adapted to deal with big data. It is an algorithm that follows a MapReduce design which has been implemented under the Hadoop framework and is described in Section 5.
- **Chi-FRBCS-BigDataCS**: This is our final proposal, the modified version of the Chi-FRBCS-CS that has been prepared to take on imbalanced big data using a MapReduce scheme which has been implemented using Hadoop combined with cost-sensitive learning. This algorithm has also been described in Section 5.

The experiments associated to the sequential versions of the Chi-FRBCS algorithm have been run using the KEEL Software Tool [75,76].

Considering the parameters used in the experimentation, these algorithms use three fuzzy labels for each attribute, the product T-norm as conjunction operator in order to compute the matching degree of the antecedent of the rule with the example, PCF or PCF-CS (depending on the use of a CS version) to compute the rule weight and the FRM of the winning rule. Finally, only the approaches adapted for big data use a parameter related to the MapReduce procedure, which is the number of subsets of the original data that are created and provided for the map tasks. We have selected two different set of values for this parameter, as it has a direct impact on the AUC performance obtained and the runtime spent by the algorithms. Specifically, for the experiments on the reduced versions (10%) of the cases of study we have used 2, 4, 6, 8 and 16 mappers to have a better insight in the comparison with the sequential versions. For the full versions of the cases of study, we use 8, 16, 32 and 64 mappers to better address the big data cases under consideration. In this manner, the number of RBs created in the intermediate step of the algorithm depends on the number of map tasks.

With respect to the infrastructure used to perform the experiments, for the MapReduce designs, we have used the Atlas research group's cluster with 12 nodes, connected with a 1 Gb/s ethernet. Each node is composed by two Intel E5-2620 microprocessors (at 2 GHz, 15MB cache) and 64GB of main memory running under Linux CentOS 6.3. Furthermore, the cluster works with Hadoop 2.0.0 (Cloudera CDH4.3.0), where one node is configured as namenode

Table 6
Summary of imbalanced datasets.

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)	3.99
kddcup_10_DOS_versus_PRB	392 447	41	(DOS; PRB)	(388 337; 4110)	(98.953; 1.047)	94.49
kddcup_10_DOS_versus_R2L	388 449	41	(DOS; R2L)	(388 337; 112)	(99.971; 0.029)	3467.29
kddcup_10_normal_versus_PRB	101 388	41	(normal; PRB)	(97 278; 4110)	(95.946; 4.054)	23.67
kddcup_10_normal_versus_R2L	97 390	41	(normal; R2L)	(97 278; 112)	(99.885; 0.115)	868.55
poker_10_0_vs_2	56 252	10	(0; 2)	(51 370; 4882)	(91.321; 8.679)	10.52
poker_10_0_vs_3	53 533	10	(0; 3)	(51 370; 2163)	(95.96; 4.04)	23.75
poker_10_0_vs_4	51 767	10	(0; 4)	(51 370; 397)	(99.233; 0.767)	129.40
poker_10_0_vs_5	51 575	10	(0; 5)	(51 370; 205)	(99.603; 0.397)	250.59
poker_10_0_vs_6	51 516	10	(0; 6)	(51 370; 146)	(99.717; 0.283)	351.85
poker_10_0_vs_7	51 393	10	(0; 7)	(51 370; 23)	(99.955; 0.045)	2233.48
poker_10_1_vs_2	48 191	10	(1; 2)	(43 309; 4882)	(89.869; 10.131)	8.87
poker_10_1_vs_3	45 472	10	(1; 3)	(43 309; 2163)	(95.243; 4.757)	20.02
poker_10_1_vs_4	43 706	10	(1; 4)	(43 309; 397)	(99.092; 0.908)	109.09
poker_10_1_vs_5	43 514	10	(1; 5)	(43 309; 205)	(99.529; 0.471)	211.26
poker_10_1_vs_6	43 455	10	(1; 6)	(43 309; 146)	(99.664; 0.336)	296.64
poker_10_1_vs_7	43 332	10	(1; 7)	(43 309; 23)	(99.947; 0.053)	1883.00
RLCP_10	574 913	2	(FALSE; TRUE)	(572 820; 2093)	(99.636; 0.364)	273.68
kddcup_DOS_versus_normal	4 856 151	41	(DOS; normal)	(3 883 370; 972 781)	(79.968; 20.032)	3.99
kddcup_DOS_versus_PRB	3 924 472	41	(DOS; PRB)	(3 883 370; 41 102)	(98.953; 1.047)	94.48
kddcup_DOS_versus_R2L	3 884 496	41	(DOS; R2L)	(3 883 370; 1126)	(99.971; 0.029)	3448.82
kddcup_DOS_versus_U2R	3 883 422	41	(DOS; U2R)	(3 883 370; 52)	(99.999; 0.001)	74 680.19
kddcup_normal_versus_PRB	1 013 883	41	(normal; PRB)	(972 781; 41 102)	(95.946; 4.054)	23.67
kddcup_normal_versus_R2L	973 907	41	(normal; R2L)	(972 781; 1126)	(99.884; 0.116)	863.93
kddcup_normal_versus_U2R	972 833	41	(normal; U2R)	(972 781; 52)	(99.995; 0.005)	18 707.33
poker_0_vs_2	562 530	10	(0; 2)	(513 702; 48 828)	(91.32; 8.68)	10.52
poker_0_vs_3	535 336	10	(0; 3)	(513 702; 21 634)	(95.959; 4.041)	23.75
poker_0_vs_4	517 680	10	(0; 4)	(513 702; 3978)	(99.232; 0.768)	129.14
poker_0_vs_5	515 752	10	(0; 5)	(513 702; 2050)	(99.603; 0.397)	250.59
poker_0_vs_6	515 162	10	(0; 6)	(513 702; 1460)	(99.717; 0.283)	351.85
poker_0_vs_7	513 938	10	(0; 7)	(513 702; 236)	(99.954; 0.046)	2176.70
poker_0_vs_8	513 719	10	(0; 8)	(513 702; 17)	(99.997; 0.003)	30 217.76
poker_0_vs_9	513 710	10	(0; 9)	(513 702; 8)	(99.998; 0.002)	64 212.75
poker_1_vs_2	481 925	10	(1; 2)	(433 097; 48 828)	(89.868; 10.132)	8.87
poker_1_vs_3	454 731	10	(1; 3)	(433 097; 21 634)	(95.242; 4.758)	20.02
poker_1_vs_4	437 075	10	(1; 4)	(433 097; 3978)	(99.09; 0.91)	108.87
poker_1_vs_5	435 147	10	(1; 5)	(433 097; 2050)	(99.529; 0.471)	211.27
poker_1_vs_6	434 557	10	(1; 6)	(433 097; 1460)	(99.664; 0.336)	296.64
poker_1_vs_7	433 333	10	(1; 7)	(433 097; 236)	(99.946; 0.054)	1835.16
poker_1_vs_8	433 114	10	(1; 8)	(433 097; 17)	(99.996; 0.004)	25 476.29
poker_1_vs_9	433 105	10	(1; 9)	(433 097; 8)	(99.998; 0.002)	54 137.13
RLCP	5 749 132	2	(FALSE; TRUE)	(5 728 201; 20 931)	(99.636; 0.364)	273.67

and jobtracker, and the rest are datanodes and task-trackers. For the sequential experiments we have used a cluster with Intel Core i7 930 microprocessors (at 2.8 GHz, 15MB cache) and 24GB of main memory connected with a 1 Gb/s ethernet. We acknowledge that the runtime comparisons between the sequential versions and the MapReduce designs are not performed in identical machines, however, the time advantage is obtained for the sequential versions which are, even in this case, notably slower than the Hadoop implementations.

6.3. Analysis of the Chi-FRBCS-BigDataCS behavior

In this part of the study, we want to analyze the behavior of the Chi-FRBCS-BigDataCS proposal in the scenario of imbalanced big data in contrast with the other learning proposals. This section is divided into two parts: the first part

(Section 6.3.1) is devoted to the presentation of the precision of our approach in terms of classification performance using the AUC measure; the second part (Section 6.3.2) is devoted to the analysis on the runtime of the model.

6.3.1. Analysis on the precision of the model

In this section, we present a set of experiments to illustrate and demonstrate the behavior of Chi-FRBCS-BigDataCS. These experiments are organized in two phases: the first one compares the behavior of the different alternatives using the cases of study that contain the 10% of the instances of the original datasets while the second one compares the behavior of the approaches over the full datasets considered in the study. The experiments where organized in this way to be able to contrast the results of the big data versions in relation with the serial versions of the algorithm for the smaller datasets. Additionally, this organization also permits to check how the results change when instead of using a reduced version of the dataset the whole dataset is utilized.

In Tables 7 and 8 we present the average results in training and test for the reduced versions (10%) of the imbalanced big data cases of study for the Chi-FRBCS and Chi-FRBCS-CS versions respectively. These tables are divided by columns in two parts: the first part corresponds to the results of the sequential variant while the second part is related to the big data variants of the Chi-FRBCS and Chi-FRBCS-CS algorithms respectively. Furthermore, the results for the big data alternatives are divided by columns in five parts, which correspond to the number of mappers used: 2, 4, 6, 8 and 16 mappers for each case.

