Chi-BD-DRF: Design of Scalable Fuzzy Classifiers for Big Data via A Dynamic Rule Filtering Approach

Fatemeh Aghaeipoor
Dept. of Mathematics and Computer Science
Shahid Bahonar University of Kerman, Iran
Email:ff.aghaei@eng.uk.ac.ir

Mohammad Masoud Javidi
Dept. of Mathematics and Computer Science
Shahid Bahonar University of Kerman, Iran
Email: javidi@uk.ac.ir

Isaac Triguero
Computational Optimisation and Learning (COL) Lab
School of Computer Science, University of Nottingham, United Kingdom
Email: Isaac.Triguero@nottingham.ac.uk

Alberto Fernández
DaSCI Andalusian Research Institute
University of Granada, Spain
Email: alberto@decsai.ugr.es

***Abstract*—Big data classification problems are known to be no longer addressable by sequential algorithms. Therefore, it is necessary to design and develop novel solutions to provide accurate yet interpretable models in a tolerable elapsed time. In this area, Fuzzy Rule-Based Classification Systems are very advantageous due to their intrinsic interpretable and accurate capabilities. However, when these systems are applied in Big Data scenarios, the size of the rule set can become too large to be useful, whereas many of the generated rules could be associated with the non-dense areas or outliers. The presence of such rules in the rule base not only increases the running time and computation overheads but also affects on the interpretability of the fuzzy system. In this contribution, we propose a novel approach to obtain compact and accurate fuzzy models for Big data problems in a linearly scalable complex time. To do so, a dynamic filtering approach is applied to remove low supporting rules. Moreover, an efficient computation of the rules' weights is presented to improve the accuracy of the predictions. This model is developed for Big Data analytics by using Apache Spark framework. This allows taking advantage of the built-in resources and directives for a transparent distributed computing, as well as the machine learning pipeline to ease the complete processing. Experimental results, using different Big Data problems, confirmed the goodness of the proposed algorithm with respect to the baseline fuzzy classifier.**

***Index Terms*—Fuzzy Rule-Based Classification Systems, Big Data, Spark, Dynamic Rule Filtering, Rule Weights**

I. Introduction

Nowadays, massive amounts of data are generated around the world. These data which are huge in the Volume, different in the Variety, and fast in the Velocity of generation are known as Big Data [1]. These characteristics cause some difficulties in the storing, processing, and analyzing Big Data so that the traditional data mining algorithms can not address these challenges well [2]. Moreover, given the amount of data, providing explainable and interpretable models that translates into the current trend for eXplainable Artificial Intelligence (XAI) [3], is not a straightforward task in big datasets. Summing up, these issues motivate the researchers to re-design and develop novel, scalable, and efficient solutions specifically for Big Data applications.

Different technologies such as Hadoop Map-Reduce and Apache Spark framework developed based on distributed computing to address the requirements of Big Data [4]. Map-Reduce is a programming paradigm that helps to the parallelization of an algorithm using two simple functions, namely Map and Reduce. Whereas the former function runs the same procedure on different partitions of a big dataset, the latter fused their results to bring about a single output. Although this data processing technique was originally employed by Apache Hadoop, it can be extended by different alternatives such as Apache Spark.

Apache Spark, introduced in 2010, is a large open source project developed for distributed operations. Even though it has some in common with Hadoop methodologies, it presents some specific features intentionally for Big Data analytic; features like in-memory computations that avoid cost-extensive I/O operations, the inherent property of Hadoop ecosystem. This capability is the most effective reason that makes Spark a framework faster than that of Hadoop [5].

Owing to the "special" requirements for the Big Data frameworks, a current target area of interest for both academia and industry is to design and develop new Machine Learning (ML) algorithms in this context. In this regard, one of the most widely used target areas is Big data classification problems.

As for standard (small-data) applications, the use of rule-based systems in general [6], and fuzzy rule-based systems [7] in particular is advisable from the point of view of XAI [8]. The reason behind this is the good trade-off of these solutions in terms of interpretability/accuracy. Furthermore, the goodness of the fuzzy representation for both the semantics/interpretability and also to cover the clusters in big data

978-1-7281-6932-3/20/$31.00 ©2020 IEEE

in a more general way [9].

