A Distributed Fuzzy Associative Classifier for Big Data

Armando Segatori, Alessio Bechini, Pietro Ducange, and Francesco Marcelloni, *Member, IEEE*

***Abstract*—Fuzzy associative classification has not been widely analyzed in the literature, although associative classifiers have proved to be very effective in different real domain applications. The main reason is that learning fuzzy associative classifiers is a very heavy task, especially when dealing with large datasets. To overcome this drawback, in this paper, we propose an efficient distributed fuzzy associative classification approach based on the MapReduce paradigm. The approach exploits a novel distributed discretizer based on fuzzy entropy for efficiently generating fuzzy partitions of the attributes. Then, a set of candidate fuzzy association rules is generated by employing a distributed fuzzy extension of the well-known FP-Growth algorithm. Finally, this set is pruned by using three purposely adapted types of pruning.**

We implemented our approach on the popular Hadoop framework. Hadoop allows distributing storage and processing of very large data sets on computer clusters built from commodity hardware. We have performed an extensive experimentation and a detailed analysis of the results using six very large datasets with up to 11,000,000 instances. We have also experimented different types of reasoning methods. Focusing on accuracy, model complexity, computation time and scalability, we compare the results achieved by our approach with those obtained by two distributed non-fuzzy associative classifiers recently proposed in the literature. We highlight that, although the accuracies result to be comparable, the complexity, evaluated in terms of number of rules, of the classifiers generated by the fuzzy distributed approach is lower than the one of the non-fuzzy classifiers.

***Index Terms*—fuzzy associative classifier, Hadoop, Big Data, MapReduce, associative classifier, fuzzy FP-Growth.**

I. Introduction

Associative classifiers (ACs) have been widely investigated in the last years and exploited in a number of successful real-world applications [1], [2]. ACs integrate *frequent pattern mining* and *classification* into a single system [3], [4]. An AC is typically generated in two steps. First, a frequent-pattern mining algorithm [5] is used to extract associations out of the dataset; then, classification rules are generated from the associations, and pruned according to their support, confidence, and redundancy.

Frequent pattern mining algorithms operate on *categorical* attributes only. Anyway, numerical *continuous* attributes are often used to describe real data objects as well; a proper discretization algorithm can then be used to map continuous values onto a greatly reduced subset of discrete values [6], so to make them amenable to be dealt with by ACs. Clearly, the AC performance strongly depends on the discretization process. In classical approaches, the discretization process splits the attribute domain into a number of intervals, thus letting us assign each numerical value to the label of the relative interval it belongs to. To better model smooth transitions between adjacent intervals [7], the adoption of fuzzy subsets has been recently proposed, thus leading to Fuzzy Associative Classifiers (FACs) [8], [9], [10], [11].

FAC generation requires frequent pattern mining algorithms specifically designed to cope with fuzziness [8], [10]. In fact, in uncovering frequent patterns, we have to take into account that each value can belong to multiple different fuzzy sets with different membership degrees. In [10], we have proposed a novel approach for generating FACs employing a fuzzy version of the FP-growth algorithm [12]: we have shown that our AC based on a Fuzzy Frequent Pattern (*AC-FFP*) mining algorithm exhibits good accuracy and scalability. Thus, AC-FFP will be used also in the classifier proposed in this paper.

The approaches proposed so far to learn both ACs and FACs have mainly aimed at improving classification accuracy, often neglecting time and space requirements [13]. Moreover, whenever we need to deal with big data, that is, datasets that cannot be captured, stored, managed, and analyzed by classical database software tools [14], most of these learning algorithms cannot be applied because of computational and storage issues.

To the best of our knowledge, so far only a few works have tackled the problem of learning ACs and FACs from big data. MRAC and MRAC+ [15] are two versions of a distributed AC scheme shaped according to the MapReduce programming model, and suitable to run on a small cluster of commodity hardware. The scheme mines classification association rules (CARs) by using a properly enhanced, distributed version of the well-known FP-Growth algorithm. The obtained CAR set is then reduced by means of a distributed rule pruning procedure, to get to the rule base actually used for classification of unlabelled patterns. Fuzzy approaches have been pursued as well: AC-FFP [16] is a distributed version of the FAC introduced in [10], which relies on a strong fuzzy partition of continuous attributes, obtained through a specific algorithm roughly shaped on the classical procedure by Fayyad and Irani [17]. Moreover, Chi-FRBCS-BigData [18] is a fuzzy rule-based classifier (but not an associative one), developed according to the MapReduce programming model [19] and based on the approach previously described by Chi et al. [20]. First, distinct rule bases are concurrently generated from each block of the training set stored in different nodes. Subsequently,

Manuscript received Month DD, YYYY; revised Month DD, YYYY; accepted Month DD, YYYY. Date of publication Month DD, YYYY; date of current version Month DD, YYYY. This work was partially supported by the University of Pisa under grant PRA_2017_37 "IoT e Big Data: metodologie e tecnologie per la raccolta e l'elaborazione di grosse moli di dati."

A. Segatori, A. Bechini, and F. Marcelloni are with the Department of Information Engineering, University of Pisa, Pisa, Italy.

P. Ducange is with eCampus University, Novedrate, Italy.

Digital Object Identifier 10.1109/TCYB.YYYY.XXXXXXX

linguistic fuzzy rules are used to merge all the rule bases. A specialized version of Chi-FRBCS-BigData for handling imbalanced big datasets has been recently presented as well [21]. Finally, the effects of the granularity of fuzzy partitions have been studied in the context of Chi-FRBCS-BigData [22]. A deep analysis on the progress and opportunities on fuzzy systems for big data has been recently discussed in [23], pinpointing that just a few works have addressed the big data mining problem from the fuzzy modelling standpoint.

In this work, we extend the AC-FFP approach along two directions. First, we propose and employ a novel discretizer based on fuzzy entropy that efficiently generates strong fuzzy partitions of the continuous attributes. Second, we perform an extensive experimentation and a very deep analysis using six datasets with up to $11,000,000$ instances. Focusing on accuracy, model complexity, and computation time, we compare the results obtained by our approach to those obtained by the recent state-of-the-art distributed ACs discussed in [15]. Besides, despite of their comparable accuracies, the number of rules in the final FAC models is significantly lower than in the corresponding ACs.

The paper is organized as follows. Background information is summarized in Section II. Section III is dedicated to discuss each single step of the proposed distributed fuzzy CAR mining algorithm. In Section IV we show the experimental results and, finally, in Section V we draw proper conclusions.

II. Background

In this section, we introduce basic concepts on fuzzy associative classification and the MapReduce distributed programming model, as well as the AC-FFP algorithm.

In the following, we assume that any pattern is described by F different *attributes* in the set $\mathbf{X} = \{X_1, \ldots, X_F\}$. Any single attribute X_f can be either categorical or continuous. A pattern is thus represented by an F-dimensional vector $\mathbf{x}$, whose f-th element x_f assumes the value v_f belonging to the universe of X_f. A pattern (or "object") is associated with a class value (or "label") out of a predefined set of L classes $\mathbf{C} = \{C_1, \ldots, C_L\}$; the object class can be considered as its $(F+1)$-th attribute. A training set $\mathbf{TS}$ of N labelled objects (or "examples") is given: $\mathbf{TS} = \{\mathbf{o}_1, \ldots, \mathbf{o}_N\}$, with $\mathbf{o}_n = (\mathbf{x}_n, y_n)$, $y_n \in \mathbf{C}$. A single *item* refers either to the couple $(X_f, v_{f,j})$, with $f = 1, \cdots, F+1$ expressing the fact that attribute X_f takes its specific value (or class label) $v_{f,j}$.

A *classification* is the assignment of a specific class label $C_{\hat{l}}$ to an observed, unlabelled pattern $\hat{\mathbf{o}}$.

A. Fuzzy Associative Classifiers

Association rules describe correlations among *itemsets*, i.e. sets of items in patterns of a dataset. They are expressed in the form $Z \rightarrow Y$, where Z and Y are itemsets and $Z \cap Y = \varnothing$. Association rules have been widely used in market basket analysis, where items identify products and rules describe dependencies among them [24].

Classification association rules (CARs) are those association rules where the itemset Y is a class label. CARs are able to catch significant relationships among itemsets when considering categorical attributes. Continuous attributes can be dealt with as categorical ones by performing a proper *partitioning* of their universe.

In the fuzzy associative classification context, for each (continuous) attribute X_f, we consider a fuzzy partition $\mathbf{P}_f = \{A_{f,1}, \ldots, A_{f,T_f}\}$. In this case, the single item is defined as the couple $(X_f, A_{f,j})$, where $A_{f,j}$ is the j-th fuzzy set defined in $\mathbf{P}_f$. In this work categorical attributes (as well as class labels) will be treated on equal terms with continuous ones, considering categories as fuzzy sets and the membership degree of one value to its category always 1.

The m-th fuzzy classification association rule (FCAR) out of the rule base $\mathbf{RB} = \{FCAR_1, \ldots, FCAR_M\}$ is therefore expressed as

$$FCAR_m : \mathit{FAnt}_m \rightarrow C_{l_m} \text{ with } RW_m \tag{1}$$

where the consequent C_{l_m} is the class label selected for the rule and the antecedent FAnt_m is represented as the conjunction

$$\mathit{FAnt}_m : X_1 \textbf{ is } \mathcal{A}_{1,m} \ \ldots \ \textbf{AND} \ \ldots \ X_F \textbf{ is } \mathcal{A}_{F,m} \tag{2}$$

where the fuzzy set $\mathcal{A}_{f,m}$ used for variable X_f refers either to a fuzzy set $A_{f,j_{f,m}}$ in the partition for attribute X_f, or to the whole universe U_f (in this last case the membership degree of any value is 1, and the corresponding term is always true). The number of terms in Eq. 2 that are not always true is referred to as *rule length*, and is indicated by g_m. Notably, in Eq. 1 the rule is paired with a corresponding weight RW_m, to express its relative importance whenever used along with all the other rules.

The extent a given rule $FCAR_m$ matches an example $\mathbf{o}_n = (\mathbf{x}_n, y_n)$ is expressed via the *strength of activation* (or *matching degree* with the input), calculated as

$$w_m(\mathbf{x}_n) = \prod_{f=1}^{F} \mu_{f,m}(x_{f,n}), \tag{3}$$

where $\mu_{f,m}(x)$ is the membership function (MF) associated with the fuzzy set $\mathcal{A}_{f,m}$.

In association rule analysis, *support* and *confidence* are the most common measures to determine the strength of an association rule, with respect to the training set $\mathbf{TS}$. For a generic $FCAR_m$ they can be expressed as

$$\text{fuzzySupp}(\mathit{FAnt}_m \rightarrow C_{l_m}) = \frac{\sum_{\mathbf{x}_n \in \mathbf{TS}_{l_m}} w_m(\mathbf{x}_n)}{N} \tag{4}$$

$$\text{fuzzyConf}(\mathit{FAnt}_m \rightarrow C_{l_m}) = \frac{\sum_{\mathbf{x_n} \in \mathbf{TS}_{l_m}} w_m(\mathbf{x}_n)}{\sum_{\mathbf{x}_n \in \mathbf{TS}} w_{\mathit{FAnt}_m}(\mathbf{x}_n)} \tag{5}$$

where $\mathbf{TS}_{l_m} = \{\mathbf{x}_n \mid (\mathbf{x}_n, y_n) \in \mathbf{TS}, y_n = C_{l_m}\}$ is the set of objects classified with label C_{l_m}, $w_m(\mathbf{x}_n)$ is the matching degree of rule $FCAR_m$, and $w_{\mathit{FAnt}_m}(\mathbf{x}_n)$ is the matching degree of all the rules whose antecedent equals FAnt_m.

Different definitions have been proposed for rule weight RW_m [25]. As discussed in [26], rule weights can improve the performance of fuzzy rule-based classifiers. In this paper, we adopt the fuzzy confidence value, or certainty factor (CF),

computed as in Eq. 5.

