A MapReduce Approach to Address Big Data Classification Problems Based on the Fusion of Linguistic Fuzzy Rules

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

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

Received 28 November 2014

Accepted 9 January 2015

Abstract

The big data term is used to describe the exponential data growth that has recently occurred and represents an immense challenge for traditional learning techniques. To deal with big data classification problems we propose the Chi-FRBCS-BigData algorithm, a linguistic fuzzy rule-based classification system that uses the MapReduce framework to learn and fuse rule bases. It has been developed in two versions with different fusion processes. An experimental study is carried out and the results obtained show that the proposal is able to handle these problems providing competitive results.

Keywords: Fuzzy rule based classification systems; Big data; MapReduce; Hadoop; Rules fusion.

1. Introduction

Along the last years, one of the interesting trends in the information technology industry is what is known as “big data.” This popular term is used when referring to massive amounts of data that are difficult to handle and analyze using traditional data management tools.[1] These include structured and unstructured, data featuring from terabytes to zettabytes and coming from diverse sources such as social networks, mobile devices, multimedia data, webpages or sensor networks among others.[2]

More data should lead to more effective analysis and therefore, should enable the extraction of more accurate and precise information. Nevertheless, the standard machine learning and data mining techniques are not able to easily scale up to big data problems. [3]In this way, it is necessary to adapt and redesign the standard learning algorithms considering the existing solutions to address these problems. [4,5,6]

Fuzzy Rule Based Classification Systems (FRBCSs) are effective and popular tools for pattern recognition and classification.[7] These techniques are able to obtain good accuracy results while providing a descriptive model for the end user through the usage of linguistic labels. When dealing with big data, one of the issues that hinders the extraction of information is the uncertainty that is associated to the vagueness or the noise inherent to the available data. Therefore, FRBCSs seem appropriate in this scenario as they can handle uncertainty, ambiguity or vagueness in a very effective manner. Another issue that complicates the learning process is the high dimensionality and the large number of instances that are present in big data, since the inductive learn-

[*]Corresponding author. Tel:+34-958-240598; Fax: +34-958-243317.
[†]E-mail: srio@decsai.ugr.es (Sara del Río), vlopez@decsai.ugr.es (Victoria López), J.M.Benitez@decsai.ugr.es (José Manuel Benítez), herrera@decsai.ugr.es (Francisco Herrera).

ing capacity of FRBCSs is affected by the exponential growth of the search space.[8] To address this problem, several approaches have appeared to build parallel fuzzy systems [9,10]; however, these models aim to reduce the processing time while preserving the accuracy and they are not designed to manage huge amounts of data. In this way, it is necessary to adapt and redesign the FRBCSs so that they are able to provide an accurate classification in a reasonable amount of time in big data problems.

One of the most popular paradigms nowadays for addressing big data is MapReduce,[11] a new distributed programming model that organizes the computation into two key operations: the *map* function that is responsible for splitting the original dataset and processing each sub-problem independently, and the *reduce* function that collects and aggregates the results from the *map* function.

In this paper, we present an FRBCS that can deal with big data classification problems providing an interpretable model and competitive accuracy results. We extend the conference contribution presented in [13]. Our proposal, denoted as Chi-FRBCS-BigData, is based on the Chi *et al.'s* approach,[12] a classical FRBCS learning method which has been adapted to deal with big data following a MapReduce scheme.

The fusion of linguistic fuzzy rules is a fundamental task in the approach due to the nature of the MapReduce procedure, which divides the original dataset into blocks. The Chi-FRBCS-BigData algorithm can generate contradictory rules (rules with the same antecedent, with or without the same consequent and with different rule weights), so it is necessary to address how the fusion of rules is performed with specific procedures. To do this, two different versions of Chi-FRBCS-BigData approach have been developed: Chi-FRBCS-BigData-Max and Chi-FRBCS-BigData-Ave. While both versions share most of their operations, they differ in the *reduce* step of the approach, where the generated rule bases are combined. Therefore, these variants of Chi-FRBCS-BigData algorithm obtain different final rule bases.

Furthermore, the choice of the Chi *et al's* method over any other FRBCS method is due to its intrinsic characteristics that make it particularly suitable to build a parallel approach:

- It is a simple approach that does not have strong dependencies between parts of the algorithm.
- It generates rules that have the same structure (rules with as many antecedents as attributes in the dataset using only a fuzzy label). Having the same structure for the rules greatly benefits both the rule generation from a subset of the data and the combination of rules, a process carried out in the *reduce* phase of the proposed approach.

In order to evaluate the performance of the Chi-FRBCS-BigData algorithm, we have designed and carried out an experimental study based in six binary big data problems. The classification is evaluated using the accuracy obtained and the runtime spent by the models, which will also help to understand the strong points and limitations of both versions of the proposal.

The rest of this paper is organized as follows. Section 2 provides some background information about big data. Section 3 introduces some general concepts about FRBCSs and describes the Chi *et al's* algorithm. Section 4 describes the approaches proposed in this work, the versions of the Chi-FRBCS-BigData algorithm to deal with big data. Then, Section 5 shows the configuration of the experimental study, a first scalability study to show the inability of the Chi *et al's* algorithm to deal with big data classification problems and the study with the results and their analysis over the six big data problems. Finally, the conclusions achieved in this work are shown in Section 6.

2. Big data and the MapReduce programming model

In this section we present some background about big data. Section 2.1 provides an introduction to big data. Section 2.2 provides a detailed description about the MapReduce programming model.

2.1. Introduction to big data

The big data term is related to the exponential growth in data generation that has taken place in the last years and has raised considerable interest because of the possibilities in the improvement in the data processing and knowledge extraction. Big data is the popular term to encompass all the data so large and complex that it becomes difficult to process or analyze using traditional software tools or data processing applications.[6] Initially, this concept was defined as a 3Vs model, namely volume, velocity and variety [6]:

- **Volume**: This characteristic refers to the huge amounts of data that need to be processed to obtain helpful information.
- **Velocity**: This property states that the data processing applications must be able to obtain results in a reasonable time.
- **Variety**: This feature indicates that the data can be presented in multiple formats; structured and unstructured, such as text, numerical data or multimedia among others.

More recently, new dimensions have been proposed [6] by different organizations to describe the big data model being the veracity, validity, volatility, variability or value some of them.

Big data problems appear in a large number of fields and sectors such as economic and business activities, public administrations, national security or researches, among others. For example, the New York Stock Exchange can generate up to one Terabyte per day of new trade data. Facebook servers store one Petabyte of multimedia daily data (about ten billion photos). Another example is the Internet Archive, which can accumulate two Petabytes of data per day.[14]

This situation tends to be a problem as the researchers, governments or enterprises have had to face the challenge to process huge amounts of data quickly and efficiently, so that they can improve the productivity (in business) or obtain new scientific breakthroughs (in scientific disciplines).

2.2. *MapReduce programming model*

The MapReduce programming model was introduced by Google in 2004.[11,15] It is a distributed programming model for writing massive, scalable and fault tolerant data applications that was developed for processing large datasets over a cluster of machines. The MapReduce model is based on two primary functions: the *map* function and the *reduce* function, which must be designed by users. In general terms, in the first phase the input data is processed by the *map* function which produces some intermediate results; afterwards, these intermediate results will be fed to a second phase in a *reduce* function which somehow combines the intermediate results to produce a final output.