Looking at the results we can observe that the performance obtained, both in training and test, is higher in most of the cases of study for the Chi-FRBCS-CS alternatives, the sequential approach and the big data adaptation for any number of mappers configuration. This situation demonstrates the positive influence of the usage of cost-sensitive learning when dealing with imbalanced data as the classifier is able to provide appropriate solutions in an arduous environment. Additionally, we can observe that the model does not present a strong overfitting on the training set in relation with the test set, as we cannot find huge differences between the results provided for both sets. For instance, for the *kddcup_10_normal_versus_PRB* dataset using the Chi-FRBCS-BigDataCS with 8 mappers, an AUC of 0.9728 in training is obtained which is closely followed by an AUC in test of 0.9723. There are even cases where the test set obtains a better performance than the training set such as *kddcup_10_normal_versus_R2L* for Chi-FRBCS-BigDataCS using 8 mappers with an AUC in training of 0.8747 and an AUC in test of 0.8784. This situation is caused by the usage of the PCF or PCF-CS to compute the rule weight as these measures try to make rules as general as possible considering the current dataset.

Next, we compare the results considering the cases of study derived from all original training sets in relation with the number of mappers considered. For the KDD Cup 1999 cases of study we find that the behavior of the Chi-FRBCS and Chi-FRBCS-CS approaches is not steady in relation to the number of mappers considered in the experiments. For instance, for the Chi-FRBCS sequential version, the test results achieved are worse than the results for the Chi-FRBCS-BigData approach. In this case, increasing the number of mappers may also increase the AUC metric, however, when the number of mappers is too high this performance is decreased. The Chi-FRBCS-CS sequential variant, is able to provide better test results than the Chi-FRBCS-BigDataCS proposal. However, there is not a clear optimal configuration for the number of mappers used, as the results are not stable when increasing that number of mappers. Furthermore, the worse results are obtained for the highest number of mappers considered in the experiment. In contrast, the training results provide more sensible results, decreasing the performance in a reasonable manner when the number of mappers is enlarged.

In the case of the Poker Hand cases of study we first discover that the results obtained for this set of data are a bit poor, as the AUC measure is usually ranging from 0.5 to 0.6. Similarly to the KDD Cup 1999 dataset, the Chi-FRBCS approaches are presenting erratic results where the sequential version provides worse AUC values than the Chi-FRBCS-BigData alternative, which is also improving when larger values for the number of mappers are used. In the case of the Chi-FRBCS-CS variants, the performance obtained is clearly related both in training and test with the number of mappers considered: the best performance is achieved by the sequential Chi-FRBCS-CS algorithm while the performance drops when bigger number of mappers are used.

The RLCP dataset is not able to properly identify instances from both classes in the Chi-FRBCS approaches, as the results obtained for all the variants and the number of mappers considered is 0.5. When the Chi-FRBCS-CS alternatives are tested, the RLCP provides reasonable AUC results with almost no variance when the sequential version is contrasted with smaller values for the number of mappers. Larger values for the number of mappers need to be compared to find a slight drop in accuracy.

Table 7
Average results for the Chi-FRBCS versions for the imbalanced big data cases of study using the AUC measure.

Datasets	Chi-FRBCS		Chi-FRBCS-BigData									
			2 mappers		4 mappers		6 mappers		8 mappers		16 mappers	
	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}
kddcup_10_DOS_versus_normal	0.9973	0.9972	0.9993	**0.9993**	0.9993	**0.9993**	0.9993	**0.9993**	0.9993	**0.9993**	0.9992	**0.9993**
kddcup_10_DOS_versus_PRB	0.8440	0.8430	0.9055	0.9055	0.9052	0.9059	0.9112	**0.9116**	0.9029	0.9009	0.9088	0.9105
kddcup_10_DOS_versus_R2L	1.0000	0.9897	0.9951	0.9954	0.9988	0.9954	0.9987	**1.0000**	0.9988	**1.0000**	0.9013	0.8651
kddcup_10_normal_versus_PRB	0.8608	0.8589	0.9364	0.9376	0.9286	0.9284	0.9304	0.9311	0.9337	0.9332	0.9376	**0.9381**
kddcup_10_normal_versus_R2L	0.5000	0.5000	0.5000	0.5000	0.5120	0.5032	0.5560	0.5234	0.5419	**0.5359**	0.5195	0.5111
Average (kddcup)	0.8404	0.8377	0.8673	0.8676	0.8688	0.8664	0.8791	0.8731	0.8753	**0.8739**	0.8533	0.8448
poker_10_0_vs_2	0.5753	0.5052	0.5917	0.5108	0.6143	0.5146	0.6343	0.5182	0.6493	0.5195	0.6791	**0.5244**
poker_10_0_vs_3	0.5955	0.5082	0.6204	0.5180	0.6443	0.5222	0.6600	0.5291	0.6725	0.5310	0.7018	**0.5381**
poker_10_0_vs_4	0.5114	0.4956	0.5185	0.4999	0.5336	0.4998	0.5575	0.4998	0.5704	0.4997	0.6112	**0.5020**
poker_10_0_vs_5	0.7662	0.7039	0.8053	0.7857	0.8110	0.7992	0.8138	**0.8002**	0.8143	**0.8002**	0.8258	0.8001
poker_10_0_vs_6	0.5928	0.4963	0.6128	0.4999	0.6321	**0.5044**	0.6454	**0.5044**	0.6659	**0.5044**	0.6972	0.5043
poker_10_0_vs_7	0.5748	0.4960	0.5902	**0.5000**	0.5891	**0.5000**	0.6044	**0.5000**	0.6044	**0.5000**	0.6595	**0.5000**
poker_10_1_vs_2	0.5558	0.4933	0.5749	0.5045	0.6027	0.5066	0.6183	0.5086	0.6330	0.5087	0.6667	**0.5111**
poker_10_1_vs_3	0.5503	0.4924	0.5756	0.5028	0.5991	0.5048	0.6134	0.5047	0.6288	0.5065	0.6502	**0.5082**
poker_10_1_vs_4	0.5022	0.4901	0.5205	**0.4999**	0.5398	0.4997	0.5419	0.4996	0.5550	0.4994	0.5862	0.4990
poker_10_1_vs_5	0.7040	0.6222	0.7171	0.6816	0.7331	**0.7049**	0.7365	0.6977	0.7332	0.7047	0.7434	0.7045
poker_10_1_vs_6	0.5545	0.4891	0.5750	**0.4999**	0.5986	0.4997	0.6037	0.4997	0.6107	0.4997	0.6388	0.4994
poker_10_1_vs_7	0.5831	0.4891	0.5831	**0.5000**	0.5792	**0.5000**	0.5992	**0.5000**	0.5750	**0.5000**	0.5950	**0.5000**
Average (poker)	0.5888	0.5235	0.6071	0.5419	0.6231	0.5463	0.6357	0.5468	0.6427	0.5478	0.6712	**0.5493**
RLCP_10	0.5000	**0.5000**	0.5000	**0.5000**	0.5000	**0.5000**	0.5000	**0.5000**	0.5000	**0.5000**	0.5000	**0.5000**
Total average	0.6538	0.6095	0.6734	0.6300	0.6845	0.6327	0.6958	0.6349	0.6994	**0.6357**	0.7123	0.6286

Table 8
Average results for the Chi-FRBCS cost-sensitive versions for the imbalanced big data cases of study using the AUC measure.

Datasets	Chi-FRBCS-CS		Chi-FRBCS-BigDataCS									
			2 mappers		4 mappers		6 mappers		8 mappers		16 mappers	
	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}
kddcup_10_DOS_versus_normal	0.9975	0.9974	0.9994	**0.9995**	0.9995	**0.9995**	0.9995	**0.9995**	0.9995	**0.9995**	0.9994	0.9993
kddcup_10_DOS_versus_PRB	0.9849	**0.9831**	0.9588	0.9578	0.9588	0.9575	0.9584	0.9573	0.9582	0.9569	0.9571	0.9569
kddcup_10_DOS_versus_R2L	0.9999	0.9897	0.9999	**0.9999**	0.9999	**0.9999**	0.9999	**0.9999**	0.9999	**0.9999**	0.9524	0.9318
kddcup_10_normal_versus_PRB	0.9707	0.9697	0.9733	**0.9730**	0.9728	0.9729	0.9733	0.9729	0.9728	0.9723	0.9687	0.9688
kddcup_10_normal_versus_R2L	0.9729	**0.9499**	0.9638	0.9161	0.9640	0.9216	0.8983	0.8909	0.8747	0.8784	0.7443	0.7428
Average (kddcup)	0.9852	**0.9780**	0.9790	0.9693	0.9790	0.9703	0.9659	0.9641	0.9610	0.9614	0.9244	0.9199
poker_10_0_vs_2	0.9075	0.5905	0.8847	**0.5911**	0.8476	0.5737	0.8315	0.5689	0.8164	0.5623	0.7865	0.5500
poker_10_0_vs_3	0.9536	0.6173	0.9119	**0.6213**	0.8652	0.5960	0.8358	0.5824	0.8148	0.5727	0.7845	0.5587
poker_10_0_vs_4	0.9899	**0.5787**	0.9523	0.5633	0.8504	0.5324	0.7800	0.5287	0.7642	0.5185	0.7224	0.5190
poker_10_0_vs_5	0.9921	**0.8756**	0.9793	0.8706	0.9238	0.8399	0.8685	0.8120	0.8554	0.8097	0.8311	0.7997
poker_10_0_vs_6	0.9977	0.5082	0.9309	**0.5148**	0.8344	0.5116	0.8165	0.5117	0.8128	0.5115	0.7955	0.5115
poker_10_0_vs_7	0.9990	0.4947	0.8666	0.4999	0.8506	0.4999	0.8245	0.4999	0.8084	0.4999	0.7936	**0.5000**
poker_10_1_vs_2	0.8818	0.5306	0.8580	**0.5481**	0.8198	0.5380	0.8016	0.5394	0.7848	0.5313	0.7563	0.5261
poker_10_1_vs_3	0.9338	0.5368	0.8874	**0.5423**	0.8206	0.5337	0.7885	0.5279	0.7664	0.5203	0.7218	0.5104
poker_10_1_vs_4	0.9800	0.5359	0.9135	**0.5402**	0.7787	0.5193	0.7219	0.5086	0.6848	0.5101	0.6459	0.5073
poker_10_1_vs_5	0.9918	**0.8782**	0.9649	0.8250	0.9101	0.7881	0.8394	0.7369	0.8144	0.7299	0.7608	0.7105
poker_10_1_vs_6	0.9939	0.4923	0.8518	0.4974	0.7488	0.4986	0.6951	0.4989	0.6940	0.4989	0.6819	**0.4991**
poker_10_1_vs_7	0.9981	0.4868	0.8867	0.4996	0.7085	**0.4999**	0.6880	**0.4999**	0.6111	**0.4999**	0.6111	**0.4999**
Average (poker)	0.9683	**0.5938**	0.9073	0.5928	0.8299	0.5776	0.7909	0.5679	0.7690	0.5638	0.7410	0.5577
RLCP_10	0.9135	**0.9135**	0.9135	**0.9135**	0.9135	**0.9135**	0.9135	**0.9135**	0.9110	0.9104	0.9070	0.9069
Total average	0.9699	**0.7183**	0.9276	0.7152	0.8759	0.7053	0.8463	0.6972	0.8302	0.6935	0.8011	0.6777