The *association degree* $h_m(\mathbf{x}_n)$ of $\mathbf{x}_n$ with $FCAR_m$ is calculated as:

$$h_m(\mathbf{x}_n) = w_m(\mathbf{x}_n) \cdot RW_m \tag{6}$$

Given a set of fuzzy rules, different approaches have been proposed in the literature to infer a conclusion. Two of the most popular are:

1) *The maximum matching*: an input pattern $\widehat{\mathbf{x}}$ is assigned to the class label indicated by the rule with the maximum association degree for $\widehat{\mathbf{x}}$.
2) *The weighted vote*: an input pattern $\widehat{\mathbf{x}}$ is assigned to the class label $C_{\widehat{l}} \in \{C_1, \ldots, C_L\}$ with the maximum *total strength of vote* for $\widehat{\mathbf{x}}$; the total strength of vote for $\widehat{\mathbf{x}}$ on the generic class C_l is computed as

$$V_{C_l}(\widehat{\mathbf{x}}) = \sum_{R_m \in \mathbf{RB}_{C_l}} h_m(\widehat{\mathbf{x}}) \tag{7}$$

where $\mathbf{RB}_{C_l} = \{FCAR_m \in \mathbf{RB} \mid C_{l_m} = C_l\}$ is the set of FCARs in the rule base $\mathbf{RB}$ having C_l as consequent. With this method, each FCAR gives a vote for its consequent class. If no fuzzy rule matches the pattern $\widehat{\mathbf{x}}$, we classify $\widehat{\mathbf{x}}$ as *unknown*.

B. MapReduce and Distributed Computing Frameworks

MapReduce is a distributed programming paradigm that aims to parallelize the computational flow across large-scale clusters of machines by splitting the work into independent tasks [19]. Any basic computation is divided into the *Map* and *Reduce* phases, supported by specific tasks; each one takes a set of $\langle key, value \rangle$ pairs as input, producing a set of $\langle key, value \rangle$ pairs as output. Each machine can host multiple Map and Reduce tasks.

The MapReduce execution environment automatically partitions the dataset into Z independent *blocks* to be processed in parallel on the available q computing units within the cluster. By default, the number of Map tasks corresponds to the number Z of input blocks, and the number R of reducers is user-defined. Furthermore, several copies of the user program are distributed on the machine cluster. One copy, denoted as *master*, is in charge of scheduling and handling tasks across the cluster. The others, called *workers*, execute both Map and Reduce tasks assigned by the master. Each Map task is fed one data block and, for each input $\langle key, value \rangle$ pair, generates a list of intermediate $\langle key, value \rangle$ pairs as output. In the Reduce phase, all the intermediate results are grouped up according to a key-partitioning scheme (by default $hash(key)$ **mod** R), so that each Reduce task processes a list of values associated with a specific key. Basically, by designing the Map and Reduce functions, developers can easily implement parallel algorithms to be executed across the cluster.

Over the past few years, Apache Hadoop[1] has become one of the most popular MapReduce frameworks for the efficient processing of very large data sets, which are stored in its distributed file system, called Hadoop Distributed File System (HDFS) [27]. However, the extra costs of distributed I/O operations and the system bookkeeping overhead significantly reduce the benefits of parallelism in Hadoop. Thus, new distributed cluster computing frameworks, such as Apache Spark[2], are catching on. Spark is able to reduce access latencies by systematically caching datasets in memory. Spark also supports different distributed programming models, such as MapReduce and Pregel, and it has proved to perform faster than Hadoop, especially in case of iterative and online applications [28]. On the other hand, when dealing with data streams that are quickly produced, change very quickly and need real-time analysis, other technologies, such as Apache Samza and Apache Flink[3] may be more suitable than Hadoop and Spark. Moreover, a number of data mining libraries for big data have been recently released for distributed computing frameworks, like e.g. Mahout and MLlib[4], which implement a wide range of machine learning algorithms.

C. AC-FFP

AC-FFP is a fuzzy associative classifier that exploits a fuzzy version of the well-known FP-Growth algorithm [12] for the FCAR generation. FP-Growth extracts first the frequent items, and sorts them by descending frequencies. Then, such a dataset of frequent items is compressed into a frequent pattern tree, called *FP-tree*. Finally, FP-Growth recursively mines patterns by dividing the compressed dataset into a set of projected datasets, each associated with a frequent item or a pattern fragment. For each pattern fragment, only its associated conditional dataset has to be examined. The problem of mining frequent itemsets is converted into recursively building and searching trees. In AC-FFP, rule learning is performed in three phases: (i) discretization and fuzzy partitioning, (ii) FCAR mining, and (iii) FCAR pruning. In the following, for the sake of brevity, we introduce just a short description of AC-FFP: more information can be found in [10].

In the first phase, strong triangular fuzzy partitions are defined by exploiting the cut-points generated by the algorithm proposed by Fayyad and Irani [17]. Triangular fuzzy sets are defined by positioning the cores in correspondence to each cut-point and middle point (computed on each pair of adjacent cut-points). As a result, the universe of attribute X_f is discretized into the sequence of T_f fuzzy sets $\{A_{f,1}, \ldots A_{f,j}, \ldots A_{f,T_f}\}$.

In the FCAR mining phase, a fuzzy version of the FP-Growth algorithm [12] is used, with two scans of the whole training set; in this fuzzy version, items correspond to fuzzy sets, and the n-th object $\mathbf{o}_n$ in $\mathbf{TS}$ is transformed into the fuzzy object $\widetilde{\mathbf{o}}_n = (\widetilde{\mathbf{x}}_n, y_n)$ whose generic f-th element $x_{f,n}$ takes the fuzzy value $A_{f_n,j_{f_n}}$, with $j_{f_n} \in [1, \ldots, T_{f_n}]$. Notably, to limit the complexity, in the transformation the original value $x_{f,n}$ is mapped only onto the single fuzzy set with the highest matching degree, denoted as $\widetilde{A}_{f,j}$.

Two further scans of $\mathbf{TS}$ are performed to prune redundant FCARs and to reduce noise: first, AC-FFP prunes FCARs whose values for fuzzy support and confidence are lower than

[1] https://hadoop.apache.org

[2] http://spark.apache.org/

[3] Available at http://samza.apache.org/ and https://flink.apache.org/

[4] Available at http://mahout.apache.org/ and http://spark.apache.org/mllib/

fixed thresholds determined by the expert; then, the algorithm keeps only the FCARs that correctly classify at least one data object.

Finally, the selected FCARs are inserted into the definitive rule base. In performing classification, AC-FFP adopts the weighted vote [29] as *reasoning method*.

III. The Proposed Fuzzy Associative Classifier for Big Data

In this section, we describe the Distributed Fuzzy Associative Classifier based on a Fuzzy Frequent Pattern (DFAC-FFP) mining algorithm. With respect to the very preliminary version presented in a conference paper [16], the DFAC-FFP version described here introduces several novelties: first, an original discretizer is proposed and used for the generation of strong fuzzy partitions; second, we employ an efficient FCAR mining approach for speeding up the execution time; third, we experiment different reasoning methods.

The DFAC-FFP learning process consists of the following procedures:

1) *Distributed Fuzzy Partitioning*: a strong fuzzy partition is directly generated on each continuous attribute using a distributed approach based on fuzzy entropy;
2) *Distributed FCAR Mining*: a distributed fuzzy frequent pattern mining algorithm extracts frequent FCARs with confidence and support higher than a given threshold. This procedure exploits the parallel FP-Growth algorithm discussed in [30].
3) *Distributed FCAR Pruning*: the mined FCARs are pruned by means of two distributed rule pruning phases based on redundancy and training set coverage. The remaining FCARs are kept in the final rule base.

The overall process involves seven MapReduce phases and four scans of the training set. For the sake of clarity, we refer to its implementation on Hadoop.

A. Distributed Fuzzy Partitioning

Fuzzy partitioning of continuous attributes is a critical step in FCAR generation, because of its influence on the overall performance of classifiers. Although the classifier accuracy comes from the concerted use of all its components, this stage of the proposed learning process provides a significant breakthrough in achieving good results.

Here, we introduce a novel supervised fuzzy partitioning algorithm for the generation of strong triangular fuzzy partitions on each continuous attribute: it recursively determines the core of each triangular fuzzy set by choosing the candidate fuzzy partitions that yield the minimal fuzzy entropy, and then splits the continuous attribute domain into two subsets. Similar to the method proposed by Fayyad and Irani [17], the process is repeated for each generated subset until a stopping condition is met. We recall that the proposed methodology is not a plain parameter tuning process. Indeed, a tuning process is usually performed after the identification of a set of parameters: in our case, the parameters of the fuzzy sets are directly generated by means of the proposed fuzzy partitioning algorithm.

The discussion of the fuzzy partitioning approach requires some preliminary definitions. Let $\mathbf{TS}_f$ be the projection of the training set $\mathbf{TS}$ along attribute X_f. We assume the values in $\mathbf{TS}_f$ sorted in increasing order. A generic interval $I_f = [l_f, u_f]$ over the universe U_f of X_f contains a set $S_{I_f} = \{x_{f,1}, ..., x_{f,N_{I_f}}\}$ of N_{I_f} distinct values of $\mathbf{TS}_f$. A fuzzy partition $\mathbf{P}_{I_f}$ on the interval I_f is represented as $\mathbf{P}_{I_f} = \{A_{I_f,1}, \ldots, A_{I_f,|\mathbf{P}_{I_f}|}\}$, where $|\mathbf{P}_{I_f}|$ is the number of fuzzy sets in $\mathbf{P}_{I_f}$. Let $S_{I_f,1}, \ldots, S_{I_f,|\mathbf{P}_{I_f}|}$ be the subsets of points in S_{I_f} included in the support of $A_{I_f,1}, \ldots, A_{I_f,|\mathbf{P}_{I_f}|}$, respectively. The weighted fuzzy entropy $\mathrm{WFEnt}(\mathbf{P}_{I_f}, I_f)$ of partition $\mathbf{P}_{I_f}$ for I_f is defined as

$$\mathrm{WFEnt}(\mathbf{P}_{I_f}; I_f) = \sum_{j=1}^{|\mathbf{P}_{I_f}|} \frac{|A_{I_f,j}|}{|S_{I_f}|} \mathrm{FEnt}(A_{I_f,j}) \tag{8}$$

where $|A_{I_f,j}|$ is the fuzzy cardinality of fuzzy set $A_{I_f,j}$, $|S_{I_f}|$ is the cardinality of set S_{I_f} and $\mathrm{FEnt}(A_{I_f,j})$ is the fuzzy entropy of $A_{I_f,j}$.

We recall that the fuzzy cardinality of a fuzzy set $A_{f,j}$ is computed as

$$|A_{f,j}| = \sum_{i=1}^{N_{I_f,j}} \mu_{A_{I_f,j}}(x_{f,i}) \tag{9}$$

where $N_{I_f,j}$ is the number of points (crisp cardinality) in the subset $S_{I_f,j}$ and $\mu_{A_{I_f,j}}(x_{f,i})$ is the membership degree of example $x_{f,i}$ to fuzzy set $A_{I_f,j}$. The fuzzy entropy of $A_{I_f,j}$ is defined as

$$\mathrm{FEnt}(A_{I_f,j}) = -\sum_{l=1}^{L} \frac{|S^{C_l}_{I_f,j}|}{|A_{I_f,j}|} log_2(\frac{|S^{C_l}_{I_f,j}|}{|A_{I_f,j}|}) \tag{10}$$

where $S^{C_l}_{I_f,j}$ is the set of examples in $S_{I_f,j}$ with class label equal to C_l and L is the total number of class labels.

Given an initial fuzzy partition $\widetilde{\mathbf{P}}_{I_f}$ and a new generated partition $\mathbf{P}_{I_f}$, both defined on I_f, we can state that $\mathbf{P}_{I_f}$ can replace $\widetilde{\mathbf{P}}_{I_f}$ if and only if:

$$\mathrm{FGain}(\mathbf{P}_{I_f}; I_f) > \frac{\log_2(|S_{I_f}| - 1)}{|S_{I_f}|} + \frac{\Delta(\mathbf{P}_{I_f}; I_f)}{|S_{I_f}|} \tag{11}$$

where

$$\mathrm{FGain}(\mathbf{P}_{I_f}; I_f) = \mathrm{WFEnt}(\widetilde{\mathbf{P}}_{I_f}, I_f) - \mathrm{WFEnt}(\mathbf{P}_{I_f}; I_f) \tag{12}$$

$$\Delta(\mathbf{P}_{I_f}; I_f) = \log_2(3^{L_f} - 2) - \left[\sum_{j=1}^{|\widetilde{\mathbf{P}}_{I_f}|} \widetilde{L}_{f,j} \cdot \mathrm{FEnt}(\widetilde{A}_{I_f,j}) - \sum_{j=1}^{|\mathbf{P}_{I_f}|} L_{f,j} \cdot \mathrm{FEnt}(A_{I_f,j}) \right] \tag{13}$$

and $\widetilde{L}_{f,j}$ and $L_{f,j}$ are the number of class labels represented in the sets $\widetilde{S}_{I_f,j}$ and $S_{I_f,j}$, respectively.