The MapReduce model is defined with respect to an essential data structure known as <key,value> pair. The processed data, the intermediate and final results work in terms of <key,value> pairs. In this way, the *map* and *reduce* functions that can be seen in a MapReduce procedure are defined as follows:

- **Map function**: In the *map* function the master node takes the input, divides it into several sub-problems and transfers them to the worker nodes. Next, each worker node processes its sub-problem and generates a result that is transmitted back to the 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. Then, these intermediate <key,value> pairs are automatically shuffled and ordered according to the intermediate key and will be the input to the *reduce* function.
- **Reduce function**: In the *reduce* function, the master node collects the answers of worker nodes and combines them in some way to form the final output of the method. Again, in terms of <key,value> pairs, the *reduce* function obtains the intermediate <key,value> pairs produced in the previous phase and generates the corresponding <key,value> pair as the final output of the algorithm.

Figure 1 illustrates a typical MapReduce program with its *map* and *reduce* steps. The terms k and v refer to the original key and value pair respectively; k' and v' are the intermediate <key,value> pair that is created after the *map* step; and v'' is the final result of the algorithm.

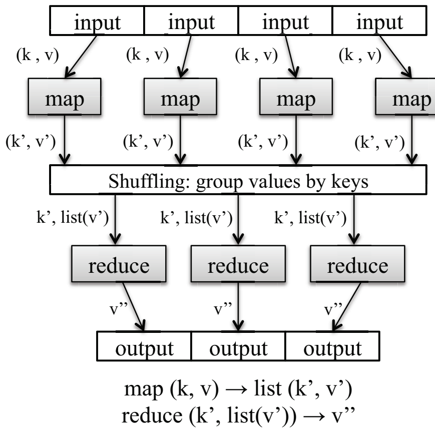

input
(k , v)
map
(k', v')
Shuffling: group values by keys
k', list(v')
reduce
v''
output
map (k, v) → list (k', v')
reduce (k', list(v')) → v''

Fig. 1. The MapReduce programming model

Nevertheless, the original MapReduce technology is a proprietary system exploited by Google, and therefore, it is not available for public use. Apache Hadoop is the most relevant open source implementation of the Google's MapReduce programming model and the Google File System (GFS).[14] It is a project written in Java and supported by the Apache Software Foundation for easily writing applications that process vast amounts of data in parallel on clusters of nodes. Hadoop provides a distributed file system similar to GFS, designated as Hadoop Distributed File System, which is highly fault tolerant, and is designed for work over large clusters of "commodity hardware."

In this paper we consider the Hadoop MapReduce implementation to develop our proposals because of its performance, open source nature, installation facilities and its associated distributed file system (Hadoop Distributed File System).

Machine learning algorithms have also been adapted using the MapReduce programming model to manage big data in a straightforward way. The Mahout project,[16] also supported by the Apache Software Foundation, aims to provide scalable machine learning applications for large-scale and intelligent data analysis techniques over Hadoop platforms or other scalable systems. It is possibly the most widely used tool that integrates scalable machine learning algorithms for clustering, recommendation systems, classification problems, pattern mining and regression, among others.

Furthermore, several MapReduce implementations have been proposed for different classification algorithms such as cost-sensitive fuzzy rule based systems for imbalanced classification,[17] ensembles of classifiers [18,19] or Support Vector Machines [20] to mention a few.

3. Fuzzy Rule Based Classification Systems: The Chi-FRBCS Approach

In this section, we first introduce in Section 3.1 some concepts related to FRBCSs and then, in Section 3.2, we describe the fuzzy learning algorithm that has been adapted in this work, Chi-FRBCS.

3.1. Introduction to Fuzzy Rule Based Classification Systems

Any classification problem is usually defined by m training instances $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn}, C_p)$, $p = 1, 2, \ldots, m$ with n attributes and M classes where x_{pi} is the value of attribute i ($i = 1, 2, \ldots, n$) and C_p is the value of class label $(C_1, \ldots, C_M)$ of the p-th training sample.

An FRBCS is composed by two elements: the Inference System and the Knowledge Base (KB). In a linguistic FRBCS, the KB is formed of the Data Base (DB), which contains the membership functions of the fuzzy partitions associated to the input attributes and the Rule Base (RB), which comprises the fuzzy rules that describe the problem. A learning procedure is needed to construct the KB from the available examples.

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

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

where R_j is the label of the j-th rule, $\mathbf{x} = (x_1, \ldots, x_n)$ is a 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. We use triangular membership functions to represent the linguistic labels.

There are many alternatives that have been proposed to compute the rule weight. Among them, a good choice is to use the heuristic method known as the Penalized Certainty Factor (PCF) [21]:

$$\mathrm{RW}_j = \mathrm{PCF}_j = \frac{\sum_{x_p \in C_j} \mu_{A_j}(x_p) - \sum_{x_p \notin C_j} \mu_{A_j}(x_p)}{\sum_{p=1}^{m} \mu_{A_j}(x_p)} \qquad (2)$$

where $\mu_{A_j}(x_p)$ is the membership degree of the $\mathbf{x}_p$ p-th example of the training set with the antecedents of the rule and C_j is the consequent class of rule j.

In order to provide the final class associated with a new pattern $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn})$, the winner rule R_w is determined through the following equation:

$$\mu_w(\mathbf{x}_p) \cdot RW_w = max\{\mu_j(\mathbf{x}_p) \cdot RW_j; j = 1 \ldots L\} \quad (3)$$

We use the fuzzy reasoning method of the wining rule [22] when predicting a class using the built KB for a given example. In this way, the class assigned to the example $\mathbf{x}_p$, C_w, is the class indicated in the consequent of the winner rule R_w.

In the event that multiple rules obtain the same maximum value for equation 3 but with different classes on the consequent, the classification of the pattern $\mathbf{x}_p$ will not be performed, that is, the pattern would not have any associated class. In the same way, if the pattern $\mathbf{x}_p$ does not match any rule in the RB, no class is associated to the example and the classification is also not carried out.

3.2. *The Chi et al.'s algorithm for Classification*

To build the KB of a linguistic FRBCS, we need to use a learning procedure that specifies how the DB and RB are created. In this work, we use the method proposed in [23], an extension of the well-known Wang and Mendel method for classification [24], which we have called the Chi *et al's* method, Chi-FRBCS.

To generate the KB, this generation method tries to find the relationship between the input attributes and the classes space following the next steps:

1. *Building the linguistic fuzzy partitions*: This step builds the DB from the domain associated to each attribute A_i using equally distributed triangular membership functions.
2. *Generating a new fuzzy rule associated to each example* $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn}, C_p)$:
   (a) Compute the matching degree $\mu(\mathbf{x}_p)$ of the example with respect to the fuzzy labels of each attribute using a conjunction operator.
   (b) Select the fuzzy region that obtains the maximum membership degree in relation with the example.
   (c) Build a new fuzzy rule whose antecedent is calculated according to the previous fuzzy region and whose consequent is the class label of the example C_p.
   (d) Compute the rule weight.

When following the previous procedure, several rules with the same antecedent can be built. If they have the same class in the consequent, then, duplicated rules are deleted. However, if the class in the consequent is different, only the rule with the highest weight is maintained in the RB.

4. The Chi-FRBCS-BigData algorithm: A MapReduce Design based on the Fusion of Fuzzy Rules

In this section, we will present the Chi-FRBCS-BigData algorithm that we have developed to deal with big data classifications problems. To do so, this method uses two different MapReduce processes:

- One MapReduce process is devoted to the building of the model from a big data training set, detailed in Section 4.1.
- The other MapReduce process is used to estimate the class of the examples belonging to big data sample sets using the previous learned model, this process is explained in Section 4.2.

Both parts follow the MapReduce design, distributing all the computations along several processing units that manage different chunks of information, aggregating the results obtained in an appropriate manner.