In all these cases of study we can say that there is not a strong degradation in the performance when using the MapReduce versions. Specifically, the Chi-FRBCS-BigDataCS is more affected by the increasing number of mappers than Chi-FRBCS-BigData, however, this behavior is expected because increasing the number of portions induces the dataset shift problem and the small sample size, situations that have a pernicious effect when dealing with imbalanced datasets. To test the influence of the small sample size problem when different number of mappers are considered, we show in Table 9 the diverse number of minority and majority class instances by mapper for the Chi-FRBCS-BigData versions. Please note that the number of instances per mapper for Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS is the same, because the initial stage in both algorithms is identical: the framework automatically divides the data in different information portions that are then copied and distributed to all the mapper processes considered.

As it is expected, the number of instances per mapper from each class is drastically reduced when higher values for the number of mappers are obtained. This decrement on the available number of instances is observed in both classes, however, it has a greater impact on the minority class. The minimum average number of samples per mapper in the most adverse situation for the majority class is 2164.75 for all the reduced versions considered, which is a reasonable number of samples to learn the associated fuzzy rules. However, when the number of minority class samples is observed for the maximum number of mappers considered, we find several cases of study that do not have at least 7 minority class samples per mapper. In these cases we encounter the small sample size problem which is responsible for the poor results achieved. The small sample size problem also influences the increasing drop in the performance of the algorithms when larger values for the number of mappers are utilized. For instance, the cases of study with the lesser number of minority class instances, like *poker_10_0_vs_7* and *poker_10_1_vs_7*, obtain very poor results being unable to properly identify instances from both classes. In the *kddcup_10_normal_versus_R2L* case of study we can also observe the dramatical drop in the performance, going from an AUC value of 0.9693 when 2 mappers are used to 0.7428 for 16 mappers, as we range from 45.60 minority class instances by mapper to 5.70.

Table 10 shows the average results in training and test for the full imbalanced big data cases of study. This table is divided by columns in two parts: the first column is related to the Chi-FRBCS-BigData algorithm while the second column is related with the cost-sensitive alternative, the Chi-FRBCS-BigDataCS algorithm. As in the preceding case, these algorithms organize their results by columns in four parts according to the number of mappers: 8, 16, 32 and 64 respectively. Please note that the sequential versions were not included in this table since these approaches were not able to complete an experiment with data this size as it was shown in the scalability studies (Sections 3.3 and 4.2).

On the first hand, we can observe a similar behavior between the reduced datasets in relation with the full datasets. Specifically, Chi-FRBCS-BigDataCS is able to provide a much better performance than Chi-FRBCS-BigData for all the diverse number of mappers tested. The differences between the training and test results are observed only for the Poker Hand cases of study which means that overfitting appears when the size of the training set is smaller.

On the other hand, the results related to the number of mappers used also resemble the results obtained for the sequential versions. For instance, when examining the number of mappers used for the big data developments, we can see that as the number of mappers increases and therefore the data available for each mapper is reduced, our proposal Chi-FRBCS-BigDataCS maintains a slight decrease in performance whereas the Chi-FRBCS-BigData alternative is not able to show a clear tendency.

When we take a closer look grouping together the cases of study that are derived from the same datasets we can observe that the general conclusions extracted can also be applied to these groups. Specifically, the KDD Cup 1999 cases of study follow this different behavior for Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS. Chi-FRBCS-BigData does not show a clear trend for diverse values of the number of mappers, while the Chi-FRBCS-BigDataCS method decrements its performance when larger number of mappers are utilized.

The Poker Hand cases of study also closely follow this disposition, not having a shift according to the number of mappers for the Chi-FRBCS-BigData method but degrading the performance of the Chi-FRBCS-BigDataCS method for high values of the number of mappers considered. In addition, we also observe that the values obtained for the AUC measure are still poor for these cases of study, however, they are better than the results obtained for the reduced 10% cases of study previously analyzed.

The RLCP dataset shows a similar behavior to the one previously analyzed. The Chi-FRBCS-BigData approach does not show a correct classification of the samples considered as it obtains an AUC value of 0.5. For the Chi-FRBCS-BigDataCS, the results achieved, while being better, do not vary much with respect to the number of mappers. For the smaller values of the number of mappers the AUC results are the same, while they are slightly diminished when larger values are considered.

Table 9
Average number of minority and majority class instances by mapper for the Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS versions.

Datasets	Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS									
	2 mappers		4 mappers		6 mappers		8 mappers		16 mappers	
	#min class	#maj class	#min class	#maj class	#min class	#maj class	#min class	#maj class	#min class	#maj class
kddcup_10_DOS_versus_normal	39 014.40	155 231.60	19 507.20	77 615.80	13 004.80	51 743.87	9753.60	38 807.90	4876.80	19 403.95
kddcup_10_DOS_versus_PRB	1612.00	155 366.80	806.00	77 683.40	537.33	51 788.93	403.00	38 841.70	201.50	19 420.85
kddcup_10_DOS_versus_R2L	40.80	155 338.80	20.40	77 669.40	13.60	51 779.60	10.20	38 834.70	5.10	19 417.35
kddcup_10_normal_versus_PRB	1639.20	38 916.00	819.60	19 458.00	546.40	12 972.00	409.80	9729.00	204.90	4864.50
kddcup_10_normal_versus_R2L	45.60	38 910.40	22.80	19 455.20	15.20	12 970.13	11.40	9727.60	5.70	4863.80
poker_10_0_vs_2	1952.00	20 548.80	976.00	10 274.40	650.67	6849.60	488.00	5137.20	244.00	2568.60
poker_10_0_vs_3	906.40	20 506.80	453.20	10 253.40	302.13	6835.60	226.60	5126.70	113.30	2563.35
poker_10_0_vs_4	162.00	20 544.80	81.00	10 272.40	54.00	6848.26	40.50	5136.20	20.25	2568.10
poker_10_0_vs_5	88.00	20 542.00	44.00	10 271.00	29.33	6847.33	22.00	5135.50	11.00	2567.75
poker_10_0_vs_6	52.40	20 554.00	26.20	10 277.00	17.46	6851.33	13.10	5138.50	6.55	2569.25
poker_10_0_vs_7	6.80	20 550.40	3.40	10 275.20	2.26	6850.13	1.70	5137.60	0.85	2568.80
poker_10_1_vs_2	1927.60	17 348.80	963.80	8674.40	642.53	5782.93	481.90	4337.20	240.95	2168.60
poker_10_1_vs_3	856.80	17 332.00	428.40	8666.00	285.60	5777.33	214.20	4333.00	107.10	2166.50
poker_10_1_vs_4	156.00	17 326.40	78.00	8663.20	52.00	5775.46	39.00	4331.60	19.50	2165.80
poker_10_1_vs_5	87.60	17 318.00	43.80	8659.00	29.20	5772.66	21.90	4329.50	10.95	2164.75
poker_10_1_vs_6	56.80	17 325.20	28.40	8662.60	18.93	5775.06	14.20	4331.30	7.10	2165.65
poker_10_1_vs_7	8.80	17 324.00	4.40	8662.00	2.93	5774.66	2.20	4331.00	1.10	2165.50
RLCP_10	827.60	229 137.60	413.80	114 568.80	275.87	76 379.20	206.90	57 284.40	103.45	28 642.20

Table 10
Average results for the big data Chi-FRBCS versions for the full imbalanced big data cases of study using the AUC measure.