Algorithm 1 shows the main steps of the proposed recursive fuzzy partitioning algorithm.

The recursive procedure *CreateFuzzyPartition* returns the

Algorithm 1 Fuzzy Partioning

Require: l_f, u_f, S_{I_f}, $\widetilde{\mathbf{P}}_{I_f}$

1: **procedure** CREATEFUZZYPARTITION(**in**: l_f, u_f, S_{I_f}, $\widetilde{\mathbf{P}}_{I_f}$)
2: $(x_f^{(best)}, \mathbf{P}_{I_f}) \leftarrow$ DETERMINEBESTPARTITION(S_{I_f}, l_f, u_f)
3: **if** STOPPINGCONDITION($\widetilde{\mathbf{P}}_{I_f}$, $\mathbf{P}_{I_f}$) **then**
4: **return** $\widetilde{\mathbf{P}}_{I_f}$
5: **end if**
6: $\mathbf{P}_{I_{f,1}} \leftarrow$ CREATEFUZZYPARTITION(l_f, $x_f^{(best)}$, $S_{I_f,1}$, $\mathbf{P}_{I_f}$)
7: $\mathbf{P}_{I_{f,2}} \leftarrow$ CREATEFUZZYPARTITION($x_f^{(best)}$, u_f, $S_{I_f,3}$, $\mathbf{P}_{I_f}$)
8: $\mathbf{P}_{I_f} \leftarrow$ MERGEFUZZYPARTITIONS($\mathbf{P}_{I_{f,1}}, \mathbf{P}_{I_{f,2}}$)
9: **return** $\mathbf{P}_{I_f}$
10: **end procedure**

partition $\mathbf{P}_{I_f}$ on the basis of a generic interval I_f (specified through l_f and u_f), the corresponding set of points S_{I_f}, and a previously defined partition $\widetilde{\mathbf{P}}_{I_f}$. To this aim, the procedure *DetermineBestPartition* scans S_{I_f} and, for each value $x_{q,f}$, defines a *candidate fuzzy partition* in the interval I_f as shown in Figure 1. We consider candidate (strong) partitions $\mathbf{P}_{I_f} = \{A_{I_f,1}, A_{I_f,2}, A_{I_f,3}\}$, containing three triangular fuzzy sets, whose cores are placed at l_f, $x_{q,f}$, and u_f. The procedure identifies the best partition

$$\mathbf{P}_{I_f} = \arg\min_{\mathbf{P}_{I_f}} \left(\mathrm{WFEnt}(\mathbf{P}_{I_f}; I_f)\right) \tag{14}$$

that minimize the weighted fuzzy entropy (Eq. 8), and the corresponding value for the central core $x_f^{(best)}$ is obtained (see Figure 2).

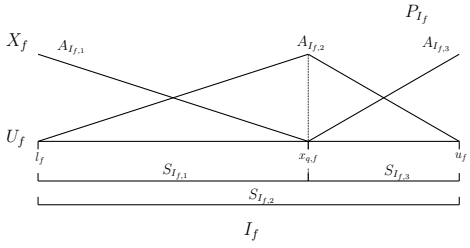

Fig. 1. An example of *candidate fuzzy partition* defined on $x_{f,q}$.

The *StoppingCondition* procedure just checks whether partition $\mathbf{P}_{I_f}$ can replace partition $\widetilde{\mathbf{P}}_{I_f}$ according to the condition on the fuzzy information gain (Eq. 11). If not, partition $\widetilde{\mathbf{P}}_{I_f}$ is returned. Otherwise, the interval I_f is split in the two intervals $I_{f,1} = [l_f, x_f^{(best)}]$ and $I_{f,2} = (x_f^{(best)}, u_f]$, and the process is recursively applied to them.

Notice that when *CreateFuzzyPartition* is recursively called on $I_{f,1}$ and $I_{f,2}$, the input initial partitions $\widetilde{\mathbf{P}}_{I_{f,1}}$ and $\widetilde{\mathbf{P}}_{I_{f,2}}$ are set equal to $\mathbf{P}_{I_f}$ as bounded, respectively, on $I_{f,1}$ and $I_{f,2}$. This means that in $I_{f,1}$ and $I_{f,2}$ the initial partitions are composed by just two triangular fuzzy sets with cores at the interval endpoints, as shown in Figure 2.

The two recursive calls of *CreateFuzzyPartition* return two adiacent partitions, namely $\mathbf{P}_{I_{f,1}}$ and $\mathbf{P}_{I_{f,2}}$, that share $x_f^{(best)}$ as a common core. The procedure *MergeFuzzyPartition* is devoted to merge the last fuzzy set and the first fuzzy set of the two adjacent partitions into a unique triangular fuzzy set and generates the final partition $\mathbf{P}_{I_f}$ (line 8).

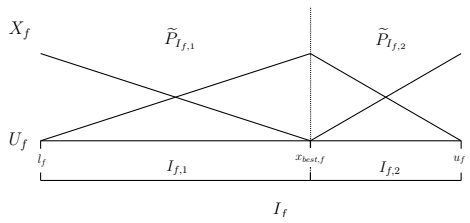

Fig. 2. An example of the two intervals generated by the *DetermineBestPartition* procedure.

The proposed fuzzy partitioning algorithm follows a top-down approach, and starts with an initial interval $I_f \equiv U_f$ corresponding to the universe of X_f, and an initial partition $\widetilde{P}_f$ with just one fuzzy set with membership degree equal to 1 throughout I_f. The initial S_{I_f} coincides with $\mathbf{TS}_f$.

At the end of the overall partitioning, a strong fuzzy partition $\mathbf{P}_f$ is defined for the universe of each input variable X_f. When the stopping condition is met at the first call of *CreateFuzzyPartition*, X_f cannot be significantly partitioned any more, thus it is discarded in any successive computation.

The complexity of the algorithm depends on the number of candidate fuzzy partitions evaluated in each interval. In fact, a candidate fuzzy partition has to be generated for each value in the interval, and the relative weighted fuzzy entropy has to be computed as well. When dealing with huge amounts of data, this approach leads to excessively long runtimes. To tackle this problem, for each attribute X_f we consider only a set of *representative values* instead of the complete $\mathbf{TS}_f$. Such values are computed by performing a *binning* of U_f, and taking the bin boundaries as representative values. To this aim, the class distribution for each bin is calculated, considering the number of actual values belonging to the different classes in each bins. It is worth noticing that the calculation of the class distributions has to be carried out only once.

The complete distributed fuzzy partitioning process requires two MapReduce steps (see Figure 3). The training set is split in Z blocks, and Z mappers are used as well, so that each block would feed one single *map-task*.

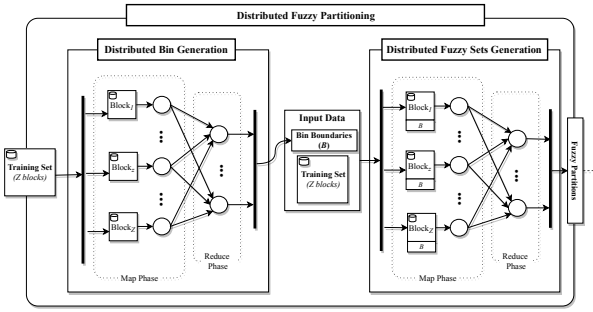

Fig. 3. MapReduce steps in the distributed Fuzzy Partitioning of DFAC-FFP.

The first MapReduce step is the *Distributed Bin Generation*, aimed at bounding the number of candidate fuzzy partitions to be evaluated. This goal is achieved by binning the universes of the continuous attributes. Each *Mapper*$_z$, working only on its relative *Block*$_z$, for each continuous attribute sorts the

values and identifies bins over them by including in each bin a number of instances equal to a given percentage γ of the data block (in our experiments, we set $\gamma = 0.1\%$). At most $\Gamma = 100/\gamma + 1$ bin boundaries of these equi-frequency bins are obtained as output. Then, all the bin boundaries relative to each attribute X_f produced by all the mappers are grouped together so that a reducer can sort them in increasing order and output a key-value pair $\langle$*key*:f, *value*:$B_f\rangle$, where B_f is the sorted list of the bin boundaries for X_f. The number of bin boundaries for attribute X_f relative to the whole $\mathbf{TS}$ is at most equal to $\Omega_f = Z \cdot \Gamma$. Given the simplicity of these operations, the relative pseudocode is not reported.

The second MapReduce step, named *Distributed Fuzzy Sets Generation*, obtains the fuzzy partitioning of each attribute X_f on the basis of the relative bin boundaries provided as input. Each mapper, for each bin of each attribute, counts the number of instances belonging to the different classes. Subsequently, each reducer generates, for each attribute, a strong fuzzy partition by using the presented fuzzy partitioning method and taking all the calculated bin boundaries as candidate central cores. The fuzzy entropy is always calculated by using the distribution of classes in each bin.

Algorithm 2 Mapper and Reducer in Distributed Fuzzy Sets Generation

Require: $\mathbf{TS}$ split into Z *Block*$_z$. Matrix B containing, for each row, a list of sorted bin boundaries B_f.

1: **procedure** MAPPER(**in**: *Block*$_z$, B, L)
2: $W_{f_z} \leftarrow$ create F vectors according to each B_f and L
3: **for each** $\mathbf{x}_n$ **in** *Block*$_z$ **do**
4: **for each** continuous attribute X_f **do**
5: $W_{f_z} \leftarrow$ update frequency according to $x_{f,n}$
6: **end for**
7: **end for**
8: **for each** continuous attribute X_f **do**
9: output $\langle$*key*:f, *value*:$W_{f,z}\rangle$
10: **end for**
11: **end procedure**
12: **procedure** REDUCER(**in**: f, $list(W_{f_z})$, B_f)
13: $W_f \leftarrow$ element-wise sum of $list(W_{f_z})$
14: $\mathbf{P}_f \leftarrow$ compute Fuzzy Partitioning with W_f and B_f
15: output $\langle$*key*:f, *value*:$\mathbf{P}_f\rangle$
16: **end procedure**

More in detail (see Algorithm 2), each *Mapper*$_z$ first loads the z-th block of $\mathbf{TS}$ and for each X_f, initializes a vector W_{f_z} of $\Omega_f - 1$ elements (line 2). Each vector element $W_{f_z,r}$ corresponds to the bin $(b_{f,r}, b_{f,r+1}]$ and contains an L-element vector that stores, for each of the L classes, the number of instances of the class belonging to the r-th bin in *Block*$_z$. Then, for each object of the block, the $Mapper_z$ updates W_{f_z} (line 5) and finally outputs a key-value pair $\langle$*key*:f, *value*:$W_{f_z}\rangle$. Each Reducer is fed the list $List(W_{f_z})$ with Z vectors W_{f_z} and it first creates a vector W_f of $\Omega_f - 1$ elements by performing an element-wise sum of all Z vectors W_{f_z}. Thus, W_f stores the frequency of each class in every bin for X_f. Then, the Reducer generates candidate fuzzy partitions (line 14) upon bin boundaries by computing the weighted fuzzy entropy using W_f. Space and time complexities of the Map phase are $O(\lceil \frac{Z}{q} \rceil \cdot N/Z)$ and $O(\lceil \frac{Z}{q} \rceil \cdot (N \cdot F \cdot \log(\Omega_f)/Z))$, respectively. Space and time complexities for each reducer are $O(\lceil \frac{F}{R} \rceil \cdot (\Omega_f - 1))$ and $O(\lceil \frac{F}{R} \rceil \cdot (2 \cdot \max(T_f) - 3) \cdot (\Omega_f - 1)^2))$, respectively, where $\max(T_f)$ is the maximum number T_f of fuzzy sets generated for an attribute.