Furthermore, we have developed two versions of the Chi-FRBCS-BigData algorithm, which we have named Chi-FRBCS-BigData-Max and Chi-FRBCS-BigData-Ave. These versions share most of their operations, however, they behave differently in the *reduce* step of the approach, when the different RBs generated by each *map* are fused. These versions obtain different RBs and thus, different KBs.

4.1. *Building the model with Chi-FRBCS-BigData*

The procedure to build the KB following a MapReduce scheme in Chi-FRBCS-BigData is depicted in Figure 2. This procedure is divided into the following phases:

1. *Initial:* In this first phase, the method computes the domain associated to each attribute A_i using the whole training set. The DB is created using equally distributed triangular membership functions as in Chi-FRBCS. Then, the system automatically segments the original training dataset into independent data blocks which are automatically transferred to the different processing units together with the created DB.

2. *Map:* In this second phase, each processing unit works independently over its available data to build its associated RB (called RB_i in Figure 2) following the original Chi-FRBCS method.

   Specifically, for each example in the data partition, an associated fuzzy rule is created: first, the membership degree of the fuzzy labels is computed according to the example values; then, the fuzzy region that obtains the greatest value is selected to become the antecedent of the rule; next, the class of the example is assigned to the rule as the consequent; and finally, the rule weight is computed *using the set of examples that belong to the current map process*.

   After the rules have been created and before finishing the *map* step, each *map* process searches for rules with the same antecedent. If the rules share the same consequent, only one rule is preserved; if the rules have different consequents, only the rule with the highest weight is kept in the *map* RB.

3. *Reduce:* In this third phase, a processing unit receives the results obtained by each *map* process (RB_i) and combines them to form the final RB (called RB_R in Figure 2). The combination of the rules is straight-forward: the rules created by each *map* $RB_1, RB_2, \ldots, RB_n$ are all integrated in one RB, RB_R. Specific procedures to fuse these rule bases are defined. These procedures determine the two variants of the Chi-FRBCS-BigData algorithm:

   (a) **Chi-FRBCS-BigData-Max:** In this approach, the method searches for the rules with the same antecedent. Among these rules, only the rule with the highest weight is maintained in the final RB, RB_R. In this case it is not necessary to check if the consequent is the same or not, as we are only maintaining the most powerful rules. Equivalent rules (rules with the same antecedent and consequent) can present different weights as they are computed in different *map* processes over different training sets.

   For instance, if we have five rules with the same antecedent and the following consequents and rule weights (see Figure 3);

   R_1 of RB_1: Class 1, $RW_1 = 0.8743$;
   R_1 of RB_2: Class 2, $RW_2 = 0.9254$;
   R_2 of RB_3: Class 1, $RW_3 = 0.7142$;
   R_1 of RB_4: Class 2, $RW_4 = 0.2143$ and
   R_2 of RB_n: Class 1, $RW_5 = 0.8215$.

   then, Chi-FRBCS-BigData-Max will keep in RB_R the rule R_1 of RB_2: Class 2, $RW_2 = 0.9254$ because it is the rule with the maximum weight.

   (b) **Chi-FRBCS-BigData-Ave:** In this approach, the method also searches for the

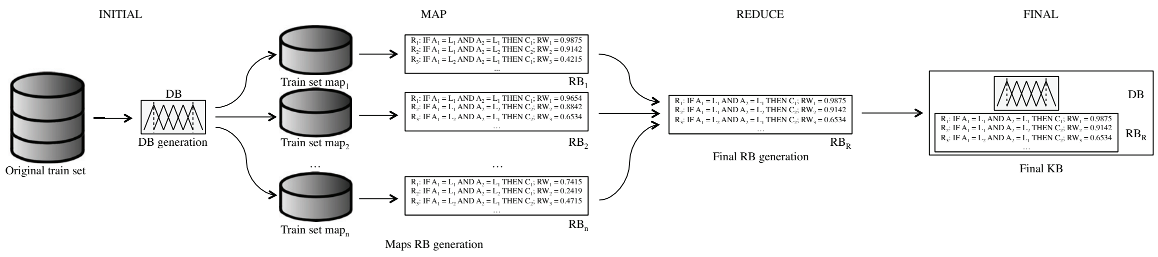

INITIAL
MAP
REDUCE
FINAL
DB
DB generation
Original train set
Train set map_1
Train set map_2
...
Train set map_n
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.9875$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
R_3: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_1; $RW_3 = 0.4215$
RB_1
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.9654$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
R_3: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
RB_2
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.7415$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_1; $RW_2 = 0.2419$
R_3: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.4715$
RB_n
Maps RB generation
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.9875$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
R_3: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
RB_R
Final RB generation
DB
RB_R
Final KB

Figure 2: A flowchart of how the building of the KB is organized in Chi-FRBCS-BigData

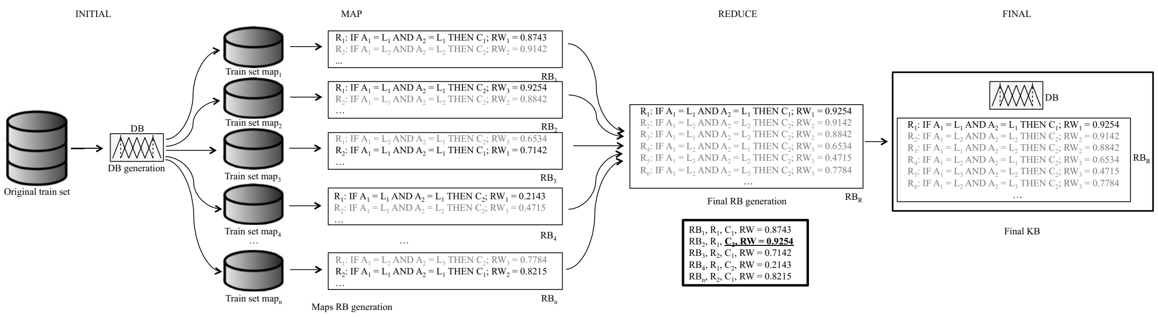

INITIAL
MAP
REDUCE
FINAL
DB
DB generation
Original train set
Train set map_1
Train set map_2
Train set map_3
Train set map_4
...
Train set map_n
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.8743$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
RB_1
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.9254$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
RB_2
R_1: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.7142$
RB_3
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_1 = 0.2143$
R_2: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
RB_4
R_1: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_2 = 0.8215$
RB_n
Maps RB generation
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.9254$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
R_3: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
R_4: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_5: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
R_6: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
Final RB generation
RB_R
RB_1, R_1, C_1, RW = 0.8743
RB_2, R_1, C_2, RW = 0.9254
RB_3, R_2, C_1, RW = 0.7142
RB_4, R_1, C_2, RW = 0.2143
RB_n, R_2, C_1, RW = 0.8215
DB
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.9254$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
R_3: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
R_4: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_5: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
R_6: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
RB_R
Final KB

Figure 3: Example of building the KB with Chi-FRBCS-BigData-Max

rules with the same antecedent. Then, the average weight of the rules that have the same consequent is computed (this step is needed because rules with the same antecedent and consequent may have different weights as they are built over different training sets). Finally, the rule with the greatest average weight is kept in the final RB, RB_R.

For instance, if we have five rules with the same antecedent and the following consequents and rule weights (see Figure 4);

R_1 of RB_1: Class 1, $RW_1 = 0.8743$;
R_1 of RB_2: Class 2, $RW_2 = 0.9254$;
R_2 of RB_3: Class 1, $RW_3 = 0.7142$;
R_1 of RB_4: Class 2, $RW_4 = 0.2143$ and
R_2 of RB_n: Class 1, $RW_5 = 0.8215$.

then, Chi-FRBCS-BigData-Ave will first compute the average weight for the rules with the same consequent, namely, R_{C1}: Class 1, $RW_{C1} = 0.8033$ and R_{C2}: Class 2, $RW_{C2} = 0.5699$, and it will keep in RB_R the rule R_{C1}: Class 1, $RW_{C1} = 0.8033$ because it is the rule with the maximum average weight.