Fuzzy Rule-Based Classification Systems (FRBCSs) are basically developed to classify input examples using fuzzy rules. They represent the relationship between variables using fuzzy sets. The same as a typical fuzzy system, each FRBCS is composed of a Knowledge Base (KB) and an inference module. KB includes all defined fuzzy sets in a Data Base and all generated rules in a Rule Base (RB).

Fuzzy rules can be defined using expert knowledge or generated using learning algorithms. One of the most well-known rule learning algorithms is for Chi et al. [10]. Chi algorithm is a fast and simple method that generates one fuzzy rule for each data sample. The method of this algorithm is commonly employed as the base strategy in developing new algorithms. The reason is that this method does not involve any kind of feature selection, sample selection, or rule selection that may bias the functionality of the systems. Moreover, the simplicity of this model allows evaluating the influence of new components.

The Chi algorithm has been also adapted for applying in Big Data applications in several studies. The initial version was [11] where the authors proposed the Chi-FRBCS-BigData and used the MapReduce framework to generate the RB. Indeed, they generated the initial rules through the Map function and then modified the possible conflicts using the maximum or average value of the rules' weight in the Reduce function. This algorithm improved as the Chi-BD in [12]. This new version considered all the data samples instead of local partitions to compute the rules' weights. In this way, exactly the same original Chi-FRBCS was obtained. However, computing the rules' weights with a whole view of all the samples was still a problem. It was extremely time-consuming which affected the scalability of the model.

In the following, Chi-BD-SF [13] proposed to handle the rules' weight computation by providing an approximated metric. Chi-BD-SF applied the standard support of a rule to modify the rule conflicts. This contribution not only reduced the time of the rule generation step but also improved the performance of the FRBCSs rather than the previous versions.

Taking a look at this progress line reveals that some designs are not linearly scalable, or when they try to overcome this issue, they missed significant components such as the rules' weights. Additionally, we acknowledge that in spite of the good coverage of fuzzy rules, especially when a low granularity is applied, there are cases in which an excessive number of rules can be generated, making difficult the actual understanding of the phenomena under study. Furthermore, in some cases, many of the generated rules could be associated with non-dense areas or outliers [13].

In this study, we propose Chi-BD-DRF, abbreviated of Chi Big Data Dynamic Rule Filtering, by using the Chi algorithm as the baseline model. It provides a linear computational procedure for obtaining fuzzy rules and manages the conflict between class consequences by computing the non-fuzzy confidence of the rules. This measure, came from association rule mining concepts [14], assists to extract more reliable rules without computation overheads. Furthermore, a novel filtering procedure is proposed to discard the aforementioned non-dense areas within the problem domain and consequently obtain a more interpretable and compact RB. This filtering procedure is dynamically adapted to the problem under study with respect to the characteristics of each dataset as well as its corresponding RB. Additionally, Chi-BD-DRF takes advantage of the rules' confidence to improve the accuracy of predictions.

Chi-BD-DRF utilizes the capabilities of Apache Spark [15] and MLlib [16] to develop an efficient distributed method based on the DataFrame partitions as well as a well-structured algorithm by using the ML pipeline. The fusion of ML pipeline and Spark DataFrames is a direct and straightforward strategy to replicate and distribute the same operations across multiple executing nodes in the case of Big Data environments.

In order to contrast the good behavior of this procedure, we used five big datasets with two performance metrics, namely accuracy and elapsed time. The obtained results revealed the effectiveness of the Chi-BD-DRF and its components rather than the previous implementations.

The remainder of this work is structured as follows: In Section II, the preliminaries of Big Data problems, tools and frameworks are described. In Section III, the Chi-BD-DRF model and the proposed filtering approach are discussed. In the following, Section IV provides the descriptions of the conducted experiments, evaluations and results. Finally, Section V concludes this contribution.