B. The Distributed FCAR Mining Approach

The distributed FCAR mining approach extracts, for each class label, the K non-redundant FCARs with the highest confidence; its implementation is based on the Parallel FP-Growth (PFP-Growth) proposed by Li et al. [30].

The PFP-Growth algorithm uses three MapReduce phases to generate frequent patterns. In the first phase, it counts the support values for all the dataset items. In the second phase, each node builds a local and independent tree and recursively mines the frequent patterns from it. Such a subdivision requires the whole dataset to be projected onto different *item-conditional datasets*. An item-conditional dataset $\mathbf{TS}_{f,j}$, also called *item-projected dataset*, is a dataset restricted to the objects where the specific item $IT_{f,j}$ occurs. In each object of $\mathbf{TS}_{f,j}$, named *item-projected object*, the items with support smaller than $IT_{f,j}$ are removed, and the others are sorted according to the descending support order. Since the FP-tree building processes are independent of each other, all the item-projected datasets can be distributed over the nodes and processed independently. In the last phase, the algorithm aggregates the previously generated results and, for each item, selects only the highest supported patterns. As shown by empirical studies, the PFP algorithm achieves a near-linear speedup [30].

We adapted the PFP-Growth algorithm to generate frequent FCARs with high fuzzy confidence. As shown in Figure 4, the overall FCAR mining process consists of three MapReduce phases: (i) *Distributed Fuzzy Counting*, (ii) *Distributed Fuzzy FP-Growth*, and (iii) *Distributed Rules Selection*.

The *Distributed Fuzzy Counting* phase (detailed in Algorithm 3) scans the dataset and counts both the fuzzy support for selecting the *frequent fuzzy sets*, and the number of occurrences of each class label. A fuzzy set is frequent if its fuzzy support is higher than a minimum threshold *minSupp* fixed by the expert. The fuzzy support of each fuzzy set $A_{f,j}$ is defined as:

$$\text{fuzzySupp}(A_{f,j}) = \frac{\sum_{n=1}^{N} \mu_{f,j}(x_{f,n})}{N} \tag{15}$$

where $x_{f,n}$ is the value of the f-th attribute in $\mathbf{x}_n$. The same equation can be used also to deal with class labels, as the label is considered as the $(F+1)$-th attribute, i.e. $A_{F+1,j} \equiv C_j$: in this case, with crisp sets, $\mu_{F+1,j}(x_{F+1,n})$ equals either 1 or 0.

As usual, *Mapper*$_z$ is fed *Block*$_z$, whose r-th object is indicated as $\mathbf{o}_r = (\mathbf{x}_r, y_r)$. For each fuzzy set $A_{f,j} \in \mathbf{P}_f$, *Mapper*$_z$ outputs a key-value pair $\langle$*key*:$A_{f,j}$, *value*:$\mu_{f,j}(x_{f,r})\rangle$ (line 4), where $\mu_{f,j}(x_{f,r})$ is the membership degree of the f-th component of $\mathbf{x}_r$ to the j-th fuzzy set of partition $\mathbf{P}_f$; only fuzzy sets with matching degree > 0 are considered. Since strong partitions are used, at most $2F$ values can be associated to each $\mathbf{x}_r$. The mapper outputs also the pair $\langle$*key*:y_r, *value*:$1\rangle$. The reducer, for each key, is fed a list with the corresponding membership values, and outputs $\langle$*key*:$A_{f,j}$, *value*: fuzzySupp$(A_{f,j})\rangle$, where fuzzySupp$(A_{f,j})$ is calculated according to the Eq. 15. Notably, whenever

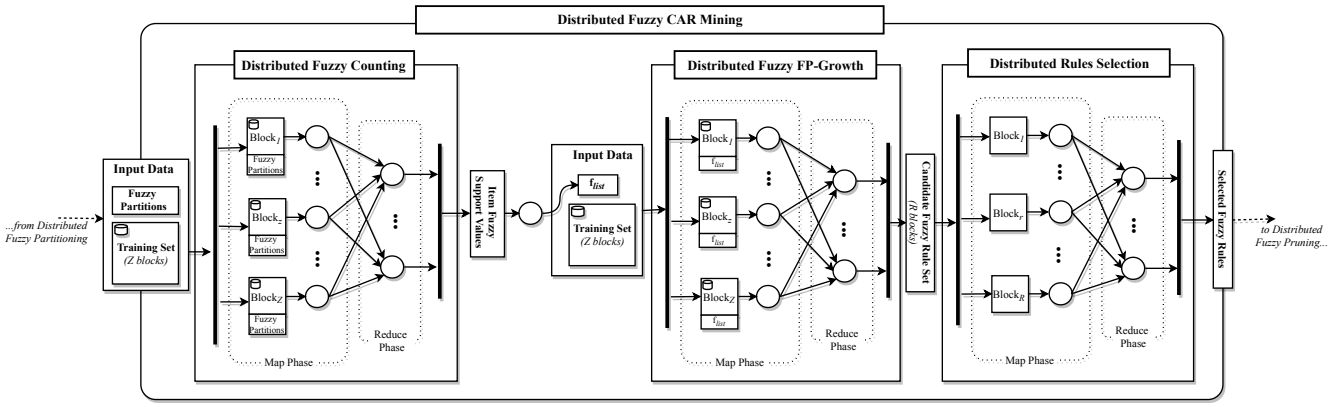

Fig. 4. The FCAR Mining process, part of the DFAC-FFP algorithm.

Algorithm 3 Distributed Fuzzy Counting

Require: **TS** split into Z *Block*$_z$. $\mathbf{P}_f$ be the fuzzy partition defined in X_f
1: **procedure** MAPPER(**in**: *Block*$_z$, $\mathbf{P}_f$)
2: **for each** object $\mathbf{o}_r = (\mathbf{x}_r, y_r)$ **in** *Block*$_z$ **do**
3: **for each** attribute X_f **do**
4: output $\langle$*key*:$A_{f,j}$, *value*:$\mu_{f,j}(x_{f,r})\rangle$
5: **end for**
6: output $\langle$*key*:y_r, *value*:1$\rangle$
7: **end for**
8: **end procedure**
9: **procedure** REDUCER(**in**: $A_{f,j}$, $list(\mu_{f,j}(x_{f,r}))$)
10: fuzzySupp$(A_{f,j}) \leftarrow$ compute fuzzy support according Eq.15
11: output $\langle$*key*:$A_{f,j}$, *value*: fuzzySupp$(A_{f,j})\rangle$
12: **end procedure**

Algorithm 4 Distributed Fuzzy FP-Growth

Require: **TS** split into Z *Block*$_z$, fs_{list}
1: **procedure** MAPPER(**in**: *Block*$_z$, fs_{list})
2: **for each** object $\mathbf{o}_r = (\mathbf{x}_r, y_r)$ **in** *Block*$_z$ **do**
3: $\widetilde{\mathbf{x}}_r \leftarrow$ create empty list
4: **for each** attribute X_f **do**
5: $\widetilde{A}_{f,j} \leftarrow$ get fuzzy sets with the highest matching degree for $x_{f,r}$
6: **if** $\widetilde{A}_{f,j}$ is in fs_{list} **then**
7: $\widetilde{\mathbf{x}}_r \leftarrow$ insert $\widetilde{A}_{f,j}$
8: **end if**
9: **end for**
10: $\widetilde{\mathbf{x}}_r \leftarrow$ sort $\widetilde{\mathbf{x}}_r$ according to fs_{list}
11: **for** $p \leftarrow 0$ to $|\widetilde{\mathbf{x}}_r| - 1$ **do**
12: output $\langle$*key*:$\widetilde{x}_r[p]$, *value*:$(\{\widetilde{x}_r[0], \ldots, \widetilde{x}_r[p]\}, y_r)\rangle$
13: **end for**
14: **end for**
15: **end procedure**
16: **procedure** REDUCER(**in**: $A_{f,j}$, $\mathbf{TS}_{f,j}$)
17: $FPTree_{f,j} \leftarrow$ build tree from $\mathbf{TS}_{f,j}$
18: $FCAR_{list} \leftarrow$ FPGROWTH($FPTree_{f,j}$)
19: **for each** $FCAR_m$ **in** $FCAR_{list}$ **do**
20: output $\langle$*key*:$null$, *value*:$FCAR_m\rangle$
21: **end for**
22: **end procedure**

$f = F+1$, supports for class values are produced as well. Space and time complexities are both $O((F+1) \cdot N/Z)$.

At the end of the first MapReduce phase, the algorithm selects only the fuzzy sets whose support is larger than threshold $minSupp$, and stores them in fs_{list} ordered for descending fuzzy support. Only frequent fuzzy sets in fs_{list} will be subsequently considered. Since fs_{list} is generally small, this step can efficiently be performed on a single machine.

The second MapReduce phase, *Distributed Fuzzy FP-Growth*, mines fuzzy CARs whose support, confidence and χ^2 values are higher than *minSupp*, *minConf* and $min\chi^2$ thresholds, respectively. The approach modifies PFP-Growth [30] by extending the concepts of projected dataset and projected object to the fuzzy context. Indeed, each mapper computes the *item projected objects* so that the reducers are able to build the *item-projected datasets*, and then to generate local conditional FP-Trees. In the following the $A_{f,j}$*-projected dataset* will be indicated as $\mathbf{TS}_{f,j}$. Since the local conditional FP-Trees are independent of each other, FCARs can be mined by each node independently of the others.

Algorithm 4 accounts for the structure of the *Distributed Fuzzy FP-Growth* phase. As in the first phase, each *Mapper*$_z$ reads all objects in *Block*$_z$, building for each of them the corresponding fuzzy object as described in Section II-C. For each value $x_{f,r}$, the mapper finds out the fuzzy set $A_{f,j}$ with the highest matching degree (line 5) out of the two activated by $x_{f,r}$; fuzzy sets not present in fs_{list} are discarded. Then, *Mapper*$_z$ sorts the fuzzy sets according to the fs_{list} (line 10), retrieves the class label C_{l_r} and, finally, for each extracted fuzzy set outputs the key-value pair $\langle$*key*:$\widetilde{x}_r[p]$, *value*:$(\{\widetilde{x}_r[0], \ldots, \widetilde{x}_r[p]\}, y_r)\rangle$, where $(\{\widetilde{x}_r[0], \ldots, \widetilde{x}_r[p]\}, y_r)$ is the $\widetilde{x}_r[p]$-projected object, that is, $A_{f,j}$-projected object.

The MapReduce framework groups up all the projected objects with the same item; each reducer receives as input the key-value pairs $\langle$*key*:$A_{f,j}$, *value*:$\mathbf{TS}_{f,j}\rangle$, builds the $A_{f,j}$-conditional FP-Tree, say $FPTree_{f,j}$ (line 17), and recursively mines the FCARs [12] (line 18). As in AC-FFP, if a node already exists in the tree, the relative counter is just incremented by 1 and no other information about the matching degrees is kept. It is worth noting that the *FPGrowth* function considers only FCARs whose support, confidence and χ^2 values exceed the relative thresholds. Finally, for each extracted rule, the reducer outputs $\langle$*key*:$null$, *value*:$FCAR_m\rangle$ pairs, where $FCAR_m$ is the m-th in $FCAR_{list}$.

The main contribution to the complexity of the overall phase is due to the reduce step. The number of pairs $\langle key, list(values)\rangle$ processed by each reducer is determined

by the default partition function $hash(key)$ **mod** R, which tries to evenly distribute the jobs by making each reducer to build $\lceil \frac{|fs_{list}|}{R} \rceil$ local conditional FP-Trees, where $|fs_{list}|$ is the number of frequent items. However, this strategy does not necessarily guarantee a perfect load balancing. In fact, the time complexity for rule mining out of a single FP-tree is exponential on the longest frequent path in such a tree [31]. In case of a large number of frequent items, the conditional FP-trees corresponding to the items with the smallest support are very deep, since they consider in their paths almost all the frequent items. Thus, datasets involving a small number of attributes can be easily managed, but conversely runtimes may excessively increase as the number of attributes becomes very large. Let $|FPTree_{f,j}|$ be the space required to store the $A_{f,j}$-conditional FP-Tree, and $|FPGrowth_{f,j}|$ the time necessary to mine rules from the $A_{f,j}$-conditional FP-Tree, then space and time complexities of each reducer are $O\left(\left\lceil \frac{|fs_{list}|}{R} \right\rceil \cdot |FPTree_{f,j}|\right)$ and $O\left(\left\lceil \frac{|fs_{list}|}{R} \right\rceil \cdot |FPGrowth_{f,j}|\right)$, respectively.