Please note that it is not needed for any Chi-FRBCS-BigData version to recompute the rule weights according to the data in the *reduce* stage, as we are calculating the new rule weights from the previously rule weights provided by each *map*.

4. *Final:* In this last phase, the results computed in the previous phases are provided as the output of the computation process. Precisely, the generated KB is composed by the DB built in the "Initial" phase and the RB, RB_R, is finally obtained in the "reduce" phase. This KB will be the model that will be used to predict the class for new examples.

4.2. *Classifying big data sample sets*

As it was previously said, Chi-FRBCS-BigData uses another MapReduce process to estimate the class of

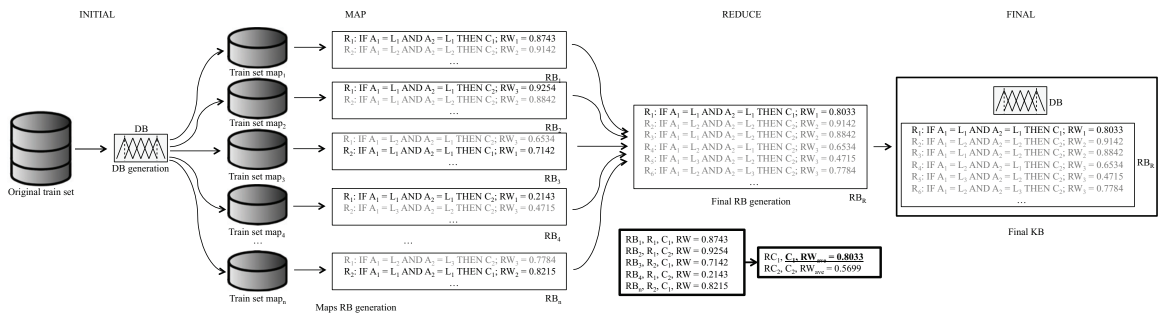

INITIAL
MAP
REDUCE
FINAL
Original train set
DB
DB generation
Train set map$_1$
Train set map$_2$
Train set map$_3$
Train set map$_4$
Train set map$_n$
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.8743$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
RB$_1$
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.9254$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
RB$_2$
R_1: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.7142$
RB$_3$
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_1 = 0.2143$
R_2: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
RB$_4$
R_1: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_2 = 0.8215$
RB$_n$
Maps RB generation
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.8033$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
R_3: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
R_4: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_5: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
R_6: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
Final RB generation
RB$_R$
$RB_1, R_1, C_1, RW = 0.8743$
$RB_2, R_1, C_2, RW = 0.9254$
$RB_3, R_2, C_1, RW = 0.7142$
$RB_4, R_1, C_2, RW = 0.2143$
$RB_n, R_2, C_1, RW = 0.8215$
RC_1, $C_1, RW_{ave} = 0.8033$
$RC_2, C_2, RW_{ave} = 0.5699$
Final KB

Figure 4: Example of building the KB with Chi-FRBCS-BigData-Ave

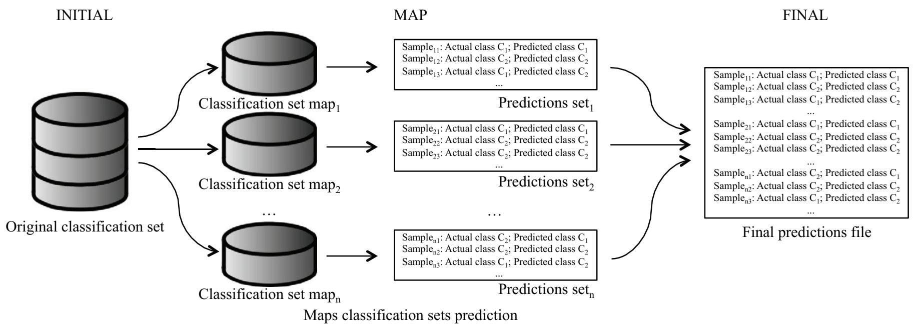

INITIAL
MAP
FINAL
Original classification set
Classification set map$_1$
Classification set map$_2$
Classification set map$_n$
Sample$_{11}$: Actual class C_1; Predicted class C_1
Sample$_{12}$: Actual class C_2; Predicted class C_2
Sample$_{13}$: Actual class C_1; Predicted class C_2
Predictions set$_1$
Sample$_{21}$: Actual class C_1; Predicted class C_1
Sample$_{22}$: Actual class C_2; Predicted class C_2
Sample$_{23}$: Actual class C_2; Predicted class C_2
Predictions set$_2$
Sample$_{n1}$: Actual class C_2; Predicted class C_1
Sample$_{n2}$: Actual class C_2; Predicted class C_2
Sample$_{n3}$: Actual class C_1; Predicted class C_2
Predictions set$_n$
Maps classification sets prediction
Final predictions file

Figure 5: A flowchart of how the classification of a big dataset is organized in Chi-FRBCS-BigData

the examples that belong to big data classification sets using the KB built in the previous step. This approach follows a similar scheme to the previous step where the initial dataset is distributed along several processing units that provide a result that will be part of the final result. Specifically, this class estimation process is depicted in Figure 5 and follows the phases:

1. *Initial:* In this first phase, the method does not need to perform a specific operation. The system automatically segments the original big data dataset that needs to be classified into independent data blocks which are automatically transferred to the different processing units together with the previously created KB.

2. *Map:* In this second phase, each *map* task estimates the class for the examples that are included in its data partition. To do so, each processing unit goes through all the examples in its data chunk and predicts its output class according to the given KB and using the fuzzy reasoning method of the wining rule. Please note that Chi-FRBCS-BigData-Max and Chi-FRBCS-BigData-Ave will produce different classification estimations because the input RBs are also different, however, the class estimation process followed is exactly the same for both approaches.

3. *Final:* In this last phase, the results computed in the previous phase are provided as the output of the computation process. Precisely, the estimated classes for the different examples of the big data classification set are aggregated just concatenating the results provided by each *map* task.

It is important to note that this mechanism does not include a *reduce* step as it is not necessary to perform a computation to combine the results obtained in the *map* phase.

5. Experimental study

In this section, we present the experimental study carried out using the Chi-FRBCS-BigData algorithm over big data problems. First, in Section 5.1, we provide some details of the classification problems chosen for the experiments and the configuration parameters for the methods analyzed. Next, in Section 5.2 we study the behavior of the original Chi-FRBCS serial version with respect to Chi-FRBCS-BigData. Then, in Section 5.3, we provide the accuracy performance results for the approaches tested in the study with respect to the number of maps considered. Finally, an analysis that evaluates the runtime spent by the algorithms over the selected data is shown in Section 5.4.

5.1. Experimental framework

This study aims to analyze the quality of the Chi-FRBCS-BigData algorithm in the big data scenario. For this, we will consider six problems from the UCI dataset repository [25]: the Record Linkage Comparison Patterns (RLCP) dataset, the KDD Cup 1999 dataset, the Poker Hand dataset, the Covertype dataset, the Census-Income (KDD) dataset and the Fatality Analysis Reporting System (FARS) dataset. A summary of the datasets features is shown in Table 1, where the number of examples (#Ex.), number of attributes (#Atts.), selected classes and the number of examples per class are included. This table is in descending order according to the number of examples of each dataset.