Datasets	Chi-FRBCS-BigData								Chi-FRBCS-BigDataCS							
	8 mappers		16 mappers		32 mappers		64 mappers		8 mappers		16 mappers		32 mappers		64 mappers	
	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}	AUC_{tr}	AUC_{tst}
kddcup_DOS_versus_normal	0.9992	0.9992	0.9992	0.9992	0.9993	**0.9993**	0.9992	0.9992	0.9993	**0.9993**	0.9993	**0.9993**	0.9993	**0.9993**	0.9993	**0.9993**
kddcup_DOS_versus_PRB	0.8639	0.8636	0.8636	0.8633	0.8639	0.8639	0.8634	0.8633	0.9558	**0.9557**	0.9556	0.9556	0.9553	0.9553	0.9546	0.9545
kddcup_DOS_versus_R2L	0.9941	0.9913	0.9881	0.9886	0.9779	0.9769	0.9942	0.9918	0.9999	**0.9999**	0.9999	**0.9999**	0.9999	**0.9999**	0.9999	**0.9999**
kddcup_DOS_versus_U2R	0.8544	0.8464	0.8544	0.8464	0.8544	0.8464	0.8544	0.8464	0.9387	**0.9366**	0.8960	0.8880	0.8960	0.8880	0.8960	0.8880
kddcup_normal_versus_PRB	0.8936	0.8932	0.8701	0.8693	0.8784	0.8788	0.8693	0.8690	0.9681	**0.9681**	0.9670	0.9671	0.9683	0.9679	0.9663	0.9659
kddcup_normal_versus_R2L	0.5086	0.5050	0.5089	0.5055	0.5197	0.5178	0.5246	0.5223	0.9626	**0.9616**	0.9460	0.9446	0.9101	0.9119	0.8298	0.8229
kddcup_normal_versus_U2R	0.5397	**0.5000**	0.5454	**0.5000**	0.5454	**0.5000**	0.5454	**0.5000**	0.5518	**0.5000**	0.5575	**0.5000**	0.5575	**0.5000**	0.5575	**0.5000**
Average (kddcup)	0.8076	0.7998	0.8042	0.7960	0.8056	0.7976	0.8072	0.7989	0.9109	**0.9030**	0.9030	0.8935	0.8981	0.8889	0.8862	0.8758
poker_0_vs_2	0.5240	0.5146	0.5310	0.5188	0.5400	0.5237	0.5480	0.5271	0.7371	**0.6689**	0.7048	0.6420	0.6597	0.6054	0.6126	0.5700
poker_0_vs_3	0.5338	0.5189	0.5477	0.5271	0.5626	0.5367	0.5751	0.5432	0.7950	**0.7132**	0.7422	0.6686	0.6744	0.6163	0.6250	0.5769
poker_0_vs_4	0.5005	0.5000	0.5012	0.5000	0.5065	0.5005	0.5140	0.5009	0.8262	**0.6755**	0.6752	0.5961	0.5884	0.5345	0.5526	0.5132
poker_0_vs_5	0.7341	0.7298	0.7483	0.7479	0.7488	0.7486	0.7489	0.7486	0.9686	**0.9588**	0.8977	0.8859	0.7707	0.7673	0.7490	0.7485
poker_0_vs_6	0.5445	0.5194	0.5553	0.5218	0.5645	0.5264	0.5719	0.5280	0.6559	**0.5611**	0.6131	0.5410	0.5969	0.5371	0.5943	0.5355
poker_0_vs_7	0.5935	0.5115	0.6220	0.5139	0.6562	0.5228	0.6704	0.5299	0.6935	0.5294	0.6776	**0.5318**	0.6825	**0.5318**	0.6825	**0.5318**
poker_0_vs_8	0.5000	0.5000	0.6262	0.5000	0.7422	0.5000	0.8333	0.6750	0.8396	**0.7750**	0.8396	**0.7750**	0.8396	**0.7750**	0.8396	**0.7750**
poker_0_vs_9	0.7500	**0.5000**	0.7708	**0.5000**	0.7708	**0.5000**	0.7708	**0.5000**	0.7708	**0.5000**	0.7708	**0.5000**	0.7708	**0.5000**	0.7708	**0.5000**
poker_1_vs_2	0.5032	0.5004	0.5071	0.5015	0.5125	0.5027	0.5185	0.5045	0.6681	**0.5761**	0.6354	0.5589	0.5942	0.5363	0.5575	0.5208
poker_1_vs_3	0.5021	0.5002	0.5056	0.5010	0.5114	0.5032	0.5180	0.5054	0.7004	**0.6088**	0.6383	0.5649	0.5760	0.5326	0.5393	0.5139
poker_1_vs_4	0.5010	0.5000	0.5016	0.5000	0.5035	0.5000	0.5057	0.4999	0.7511	**0.6191**	0.6054	0.5593	0.5283	0.5062	0.5154	0.5005
poker_1_vs_5	0.7483	0.7481	0.7484	0.7483	0.7486	0.7483	0.7498	0.7490	0.9745	**0.9617**	0.9017	0.8911	0.7814	0.7769	0.7488	0.7486
poker_1_vs_6	0.5047	0.5000	0.5146	0.5010	0.5244	0.5023	0.5313	0.5034	0.5826	**0.5163**	0.5457	0.5053	0.5387	0.5043	0.5388	0.5044
poker_1_vs_7	0.5077	**0.5016**	0.5089	0.5000	0.5180	0.5000	0.5278	0.5000	0.5517	0.5000	0.5492	0.5000	0.5457	0.5000	0.5438	0.5000
poker_1_vs_8	0.5125	**0.5000**	0.6583	**0.5000**	0.7140	**0.5000**	0.7693	**0.5000**	0.7745	**0.5000**	0.7745	**0.5000**	0.7745	**0.5000**	0.7745	**0.5000**
poker_1_vs_9	0.8383	**0.6000**	0.8383	**0.6000**	0.8383	**0.6000**	0.8383	**0.6000**	0.8383	**0.6000**	0.8383	**0.6000**	0.8383	**0.6000**	0.8383	**0.6000**
Average (poker)	0.5811	0.5403	0.6053	0.5426	0.6226	0.5447	0.6369	0.5572	0.7580	**0.6415**	0.7131	0.6137	0.6725	0.5827	0.6552	0.5712
RLCP	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.9134	**0.9134**	0.9134	**0.9134**	0.9134	**0.9134**	0.9093	0.9092
Total average	0.6438	0.6143	0.6590	0.6147	0.6709	0.6166	0.6809	0.6253	0.8091	**0.7291**	0.7768	0.7078	0.7483	0.6858	0.7331	0.6741

The general tendency that incurs in a drop in the performance for good performing algorithms appears usually when a more parallel solution is compared with a less parallel solution or sequential solution, as only partial information is available for the computation in contrast with larger portions of information that can even cover the whole information available. However, this undesirable effect is not only related to the less quantity of data available but also to the induction of the small sample size problem that further hinders the classification performance in imbalanced situations, which is noticeable in Chi-FRBCS-BigDataCS. To measure the effect of this problem, we present in Table 11 the number of minority and majority class instances by mapper for the Chi-FRBCS-BigData versions. We would like to remind the reader that the number of instances per mapper for Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS is the same, because the initial stage in both algorithms is identical.

This table displays the enormous reduction in the number of samples from each class when larger values for the number of mappers are utilized. In this case, as in the reduced versions, the decrement of the available samples from each class is noticeable, but the influence of the minority class is greater than the influence of the majority class. For the full datasets, in the most difficult scenario, the average number of majority class instances per mapper is 5413.60, which is clearly a fair amount of instances to build a model. However, when we turn to the minority class instances, in the worst case scenario we can find several cases of study that are not even able to provide 1 minority class per mapper, which are usually cases that are not able to properly identify both classes in the test set. Furthermore, when we look at not so dramatic cases of study, we can also find problems with 15 to 20 minority class samples. In these cases, even when there are more instances, the quantity of them is risible with respect to the number of majority class samples, which means that these cases also suffer from the small sample size problem. Furthermore, the small sample size problem aggravates the decrement in the performance for the larger values of the number of mappers. For instance, the *kddcup_normal_versus_R2L* dataset shows an AUC metric of 0.9616 when 8 mappers are used, while this value lowers to 0.8229 when the number of mappers is set to 64.

We acknowledge that this decrement in precision is inevitable when a division of the input data is needed to speed up the classification process; however, these results show that it is of the utmost importance to select an appropriate threshold to perform the data division for the processing, especially in the presence of imbalanced datasets. When a good threshold is established, the downfall in precision is admissible but when that threshold does not fit the problem considered, we can see a lethal reduction in the performance invalidating all the learning process followed due to the small sample size problem.

6.3.2. Analysis on the runtime of the model

Tables 12 and 13 show the time elapsed in seconds and in the hh:mm:ss.SSS format (hours, minutes, seconds, milliseconds) for the reduced versions (10%) of the imbalanced big data cases of study for the Chi-FRBCS and the Chi-FRBCS-BigData alternatives, and for the Chi-FRBCS-CS and Chi-FRBCS-BigDataCS methods respectively. These tables are divided by columns in two parts: the first part corresponds to the results of the sequential variant while the second part is related to the big data variants of the Chi-FRBCS and Chi-FRBCS-CS algorithms respectively. Moreover, the results for the big data versions are divided by columns in five parts which correspond to the number of mappers used: 2, 4, 6, 8 and 16 mappers for each case.

Looking at these tables we can see that, in general, the runtimes obtained by the Chi-FRBCS approaches are slightly lower than the ones obtained by the Chi-FRBCS-CS methods. This behavior is expected as the Chi-FRBCS-CS methods need to perform additional operations with respect to Chi-FRBCS as they include the misclassification costs in their inner way of running. Moreover, the results obtained show that the sequential versions are notably slower than the big data alternatives, even when they are compared with the performance of the big data versions on 2 mappers, as the speed gain is not linearly related to the number of mappers considered. Furthermore, this trend can also be seen among the different number of mappers considered, as the decrement in the running time is reduced meaningfully when the number of mappers is increased. This reduction in the processing time is again not lineal, as this decrement in time is more tangible at the beginning with a lower number of mappers than with a larger number of mappers.

When analyzing the behavior of the groups of cases of study derived from the original datasets we can find different groups of behavior for the cases under consideration. A first group corresponds to the bigger cases of study, the ones derived from the KDD Cup 1999 dataset and the RLCP dataset. In this case, we can see that the general trend perfectly applies to this data: the sequential versions provide runtimes that greatly exceed the results obtained by the MapReduce designs. Furthermore, the usage of higher number of mappers is able to improve the execution times, however, that

Table 11
Average number of minority and majority class instances by mapper for the Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS versions.