A pre-pruning strategy is used to speed up the FCAR mining process in dealing with big data, by avoiding the inspection of sub-trees that likely lead to redundant FCARs. Some preliminary definitions are needed; a rule $FCAR_m$ is *more significant* than $FCAR_s$ if and only if:

$$\begin{aligned} &1.\ \textit{conf}(FCAR_m) > \textit{conf}(FCAR_s) \\ &2.\ \textit{conf}(FCAR_m) = \textit{conf}(FCAR_s)\ \textbf{AND} \\ &\quad\ \textit{supp}(FCAR_m) > \textit{supp}(FCAR_s) \\ &3.\ \textit{conf}(FCAR_m) = \textit{conf}(FCAR_s)\ \textbf{AND} \\ &\quad\ \textit{supp}(FCAR_m) = \textit{supp}(FCAR_s)\ \textbf{AND} \\ &\quad\ \textit{RL}(FCAR_m) < \textit{RL}(FCAR_s) \end{aligned} \tag{16}$$

where $\textit{conf}(\cdot)$, $\textit{supp}(\cdot)$, and $\textit{RL}(\cdot)$ indicate confidence, support, and rule length. A rule $FCAR_m$: $\textit{FAnt}_m \rightarrow C_{l_m}$ is *more general* than $FCAR_s$: $\textit{FAnt}_s \rightarrow C_{l_s}$, if and only if $\textit{FAnt}_m \subseteq \textit{FAnt}_s$. A fuzzy rule $FCAR_s$ is pruned in case there exists an $FCAR_m$ more significant and more general than $FCAR_s$.

The rules mined by the FP-Growth algorithm (line 18) are more and more specialized as the recursive visit of an $A_{f,j}$-conditional FP-tree goes deeper. We stop the visit of an FP-tree at a specific node ι if, visiting two further nodes, the rules generated at such nodes are not more significant than the rule generated at node ι. Let us assume that $FCAR_m$: $\textit{FAnt}_m \rightarrow C_{l_m}$, $FCAR_s$: $\textit{FAnt}_s \rightarrow C_{l_s}$, and $FCAR_d$: $\textit{FAnt}_d \rightarrow C_{l_d}$, with $\textit{FAnt}_s = \{\textit{FAnt}_m, \widetilde{A}_{f,j}\}$ and $\textit{FAnt}_d = \{\textit{FAnt}_s, \widehat{A}_{f,j}\}$, where $\widetilde{A}_{f,j}$ and $\widehat{A}_{f,j}$ are frequent items for the $\textit{FAnt}_s$ and $\textit{FAnt}_d$ antecedents, respectively. We stop the generation of other rules with sub-pattern $\textit{FAnt}_d$ if and only if:

$$\begin{aligned} &\textit{conf}(FCAR_d) - \textit{conf}(FCAR_s) \leq 1 - \tfrac{\textit{supp}(FCAR_d)}{\textit{supp}(FCAR_s)} \\ &\qquad\qquad\qquad\qquad\qquad\qquad \textbf{AND} \\ &\textit{conf}(FCAR_s) - \textit{conf}(FCAR_m) \leq 1 - \tfrac{\textit{supp}(FCAR_s)}{\textit{supp}(FCAR_m)} \end{aligned} \tag{17}$$

The pre-pruning method does not evaluate Eq. 17 for rules with one or two antecedents, and for those generated from $\textit{FAnt}_d$ whose class is different from $FCAR_d$. In this way, we avoid to visit the FP-tree paths likely leading to redundant rules. Notably, such paths would be pruned in the next steps anyway. This modification of FP-Growth let us significantly reduce the execution time of the mining process and, thus, let us reduce the $minSupp$ threshold. In this way, highly confident and specialized rules with a small support can be generated. In our experiments, we have verified that this pre-pruning yields a good trade-off between execution time and accuracy of the generated classifiers.

The next MapReduce phase, named *Distributed Rule Selection*, is aimed at keeping sufficiently low the number of rules in the final rule base, by selecting only the top K non-redundant rules for each class label C_l. Each mapper is fed the block containing the extracted rules, and outputs a pair $\langle \textit{key}{:}C_{l_m}, \textit{value}{:}FCAR_m \rangle$, where C_{l_m} is the class label associated with $FCAR_m$. Each reducer processes all the rules with the same class label, say $list(FCAR_{C_l})$, and selects the best K significant non-redundant rules. For each of these K rules the reducer outputs a pair $\langle \textit{key}{:}null, \textit{value}{:}FCAR_m \rangle$; thus, the process outputs at most $K \cdot L$ rules. Space and time complexities of each reducer are $O(K)$ and $O(\lceil \frac{L}{q} \rceil \cdot |FCAR_{C_l}| \cdot \log(K))$, where $|FCAR_{C_l}|$ is the number of generated rules by the *Distributed Fuzzy FP-Growth* phase for C_l.

C. Distributed Pruning

The mined FCARs are pruned (see Figure 5) mainly to reduce noisy information. At this stage, **TS** has to be scanned two further times.

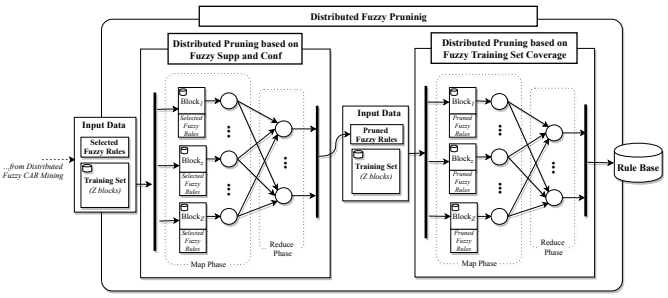

Fig. 5. The overall FCAR Pruning process of the DFAC-FFP algorithm.

The first scan is aimed to check whether, for each rule $FCAR_m$, the relative confidence and fuzzy support are lower than the *minConf* and $\textit{minFuzzySupp}_m$ thresholds, where $\textit{minFuzzySupp}_m$ is calculated as

$$\textit{minFuzzySupp}_m = \textit{minSupp} \cdot 0.5^{g_m - 1} \tag{18}$$

where *minSupp* is the minimum support determined by the expert, and g_m is the rule length of $FCAR_m$. In practice, the $\textit{minFuzzySupp}_m$ threshold corresponds to *minSupp* adapted to the number of conditions, so as to take into account the effect of the product t-norm as conjunction operator. The pseudocode for these operations is reported in Algorithm 5.

As usual, each $\textit{Mapper}_z$ is fed $\textit{Block}_z$ of **TS**. Moreover, each mapper loads the rules previously mined, rank them, and computes the matching degree $w_m(\mathbf{x}_r)$ of each rule $FCAR_m$ for $\mathbf{x}_r$ according to Eq. 3. We recall that in our experiments we have exploited the product as t-norm for implementing the conjunction operator. For each $FCAR_m$ with matching degree > 0, the mapper outputs the pair $\langle \textit{key}{:}m, \textit{value}{:}w_m(\mathbf{x}_r) \rangle$,

Algorithm 5 Distributed Pruning based on Fuzzy Support and Confidence

Require: **TS** split into Z $Block_z$, $FCAR_{list}$ be a list of rules sorted according to Eq. 16
1: **procedure** MAPPER(**in**: $Block_z$, $FCAR_{list}$)
2: **for each** object $\mathbf{o}_r = (\mathbf{x}_r, y_r)$ **in** $Block_z$ **do**
3: **for each** $FCAR_m$ **in** $FCAR_{list}$ **do**
4: $w_m(\mathbf{x}_r) \leftarrow$ compute matching deg as per Eq. 3
5: **if** $y_r == C_{l_m}$ **then**
6: output $\langle$*key*:m, *value*:$w_m(\mathbf{x}_r)\rangle$
7: **else**
8: output $\langle$*key*:m, *value*: $-w_m(\mathbf{x}_r)\rangle$
9: **end if**
10: **end for**
11: **end for**
12: **end procedure**
13: **procedure** REDUCER(**in**: m, $list(w_m)$, $FCAR_{list}$)
14: $FCAR_m \leftarrow$ update support according to $list(w_m)$ and Eq. 4
15: $FCAR_m \leftarrow$ update confidence according to $list(w_m)$ and Eq. 5
16: **if** $FCAR_m$ is not pruned **then**
17: output $\langle$*key*:$null$, *value*:$FCAR_m\rangle$
18: **end if**
19: **end procedure**

Algorithm 6 Distributed Pruning based on Fuzzy Training Set Coverage

Require: **TS** split into Z $Block_z$, $FCAR_{list}$ be a list of rules sorted according to Eq. 16
1: **procedure** MAPPER(**in**: $Block_z$, $FCAR_{list}$)
2: **for each** object $\mathbf{o}_r = (\mathbf{x}_r, y_r)$ **in** $Block_z$ **do**
3: $matchesCount \leftarrow 0$
4: **for each** $FCAR_m$ **in** $FCAR_{list}$ **do**
5: $w_m(\mathbf{x}_r) \leftarrow$ compute matching deg as per Eq. 3
6: **if** $w_m(\mathbf{x}_r) \geq \overline{w}_m$ **then**
7: $matchesCount \leftarrow matchesCount + 1$
8: **if** $C_{l_m} == y_r$ **then**
9: output $\langle$*key*:m, *value*:$null\rangle$
10: **end if**
11: **end if**
12: **if** $matchesCount \geq \delta$ **then**
13: **break**
14: **end if**
15: **end for**
16: **end for**
17: **end procedure**
18: **procedure** REDUCER(**in**: m , $FCAR_{list}$)
19: $FCAR_m \leftarrow$ get the m-th rule from $FCAR_{list}$
20: output $\langle$*key*:$null$, *value*:$FCAR_m\rangle$
21: **end procedure**

where m corresponds to the index of $FCAR_m$ in $FCAR_{list}$. Note that since $FCAR_{list}$ is sorted according to the Eq. 16, m corresponds also to the rank of the rule. When the class label C_{l_m} of rule $FCAR_m$ is not y_r, $Mapper_z$ outputs the negative value $-w_m(\mathbf{x}_r)$ (line 5), thus enabling the downstream reducer to properly compute the fuzzy support and the fuzzy confidence. More precisely, each reducer takes a pair $\langle$*key*:m, *value*:$list(w_m)\rangle$, where $list(w_m)$ is the list of all non-zero matching degrees for $FCAR_m$. The fuzzy support and the fuzzy confidence are calculated according to Equations 4 and 5, where the numerators of both equations are computed by considering only the positive w_m, and the denominator of the second one is computed by considering the absolute value of each element in $list(w_m)$. Finally, each reducer produces (line 17) only FCARs with confidence and support higher than *minConf* and *minFuzzySupp*, respectively (line 16). Space and time complexities of each *map-task* are $O(\lceil \frac{N}{Z} \rceil \cdot K \cdot L)$.

The last step of the pruning process is a training set coverage analysis. Algorithm 6 shows the pseudo code of *Distributed Pruning based on Fuzzy Training Set Coverage* phase.

$Mapper_z$ loads both the associated $Block_z$ and the filtered rule set just generated. We underline that rules are ranked and placed in $FCAR_{list}$ in descending order according to their fuzzy confidence, fuzzy support, and length. Moreover, like in AC-FFP, each object $\mathbf{o}_r$ is associated with a counter of FCAR matches. The mapper scans the rule list to determine, for each object $\mathbf{o}_r$, how many FCARs are matched on $\mathbf{x}_r$; only rules whose matching degree is higher than the *fuzzy matching degree threshold* $\overline{w}_m = 0.5^{g_m - 1}$, where g_m is the rule length of $FCAR_m$, are taken into account (line 6). If $FCAR_m$ correctly classifies $\mathbf{x}_r$ (line 8), the mapper outputs a pair $\langle$*key*:m, *value*:$null\rangle$, where the key is the rank of the $FCAR_m$. When the counter of FCAR matches is higher than a *coverage threshold* δ, the corresponding object is not considered anymore. The key-value pair $\langle$*key*:m, *value*:$null\rangle$ is used to feed the reducer, which gets the rule from the ranked list and outputs $\langle$*key*:$null$, *value*:$FCAR_m\rangle$. Finally, the rules survived to the last pruning steps are inserted into the rule base and exploited to classify new unlabeled patterns. Space and time complexities of $Mapper_z$ are $O(\lceil \frac{N}{Z} \rceil \cdot K \cdot L)$.