Table 1. Summary of datasets

Datasets	**#Ex.**	**#Atts.**	**Selected classes**	**#Samples per class**
RLCP	5749132	2	(FALSE; TRUE)	(5728201; 20931)
Kddcup_DOS_vs_normal	4856151	41	(DOS; normal)	(3883370; 972781)
Poker_0_vs_1	946799	10	(0; 1)	(513702; 433097)
Covtype_2_vs_1	495141	54	(2; 1)	(283301; 211840)
Census	141544	41	(-_50000.; 50000+.)	(133430; 8114)
Fars_Fatal_Inj_vs_No_Inj	62123	29	(Fatal_Inj; No_Inj)	(42116; 20007)

Since several of the selected datasets contain multiple classes, in this work we have decided to reduce multi-class problems to two classes. Despite the ability of the Chi-FRBCS-BigData algorithm to address with multi-class problems, we want to avoid the imbalance in the data that arises in many real-world problems.[26] Moreover, the division approach of the presented MapReduce scheme aggravates the small sample size problem, which degrades the performance of classifiers in the imbalanced scenario.

To develop the different experiments we use a 10-fold stratified cross-validation partitioning scheme, i.e., ten random partitions of data with a 10% of the samples with the combination of nine of them (90%) as training set and the remaining one as test set. The results obtained for each dataset are the average results obtained by computing the mean of all the partitions.

To demonstrate the inability of the original Chi-FRBCS serial version to deal with big data classification problems, we have compared the results obtained by the serial version with respect to Chi-FRBCS-BigData for the selected datasets.

In order to verify the performance of the proposed model, we have compared the results obtained by Chi-FRBCS-BigData-Max with Chi-FRBCS-BigData-Ave so that we can compare their behavior with respect to the selected big data problems.

Regarding the parameters used in the experiments, these algorithms use:

- Three fuzzy labels for each attribute.
- The product T-norm to compute the matching degree of the antecedent of the rule with the example.
- The PCF to compute the rule weight.
- The winning rule is used as fuzzy reasoning method.

Additionally, another parameter is used in the MapReduce procedure, which is the number of maps associated to the computation. This value has been set to 8, 16, 32, 64 and 128 maps and represents the number of subsets of the original dataset that are created and are provided to the map processes.

The measures of the quality of classification are built from a confusion matrix (Table 2), which organizes the examples of each class in accordance with their correct or incorrect identification.

Table 2. 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)

The effectiveness in classification for the proposed approach will be evaluated using the most frequently used empirical measure, accuracy, which is defined as follows:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4}$$

With respect to the infrastructure used to perform the experiments, we have used the research group's cluster with 16 nodes connected with a 40Gb/s Infiniband. Each node is equipped with two Intel E5-2620 microprocessors (at 2 GHz, 15MB cache) and 64GB of main memory running under Linux CentOS 6.5. The head node of the cluster is equipped with two Intel E5645 microprocessors (at 2.4 GHz, 12MB cache) and 96GB of main memory. Furthermore, the cluster works with Hadoop 2.0.0 (Cloudera CDH4.5.0), where the head node is configured as name-node and job-tracker, and the rest are datanodes and task-trackers.

5.2. *Analysis of the Chi-FRBCS serial version with respect to Chi-FRBCS-BigData*

In this section, we present a set of experiments to illustrate the behavior of the original Chi-FRBCS serial version with respect to Chi-FRBCS-BigData. The experiments have been designed to contrast the results of the serial version in relation to the big data versions of the algorithm for the selected datasets. Table 3 presents the average results in training and test for the Chi-FRBCS versions and is divided by columns into 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 algorithm using 8 maps. The bold values highlight the best performing algorithm.

Firstly, we can observe that the table does not present the results for the "RLCP" and "Kddcup_DOS_vs_normal" datasets. This means that the sequential version of Chi-FRBCS was not able to complete the associated experiments.

Table 3. Average Accuracy results for the Chi-FRBCS versions

Datasets			**8 maps**			
	Chi-FRBCS		Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
Poker_0_vs_1	63.72	**61.77**	62.93	60.74	63.12	60.91
Covtype_2_vs_1	74.65	74.57	74.69	**74.63**	74.66	74.61
Census	96.52	86.06	97.12	**93.89**	97.12	93.86
Fars_Fatal_Inj_vs_No_Inj	99.66	89.26	97.01	95.07	97.18	**95.25**
Average	83.64	77.92	82.94	81.08	83.02	**81.16**

In terms of the accuracy achieved by the algorithms considered in the study we can see that, in general, the Chi-FRBCS-BigData versions are able to provide better classification results than the serial version. The unique exception to this tendency can be observed in the "Poker_0_vs_1" dataset, for which the serial variant obtains better results.

In addition, we can also observe that the results in training indicate that there is some overfitting in the serial version as there are any differences between the training and test results, mainly on the results for the "Census" and "Fars_Fatal_Inj_vs_No_Inj" datasets.

Moreover, Table 4 shows the time elapsed in seconds for the serial version and the big data alternatives. This table is also divided by columns into 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 Chi-FRBCS for 8 maps. The bold values highlight the quickest algorithm.

Table 4. Average runtime elapsed in seconds for the Chi-FRBCS versions

Datasets		**8 maps**	
	Chi-FRBCS	Chi-BigData-Max	Chi-BigData-Ave
	Runtime (s)	Runtime (s)	Runtime (s)
Census	38655.60	**1102.45**	1343.92
Covtype_2_vs_1	86247.70	**2482.09**	2512.16
Fars_Fatal_Inj_vs_No_Inj	8056.60	**241.96**	311.95
Poker_0_vs_1	114355.80	**5672.80**	7682.19
Average	61828.93	**2374.82**	2962.56

Considering these results, we can see that the sequential version is notably slower than the MapReduce alternatives. Furthermore, the results obtained show that the runtime spent is directly related to the operations that need to be performed by the big data approaches. In this way, we can observe that the Chi-BigData-Ave method is slower than the Chi-BigData-Max algorithm, since it performs fewer operations.

Furthermore, in Table 5 we compare the runtime spent by the big data versions with respect to the runtime for the sequential version divided by the number of parallel processes considered or maps (8 in this analysis). This table follows the same structure as Table 4. Even in this case we can see that the big data alternatives are significantly faster than the sequential version.

Table 5. Average runtime elapsed in seconds for the Chi-FRBCS versions supposing that the sequential version was executed in parallel

Datasets		**8 maps**	
	Chi-FRBCS	Chi-BigData-Max	Chi-BigData-Ave
	Runtime (s) / 8 maps	Runtime (s)	Runtime (s)
Census	4831.95	**1102.45**	1343.92
Covtype_2_vs_1	10780.96	**2482.09**	2512.16
Fars_Fatal_Inj_vs_No_Inj	1007.08	**241.96**	311.95
Poker_0_vs_1	14294.48	**5672.80**	7682.19
Average	7728.62	**2374.82**	2962.56

Finally, Table 6 shows the average number of rules generated for the sequential version and the big data alternatives. This table is also divided by columns into two parts: the first part corresponds to the average number of rules generated for the sequential variant while the second part is related to the average number of rules generated by the big data variants of Chi-FRBCS for 8 maps. The values in boldface highlight the lowest numbers of rules obtained for a specific dataset.

Table 6. Average number of rules generated for the Chi-FRBCS versions

Datasets		**8 maps**	
	Chi-FRBCS	Chi-BigData-Max	Chi-BigData-Ave
	Average NumRules	Average NumRules	Average NumRules
Census	**31518.3**	34278.0	34278.0
Covtype_2_vs_1	**6962.7**	7079.1	7079.1
Fars_Fatal_Inj_vs_No_Inj	**16843.3**	17114.9	17114.9
Poker_0_vs_1	**51265.4**	52798.1	52798.1

We can see that the numbers of rules generated by the serial version are slightly lower than the ones obtained by the big data variants. However, the Chi-FRBCS-BigData versions are able to obtain better classification results as we can observed in Table 3.