Datasets	Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS							
	8 mappers		16 mappers		32 mappers		64 mappers	
	#min class	#maj class	#min class	#maj class	#min class	#maj class	#min class	#maj class
kddcup_DOS_versus_normal	97 254.60	388 360.50	48 627.30	194 180.25	24 313.65	97 090.13	12 156.83	48 545.06
kddcup_DOS_versus_PRB	4120.40	388 326.80	2060.20	194 163.40	1030.10	97 081.70	515.05	48 540.85
kddcup_DOS_versus_R2L	112.20	388 337.40	56.10	194 168.70	28.05	97 084.35	14.03	48 542.18
kddcup_DOS_versus_U2R	4.80	388 337.40	2.40	194 168.70	1.20	97 084.35	0.60	48 542.18
kddcup_normal_versus_PRB	4076.20	97 312.10	2038.10	48 656.05	1019.05	24 328.03	509.53	12 164.02
kddcup_normal_versus_R2L	117.20	97 273.50	58.60	48 636.75	29.30	24 318.38	14.65	12 159.19
kddcup_normal_versus_U2R	4.10	97 279.20	2.05	48 639.60	1.03	24 319.80	0.51	12 159.90
poker_0_vs_2	4933.30	51 319.70	2466.65	25 659.85	1233.32	12 829.93	616.66	6414.96
poker_0_vs_3	2151.60	51 382.00	1075.80	25 691.00	537.90	12 845.50	268.95	6422.75
poker_0_vs_4	398.30	51 369.70	199.15	25 684.85	99.57	12 842.43	49.79	6421.21
poker_0_vs_5	210.40	51 364.80	105.20	25 682.40	52.60	12 841.20	26.30	6420.60
poker_0_vs_6	152.00	51 364.20	76.00	25 682.10	38.00	12 841.05	19.00	6420.52
poker_0_vs_7	22.60	51 371.20	11.30	25 685.60	5.65	12 842.80	2.83	6421.40
poker_0_vs_8	1.90	51 370.00	0.95	25 685.00	0.48	12 842.50	0.24	6421.25
poker_0_vs_9	0.80	51 370.20	0.40	25 685.10	0.20	12 842.55	0.10	6421.27
poker_1_vs_2	4863.20	43 329.30	2431.60	21 664.65	1215.80	10 832.32	607.90	5416.16
poker_1_vs_3	2162.80	43 310.30	1081.40	21 655.15	540.70	10 827.57	270.35	5413.79
poker_1_vs_4	394.10	43 313.40	197.05	21 656.70	98.52	10 828.35	49.26	5414.17
poker_1_vs_5	197.90	43 316.80	98.95	21 658.40	49.47	10 829.20	24.74	5414.60
poker_1_vs_6	142.40	43 313.30	71.20	21 656.65	35.60	10 828.32	17.80	5414.16
poker_1_vs_7	24.50	43 308.80	12.25	21 654.40	6.12	10 827.20	3.06	5413.60
poker_1_vs_8	2.00	43 309.40	1.00	21 654.70	0.50	10 827.35	0.25	5413.67
poker_1_vs_9	1.20	43 309.30	0.60	21 654.65	0.30	10 827.32	0.15	5413.66
RLCP	2097.60	572 815.60	1048.80	286 407.80	524.40	143 203.90	262.20	71 601.95

Table 12
Runtime elapsed in seconds and in the hh:mm:ss.SSS format for the Chi-FRBCS versions.

Datasets	Chi-FRBCS		Chi-FRBCS-BigData									
			2 mappers		4 mappers		6 mappers		8 mappers		16 mappers	
	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS
kddcup_10_DOS_versus_normal	33 670.214	9:21:10.214	14 472.846	4:01:12.846	4140.643	1:09:00.643	1778.238	0:29:38.238	1052.501	0:17:32.501	373.428	**0:06:13.428**
kddcup_10_DOS_versus_PRB	22 510.587	6:15:10.587	9205.046	2:33:25.046	2547.166	0:42:27.166	1165.548	0:19:25.548	704.896	0:11:44.896	249.251	**0:04:09.251**
kddcup_10_DOS_versus_R2L	25 448.700	7:04:08.700	9234.977	2:33:54.977	2492.280	0:41:32.280	1197.892	0:19:57.892	707.083	0:11:47.083	228.386	**0:03:48.386**
kddcup_10_normal_versus_PRB	1756.513	0:29:16.513	1068.581	0:17:48.581	234.787	0:03:54.787	136.951	0:02:16.951	99.986	0:01:39.986	67.127	**0:01:07.127**
kddcup_10_normal_versus_R2L	1599.831	0:26:39.831	875.615	0:14:35.615	215.648	0:03:35.648	128.611	0:02:08.611	94.273	0:01:34.273	62.629	**0:01:02.629**
Average (kddcup)	16 997.169	4:43:17.169	6971.413	1:56:11.413	1926.105	0:32:06.105	881.448	0:14:41.448	531.748	0:08:51.748	196.164	**0:03:16.164**
poker_10_0_vs_2	1022.735	0:17:02.735	586.696	0:09:46.696	563.302	0:09:23.302	519.465	**0:08:39.465**	562.220	0:09:22.220	624.649	0:10:24.649
poker_10_0_vs_3	832.182	0:13:52.182	575.673	0:09:35.673	515.892	**0:08:35.892**	543.217	0:09:03.217	539.140	0:08:59.140	579.760	0:09:39.760
poker_10_0_vs_4	798.369	0:13:18.369	605.735	0:10:05.735	504.460	**0:08:24.460**	552.799	0:09:12.799	610.555	0:10:10.555	567.566	0:09:27.566
poker_10_0_vs_5	1209.566	0:20:09.566	596.255	0:09:56.255	522.352	0:08:42.352	512.363	0:08:32.363	507.853	0:08:27.853	504.863	**0:08:24.863**
poker_10_0_vs_6	1051.263	0:17:31.263	682.470	0:11:22.470	614.844	0:10:14.844	536.008	0:08:56.008	581.397	0:09:41.397	459.080	**0:07:39.080**
poker_10_0_vs_7	963.291	0:16:03.291	520.087	0:08:40.087	460.601	**0:07:40.601**	481.295	0:08:01.295	479.882	0:07:59.882	473.661	0:07:53.661
poker_10_1_vs_2	796.636	0:13:16.636	439.989	0:07:19.989	390.712	**0:06:30.712**	398.359	0:06:38.359	406.932	0:06:46.932	399.267	0:06:39.267
poker_10_1_vs_3	734.307	0:12:14.307	410.816	0:06:50.816	383.222	**0:06:23.222**	416.784	0:06:56.784	409.623	0:06:49.623	424.023	0:07:04.023
poker_10_1_vs_4	645.596	0:10:45.596	442.978	0:07:22.978	421.791	0:07:01.791	411.050	0:06:51.050	401.203	0:06:41.203	377.247	**0:06:17.247**
poker_10_1_vs_5	547.979	0:09:07.979	395.322	0:06:35.322	366.879	0:06:06.879	358.951	**0:05:58.951**	377.911	0:06:17.911	379.750	0:06:19.750
poker_10_1_vs_6	697.428	0:11:37.428	393.996	0:06:33.996	366.409	0:06:06.409	370.342	0:06:10.342	366.044	0:06:06.044	360.484	**0:06:00.484**
poker_10_1_vs_7	690.171	0:11:30.171	381.735	0:06:21.735	363.016	0:06:03.016	353.859	0:05:53.859	347.198	**0:05:47.198**	353.263	0:05:53.263
Average (poker)	832.460	0:13:52.460	502.646	0:08:22.646	456.123	0:07:36.123	454.541	**0:07:34.541**	465.830	0:07:45.830	458.634	0:07:38.634
RLCP_10	8683.187	2:24:43.187	4420.900	1:13:40.900	1174.823	0:19:34.823	562.273	0:09:22.273	369.080	0:06:09.080	179.112	**0:02:59.112**
Total average	5758.809	1:35:58.809	2517.207	0:41:57.207	904.379	0:15:04.379	579.111	0:09:39.111	478.765	0:07:58.765	370.197	**0:06:10.197**

Table 13
Runtime elapsed in seconds and in the hh:mm:ss.SSS format for the Chi-FRBCS cost-sensitive versions.