D. Classification stage

The distributed approach described in the previous sections generates a set of FCARs and, for all the continuous attributes, the fuzzy partitions to be used in the fuzzy rules. In the classification phase, generally multiple rules are activated by an unlabeled instance, and a proper reasoning method to choose a unique classification label has to be adopted. In the experiments discussed here later, we employ both the *weighted vote* and the *maximum matching* reasoning methods. Since we adopt the product t-norm as conjunction operator, rules with a higher number of conditions in the antecedent generally have a lower matching degree than rules with a lower number of conditions. Hence, more general rules are more influential than specific ones. To re-balance the influence of rules, for a given input pattern $\widehat{\mathbf{x}} = [\hat{x}_1 \ldots, \hat{x}_F]$ and a given rule $FCAR_m$, we compute a *normalized matching degree*

$$\widehat{w}_m(\widehat{\mathbf{x}}) = w_m(\widehat{\mathbf{x}}) \cdot 2^{g_m} \tag{19}$$

where g_m is the rule length of $FCAR_m$. In case $\widehat{\mathbf{x}}$ activates no rule, the input pattern is classified as *unknown*.

The quality of fuzzy association rules is mainly driven by the confidence value. Even though both the maximum matching and the weighted vote methods employ such a value as certainty factor, rules with high confidence might not be properly rewarded. For instance, the maximum matching tends to select rules characterized by higher association degrees, and the weighted vote considers the contribution of each rule, low-confidence rules included. On the other hand, a rule with 100% confidence should be always selected for the classification, independently of the vote/or the activation degree of the other rules; rules of this kind are very representative of the training

set, because in the learning phase their activation always corresponds to a correct classification on their behalf.

To hopefully improve the accuracy of our fuzzy associative classifier, yet another reasoning method named *best rule* has been tested. Rules are sorted in decreasing order according to their confidence, support, and length. For an unlabeled pattern $\widehat{\mathbf{x}}$, we select the class label $\widehat{C}_l$ indicated by the first rule with $\widehat{w}_m(\widehat{\mathbf{x}})$ greater than a given threshold.

In our experiments, we adopted both 1 and 0 as threshold values. With the first threshold, a rule is considered only when on average the membership degree of each fuzzy set in the rule antecedent is ≥ 0.5; this therefore helps in filtering out the rules not representative for $\widehat{\mathbf{x}}$. On the other hand, threshold 0 allows considering any rule activated by instance $\widehat{\mathbf{x}}$.

IV. Experimental Study

To characterize the behavior of the proposed approach for generating fuzzy associative classifiers for big data, the experimental study focuses on two aspects. First, we investigate the performance of DFAC-FFP in terms of classification accuracy, model complexity, and computation time. Then, we carry out a general scalability analysis, aimed at showing how the characteristics of the underlying hardware platform actually affect the algorithm runtime and thus its practical use with real big datasets.

We tested our algorithm on six well-known big datasets, extracted from the UCI[5] and LIBSVM[6] repositories. As summarized in Table I, datasets are characterized by different numbers of input/output instances (up to 11,000,000), classes (from 2 to 23), and attributes (up to 54)[7]. For each dataset, we report the number of numeric (N) and categorical (C) attributes. For all algorithms, a five-fold cross-validation, using the same data partitions, has been carried out.

TABLE I
BIG DATASETS USED IN EXPERIMENTS

Dataset	# Instances	# Attributes	# Classes
Cover Type (COV)	581,012	54 (10 N, 44 C)	2
HIGGS (HIG)	11,000,000	28 (28 N)	2
KDDCup 1999 (KDD99)	4,898,431	41 (26N, 15C)	23
KDD99 restricted to 2 Classes (KDD99_2)	4,856,151	41 (26N, 15C)	2
KDD99 restricted to 5 Classes (KDD99_5)	4,898,431	41 (26N, 15C)	5
Susy (SUS)	5,000,000	18 (18 N)	2

All the experiments have been run over a typical low-end system testbed for supporting the target classification service: a small cluster comprising one master node with a 4-core CPU (Intel Core i5 CPU 750 x 2.67 GHz), 4 GB of RAM and a 500GB Hard Drive, and four slave nodes each with 4-core CPU with Hyperthreading (Intel Core i7-2600K CPU x 3.40 GHz), 16GB of RAM and 1 TB Hard Drive. All the nodes are connected by a Gigabit Ethernet (1 Gbps) and run Ubuntu 12.04. The algorithm has been deployed upon Apache Hadoop 1.0.4: the master hosts the *NameNode* and *JobTracker* processes, while each slave runs a *DataNode* and a *TaskTracker*.

[5]Available at https://archive.ics.uci.edu/ml/datasets.html

[6]Available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[7]KDDCup contains a wide variety of intrusions/attacks simulated in a military network environment: each connection (an instance of the dataset) is labeled either as one out of 22 different attacks/intrusion (bad connection) or as "No attacks" (normal connection). Attacks fall into four categories: DOS, R2L, U2R, and probing. KDD99_5 is built from KDD99 by employing such four categories, plus the normal connection. KDD99_2 is built from KDD99 by filtering just the instances labeled with the two most frequent categories, namely "Normal" and "DOS".

A. Performance Analysis of DFAC-FFP

In this section, we first show the results obtained by DFAC-FFP. We have found that the DFAC-FFP accuracy is comparable with those achieved by recent state-of-the-art similar distributed associative classifiers; a discussion on the results achieved by MRAC and MRAC+ [15] will substantiate this claim.

MRAC is a distributed extension of the well-known CMAR [32] algorithm and its learning procedure consists of three main phases: 1) discretization is carried out through a MapReduce implementation of the approach proposed by Fayyad and Irani [17]; 2) a parallel, specialized version of the FP-growth algorithm [30] extracts significant, frequent CARs; 3) the final rule base is obtained upon the application of two rule pruning steps. In classification, MRAC uses the weighted chi-squared as reasoning method. MRAC+ represents an enhanced version of MRAC, intended to improve both accuracy and execution time; as reasoning mechanism, MRAC+ employs a crisp version of the best rule. Both algorithms have shown to be very effective in comparison with two distributed implementations of well-known classifiers, namely Decision Tree and Random Forest. In particular, the experimental study shows that MRAC+ outperforms the Decision Tree and achieves comparable results with Random Forest in terms of accuracy and execution time.

Table II summarizes the parameters used for learning the three associative classifiers. Results for MRAC and MRAC+ have been taken from the related paper [15]. The $MinConf$ value guarantees that rules with the same antecedent but different consequents would not co-exist in the final rule base.

TABLE II
PARAMETER VALUES FOR ALGORITHMS USED IN EXPERIMENTS

Method	Parameters
DFAC-FFP	$\gamma = 0.1\%, \phi = 2\%, MinSupp = 0.01\%$
	$MinConf = 50\%, min\chi^2 = 20\%, K = 15000, \delta = 4$
MRAC+	$\gamma = 0.1\%, \phi = 2\%, MinSupp = 0.01\%$
	$MinConf = 50\%, min\chi^2 = 20\%, K = 15000$
MRAC	$\gamma = 0.1\%, \phi = 2\%, MinSupp = 0.01\%$
	$MinConf = 50\%, min\chi^2 = 20\%, K = 15000, \delta = 4$

Table III reports, for each dataset and for each algorithm, the average accuracy $\pm$ its standard deviation, both on the training (Acc_{Tr}) and test (Acc_{Ts}) sets. The highest accuracy values for each dataset are shown in bold. Regarding DFAC-FFP, we also show, for each dataset, the average number of fuzzy sets (FS) generated by the distributed fuzzy partitioning method for the continuous attributes. Moreover, we report also the results for each of four experimented reasoning methods (see Section II-A); here WV, MM, $BFR(0)$, and $BFR(1)$ stand for *Weighting Vote*, *Maximum Matching*, *Best Rule* with no threshold and *Best Rule* with threshold equal to 1, respectively.

TABLE III
AVERAGE ACCURACY ± STANDARD DEVIATION ACHIEVED BY DFAC-FFP, MRAC+ AND MRAC

			DFAC-FFP		**MRAC+**		**MRAC**	
Dataset	**FS**	**Reasoning Method**	Acc_{Tr}	Acc_{Ts}	Acc_{Tr}	Acc_{Ts}	Acc_{Tr}	Acc_{Ts}
COV	10.18	WV	77.227 ± 0.043	77.190 ± 0.212	$\mathbf{78.329} \pm 0.091$	$\mathbf{78.092} \pm 0.157$	74.246 ± 0.100	74.261 ± 0.156
		MM	75.194 ± 0.045	75.052 ± 0.207				
		BFR (0)	75.550 ± 0.059	75.519 ± 0.170				
		BFR (1)	76.943 ± 0.124	76.877 ± 0.174				
HIG	10.22	WV	$\mathbf{66.019} \pm 0.062$	$\mathbf{66.005} \pm 0.078$	65.942 ± 0.058	65.904 ± 0.045	65.079 ± 0.054	65.050 ± 0.061
		MM	63.486 ± 0.066	63.472 ± 0.066				
		BFR (0)	65.514 ± 0.033	65.507 ± 0.022				
		BFR (1)	65.738 ± 0.036	65.733 ± 0.023				
KDD99_2	2.65	WV	99.987 ± 0.011	99.986 ± 0.011	$\mathbf{99.999} \pm 0.000$	$\mathbf{99.998} \pm 0.000$	99.959 ± 0.002	99.957 ± 0.004
		MM	99.997 ± 0.000	99.997 ± 0.001				
		BFR (0)	99.998 ± 0.000	$\mathbf{99.998} \pm 0.001$				
		BFR (1)	$\mathbf{99.999} \pm 0.001$	$\mathbf{99.998} \pm 0.001$				
KDD99_5	3.30	WV	99.935 ± 0.029	99.935 ± 0.031	99.863 ± 0.046	99.858 ± 0.047	99.898 ± 0.034	99.898 ± 0.035
		MM	99.915 ± 0.054	99.914 ± 0.054				
		BFR (0)	$\mathbf{99.941} \pm 0.031$	$\mathbf{99.939} \pm 0.032$				
		BFR (1)	$\mathbf{99.941} \pm 0.031$	$\mathbf{99.939} \pm 0.032$				
KDD99	3.26	WV	99.839 ± 0.123	99.838 ± 0.123	99.582 ± 0.020	99.579 ± 0.020	99.640 ± 0.024	99.639 ± 0.024
		MM	99.845 ± 0.134	99.843 ± 0.135				
		BFR (0)	$\mathbf{99.887} \pm 0.150$	$\mathbf{99.886} \pm 0.150$				
		BFR (1)	$\mathbf{99.887} \pm 0.152$	$\mathbf{99.886} \pm 0.150$				
SUS	13.98	WV	77.728 ± 0.022	77.716 ± 0.039	78.247 ± 0.013	78.220 ± 0.035	76.245 ± 0.055	76.232 ± 0.068
		MM	76.670 ± 0.025	76.664 ± 0.037				
		BFR (0)	77.526 ± 0.038	77.528 ± 0.076				
		BFR (1)	$\mathbf{78.274} \pm 0.004$	$\mathbf{78.267} \pm 0.050$				

Table III shows that, for the different versions of the KDD dataset, accuracy is not influenced very much by the reasoning method. Indeed, for these datasets all the rules have a confidence higher than 99.6%, thus, regardless of the used reasoning method, more or less the same rules are always chosen for the actual classification. Regarding the SUS dataset, the best result is achieved with the BRF(1) reasoning method: this is due to the fact that the generated rule bases contain, on average, 28% and 64.6% of rules with confidence higher than 90% and 80%, respectively. Finally, considering the COV and HIG datasets, the highest accuracies are obtained with the WV reasoning method; in these cases, the number of rules with very high confidence is limited, and most of the rules are not able to satisfactorily represent the training set. Thus, a reasoning method that relies on all the rules is more suitable than an approach that considers only one rule for classification.