5.3. *Analysis of the Chi-FRBCS-BigData accuracy*

In this section, we will compare the two versions of the proposed approach, Chi-FRBCS-BigData-Max and Chi-FRBCS-BigData-Ave, to see if there are differences between them. For the sake of space, these algorithms are named Chi-BigData-Max and Chi-BigData-Ave respectively in the result tables.

In this way, in Table 7, we present the average results in training and test for the Chi-FRBCS-BigData algorithms using 8, 16, 32, 64 and 128 maps over the selected datasets and considering the accuracy performance measure. This table is divided into five horizontal parts that correspond to the performance results obtained with the different number of maps. The bold values highlight the most effective method in test related to the number of maps considered and the underlined values indicate which is the best performing algorithm in test for all the experiments.

Table 7. Average Accuracy results for the Chi-FRBCS-BigData versions using 8, 16, 32, 64 and 128 maps

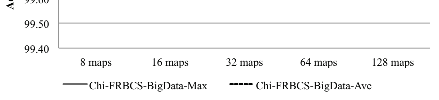
Datasets	**8 maps**			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	**99.63**	99.63	**99.63**
Kddcup_DOS_vs_normal	99.93	**99.93**	99.93	**99.93**
Poker_0_vs_1	62.93	60.74	63.12	**60.91**
Covtype_2_vs_1	74.69	**74.63**	74.66	74.61
Census	97.12	**93.89**	97.12	93.86
Fars_Fatal_Inj_vs_No_Inj	97.01	95.07	97.18	**95.25**
Average	88.55	87.31	88.61	**87.37**
	16 maps			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	**99.63**	99.63	**99.63**
Kddcup_DOS_vs_normal	99.93	**99.93**	99.93	**99.93**
Poker_0_vs_1	62.18	59.88	62.58	**60.35**
Covtype_2_vs_1	74.77	**74.72**	74.77	74.69
Census	97.14	**93.75**	97.15	93.52
Fars_Fatal_Inj_vs_No_Inj	96.69	94.75	97.06	**95.01**
Average	88.39	87.11	88.52	**87.19**
	32 maps			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	**99.63**	99.63	**99.63**
Kddcup_DOS_vs_normal	99.92	**99.92**	99.92	**99.92**
Poker_0_vs_1	61.27	58.93	61.82	**59.30**
Covtype_2_vs_1	74.69	74.62	74.88	**74.85**
Census	97.11	**93.48**	97.12	93.32
Fars_Fatal_Inj_vs_No_Inj	96.49	94.26	96.87	**94.63**
Average	88.19	86.81	88.37	**86.94**
	64 maps			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	**99.63**	99.63	**99.63**
Kddcup_DOS_vs_normal	99.92	99.92	99.93	**99.93**
Poker_0_vs_1	60.45	57.95	60.88	**58.12**
Covtype_2_vs_1	74.67	74.52	75.05	**74.96**
Census	97.07	**93.30**	97.13	93.11
Fars_Fatal_Inj_vs_No_Inj	96.27	93.98	96.76	**94.56**
Average	88.00	86.55	88.23	**86.72**
	128 maps			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	**99.63**	99.63	**99.63**
Kddcup_DOS_vs_normal	99.93	**99.93**	99.93	**99.93**
Poker_0_vs_1	59.59	56.96	60.09	**57.12**
Covtype_2_vs_1	74.12	74.01	75.04	**74.99**
Census	96.95	**92.97**	97.05	92.91
Fars_Fatal_Inj_vs_No_Inj	96.07	93.82	96.67	**94.20**
Average	87.71	86.22	88.07	**86.46**

In a first glance, we can see that the best performing algorithm in average is the Chi-FRBCS-

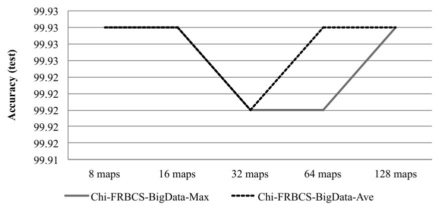
BigData-Ave method for any number of maps considered. In this way, obtaining the average rule weight of all the partial RBs obtained shows a positive impact in the classification results since we aim to make the rules as general as possible. However, we can find an exception in the "Census" dataset, which does not follow the same tendency as the other datasets. In this case, the Chi-FRBCS-BigData-Max variant gets slightly better results. This behavior may be related to the results in training and a possible overfitting, since it seems that this particular dataset presents a huge gap between training and test results.

On the other hand, we can observe a reduction in classification accuracy when using a larger number of maps in both Chi-FRBCS-BigData versions and for both training and test results. This is an expected behavior of the MapReduce design used, since the rule weights are calculated from smaller data partitions when the number of maps is incremented. However, this trend is not observed in the case of the "Covtype_2_vs_1" dataset, where the Chi-FRBCS-BigData-Ave alternative provides better accuracy results when the number of maps is increased.

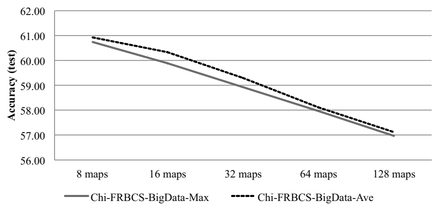
In Figure 6 we represent the average results for the Chi-FRBCS-BigData versions and for all the datasets considered: the "RLCP" dataset (Figure 6a), the "Kddcup_DOS_vs_normal" dataset (Figure 6b), the "Poker_0_vs_1" dataset (Figure 6c), the "Covtype_2_vs_1" dataset (Figure 6d), the "Census" dataset (Figure 6e) and "Fars_Fatal_Inj_vs_No_Inj" dataset (Figure 6f). This figure shows the evolution of the accuracy measure when the number of maps is varied.



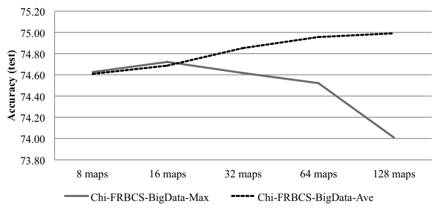
(a) RLCP dataset



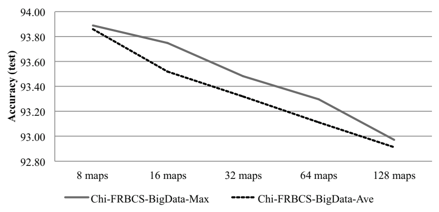
(b) Kddcup_DOS_vs_normal dataset



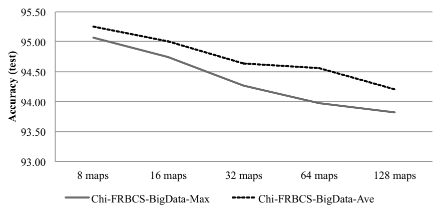
(c) Poker_0_vs_1 dataset



(d) Covtype_2_vs_1 dataset



(e) Census dataset



(f) Fars_Fatal_Inj_vs_No_Inj dataset

Fig. 6. Average results for Chi-FRBCS-BigData versions using the accuracy measure

In order to illustrate how the Chi-FRBCS-BigData proposal is able to reduce the complexity of the model by decreasing the number of final rules, we present in Table 8 the number of rules created by each map process and the number of final rules, when the distinct RBs generated by each map are fused. To do this, we have selected the "Kddcup_DOS_vs_normal" dataset with 41 attributes and 4856151 instances. More concretely, we have chosen the 1th partition of the 10-fcv that uses 4369790 instances for training and 486361 instances for test. The number of maps considered in this analysis is 8. In this table we can observe that the number of rules has dramatically decreased from the 1708 rules that were created by all the maps to the 301 rules that finally compose the rule base (RB_R).