Datasets	Chi-FRBCS-CS		Chi-FRBCS-BigDataCS									
			2 mappers		4 mappers		6 mappers		8 mappers		16 mappers	
	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS
kddcup_10_DOS_versus_normal	32 892.936	9:08:12.936	15 414.766	4:16:54.766	4162.916	1:09:22.916	1843.283	0:30:43.283	1076.427	0:17:56.427	415.060	**0:06:55.060**
kddcup_10_DOS_versus_PRB	29 981.173	8:19:41.173	9798.818	2:43:18.818	2585.928	0:43:05.928	1188.691	0:19:48.691	758.516	0:12:38.516	250.027	**0:04:10.027**
kddcup_10_DOS_versus_R2L	25 498.727	7:04:58.727	9649.161	2:40:49.161	2543.190	0:42:23.190	1206.790	0:20:06.790	673.231	0:11:13.231	266.537	**0:04:26.537**
kddcup_10_normal_versus_PRB	1730.916	0:28:50.916	1066.950	0:17:46.950	236.391	0:03:56.391	139.344	0:02:19.344	101.652	0:01:41.652	68.790	**0:01:08.790**
kddcup_10_normal_versus_R2L	1578.991	0:26:18.991	1072.229	0:17:52.229	212.853	0:03:32.853	134.370	0:02:14.370	94.847	0:01:34.847	65.116	**0:01:05.116**
Average (kddcup)	18 336.549	5:05:36.549	7400.385	2:03:20.385	1948.256	0:32:28.256	902.496	0:15:02.496	540.934	0:09:00.934	213.106	**0:03:33.106**
poker_10_0_vs_2	1804.004	0:30:04.004	777.028	0:12:57.028	552.463	**0:09:12.463**	557.443	0:09:17.443	579.395	0:09:39.395	572.595	0:09:32.595
poker_10_0_vs_3	1315.612	0:21:55.612	641.281	0:10:41.281	609.231	0:10:09.231	516.789	0:08:36.789	507.824	**0:08:27.824**	558.286	0:09:18.286
poker_10_0_vs_4	1630.399	0:27:10.399	764.142	0:12:44.142	607.250	0:10:07.250	582.549	0:09:42.549	581.717	0:09:41.717	572.004	**0:09:32.004**
poker_10_0_vs_5	1234.801	0:20:34.801	636.422	0:10:36.422	568.459	0:09:28.459	551.079	0:09:11.079	546.558	0:09:06.558	534.967	**0:08:54.967**
poker_10_0_vs_6	1456.625	0:24:16.625	828.762	0:13:48.762	652.770	0:10:52.770	659.502	0:10:59.502	620.019	0:10:20.019	470.053	**0:07:50.053**
poker_10_0_vs_7	1778.488	0:29:38.488	638.797	0:10:38.797	510.469	0:08:30.469	479.955	**0:07:59.955**	489.636	0:08:09.636	485.763	0:08:05.763
poker_10_1_vs_2	1137.676	0:18:57.676	496.250	0:08:16.250	425.849	0:07:05.849	396.363	**0:06:36.363**	396.681	0:06:36.681	397.112	0:06:37.112
poker_10_1_vs_3	1116.075	0:18:36.075	464.625	0:07:44.625	371.895	0:06:11.895	350.017	0:05:50.017	349.021	**0:05:49.021**	443.020	0:07:23.020
poker_10_1_vs_4	1220.649	0:20:20.649	498.319	0:08:18.319	403.404	0:06:43.404	385.980	0:06:25.980	398.769	0:06:38.769	368.581	**0:06:08.581**
poker_10_1_vs_5	1318.547	0:21:58.547	446.729	0:07:26.729	367.710	**0:06:07.710**	367.759	0:06:07.759	369.085	0:06:09.085	380.618	0:06:20.618
poker_10_1_vs_6	1453.041	0:24:13.041	478.021	0:07:58.021	394.152	0:06:34.152	377.446	0:06:17.446	351.418	**0:05:51.418**	362.195	0:06:02.195
poker_10_1_vs_7	1124.129	0:18:44.129	499.303	0:08:19.303	385.227	0:06:25.227	373.933	0:06:13.933	367.799	0:06:07.799	362.491	**0:06:02.491**
Average (poker)	1382.504	0:23:02.504	597.473	0:09:57.473	487.407	0:08:07.407	466.568	0:07:46.568	463.160	0:07:43.160	458.974	**0:07:38.974**
RLCP_10	8159.285	2:15:59.285	4165.285	1:09:25.285	1154.706	0:19:14.706	505.405	0:08:25.405	295.675	0:04:55.675	188.454	**0:03:08.454**
Total average	6468.449	1:47:48.449	2685.383	0:44:45.383	930.270	0:15:30.270	589.817	0:09:49.817	475.459	0:07:55.459	375.648	**0:06:15.648**

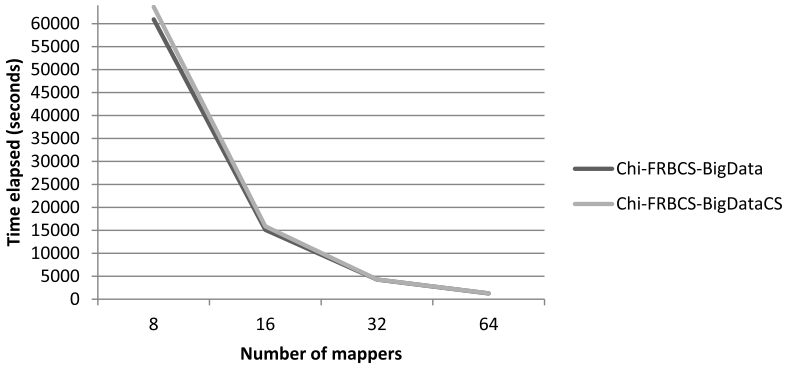

Fig. 4. Execution times for the *kddcup_full_DOS_versus_U2R* dataset.

advance is better observed when the number of mappers is smaller than in the larger cases, that is, when the data available per mapper is considerable.

The second group is related to the Poker Hand cases of study, where the processing time gain is not as clear as in the previous cases. Without a doubt, we can state there are huge differences between the sequential versions and the Hadoop implementations. When the big data versions are compared, the runtime improvement can only be detected for the smaller values of the number of mappers. The Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS algorithms seem to no further improve their behavior starting from 16 mappers.

Table 14 shows the average runtime spent in seconds and in the hh:mm:ss.SSS format for the full cases of study by the Chi-FRBCS-BigData and Chi-FRBCS-BigDataCS algorithms. This table is organized in two big parts: the first part is related to the results obtained by the Chi-FRBCS-BigData algorithm while the second part is related to the Chi-FRBCS-BigDataCS method. Similarly to the preceding tables, these algorithms present their information in four columns related to the number of mappers considered: 8, 16, 32 and 64 respectively. The sequential versions are not included in this table as they are not able to provide a result, as it was shown in the scalability studies (Sections 3.3 and 4.2).

In this table, we can observe that the Chi-FRBCS-BigData approach shows a trend that slightly benefits its runtime, however, it does not always surpass the runtime achieved by the Chi-FRBCS-BigDataCS algorithm for any number of mappers. These results can be understood in the following manner: the Chi-FRBCS-BigData approach is a less complex approach than the Chi-FRBCS-BigDataCS method and therefore, the second algorithm is bound to spend more processing time due to its additional operations. The usage of cost-sensitive learning is thus a good alternative as this time addition is insignificant compared to the performance improvement gained in imbalanced datasets. In Fig. 4, we can see the difference between the performance of the big data alternatives for the *kddcup_full_DOS_versus_U2R* dataset, where the Chi-FRBCS-BigDataCS version consumes a bit more of time. However, the Chi-FRBCS-BigDataCS tends to produce a lesser number of rules (scalability studies in Sections 3.3 and 4.2), and therefore the search for identical rules may also be less computationally demanding.

In general, when larger values for the number of mappers are used, better runtime results are obtained for both the Chi-FRBCS-BigData and the Chi-FRBCS-BigDataCS algorithms. However, the improvement in the processing times is not linearly related to the number of mappers, as smaller number of mappers show a greater performance gain than larger values of mappers.

If we analyze the behavior of the groups of cases of study derived from the original datasets we can also observe the same groups of behavior as in the reduced cases of study previously considered. Again, a first group corresponds to the bigger cases of study, the ones derived from the KDD Cup 1999 dataset and the RLCP dataset. This group displays the general trend extracted from all the data: the usage of higher number of mappers can get faster execution times, however, the runtime improvement is better appreciated with a reduced number of mappers instead of with larger values, that means, when the data available per mapper is abundant. Fig. 4 also presents the trend in the usage of different mappers.

The second group is related to the Poker Hand cases of study, where it is not possible to discern an improvement in the processing times. For the smaller values of the number of mappers, the results obtained show equivalent results,

Table 14
Runtime elapsed in seconds and in the hh:mm:ss.SSS format for the big data Chi-FRBCS versions.