On the basis of the aforementioned results, we can suggest to use the BRF reasoning method whenever a significant portion of high-confidence rules are present in the final rule base. If most of the rules are high-confidence ones, the chosen reasoning method does not affect the outcomes. In all the other cases, the weighted vote should be preferred.

As regards the accuracy level, Table III highlights that, on average, DFAC-FFP outperforms MRAC in all datasets, both on training and test sets. On the other hand, DFAC-FFP and MRAC+ achieve similar accuracies on 4 datasets, while on COV dataset MRAC+ performs better than DFAC-FFP and on HIG dataset DFAC-FFP with WV outperforms MRAC+. In general, we can state that the proposed method delivers similar to or better accuracies than two recent associative classifiers for big data. This is mainly due to the ability of fuzzy sets to manage uncertainty.

The DFAC-FFP results have also been compared to those achieved by Chi-FRBCS-BigData [18], which is the state-of-the-art algorithm for generating fuzzy rule-based classifiers for big data. The average accuracies on the test set achieved by this algorithm are 51.25, 55.89, 99.93, 96.30, 99.61 and 55.75 for the COV, HIG, KDD99_2, KDD99_5, KDD99 and SUS datasets, respectively. The proposed DFAC-FFP clearly outperforms Chi-FRBCS-BigData.

Before discussing other aspects, we would like to highlight the role of the proposed discretizer stage in getting the reported performance figures. Using the SUS dataset, we observed that substituting this stage with procedures, which create uniform fuzzy partitions with 3 and 5 fuzzy sets, leads to accuracies always more than 9% worse than those reported in Table III. Moreover, procedures, which generate the fuzzy partitions by adopting a crisp version of the proposed discretizer and place a triangular fuzzy set in each cut point, obtain accuracies always more than 1.2% worse than those in Table III.

By analyzing the results shown in Table IV, we realize that the complexities, expressed in terms of average number of rules of the models generated employing the DFAC-FFP are much lower than the complexities associated with both MRAC+ and MRAC. In particular, the number of rules of DFAC-FFP is typically one order of magnitude lower than MRAC+ and of the same order of magnitude of MRAC. In conclusion, the fuzzy associative classifiers generated by the proposed approach are characterized by a lower number of parameters (rules) than the ones generated by the two recent algorithms, which generate crisp rule-based classifiers. On the other hand, even though DFAC-FFP employs a lower number of rules than both MRAC and MRAC+, the generated models are not still very interpretable. However, since the activated rules usually contain a few conditions (at most four or five on average for all datasets), we can affirm that a reasoning method such as the "best rule," which considers one single rule, can provide a very interpretable explanation for each conclusion. Thus, even though the number of rules is still quite large in the final rule base, we can consider that, for each unlabeled

pattern, the classifier can provide a very intuitive justification of its reasoning.

TABLE IV
AVERAGE NUMBER OF ASSOCIATION RULES FOR DFAC-FFP, MRAC+ AND MRAC

Dataset	DFAC-FFP	MRAC+	MRAC
COV	2,646	15,612	6,714
HIG	9,365	29,999	19,468
KDD99_2	890	30,000	1,174
KDD99_5	2,419	49,349	2,878
KDD99	2,347	125,294	2,806
SUS	10,970	30,000	21,963

Table V summarizes the computation times (in seconds) on a cluster of four slaves with the same Hadoop configuration for all the algorithms: by default, the number of mappers and reducers is set equal to the available cores on the cluster. Because of memory constraints, in all the algorithms the number of reducers for the *Parallel FP-Growth* is set to 4 and, only for HIG, is set to 2. The number of HDFS blocks (Z) and instances per block (N_{Block}) is reported as well. As shown in Table V, DFAC-FFP is slower than the others; this is principally due to the additional pruning step in DFAC-FFP.

TABLE V
COMPUTATION TIMES (IN SECONDS) FOR THE LEARNING PROCESS IN DFAC-FFP, MRAC+, AND MRAC

Dataset	Z	N_{Block}	DFAC-FFP	MRAC+	MRAC
COV	1	464,809	1,472	516	1,056
HIG	96	91,667	24,077	4,660	7,382
KDD99_2	6	47,487	9,954	657	702
KDD99_5	6	653,124	23,508	1,314	1,615
KDD99	6	653,124	70,925	1,500	2,125
SUS	29	137,932	11,132	575	3,655

B. Scalability analysis

In this section, the proposed algorithm is analysed in terms of its ability to take advantage of multiple nodes in a distributed deployment, i.e. evaluating its *horizontal* scalability [33]. Such a study is needed to assess how the proposed approach can exploit a cluster of commodity hardware. The study is focused on the impact of using q distinct Computing Units (CUs) on global runtimes. In principle, such CUs can belong to different nodes, or can be placed on the same node of a cluster of computers.

Indeed, the hardware resources at each node can be boosted up in the attempt to improve the global runtime. It has been shown that *vertical* scalability does not play a primary role in the context of platforms for big data analytics [33]: variations of the processor clock rate in the range of typical modern machines do not change the order of magnitude of algorithm runtimes, as well as other possible local enhancements, like using GPUs or dedicated FPGAs. Notably, in scalability analysis the presence of hyper-threading in CPUs can be roughly accounted by CU-based results. The per-node memory availability affects the actual algorithm workability: we just assume that each node has a sufficient amount of memory to support the local execution.

A precise runtime characterization can be useful to spot out possible system bottlenecks. For this reason, we investigate also how the *different phases* affect the performance of the overall algorithm.

The prime measure used to assess scalability is the classical *speedup* index σ, which compares the runtime of the parallel implementation to the corresponding sequential version. Operating with large-sized datasets, the sequential implementation of the algorithm would take an unreasonable time to be executed. To overcome this drawback, we adopt a slightly different definition [15], taking as reference a run over Q^* identical cores, with $Q^* > 1$. In particular, the speedup formula on n CUs can be redefined as $\sigma_{Q^*}(q) = Q^* \cdot \tau(Q^*)/\tau(q)$, where $\tau(q)$ is the runtime using q CUs, and Q^* is the number of CUs used to run the reference execution. Obviously, $\sigma_{Q^*}(q)$ makes sense only for $q \geq Q^*$, where the speedup is expected to be sub-linear due to the overhead from the Hadoop procedures, the contention of shared resources among cores, and the increased communication times. In our experiments, we have taken $Q^* = 8$ and performed several executions, varying the number of CUs, with $q \geq 8$. To avoid unbalanced loads, we have kept the same number of running cores per node, varying the number of slave nodes in the cluster. Practically, we have recorded the runtimes of each experiment, starting from 1 to 4 slave nodes (from 8 to 32 cores).

Table VI summarizes the results on the SUS dataset. Similar results can be recorded with the other datasets. By default in Hadoop the number Z of mappers is automatically determined by the HDFS block size. Only q mappers can be run simultaneously without preemption, and the rest $(Z-q)$ are queued to wait for the next available CU. Thus, in the ideal case of the same execution time for all the mappers, the map phase for each MapReduce stage would require $\lceil \frac{Z}{q} \rceil$ iterations. In case $Z \leq q$, the global runtime is driven by the slowest mapper. Regarding the SUS dataset, Hadoop instantiates 36 mappers; thus the number of iterations is 5, 3, 2, and 2 over 8, 16, 24, and 32 cores, respectively. The same considerations apply to the reduce phase. In this case, the number R of reducers is user-defined and in our experiments we have assigned R to the number of available cores, with the only exception of the *Distributed Fuzzy FP-Growth* phase where, because of memory limitations, we set R so to have four mappers per machine. Each reducer sequentially processes a set of keys generated during the map phase, and each key is assigned to only one reducer according to the default partitioning function $hash(key)$ **mod** R.

TABLE VI
SPEEDUP $\sigma_8(q)$ OF THE OVERALL ALGORITHM FOR THE SUSY DATASET

q (# CUs)	Time (s)	Speedup	$\sigma_8(q)/q$ (Utilization)
8	50,740	8	1.000
16	27,251	14.896	0.931
24	19,678	20.628	0.860
32	15,482	26.219	0.819

As reported in Table VI, the actual utilization index $\sigma_8(q)/q$ tends to decrease quite rapidly, because of the increasing overhead due to the Hadoop procedures (adding a new slave node implies adding a new *Datanode* and *Tasktracker* as well)

and to the master/slaves communications. However, within the limitations due to the different experimental settings, this result is in line with [30] and MRAC+ [15], where the value of the utilization index are 0.768 and 0.80, respectively.

A better understanding of the scalability performances requires a deeper insight of the contributions from the different MapReduce jobs. To this aim, in Table VII and Figure 6 we report the speedup σ_8 of the fuzzy partitioning and the four most time-consuming jobs, namely *Distributed Fuzzy Partitioning*, *Distributed Fuzzy FP-Growth*, *Distributed Pruning based on Fuzzy supp and Conf*, and *Distributed Training Set Coverage Pruning*. The contribution of the other two phases (*Distributed Fuzzy Counting* and *Candidate Rule Filtering*) is negligible.

The four charts in Figure 6 show different scalability trends for different phases. The speedup of *Distributed Fuzzy Partitioning* rapidly moves away from linearity (see Figure 6*a*), with no significant advantage in passing from 24 to 32 cores. The overall result is mainly driven by the runtime of the last reduce step of the *Distributed Fuzzy Partitioning*. In fact, the level of parallelism is bounded by the number of continuous attributes that actually undergo discretization. Since SUS is characterized by 18 continuous attributes, each reducer processes approximately 3, 2, 1, and 1 attributes in case of 8, 16, 24, and 32 CUs, respectively. The result highlights that, as regards the distribution of the computational flow of the reduce phase, no real advantage comes from using more than 18 cores. On the other hand, the global execution time of the map phases can be effectively shrunk by exploiting a proper number of cores. Further runtime improvements might be obtained by recurring, for the software implementation of this phase, to cluster computing frameworks that better support iterative computations [28].

As in the *Distributed Fuzzy Partitioning* phase, the computational weight of the reducing activity adversely affects the efficiency of the *Distributed Fuzzy FP-Growth* phase (Figure 6*b*). Indeed, the average runtime of each mapper is quite short (a few seconds), and the global execution time is dominated by the mining of FCARs from each conditional FP-Tree. With the Hadoop default settings, all the conditional FP-Trees are evenly distributed among the reducers, and adding more cores helps improve the FP-Growth parallelization by decreasing the number of conditional FP-Trees processed by each reducer. Because of memory constraints, in the *Distributed Fuzzy FP-Growth* phase we set 4 reducers per machine. Hence, since in our tests SUS has 290 frequent items, each reducer processes more or less 73, 37, 25, and 19 FP-Trees in case of 4, 8, 12 and 16 reducers, respectively. These results are very similar to those of the Parallel FP-Growth phase in both MRAC+ and MRAC [15]. On the other hand, both *Distributed Pruning based on Fuzzy Supp and Conf* and *Distributed Training Set Coverage Pruning* (Figures 6*c* and 6*d*, respectively) present a similar behaviour, with satisfactory utilization values. In this case, the overall runtime is dominated by the map phase and the global execution time can be shrunk by exploiting additional cores, as witnessed by the results in Table VII.

V. Conclusion

In this paper, we have presented DFAC-FFP, a distributed fuzzy associative classifier for Big Data based on frequent pattern mining. It has been designed by extending the AC-FFP algorithm, which delivers very accurate results, but with consistent computational complexity and memory occupation. Such problems have been overcome by adopting the MapReduce programming model, which provides a lean, effective, and fault-tolerant means to parallelize the computation flow across clusters of machines. The DFAC-FFP learning algorithm operates in three main phases: 1) definition of a strong fuzzy partition on each attribute, by employing a novel approach based on fuzzy entropy; 2) extraction of frequent $FCAR$s by exploiting a distributed fuzzy version of the FP–Growth algorithm; 3) pruning of redundant rules by applying specific techniques based on fuzzy support, confidence, and distributed training set coverage. Moreover, we have discussed different reasoning methods, defining general guidelines on the most convenient choice for associative classification.