Table 8. Example of number of rules generated by map and number of final rules for the Chi-FRBCS-BigData-Max version with 8 maps

Kddcup_DOS_vs_normal dataset	
NumRules by map	Final numRules
RB_1 size: 211	RB_R size: 301
RB_2 size: 212	
RB_3 size: 221	
RB_4 size: 216	
RB_5 size: 213	
RB_6 size: 210	
RB_7 size: 211	
RB_8 size: 214	

Finally, in Table 9, we present the average number of rules generated for the Chi-FRBCS-BigData algorithms using 8, 16, 32, 64 and 128 maps over the selected datasets. This table is also divided into five horizontal parts that correspond to the average number of rules obtained with the different number of maps. The values in boldface correspond to the lowest numbers of rules obtained for a specific dataset. First, we can see that in general we obtain a smaller number of rules for a lower number of maps. On the other hand, we can see that the number of generated rules is higher when the number of maps is increased.

Table 9. Average number of rules generated for the Chi-FRBCS-BigData versions using 8, 16, 32, 64 and 128 maps

Datasets	**8 maps** – Average NumRules	
	Chi-BigData-Max	Chi-BigData-Ave
RLCP	**6.0**	**6.0**
kddcup_DOS_vs_normal	**298.9**	**298.9**
Poker_0_vs_1	**52798.1**	**52798.1**
Covtype_2_vs_1	**7079.1**	**7079.1**
Census	**34278.0**	**34278.0**
Fars_Fatal_Inj_vs_No_Inj	**17114.9**	**17114.9**
	16 maps – Average NumRules	
	Chi-BigData-Max	Chi-BigData-Ave
RLCP	**6.0**	**6.0**
kddcup_DOS_vs_normal	299.9	299.9
Poker_0_vs_1	53168.9	53168.9
Covtype_2_vs_1	7134.3	7134.3
Census	34341.4	34341.4
Fars_Fatal_Inj_vs_No_Inj	17158.1	17158.1
	32 maps – Average NumRules	
	Chi-BigData-Max	Chi-BigData-Ave
RLCP	**6.0**	**6.0**
kddcup_DOS_vs_normal	300.5	300.5
Poker_0_vs_1	53403.7	53403.7
Covtype_2_vs_1	7210.2	7210.2
Census	34376.5	34376.5
Fars_Fatal_Inj_vs_No_Inj	17182.0	17182.0
	64 maps – Average NumRules	
	Chi-BigData-Max	Chi-BigData-Ave
RLCP	**6.0**	**6.0**
kddcup_DOS_vs_normal	300.5	300.5
Poker_0_vs_1	53503.4	53503.4
Covtype_2_vs_1	7278.9	7278.9
Census	34392.5	34392.5
Fars_Fatal_Inj_vs_No_Inj	17196.4	17196.4
	128 maps – Average NumRules	
	Chi-BigData-Max	Chi-BigData-Ave
RLCP	**6.0**	**6.0**
kddcup_DOS_vs_normal	300.5	300.5
Poker_0_vs_1	53541.0	53541.0
Covtype_2_vs_1	7343.3	7343.3
Census	34397.3	34397.3
Fars_Fatal_Inj_vs_No_Inj	17202.1	17202.1

5.4. *Analysis of the Chi-FRBCS-BigData runtime*

In this section we compare the runtime of the two versions of the Chi-FRBCS-BigData proposal for the different problems selected and the diverse number of maps used in the experiments.

We present the average results for the runtime in a similar way to the analysis of the accuracy given in the previous Section. Table 10 shows the average time elapsed in seconds by the Chi-FRBCS-BigData-Max and Chi-FRBCS-BigData-Ave algorithms for the selected datasets with 8, 16, 32, 64 and 128 maps, respectively. This table is divided into five horizontal parts, which show the results for each dataset with respect to the different number of maps. The bold values correspond to the fastest algorithm for the same number of maps and the underlined values highlight the quickest algorithm for a specific dataset.

The results obtained show that, in average, the quickest method is the Chi-FRBCS-BigData-Max algorithm for all the values of the number of maps considered. This behavior is directly related to the operations performed by each variant, since the Chi-FRBCS-BigData-Max algorithm executes fewer operations than the Chi-FRBCS-BigData-Ave alternative.

Moreover, there are some cases where the Chi-FRBCS-BigData-Ave version is slightly faster, although this improvement is not able to compensate the slowness of the algorithm in other cases. For example, this alternative obtains better runtimes when 16 maps are used, however, such improvement does not compensate the case of the "Poker_0_vs_1" dataset, where the algorithm is much slower. We can also observe this behavior when 32 and 128 maps are considered.

Furthermore, we can see a reduction on the runtime for both versions of Chi-FRBCS-BigData when the number of maps is increased. However, this decrease in the runtime does not follow a linear relationship. For example, it can be seen that when we double the number of processing units, the speed gain obtained is much higher than reducing the processing time by half.

We can also see that this decrement in the runtime is not uniform over the different datasets, since the smaller datasets are not able to improve their runtime performance in the same proportion as the largest datasets. In addition, the Chi-FRBCS-BigData-Max method is able to scale up better than the Chi-FRBCS-BigData-Ave alternative.

Table 10. Average runtime elapsed in seconds for the Chi-FRBCS-BigData versions using 8, 16, 32, 64 and 128 maps

Datasets	Chi-BigData-Max	Chi-BigData-Ave
	8 maps – Runtime (s)	
RLCP	**31942.38**	32027.37
Kddcup_DOS_vs_normal	**115839.09**	116218.26
Poker_0_vs_1	**5672.80**	7682.19
Covtype_2_vs_1	**2482.09**	2512.16
Census	**1102.45**	1343.92
Fars_Fatal_Inj_vs_No_Inj	**241.96**	311.95
Average	**26213.46**	26682.64
	16 maps – Runtime (s)	
RLCP	9023.82	**8868.84**
Kddcup_DOS_vs_normal	30120.03	**29820.01**
Poker_0_vs_1	**3075.50**	6582.32
Covtype_2_vs_1	1477.67	**924.65**
Census	939.32	**884.30**
Fars_Fatal_Inj_vs_No_Inj	363.05	**236.40**
Average	**7499.90**	7886.09
	32 maps – Runtime (s)	
RLCP	2460.89	**2303.02**
Kddcup_DOS_vs_normal	7890.87	**7708.96**
Poker_0_vs_1	**2210.13**	6331.09
Covtype_2_vs_1	**391.40**	493.00
Census	**388.64**	771.04
Fars_Fatal_Inj_vs_No_Inj	**141.92**	228.96
Average	**2247.31**	2972.68
	64 maps – Runtime (s)	
RLCP	**701.31**	714.41
Kddcup_DOS_vs_normal	**2079.93**	2096.34
Poker_0_vs_1	**1635.98**	8373.40
Covtype_2_vs_1	**252.19**	348.86
Census	**325.24**	764.94
Fars_Fatal_Inj_vs_No_Inj	**136.24**	241.75
Average	**855.15**	2089.95
	128 maps – Runtime (s)	
RLCP	288,52	**284,41**
Kddcup_DOS_vs_normal	1669,02	**1579,77**
Poker_0_vs_1	**1022,08**	6492,28
Covtype_2_vs_1	**189,24**	259,20
Census	**208,05**	431,71
Fars_Fatal_Inj_vs_No_Inj	**92,74**	165,51
Average	**578,28**	1535,48

In summary, in this study we have tested two different approaches developed in this work over a set of datasets that have helped us to have an insight into classifications big data problems:

- The Chi-FRBCS-BigData-Ave version obtains more accurate classification results than the Chi-FRBCS-BigData-Max approach, however, it provides slower models.
- The Chi-FRBCS-BigData-Max alternative does not have a strong degradation in the accuracy

performance with respect to the Chi-FRBCS-BigData-Ave version and provides better response times.