II. Big Data in Machine Learning: Tools and Frameworks

Analyzing the vast amount of data needs adapting the standard data mining and ML algorithms and proposing the scalable tools and frameworks. In this regard, one of the most efficient strategies is referring to distributed computing where the data is processed through a cluster of nodes [1]. Indeed, the data is split into several partitions and spread along different nodes. Then, each node is responsible to proceed with its local partitions. In this way, the data would be processed in parallel using the capabilities of multiple interconnected machines instead of a single one.

The widespread paradigm for distributed implementations in Big Data is MapReduce. The basis for it is HDFS, in which data is stored by chunks along with all nodes of the cluster, and therefore computation is moved to where data is located, diminishing the network usage [4]. Among different environments that make possible MapReduce-like implementations, Apache Spark due to its exclusive characteristics attracts the attention between data scientists.

Spark is a framework 10-100X faster than Hadoop MapReduce. This is due to the in-memory computation that Spark has supplied for the cluster executors. Moreover, Spark has also other favorable characteristics which make it a powerful tool in Big Data environment [17]. Along this section, we will introduce these characteristics of Apache Spark, especially regarding the data structures and pipelines to manage the

distributed computation transparently in ML implementations (Section II-A). Then, we will focus on the use of pipelines for a simpler yet robust design of ML and Big Data solutions (Section II-B).

A. Spark and MLlib

Apache Spark is a fast and powerful computing engine for large-scale distributed data processing. It provides high-level APIs in Java, Scala, Python and R [15]. The main characteristic of the Spark framework that leads to fast data processing is in-memory cluster computing that emerged from Resilient Distributed Datasets (RDDs). The former is the base data structure of Spark. It is composed of a distributed collection of objects partitioned for different processing nodes. In a higher level of abstraction, DataFrame appeared on the top of RDD in Spark Release 1.3.0. It preserves the features of RDD besides providing the capability of processing a large amount of structured data in a more efficient manner [17].

On the top of Spark core APIs, there have been extended several specific-purpose libraries for SQL (Spark SQL), stream processing (Spark Streaming), machine learning (MLlib), and graph processing (GraphX) [16]. Among these, MLlib by providing several tools such as ML Algorithms, Featurization, and Pipelines facilitates the implementation of scalable data mining algorithms. MLlib presents the algorithms through the RDD-based APIs (spark.mllib package) and the DataFrame-based (spark.ml package). The former is now in maintenance mode, the focus is on the DataFrame-based APIs due to more user-friendly nature, benefiting of SQL queries, providing uniform APIs across multiple languages, and also Tungsten and Catalyst optimizations [18]. Another advantage that should be taken into account, is the capability of DataFrames to assist to create and tune practical ML pipelines [19].

B. ML Pipelines

In software engineering, a chain of processing units in which the output of each unit is the input of the next is called a pipeline. Indeed, a stream of information flows through multiple consecutive components and while each component processes the information individually, they all collaborate to make a comprehensive schema. The same idea can be utilized in ML.

A typical ML application includes several critical components such as pre-processing tasks, model construction, evaluation, and tuning that all should be run in order. These components can be organized into a single pipeline and it helps to provide a uniform framework for training and test data. Moreover, several advantages like standardization of components, ease of execution by other users, integration into standard libraries can be also obtained.

MLlib supports several standard APIs to implement ML applications in the structure of pipeline easily. In these APIs, each ML pipeline consists of several stages that each one can be either a Transformer or an Estimator. The input DataFrame is processed using the operation of each Transformer or Estimator as the following description.

1) Transformer: is an abstract concept for the components who are responsible to make new DataFrame from the available one. Technically, each transformer has a `transform()` method to run a specific algorithm and convert one DataFrame into another DataFrame. Feature transformers and learned models are examples of widely-used transformer.

2) Estimator: refers to the components that learn a model based on the input DataFrames. Technically, each Estimator includes a `fit()` method which returns a trained model as a transformer. Different classification algorithms are examples of Estimator. It is worth to mention that a pipeline itself is an Estimator that must be fitted by the input training data.

III. Chi-BD-DRF: A Scalable Fuzzy Big Data Classifier based on a Novel Dynamic Rule Filtering Approach

This section describes the details of our new proposed Chi-BD-DRF. At first, we review the method of the Chi algorithm as the baseline of this study in Section III-A. Then, the working procedure of the Chi-BD-DRF including the adaptation of the rule generation steps will be discussed in Section III-B. Next, the main contribution of this study, namely the significance of the dynamic filtering approach is described in Section III-C. Finally, we will explain how the Chi-BD-DRF is fitted with an ML pipeline in Section III-D.