Datasets	Chi-FRBCS-BigData								Chi-FRBCS-BigDataCS							
	8 mappers		16 mappers		32 mappers		64 mappers		8 mappers		16 mappers		32 mappers		64 mappers	
	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS	seconds	hh:mm:ss.SSS
kddcup_DOS_versus_normal	95 135.040	26:25:35.040	26 422.546	7:20:22.546	9678.697	2:41:18.697	4060.908	1:07:40.908	96 833.551	26:53:53.551	25 824.469	7:10:24.469	7692.155	2:08:12.155	3407.801	**0:56:47.801**
kddcup_DOS_versus_PRB	62 034.217	17:13:54.217	17 206.961	4:46:46.961	5336.043	1:28:56.043	5310.406	1:28:30.406	64 827.368	18:00:27.368	18 003.649	5:00:03.649	6094.751	1:41:34.751	5134.697	**1:25:34.697**
kddcup_DOS_versus_R2L	60 908.738	16:55:08.738	15 615.652	4:20:15.652	6315.864	1:45:15.864	1789.012	**0:29:49.012**	62 059.365	17:14:19.365	16 897.451	4:41:37.451	6122.615	1:42:02.615	2047.410	0:34:07.410
kddcup_DOS_versus_U2R	60 942.589	16:55:42.589	15 114.415	4:11:54.415	4288.956	1:11:28.956	1266.369	**0:21:06.369**	63 665.339	17:41:05.339	15 870.638	4:24:30.638	4302.037	1:11:42.037	1281.801	0:21:21.801
kddcup_normal_versus_PRB	6059.310	1:40:59.310	1765.673	0:29:25.673	548.857	0:09:08.857	262.545	**0:04:22.545**	6155.110	1:42:35.110	1523.940	0:25:23.940	753.089	0:12:33.089	301.214	0:05:01.214
kddcup_normal_versus_R2L	4362.339	1:12:42.339	1807.435	0:30:07.435	451.027	0:07:31.027	279.080	**0:04:39.080**	4502.856	1:15:02.856	1274.320	0:21:14.320	503.814	0:08:23.814	329.662	0:05:29.662
kddcup_normal_versus_U2R	4279.778	1:11:19.778	1899.410	0:31:39.410	597.729	0:09:57.729	350.514	0:05:50.514	5064.459	1:24:24.459	1290.801	0:21:30.801	730.109	0:12:10.109	327.904	**0:05:27.904**
Average (kddcup)	41 960.287	11:39:20.287	11 404.584	3:10:04.584	3888.167	1:04:48.167	1902.691	0:31:42.691	43 301.150	12:01:41.150	11 526.467	3:12:06.467	3742.653	1:02:22.653	1832.927	**0:30:32.927**
poker_0_vs_2	12 320.901	3:25:20.901	12 325.506	3:25:25.506	12 839.608	3:33:59.608	13 612.564	3:46:52.564	12 506.996	3:28:26.996	12 083.205	**3:21:23.205**	12 851.936	3:34:11.936	13 292.345	3:41:32.345
poker_0_vs_3	11 401.855	3:10:01.855	11 659.858	3:14:19.858	12 448.827	3:27:28.827	13 212.002	3:40:12.002	11 484.098	3:11:24.098	11 393.884	**3:09:53.884**	12 059.245	3:20:59.245	12 349.548	3:25:49.548
poker_0_vs_4	11 093.366	**3:04:53.366**	11 244.520	3:07:24.520	12 155.162	3:22:35.162	12 350.617	3:25:50.617	11 161.645	3:06:01.645	11 380.513	3:09:40.513	12 096.645	3:21:36.645	12 087.170	3:21:27.170
poker_0_vs_5	10 947.363	3:02:27.363	10 870.675	3:01:10.675	11 926.168	3:18:46.168	12 341.788	3:25:41.788	11 370.870	3:09:30.870	10 810.724	**3:00:10.724**	11 944.131	3:19:04.131	12 262.003	3:24:22.003
poker_0_vs_6	10 977.253	**3:02:57.253**	11 041.819	3:04:01.819	11 674.877	3:14:34.877	12 194.412	3:23:14.412	11 344.606	3:09:04.606	11 260.872	3:07:40.872	11 915.543	3:18:35.543	11 807.885	3:16:47.885
poker_0_vs_7	10 971.631	**3:02:51.631**	11 158.933	3:05:58.933	11 778.574	3:16:18.574	12 228.561	3:23:48.561	11 851.595	3:17:31.595	11 624.443	3:13:44.443	11 963.442	3:19:23.442	11 887.682	3:18:07.682
poker_0_vs_8	11 040.804	**3:04:00.804**	11 088.482	3:04:48.482	11 615.557	3:13:35.557	12 280.418	3:24:40.418	11 790.836	3:16:30.836	11 227.721	3:07:07.721	11 679.059	3:14:39.059	11 809.133	3:16:49.133
poker_0_vs_9	11 059.629	**3:04:19.629**	11 130.037	3:05:30.037	12 039.400	3:20:39.400	11 956.152	3:19:16.152	11 386.511	3:09:46.511	11 681.637	3:14:41.637	11 977.673	3:19:37.673	12 152.204	3:22:32.204
poker_1_vs_2	10 502.985	2:55:02.985	10 592.520	2:56:32.520	10 823.188	3:00:23.188	11 550.568	3:12:30.568	10 256.908	**2:50:56.908**	10 395.012	2:53:15.012	10 769.729	2:59:29.729	11 198.647	3:06:38.647
poker_1_vs_3	9734.080	2:42:14.080	10 232.695	2:50:32.695	10 770.971	2:59:30.971	10 643.134	2:57:23.134	9661.590	**2:41:01.590**	9769.442	2:42:49.442	10 434.828	2:53:54.828	10 584.726	2:56:24.726
poker_1_vs_4	9362.164	**2:36:02.164**	9599.178	2:39:59.178	9981.443	2:46:21.443	10 553.633	2:55:53.633	9443.253	2:37:23.253	9752.424	2:42:32.424	9765.667	2:42:45.667	9806.559	2:43:26.559
poker_1_vs_5	9298.083	**2:34:58.083**	9637.974	2:40:37.974	10 428.014	2:53:48.014	10 399.248	2:53:19.248	9412.589	2:36:52.589	9506.839	2:38:26.839	9942.829	2:45:42.829	10 262.902	2:51:02.902
poker_1_vs_6	9009.779	**2:30:09.779**	9591.369	2:39:51.369	10 112.862	2:48:32.862	10 407.752	2:53:27.752	9739.623	2:42:19.623	9854.607	2:44:14.607	9963.349	2:46:03.349	10 095.291	2:48:15.291
poker_1_vs_7	9285.360	2:34:45.360	9250.462	**2:34:10.462**	9962.175	2:46:02.175	10 333.898	2:52:13.898	9580.927	2:39:40.927	9670.806	2:41:10.806	10 300.841	2:51:40.841	10 276.786	2:51:16.786
poker_1_vs_8	9545.055	2:39:05.055	9380.564	**2:36:20.564**	9872.084	2:44:32.084	10 226.082	2:50:26.082	9830.342	2:43:50.342	9422.569	2:37:02.569	9912.194	2:45:12.194	10 300.646	2:51:40.646
poker_1_vs_9	9179.436	**2:32:59.436**	9438.347	2:37:18.347	9893.532	2:44:53.532	10 335.326	2:52:15.326	9776.855	2:42:56.855	9844.250	2:44:04.250	10 195.108	2:49:55.108	10 476.054	2:54:36.054
Average (poker)	10 358.109	**2:52:38.109**	10 515.184	2:55:15.184	11 145.153	3:05:45.153	11 539.135	3:12:19.135	10 662.453	2:57:42.453	10 604.934	2:56:44.934	11 110.764	3:05:10.764	11 290.599	3:08:10.599
RLCP	26 551.162	7:22:31.162	7089.999	1:58:09.999	1922.670	0:32:02.670	606.831	**0:10:06.831**	27 547.418	7:39:07.418	7270.635	2:01:10.635	1830.273	0:30:30.273	721.305	0:12:01.305
Final average	20 250.122	5:37:30.122	10 631.876	2:57:11.876	8644.262	2:24:04.262	8272.992	2:17:52.992	20 885.613	5:48:05.613	10 734.785	2:58:54.785	8575.044	2:22:55.044	8091.724	**2:14:51.724**

however, when larger values of mappers are considered, the runtime does not improve and it can even become worse. This situation arises due to the smaller size of the Poker Hand cases of study.

Finally, it is necessary to recall that even when a larger number of mappers tend to provide better response times it may not be wise to try to expand that number as much as possible. As we observed in Section 6.3.1, a large number of mappers may cause a dramatically drop in the performance, an unwanted case when trying to extract information from data. Therefore, it is needed to analyze the case under consideration to select an appropriate number of mappers for the experiment. This number of mappers needs to provide a reasonable number of samples for each class to avoid the small sample size problem and also enough data so that the experiments obtain lesser response times.

To sum up, our experimental study shows that cost-sensitive learning allows us to obtain better classification results for the Chi-FRBCS algorithm. We have also observed that, in the big data versions, increasing the number of mappers decreases the accuracy of the model, not only because the full information is not available but also because of the induction of data intrinsic problems that difficult the classification with imbalanced datasets, such as the small sample size problem. Finally, big data versions allow us to deal with huge amounts of data and obtain better response times which are generally significantly decremented when the number of mappers of the original dataset is increased.

7. Concluding remarks

In this paper, we have introduced a linguistic cost-sensitive fuzzy rule-based classification method for imbalanced big data called Chi-FRBCS-BigDataCS. Our aim was to obtain a model that is able to handle imbalanced big data obtaining a good precision without incrementing the processing times. To do so, we use one of the most popular approaches nowadays to deal with big data: the MapReduce framework, distributing the algorithm computing along different processing units using the map and reduce operations that have been adapted to the calculations of the fuzzy rule based classification system. We have also used cost-sensitive learning operations which have also modified the algorithm to consider the misclassification costs, proposing a new approach, PCF-CS, to compute the rule weight that consider these costs in its operations.

The experiments conducted in this work demonstrate that the MapReduce framework is able of dealing with big data for fuzzy rule based classification systems. The use of a simple but effective fuzzy rule based classification system such as the Chi et al.'s method as base of the approach has enabled the development of a proposal that can profit from this simplicity to create an efficient approach. The proposal, Chi-FRBCS-BigDataCS, can obtain classification results when its sequential counterpart was not able to provide results. Furthermore, the runtime needed by the proposal is admissible according to the results presented. The inclusion of cost-sensitive learning in its way of working, using the new rule weight procedure PCF-CS, has demonstrated to be a powerful collaborator when dealing with imbalanced datasets providing effective classification results without largely increasing the processing times.

The performance of our model, Chi-FRBCS-BigDataCS, has been tested in an experimental study including twenty-four imbalanced big data cases of study. These results corroborate the goodness of the integration of the approaches that are used to solve the imbalanced problem and big data separately, namely the usage of the MapReduce framework and cost-sensitive learning. Furthermore, the synergy between both strategies alleviates some data intrinsic problems, like the small sample size problem, that are induced because of the way the learning is done.

Acknowledgements

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] IBM, What is big data? Bringing big data to the enterprise, [Online; accessed December 2013], http://www-01.ibm.com/software/data/bigdata/, 2012.
[2] P. Zikopoulos, C. Eaton, D. DeRoos, T. Deutsch, G. Lapis, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw-Hill, 2011.
[3] S. Madden, From databases to big data, IEEE Internet Comput. 16 (3) (2012) 4–6.
[4] A. Sathi, Big Data Analytics: Disruptive Technologies for Changing the Game, MC Press, 2012.