- Both the Chi-FRBCS-BigData-Ave version and the Chi-FRBCS-BigData-Max alternative, the accuracy of the model is decreased by the increasing number of maps, although, the speed gain is significant in this cases.

In this way, it is necessary to establish a trade-off for each occasion to select the most appropriate Chi-FRBCS-bigdata version.

6. Concluding remarks

In this work we have presented a linguistic fuzzy rule-based classification algorithm for big data problems called Chi-FRBCS-BigData. This algorithm obtains an interpretable model that is able to handle big collections of data providing a good accuracy and with fast response times. To do so, our method uses the MapReduce programming model on the Hadoop platform, one of the most popular solutions to effectively deal with big data nowadays. In this way, our model distributes the computation using the *map* function and then, combines the outputs through the *reduce* function.
The Chi-FRBCS-BigData algorithm has been developed in two different versions: Chi-FRBCS-BigData-Max and Chi-FRBCS-BigData-Ave.
The performance of the Chi-FRBCS-BigData alternatives is supported by an experimental study that is carried out over six classification big data problems. The results obtained show that the proposal is able to handle these problems providing competitive results. However, it is not possible to identify a best approach and is necessary to select the model that best meets our needs according to the speed-accuracy trade-off:

- The Chi-FRBCS-BigData-Ave method with low values for the number of maps seems to be the most appropriate choice when our goal is to achieve the best precision results without caring too much about the lower response times.
- The Chi-FRBCS-BigData-Max alternative with a large number of maps seems to be the best option if we are interested in getting faster results without deeply degrading the performance.

As future work we will study the combination of a FRBCS learning method with bagging [27] in order to deal with big data classification problems. In this way we can analyze the behavior of the use of a bagging approach together with data mapping for using a FRBCS as a base classifier in a MapReduce scheme.

Acknowledgments

This work was partially supported by the Spanish Ministry of Science and Technology under project TIN2011-28488 and the Andalusian Research Plans P12-TIC-2958, P11-TIC-7765 and P10-TIC-6858.

References

1. P. Zikopoulos, C. Eaton, D. DeRoos, T. Deutsch and G. Lapis, “Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data,” *McGraw-Hill*, (2011).
2. S. Madden, “From Databases to Big Data,” *IEEE Internet Computing*, **vol. 16, no. 3**, 4–6, (2012).
3. A. Sathi, “Big Data Analytics: Disruptive Technologies for Changing the Game,” *MC Press*, (2012).
4. C.L. Philip Chen and C.Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” *Information Sciences*, **vol. 275**, 314–347, (2014)
5. X. Wu and X. Zhu and G.Q. Wu and W. Ding, “Data Mining with Big Data,” *IEEE Transactions on Knowledge and Data Engineering*, **vol. 26, no. 1**, 97–107, (2014)
6. A. Fernández, S. Río, V. López, A. Bawakid, M.J. del Jesus, J.M. Benítez and F. Herrera, “Big Data with Cloud Computing: An Insight on the Computing Environment, MapReduce and Programming Framework,” *WIREs Data Mining and Knowledge Discovery*, **vol. 4, no. 5**, 380–409, (2014).
7. H. Ishibuchi, T. Nakashima and M. Nii, “Classification and modeling with linguistic information granules: Advanced approaches to linguistic Data Mining,” *SpringerVerlag*, (2004).
8. Y. Jin, “Fuzzy modeling of high-dimensional systems: complexity reduction and interpretability improvement,” *IEEE Transactions on Fuzzy Systems*, **vol. 8, no. 2**, 212–221, (2000).
9. T.P. Hong, Y.C. Lee and M.T. Wu, “An effective parallel approach for genetic-fuzzy data mining,” *Expert*

Systems with Applications, **vol. 41, no. 2**, 655–662, (2014).

10. H. Ishibuchi, S. Mihara and Y. Nojima, "Parallel distributed hybrid fuzzy GBML models with rule set migration and training data rotation," *IEEE Transactions on Fuzzy Systems*, **vol. 21, no. 2**, 355–368, (2013).
11. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, **vol. 51, no. 1**, 107–113, (2008).
12. Z. Chi, H. Yan and T. Pham, "Fuzzy algorithms with applications to image processing and pattern recognition," *World Scientific*, (1996).
13. V. López, S. Río, J.M. Benítez and F. Herrera, "On the use of MapReduce to build Linguistic Fuzzy Rule Based Classification Systems for Big Data," *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2014), Beijing (China)*, 1905–1912, 6–11 July, (2014).
14. T. White, "Hadoop, The Definitive Guide," *OReilly Media, Inc.*, (2012).
15. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *OSDI'04: Proceedings of the 6th Symposium on Operating System Design and Implementation, San Francisco, California, USA. USENIX Association*, 137–150, (2004).
16. S. Owen, R. Anil, T. Dunning and E. Friedman, "Mahout in Action," *Manning Publications Co.*, (2011).
17. V. López, S. del Río, J.M. Benítez and F. Herrera, "Cost-sensitive linguistic fuzzy rule based classification systems under the mapreduce framework for imbalanced big data," *Fuzzy Sets and Systems*, **vol. 258**, 5–38, (2015).
18. S. Río, V. López, J.M. Benítez and F. Herrera, "On the use of MapReduce for Imbalanced Big Data using Random Forest," *Information Sciences*, **vol. 285**, 112–137, (2014).
19. I. Palit and C.K. Reddy, "Scalable and parallel boosting with mapreduce," *IEEE Transactions on Knowledge and Data Engineering*, **vol. 24, no. 10**, 1904–1916, (2012).
20. Q. He, C. Du, Q. Wang, F. Zhuang and Z. Shi, "A parallel incremental extreme SVM classifier," *Neurocomputing*, **vol. 74, no. 16**, 2532–2540, (2011).
21. H. Ishibuchi and T. Yamamoto, "Rule Weight Specification in Fuzzy Rule-Based Classification Systems," *IEEE Transactions on Fuzzy Systems*, **vol. 13, no. 4**, 428–435, (2005).
22. O. Cordón, M.J. del Jesus and F. Herrera, "A proposal on Reasoning Methods in Fuzzy Rule-Based Classification Systems,"*International Journal of Approximate Reasoning*, **vol. 20, no. 1**, 21–45, (1999).
23. Z. Chi, H. Yan and T. Pham, "Fuzzy algorithms with applications to image processing and pattern recognition," *World Scientific*, (1996).
24. L.X. Wang and J.M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man, and Cybernetics*, **vol. 22, no. 6**, 1414–1427, (1992).
25. K. Bache and M. Lichman, "UCI Machine Learning Repository," *[Online; accessed November 2014] (http://archive.ics.uci.edu/ml)*, (2014).
26. V. López, A. Fernández, S. García, V. Palade and Francisco Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, **vol. 250**, 113–141, (2013).
27. K. Trawinski, O. Cordón, A. Quirin, "On designing fuzzy multiclassifier systems by combining FURIA with bagging and feature selection," *International Journal of Uncertainty, Fuzziness, and Knowledge-based Systems*, **vol. 19, no. 4**, 589–633, (2011).