A. The basics of Chi-FRBCS

Chi et al's algorithm [10] is a fast and simple rule learning method for FRBCSs. It tries to learn the fuzzy rules from input examples through the following three main steps:

- Step 1: Define fuzzy sets and membership functions for all the variables and then fuzzify the input values using them.
- Step 2: Generate one candidate fuzzy rule for each data sample using labels with maximum membership value. Suppose that there is a dataset with n input variables and m class label as $C = \{c_1, ..., c_m\}$. In this way, for data sample $\overrightarrow{x_i} = \left(x_i^1, x_i^2, ..., x_i^n \ : \ c_i\right)$, fuzzy rule R_i will be generated in the following format:

$$R_i : A_i \rightarrow c_i : RW_i \quad ; \quad A_i = \left\{A_i^1, ..., A_i^n\right\} \qquad (1)$$

  where A_i is the antecedent part of the rule and c_i is its consequence. Each rule can also have a rule weight as RW_i.
- Step 3: Modify conflicts between rules with the same antecedent but different consequent.
- Step 4: Apply a fuzzy reasoning method to make predictions.

B. The core procedure of Chi-BD-DRF

In the introduction, we have stressed the need for developing scalable algorithms to address Big Data problems. Therefore, any procedure that involves processing all the data samples several times must be avoided. In the particular context of FRBCS, this issue is related to the rules' weight computation.

Furthermore, for the sake of obtaining a compact and interpretable system, a limitation in the size of the RB must be applied.

Following these requirements, a novel procedure for learning FRBCS, named as Chi-BD-DRF, is proposed in this work. To do so, we extend the main steps of the Chi algorithm as follows:

1) Fuzzification: This step builds the DB using triangular membership functions and uniform fuzzy partitioning. It requires data to be normalized to have the same universe of discourse. In big datasets, these processes are carried out by the standard distributed operations via Spark.

2) Rule Learning and Rule Weight Computation: For each available sample, one candidate rule in the format of Eq. (1) should be generated. For this purpose, the input DataFrames are split into several partitions and the standard Chi learning procedure (Section III-A) is applied to map the data samples into initial fuzzy rules in each node. Then, an aggregation stage is devoted to computing the actual consequents of the rules, i.e., if a generated rule would be unique, it is a member of the final RB without any change, however, if it conflicts with the other generated rules, its final consequent is obtained using the principles of the next step. To supply it, the confidence of each is also calculated in this step.

The confidence of the i-th rule is calculated as follows [14]:

$$Conf(R_i) = \frac{\sigma(A_i, c_i)}{\sigma(A_i)} \tag{2}$$

where $\sigma(A_i, c_i)$ counts the number of occurrence of this rule (both antecedent and consequence parts) in the RB, and $\sigma(A_i)$ counts the number of rules which have exactly the same antecedent as rule i. Since this measure shows an approximation of the strength of a certain rule among a collection of rules, we propose to apply this computation as the rule weight as follows:

$$RW_i = Conf(R_i) \tag{3}$$

This rule weight not only reduces the computation time but also is an approximation of the reliability of each rule. Indeed, this crisp confidence is not directly involved with the data samples, while the original fuzzy version must iterate over all the input samples and it is completely inefficient in a MapReduce programming framework.

3) Modify Conflict: As the basic procedure when all rules are individually generated, there may be several rules that share the fuzzy antecedent but comprises a different class output. In these situations, that rule which has the highest confidence is more reliable and would be chosen among them.