We implemented our approach on the popular Hadoop framework. Experimental results on six big datasets show that DFAC-FFP is able to properly process millions of data for learning the associative classifier. Indeed, considering its scalability characteristics, DFAC-FFP achieves speedup figures close to the ideal ones. Notably, such results can be obtained by recurring to commodity hardware. Furthermore, results obtained by DFAC-FFP have been compared against those achieved on the same hardware platform by MRAC+ and MRAC, two recent associative classifiers. We have concluded that, considering accuracy, DFAC-FFP and MRAC+ achieve comparable classification rates, outperforming MRAC. Runtimes on MRAC+ are shorter than in DFAC-FFP, essentially because DFAC-FFP employs an additional pruning step. From the complexity perspective, the number of rules in DFAC-FFP is one order of magnitude lower than in MRAC+ and of the same order of magnitude of MRAC. Thus, we can state that, although DFAC-FFP and MRAC+ deliver similar accuracies, the number of rules in the classifiers generated by the fuzzy distributed approach is lower than in non-fuzzy classifiers.

As future work, we primarily aim to investigate on possible runtime-effective portings of DFAC-FFP onto memory-efficient frameworks like Apache Spark, so to make the algorithm even more practically usable by the research community. Moreover, we plan to explore approaches to reduce the complexity of the rule base generated by DFAC-FFP, which hampers the interpretability of the classifier. A possible solution is based on a multi-objective evolutionary approach we have proposed in the last years for selecting rules and conditions from an initial rule base. The aim of the approach is to generate compact fuzzy associative classifiers with different trade-offs between accuracy and complexity. We intend to implement this approach in a distributed environment and use the rules generated by DFAC-FFP as the initial rule base.

References

[1] V. Mangat and R. Vig, "Novel associative classifier based on dynamic adaptive PSO: Application to determining candidates for thoracic

TABLE VII
RUNTIME AND SPEEDUP OF EACH DFAC-FFP PHASE IN THE SUSY DATASET.

	Distributed Fuzzy Partitioning			**Distributed Fuzzy FP-Growth**			**Distributed Pruning based on Fuzzy Supp and Conf**			**Distributed Training Set Coverage Pruning**		
q **(# CUs)**	**Time (s)**	**Speedup** σ_8	$\sigma_8(q)/q$	**Time (s)**	**Speedup** σ_8	$\sigma_8(q)/q$	**Time (s)**	**Speedup** σ_8	$\sigma_8(q)/q$	**Time (s)**	**Speedup** σ_8	$\sigma_8(q)/q$
8	787	8	1	3,161	8	1	40,790	8	1	5,557	8	1
16	625	10.074	0.630	1,688	14.981	0.936	21,612	15.099	0.944	2,929	15.178	0.949
24	504	12.492	0.521	1,249	20.247	0.844	15,556	20.977	0.874	2,104	21.129	0.880
32	484	13.008	0.407	987	25.621	0.801	12,137	26.886	0.840	1,641	27.091	0.847

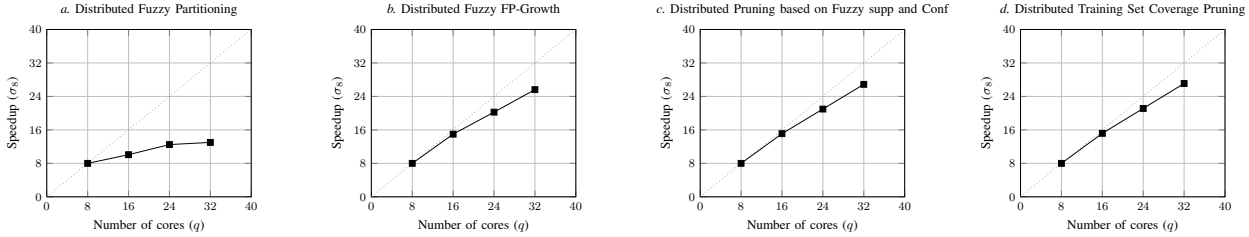

Fig. 6. Speedup of the different phases of DFAC-FFP on SUS dataset for increasing number of cores

surgery," *Expert Systems with Applications*, vol. 41, no. 18, pp. 8234–8244, 2014.

[2] N. Thasleena and S. Varghese, "Enhanced associative classification of XML documents supported by semantic concepts," *Procedia Computer Science*, vol. 46, pp. 194–201, 2015.

[3] E. Baralis and P. Garza, "I-prune: Item selection for associative classification," *International Journal of Intelligent Systems*, vol. 27, no. 3, pp. 279–299, 2012.

[4] N. Abdelhamid, A. Ayesh, F. Thabtah, S. Ahmadi, and W. Hadi, "MAC: A multiclass associative classification algorithm," *Journal of Information & Knowledge Management*, vol. 11, no. 02, 2012.

[5] C. C. Aggarwal, A. M. Bhuiyan, and A. M. Hasan, *Frequent Pattern Mining*. Springer, 2014, ch. Frequent Pattern Mining Algorithms: A Survey, pp. 19–64.

[6] S. Garcia, J. Luengo, J. A. Sáez, V. López, and F. Herrera, "A survey of discretization techniques: taxonomy and empirical analysis in supervised learning," *Knowledge and Data Engineering, IEEE Trans. on*, vol. 25, no. 4, pp. 734–750, 2013.

[7] Y. Ma, G. Chen, and Q. Wei, "A novel business analytics approach and case study–fuzzy associative classifier based on information gain and rule-covering," *Journal of Management Analytics*, vol. 1, no. 1, pp. 1–19, 2014.

[8] J. Alcala-Fdez, R. Alcala, and F. Herrera, "A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning," *IEEE Trans. on Fuzzy Systems*, vol. 19, no. 5, pp. 857–872, 2011.

[9] M. Fazzolari, R. Alcalá, and F. Herrera, "A multi-objective evolutionary method for learning granularities based on fuzzy discretization to improve the accuracy-complexity trade-off of fuzzy rule-based classification systems: D-MOFARC algorithm," *Applied Soft Computing*, vol. 24, pp. 470–481, 2014.

[10] M. Antonelli, P. Ducange, F. Marcelloni, and A. Segatori, "A novel associative classification model based on a fuzzy frequent pattern mining algorithm," *Expert Systems with Applications*, vol. 42, no. 4, pp. 2086–2097, 2015.

[11] M. Elkano, M. Galar, J. Sanz, A. Fernández, E. Barrenechea, F. Herrera, and H. Bustince, "Enhancing multi-class classification in FARC-HD fuzzy classifier: On the synergy between n-dimensional overlap functions and decomposition strategies," *IEEE Trans. on Fuzzy Systems*, vol. 23, no. 5, pp. 1562–1580, 2015.

[12] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *SIGMOD Rec.*, vol. 29, no. 2. ACM, 2000, pp. 1–12.

[13] L. T. Nguyen, B. Vo, T.-P. Hong, and H. C. Thanh, "CAR-Miner: An efficient algorithm for mining class-association rules," *Expert Systems with Applications*, vol. 40, no. 6, pp. 2305–2311, 2013.

[14] M. Xiaofeng and C. Xiang, "Big data management: concepts, techniques and challenges," *Journal of Computer Research and Development*, vol. 1, p. 98, 2013.

[15] A. Bechini, F. Marcelloni, and A. Segatori, "A MapReduce solution for associative classification of big data," *Information Sciences*, vol. 332, pp. 33–55, 2016.

[16] P. Ducange, F. Marcelloni, and A. Segatori, "A MapReduce-based fuzzy associative classifier for big data," in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Aug 2015, pp. 1–8.

[17] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proc. of the 13th Int'l Joint Conf. on Artificial Intelligence*. Morgan-Kaufmann, 1993, pp. 1022–1029.

[18] S. del Río, V. López, J. M. Benítez, and F. Herrera, "A MapReduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules," *Int'l Journal of Computational Intelligence Systems*, vol. 8, no. 3, pp. 422–437, 2015.

[19] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Comm. of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[20] Z. Chi, H. Yan, and T. Phm, *Fuzzy algorithms: with applications to image processing and pattern recognition*, ser. Advances in Fuzzy Systems - Applications and Theory. World Scientific, 1996, vol. 10.

[21] 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, pp. 5–38, 2015.

[22] 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*, pp. 1–20, 2016.

[23] A. Fernández, C. J. Carmona, M. J. del Jesus, and F. Herrera, "A view on fuzzy systems for Big Data: Progress and opportunities," *Int'l Journal of Computational Intelligence Systems*, vol. 9, no. sup1, pp. 69–80, 2016.

[24] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, Jun 1993.

[25] H. Ishibuchi, T. Nakashima, and M. Nii, *Classification and modeling with linguistic information granules: Advanced approaches to linguistic Data Mining*. Springer, 2006.

[26] H. Ishibuchi and T. Nakashima, "Effect of rule weights in fuzzy rule-based classification systems," *IEEE Trans. on Fuzzy Systems*, vol. 9, no. 4, pp. 506–515, 2001.

[27] T. White, *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[28] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proc. of the 2nd USENIX Conf. on Hot topics in Cloud Computing*, vol. 10, 2010, p. 10.

[29] 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, pp. 21–45, 1999.

[30] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "PFP: parallel FP-growth for query recommendation," in *Proc. of 2008 ACM Conf. on Recommender Systems*. ACM, 2008, pp. 107–114.

[31] I. Pramudiono and M. Kitsuregawa, *Parallel FP-growth on PC cluster*, ser. LNCS. Springer, 2003, vol. 2637, pp. 467–473.

[32] W. Li, J. Han, and J. Pei, "CMAR: Accurate and efficient classification based on multiple class-association rules," in *ICDM 2001, Proc. of IEEE Int'l Conf. on Data Mining*. IEEE, 2001, pp. 369–376.

[33] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data*, vol. 2, no. 1, p. 8, 2014.

Armando Segatori received the M.Sc. degree in Computer Engineering and the Ph.D. degree in Information Engineering from the University of Pisa, Pisa, Italy, in 2012 and 2016, respectively. His research interests are focused on design, implementation and performance evaluation of scalable and distributed classification algorithms as well as multi-objective evolutionary fuzzy systems for handling Big Data.

Alessio Bechini earned his Laurea degree in Electronics Engineering and the PhD degree in Information Engineering both from the University of Pisa in 1996 and 2003. At present he is a researcher at the Department of Information Engineering of the University of Pisa. His research interests span the fields of concurrent and distributed systems, enterprise information systems, data management and integration, service management, with particular interest in bioinformatics problems. Recently, his work has been focused on issues in data mining for Big Data. He has served as TPC co-chair, general co-chair and program co-chair of ACM international conferences.

Pietro Ducange received the M.Sc. degree in Computer Engineering and the Ph.D. degree in Information Engineering from the University of Pisa, Pisa, Italy, in 2005 and 2009, respectively. Currently, he is an associate professor at eCampus University, Novedrate, Italy, where he is the director of the SMART Engineering Solutions & Technologies (SMARTEST) Research Centre. His main research interests include evolutionary fuzzy systems, big data mining, social sensing and sentiment analysis. Further, he has been involved in a number of R&D projects in which data mining and computation intelligence algorithms have been successfully employed. He has coauthored more than 40 papers in international journals and conference proceedings. Prof. Ducange is a Member of the Editorial Boards for Soft Computing and the International Journal of Swarm Intelligence and Evolutionary Computation.

Francesco Marcelloni received the Laurea degree in Electronics Engineering and the PhD degree in Computer Engineering from the University of Pisa in 1991 and 1996, respectively. He is currently a full professor at the Department of Information Engineering of the University of Pisa, where he has co-founded the Computational Intelligence Group in 2002. Further, he is the founder and head of the Competence Centre on MObile Value Added Services (MOVAS). Currently, he is the Rectors Delegate for Internationalization of the University of Pisa. Further, he is member of the Management Board of the Pisa University Press. His main research interests include data mining for big data, multi-objective evolutionary algorithms, genetic fuzzy systems, fuzzy clustering algorithms, pattern recognition, signal analysis, mobile information systems, and data compression and aggregation in wireless sensor networks. He has co-edited three volumes, four journal special issues, and is (co-)author of a book and of more than 200 papers in international journals, books and conference proceedings. He has been TPC co-chair, general co-chair and tutorial chair of some international conferences and has held invited talks in a number of events. Currently, he serves as associate editor of Information Sciences (Elsevier) and Soft Computing (Springer), and is on the editorial board of a number of other international journals. He has coordinated various research projects funded by both public and private entities.