[5] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[6] 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.
[7] 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, Inf. Sci. 250 (2013) 113–141.
[8] H. Ishibuchi, T. Nakashima, M. Nii, Classification and Modeling with Linguistic Information Granules: Advanced Approaches to Linguistic Data Mining, Springer-Verlag, 2004.
[9] Y. Jin, Fuzzy modeling of high-dimensional systems: complexity reduction and interpretability improvement, IEEE Trans. Fuzzy Syst. 8 (2) (2000) 212–221.
[10] T.-P. Hong, Y.-C. Lee, M.-T. Wu, An effective parallel approach for genetic-fuzzy data mining, Expert Syst. Appl. 41 (2) (2014) 655–662.
[11] M. Rodríguez, D. Escalante, A. Peregrín, Efficient distributed genetic algorithm for rule extraction, Appl. Soft Comput. 11 (1) (2011) 733–743.
[12] Y. Nojima, H. Ishibuchi, I. Kuwajima, Parallel distributed genetic fuzzy rule selection, Soft Comput. 13 (5) (2009) 511–519.
[13] I. Robles, R. Alcalá, J. Benítez, F. Herrera, Evolutionary parallel and gradually distributed lateral tuning of fuzzy rule-based systems, Evol. Intel. 2 (1–2) (2009) 5–19.
[14] H. Ishibuchi, S. Mihara, Y. Nojima, Parallel distributed hybrid fuzzy GBML models with rule set migration and training data rotation, IEEE Trans. Fuzzy Syst. 21 (2) (2013) 355–368.
[15] 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, Expert Syst. Appl. 39 (7) (2012) 6585–6608.
[16] 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.
[17] 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.
[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] 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.
[20] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[21] 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.
[22] 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.
[23] Z. Chi, H. Yan, T. Pham, Fuzzy Algorithms with Applications to Image Processing and Pattern Recognition, World Scientific, 1996.
[24] T. Nakashima, G. Schaefer, Y. Yokota, H. Ishibuchi, Weighted fuzzy classifier and its application to image processing tasks, Fuzzy Sets Syst. 158 (2007) 284–294.
[25] 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.
[26] A. Fernández, S. García, M.J. del Jesus, F. Herrera, A study of the behaviour of linguistic fuzzy rule based classification systems in the framework of imbalanced data-sets, Fuzzy Sets Syst. 159 (18) (2008) 2378–2398.
[27] A. Fernández, M.J. del Jesus, F. Herrera, Hierarchical fuzzy rule based classification systems with genetic rule selection for imbalanced data-sets, Int. J. Approx. Reason. 50 (3) (2009) 561–577.
[28] 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, in: Lecture Notes on Artificial Intelligence, vol. 6086, 2010, pp. 158–167.
[29] J.A. Sáez, J. Luengo, F. Herrera, A first study on the noise impact in classes for fuzzy rule based classification systems, in: Proceedings of the 2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering, ISKE'10, IEEE Press, 2010, pp. 153–158.
[30] G.M. Weiss, The impact of small disjuncts on classifier learning, in: R. Stahlbock, S.F. Crone, S. Lessmann (Eds.), Data Mining, in: Annals of Information Systems, vol. 8, Springer, 2010, pp. 193–226.
[31] J. Huang, C.X. Ling, Using AUC and accuracy in evaluating learning algorithms, IEEE Trans. Knowl. Data Eng. 17 (3) (2005) 299–310.
[32] T. White, Hadoop, The Definitive Guide, O'Reilly Media, Inc., 2012.
[33] D. Laney, 3D data management: Controlling data volume, velocity, and variety, META Group, 2001, Tech. rep., [Online; accessed December 2013], http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.
[34] M. Beyer, Gartner says solving big data challenge involves more than just managing volumes of data, [Online; accessed December 2013], 2011, http://www.gartner.com/newsroom/id/1731916.
[35] M. Beyer, D. Laney, The importance of big data: A definition, ID: G00235055, Retrieved from Gartner database [Online; accessed December 2013], 2012, http://www.gartner.com/id=2057415.
[36] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, in: Proceedings of the 6th Symposium on Operating System Design and Implementation, OSDI 2004, 2004, pp. 137–150.
[37] J. Dean, S. Ghemawat, MapReduce: A flexible data processing tool, Commun. ACM 53 (1) (2010) 72–77.
[38] C. Lam, Hadoop in Action, Manning Publications Co., 2010.
[39] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, Manning Publications Co., 2011.
[40] J. Lin, MapReduce is good enough? If all you have is a hammer, throw away everything that's not a nail!, Big Data 1 (1) (2013) 28–37.
[41] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, 2012.

[42] Apache Drill Project, Apache Drill, 2013, [Online; December 2013, accessed], http://incubator.apache.org/drill/.
[43] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, G. Fox, Twister: a runtime for iterative MapReduce, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010), 2010, pp. 810–818.
[44] S. Das, Y. Sismanis, K.S. Beyer, R. Gemulla, P.J. Haas, J. McPherson, Ricardo: integrating R and Hadoop, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2010), 2010, pp. 987–998.
[45] M. Bostock, O.V., J. Heer, D3 data-driven documents, IEEE Trans. Vis. Comput. Graph. 17 (12) (2011) 2301–2309.
[46] HCatalog, Hcatalog, [Online; accessed December 2013, accessed] http://hive.apache.org/hcatalog/ (2013).
[47] J. Leibiusky, G. Eisbruch, D. Simonassi, Getting Started with Storm, O'Reilly Media, Inc., 2012.
[48] Cloudera, Cloudera Impala, [Online; accessed December 2013] (2013). http://www.cloudera.com/content/cloudera/en/products/cdh/impala.html.
[49] Q. Yang, X. Wu, 10 challenging problems in data mining research, Int. J. Inf. Technol. Decis. Mak. 5 (4) (2006) 597–604.
[50] T. Khoshgoftaar, K. Gao, A. Napolitano, R. Wald, A comparative study of iterative and non-iterative feature selection techniques for software defect prediction, Inf. Syst. Front., in press, http://dx.doi.org/10.1007/s10796-013-9430-0.
[51] S. Wang, X. Yao, Using class imbalance learning for software defect prediction, IEEE Trans. Reliab. 62 (2) (2013) 434–443.
[52] L. Zhou, Performance of corporate bankruptcy prediction models on imbalanced dataset: The effect of sampling methods, Knowl.-Based Syst. 41 (2013) 16–25.
[53] A. Gudys, M. Szczesniak, M. Sikora, I. Makalowska, HuntMi: An efficient and taxon-specific approach in pre-miRNA identification, BMC Bioinform. 14 (2013) 1–10, Article number 83.
[54] Q. Wei, R. Dunbrack Jr., The role of balanced training and testing data sets for binary classifiers in bioinformatics, PLoS ONE 8 (7) (2013) 1–12, Article number e67863.
[55] 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.
[56] Y.-H. Lee, P. Hu, T.-H. Cheng, T.-C. Huang, W.-Y. Chuang, A preclustering-based ensemble learning technique for acute appendicitis diagnoses, Artif. Intell. Med. 58 (2) (2013) 115–124.
[57] J. Nahar, T. Imam, K. Tickle, Y.-P. Chen, Computational intelligence for heart disease diagnosis: A medical knowledge driven approach, Expert Syst. Appl. 40 (1) (2013) 96–104.
[58] A. Orriols-Puig, E. Bernadó-Mansilla, Evolutionary rule-based systems for imbalanced datasets, Soft Comput. 13 (3) (2009) 213–225.
[59] 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.
[60] 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, Inf. Sci. 259 (2014) 571–595.
[61] J. Stefanowski, Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data, in: Smart Innovation, Systems and Technologies, vol. 13, 2013, pp. 277–306.
[62] A. Storkey, When training and test sets are different: Characterizing learning transfer, in: J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, N.D. Lawrence (Eds.), Dataset Shift in Machine Learning, MIT Press, 2009, pp. 3–28.
[63] H. Shimodaira, Improving predictive inference under covariate shift by weighting the log-likelihood function, J. Stat. Plan. Inference 90 (2) (2000) 227–244.
[64] V. López, I. Triguero, C. Carmona, S. García, F. Herrera, Addressing imbalanced classification with instance generation techniques: IPADE-ID, Neurocomputing 126 (2014) 15–28.
[65] 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.
[66] 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.
[67] 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 Cybern., Part C, Appl. Rev. 42 (4) (2012) 463–484.
[68] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.
[69] H. Ishibuchi, T. Nakashima, Effect of rule weights in fuzzy rule-based classification systems, IEEE Trans. Fuzzy Syst. 9 (4) (2001) 506–515.
[70] H. Ishibuchi, T. Yamamoto, Rule weight specification in fuzzy rule-based classification systems, IEEE Trans. Fuzzy Syst. 13 (2005) 428–435.
[71] O. Cordón, M.J. del Jesus, F. Herrera, A proposal on reasoning methods in fuzzy rule-based classification systems, Int. J. Approx. Reason. 20 (1) (1999) 21–45.
[72] L.X. Wang, J.M. Mendel, Generating fuzzy rules by learning from examples, IEEE Trans. Syst. Man Cybern. 22 (6) (1992) 1414–1427.
[73] K. Bache, M. Lichman, UCI machine learning repository, [Online; accessed December 2013], 2013, http://archive.ics.uci.edu/ml.
[74] M. Fazzolari, B. Giglio, R. Alcalá, F. Marcelloni, F. Herrera, A study on the application of instance selection techniques in genetic fuzzy rule-based classification systems: Accuracy-complexity trade-off, Knowl.-Based Syst. 54 (2014) 32–41.
[75] J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, F. Herrera, KEEL: a software tool to assess evolutionary algorithms for data mining problems, Soft Comput. 13 (2009) 307–318.
[76] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework, J. Mult.-Valued Log. Soft Comput. 17 (2–3) (2011) 255–287.