4) Prediction: After obtaining the final RB, the fuzzy reasoning is carried out through the Winning-Rule approach. It classifies each sample in the class of the most compatible rule, i.e., the class of sample $\overrightarrow{x_p} = \left(x_p^1, x_p^2, ..., x_p^n\right)$ is predicted through the following computations. At first, the matching degree of each rule with this sample, $h_{R_i}(x_p)$), is calculated as follows:

$$h_{R_i}(x_p) = \prod_{k=1}^{n} \mu_{A_i^k}(x_p^k) \tag{4}$$

then, the rule with the maximum matching degree is selected as the winner, R_o, and the prediction is done as Eq. (6).

$$R_o = arg\ max_{R_i \in RB_{final}} \left\{h_{R_i}(x_p)\,.\,RW_i\right\} \tag{5}$$

$$class(x_p) = c_o \tag{6}$$

C. DRF: Dynamic Rule Filtering Method

This section focuses on the main contribution of this study, namely the Dynamic Rule Filtering (DRF) approach. Given that each data sample generates one candidate rule in the Chi-based algorithm, the number of final rules would be numerous in the scenario of Big Data, especially when higher levels of granularity are applied. On the other hand, as previously discussed, excessive number of rules affect on the interpretability of the systems so that the actual understanding of the phenomena under study would not be possible. Moreover, a large RB may include irrelevant or ineffective rules which might affect the generalization ability of the classifier, leading to a sub-optimal predictive performance. Therefore, a large RB is undesirable not only in terms of interpretability but also from the accuracy perspective. To address this problem, a novel dynamic filtering approach is integrated into the Chi-BD-DRF.

The premise of this novel approach is to find and discard areas of low influence within the problem for the sake of obtaining a compact yet accurate RB. Therefore, we must design a metric that establishes the interestingness of each rule, and to define a threshold to maintain those rules that are truly necessary to describe the problem under study. Among different alternatives, support is selected for being informative and straightforward to compute. Indeed, low-support rules are not frequent and are usually related to low-dense areas. These rules can be recognized using a minimum support threshold.

DRF method proposes filtering infrequent rules by setting a dynamic minimum threshold for the rule' support. In a certain RB, the support of the i-th rule (1) is defined as follows [14]:

$$Supp(R_i) = \frac{\sigma(A_i, c_i)}{|RB|} \tag{7}$$

where $\sigma(A_i, c_i)$ is the number of occurrences of this rule in the RB, and $|RB|$ is the size of the RB which is equal to the number of all available rules. For the sake of simplicity, we consider support of the rules in the scale of the RB, i.e., $Supp(R_i) = \sigma(A_i, c_i)$. DRF considers low-support rules as the infrequent ones targeted to be filtered. Indeed, those rules whose support are less than average of samples will be removed from the RB. For this purpose, a threshold $MinSupp$ is defined for the support values as follows:

$$MinSupp = \lfloor \frac{\#Samples - 1}{\#Rules} \rfloor \tag{8}$$

where $\#Samples$ is the number of available samples and $\#Rules$ is the number of all generated rules in the initial RB. After determination of $MinSupp$, the RB is scanned and those

rules that do not satisfy $MinSupp$ are removed, following as stated in Eq. (9):

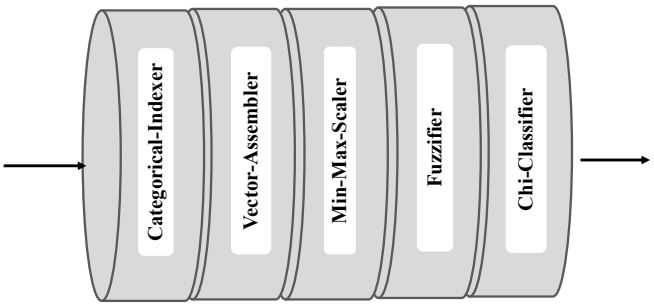
$$RB_{final} = \{R_i \in RB_{initial} \,|\, Supp\,(R_i) > MinSupp\} \quad (9)$$

By these considerations, based on the pigeon hole principle, there will be always remained some rules after DRF, even in the worst case scenario where all samples are equally divided. The goodness of the DRF approach, as its name implies, is setting the filtering threshold dynamically with respect to the characteristics of each dataset as well as its corresponding RB. It avoids considering a fixed value for all the cases or finding a user-defined one by trial and error, both being non-suitable procedures.

D. Pipeline Stages

To implement the Chi-BD-DRF algorithm introduced in Section II-B through an ML pipeline, five stages, including three Estimators and two Transformers, are developed. The first four stages, namely Categorical-Indexer, Vector-Assembler, Min-Max-Scalar, and Fuzzifier focus on doing pre-processing tasks. These components prepare the input DataFrames to train the classifier.

In the last stage of the Chi-BD-DRF pipeline, there is a Chi-Classifier Estimator that is the main processing core of the model and will be run on the top of a DataFrame containing rows of samples and a column of features. Features are initially individual columns, either numerical or categorical. They are transformed into a vector of normalized values to fit the pipeline. Fig. 1 shows the workflow of the Chi-BD-DRF pipeline. In the following, the processing stages of this pipeline are respectively described.



Chi-BD-DRF_ Pipeline

- Chi-BD-DRF_ Pipeline.fit (Training _ Dataframe) ⟶ Chi-BD-DRF_Model
- Chi-BD-DRF_ Model.transform (Test _ Dataframe) ⟶ Prediction

Fig. 1. The workflow of the Chi-BD-DRF pipeline.

1) Categorical-Indexer: encodes all the categorical columns of a DataFrame into the numerical columns. Indeed, this Estimator takes the categorical columns of a training DataFrame and maps the values of each column into indices in the range $[0, numCat_i)$, where $numCat_i$ is the number of categories in column i.

2) Vector-Assembler: is a Transformer that builds a single feature vector by concatenating multiple individual input columns, either the raw numerical columns or the columns generated by the previous stage.

3) Min-Max-Scalar: is an Estimator that gives a numerical feature vector and re-scales its values to the range $[0, 1]$.

4) Fuzzifier: is a Transformer to fuzzify the values of the input columns using their corresponding fuzzy sets. This component follows exactly the same fuzzification procedure of the Chi-BD-DRF. At the end of this stage, feature vectors are ready to train each kind of learning algorithm.

5) Chi-Classifier: is the main processing stage of the Chi-BD-DRF pipeline. This Estimator generates the final RB and produces a classification model following the principles of steps 2 and 3 of Section III-B as well as the DRF approach of Section III-C. The obtained model is a Transformer that is applied to make predictions as the last step of the Chi-BD-DRF procedure.

These stages are run respectively, and the input DataFrames are converted to the output ones by moving from the first stage toward the last one.

IV. Experimental Study

In this section, the experiments conducted to evaluate the performance of Chi-BD-DRF from different perspectives are detailed. To do so, the experimental framework is first included (Section IV-A) where we introduce the selected datasets and the evaluation metrics. Then, the proposed filtering method, namely DRF is evaluated (Section IV-B). Finally, the influence of integrating the new confidence-based rule weights into the Chi-BD-DRF is investigated (Section IV-C).

A. Experimental Set-Up

In this study, 5 big classification datasets, obtained from UCI and OpenML data repository, were employed to execute the experiments. These datasets have different numbers of features and samples summarized in Table I. Moreover, the mechanism of 5-fold cross-validation was applied to generate training and test data and the average result of five runs was reported as the final output of each model. All the experiments were developed using the Spark framework and Scala language. These experiments run on a cluster including 15 nodes, one master and 14 executor, with the following configuration:

TABLE I
Properties of the used datasets

Datasets	Abbr.	#Samples	#Features	#Classes
susy	susy	5,000,000	18	2
covtype1	cov	581,012	54	2
BNG-heart	BNG-h	1,000,000	13	2
BNG-Australian	BNG-Au	1,000,000	14	2
poker0-vs-2	poker	562,529	10	2

- Processor: Intel Xeon CPU E5645 @ (2.40GHz) x2.

- Cores: 12 threads (6 cores).
- Main memory: 96GB.
- Cache: 12 MB.
- Network: 40 Gb/s Infiniband.
- Operative System: CentOS 6.9.
- HDFS: Version 2.6.0-CDH5.8.0.

the total time needed by a given method to produce the results, including all the MapReduce/Spark stages (reading, writing,network communications, etc.) In the following tables, four measures including the test accuracy (Acc), the number of generated rules (#Rule), the running time (hh:mm:ss), and the rule filtering threshold (MinSupp) have been indicated. Running times are obtained considering the whole procedure of the FRBCS algorithms, including reading Spark DataFrames, learning, classifying, and network communications. Additionally, the accuracy of prediction is calculated using the confusion matrix as follows:

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{10}$$

B. Evaluating the Influence of DRF Method

Owing to the high number of generated rules in big datasets, DRF is the most essential component of Chi-BD-DRF to obtain an efficient and scalable FRBCS. Hence, to provide a comprehensive evaluation firstly, the influence of DRF in the performance of Chi-BD-DRF is individually investigated.

For this purpose, the Chi-BD-DRF algorithm run in two modes, one without integrating the DRF into the Chi-classifier stage of Section III-D and another mode using it. These modes have been called "Before Filtering" and "After Filtering" in Table II.

TABLE II
RESULTS OF DYNAMIC RULE FILTERING METHOD

	Before Filtering			**After Filtering**			
Datasets	Acc	#Rule	Time	Acc	#Rule	Time	MinSupp
susy	0.6553	10852.2	00:12:32	0.6515	1002.6	00:01:37	369.2
cov	0.7880	8152.2	00:13:28	0.7687	1408.4	00:02:40	56.6
BNG-h	0.8638	29570.8	00:47:24	0.8589	3230.8	00:04:03	26.4
BNG-Au	0.8490	9773.8	00:12:10	0.8441	910.8	00:01:29	82.2
poker	0.9081	51132.4	01:49:25	0.9122	15815.0	00:28:36	8.0
Avg.	0.8129	21896.28	00:34:59	0.8071	4473.52	00:07:41	108.48

Comparing the results of Table II shows that the number of rules and consequently the execution times considerably reduced After Filtering. Indeed, we obtained an average of 79.5% reduction in the number of rules by an approximately 5x faster model. However, removing this quantity of rules from the RB did not affect severely the accuracy of the models. i.e., it decreased by around 1% in average.

These findings signify the effectiveness of the proposed dynamic filtering threshold in providing a fast and scalable model. Moreover, due to the significant reduction in the number of generated rules, DRF could improve the interpretability of the model in the level of RB [20] and it can be successfully integrated into the Chi-BD-DRF besides the other components.

It is worth to mention that since the last version of Chi-FRBCS for Big Data, namely Chi-BD-SF [13], is the same Chi-BD-DRF without rule weight and filtering, namely "Before Filtering" mode in Table II, and given that the Chi-BD-SF outperformed the previous versions [11], [12], we discarded comparisons to the former two approaches.

C. Evaluating the Influence of Rule Weight

In this section, we aim to evaluate the influence of the proposed rule weight (Section III-B) to improve the accuracy of the Chi-BD-DRF. Hence, a Chi-BD-DRF model was run using two configurations, namely "without RW" and "with RW" in Table III. In the former, $RW_i = 1$ in all the cases of Eq. (5), while in the latter, the rules' weights have been set as Eq. (3).

Comparing the results of these two runs reveals that employing rule weight affects positively the accuracy of all the problems. In addition, since these weights were calculated simultaneously to the rule generation, they did not affect the computation time of the algorithm. Therefore, applying such as these weights instead of the traditional high-cost ones is suggested, especially in the accuracy-oriented tasks.

TABLE III
RESULTS OF APPLYING RULE WEIGHT

	without RW		**with RW**	
Datasets	Acc	Time	Acc	Time
susy	0.6515	00:01:37	**0.6604**	00:01:34
cov	0.7687	00:02:40	**0.7735**	00:02:40
BNG-h	0.8589	00:04:03	**0.8641**	00:04:09
BNG-Au	0.8441	00:01:29	**0.8453**	00:01:28
poker	0.9122	00:28:36	**0.9147**	00:30:22
Avg.	0.8071	00:07:41	0.8116	00:08:02

V. CONCLUSION

In this study, a fuzzy rule-based classification system based on a new dynamic rule filtering method was proposed for big data applications. This system employed the concepts of support and confidence to develop a scalable yet accurate and interpretable model using the well-known Chi et al's rule learning idea. Indeed, the non-fuzzy confidence of the rules was employed to modify the conflicting rules who had the same antecedents and different consequences. Given that this measure has a lower computational effort than its fuzzy version, it was also applied as an alternative rules' weight in Big Data applications where the traditional rules' weights are not practical. Additionally, the most critical component of this study, namely Dynamic Rule Filtering was developed using the rule's support and the infrequent rules were removed from the rule set to provide a compact collection.

This study took advantage of ML pipeline to implement the algorithm in a straightforward distributed schema. This

structure helped to ensure that the training and test data go through the identical pre-processing and core processing stages for all partitions of all executors.

The experimental results confirmed the effectiveness of the prospered Filtering method as well as the rules' weights. It was revealed that employing filtering methods based on low-computational measures, such as rule's support, could significantly reduce the running time without losing a great deal of accuracy. However, since the number of rules may still be high in some cases, this fast standard grid-based approach can be improved to obtain a reduced yet accurate RB, mainly by considering shorter antecedent rules with higher supports. This will be intended in our future works.

ACKNOWLEDGMENT

This work has been partially supported by the Spanish Ministry of Science and Technology under project TIN2017-89517-P, including European Regional Development Funds, and the Andalusian regional project P18-TP-5035.

REFERENCES

[1] M. A. Wani and S. Jabin, "Big data: issues, challenges, and techniques in business intelligence," in *Big data analytics*. Springer, 2018, pp. 613–628.

[2] A. Fernández, S. del 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 frameworks," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 380–409, 2014.

[3] A. B. Arrieta, N. Díaz-Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82 – 115, 2020.

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

[5] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015.

[6] H. Liu, A. Gegov, and M. Cocea, "Rule-based systems: a granular computing perspective," *Granular Computing*, vol. 1, no. 4, pp. 259–274, 2016.

[7] A. Fernández, S. del Río, A. Bawakid, and F. Herrera, "Fuzzy rule based classification systems for big data with mapreduce: granularity analysis," *Advances in Data Analysis and Classification*, vol. 11, no. 4, pp. 711–730, 2017.

[8] A. Fernandez, F. Herrera, O. Cordon, M. J. del Jesus, and F. Marcelloni, "Evolutionary fuzzy systems for explainable artificial intelligence: Why, when, what for, and where to?" *IEEE Computational Intelligence Magazine*, vol. 14, no. 1, pp. 69–81, 2019.

[9] P. Ducange, M. Fazzolari, and F. Marcelloni, "An overview of recent distributed algorithms for learning fuzzy models in big data classification," *Journal of Big Data*, vol. 7, no. 1, pp. 1–29, 2020.

[10] Z. Chi, H. Yan, and T. Pham, *Fuzzy algorithms: with applications to image processing and pattern recognition*. World Scientific, 1996, vol. 10.

[11] S. del Rio, V. Lopez, J. M. Benítez, and F. Herrera, "A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules," *International Journal of Computational Intelligence Systems*, vol. 8, no. 3, pp. 422–437, 2015.

[12] M. Elkano, M. Galar, J. Sanz, and H. Bustince, "CHI-BD: a fuzzy rule-based classification system for big data classification problems," *Fuzzy Sets and Systems*, vol. 348, pp. 75–101, 2018.

[13] L. Íñiguez, M. Galar, and A. Fernández, "Improving fuzzy rule based classification systems in big data via support-based filtering," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2018, pp. 1–8.

[14] N. Bhargava and M. Shukla, "Survey of interestingness measures for association rules mining: data mining, data science for business perspective," *IRACST-International Journal of Computer Science and Information Technology & Security (IJCSITS)*, vol. 6, no. 2, 2016.

[15] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[16] S. C. Pandey, "Recent developments in big data analysis tools and apache spark," in *Big Data Processing Using Spark in Cloud*. Springer, 2019, pp. 217–236.

[17] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3-4, pp. 145–164, 2016.

[18] K. Ishizaki, "Analyzing and optimizing java code generation for apache spark query plan," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 91–102.

[19] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[20] M. J. Gacto, R. Alcalá, and F. Herrera, "Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures," *Inf. Sci.*, vol. 181, no. 20, pp. 4340–4360, 2011.