On Distributed Fuzzy Decision Trees for Big Data

Armando Segatori, Francesco Marcelloni, *Member, IEEE,* and Witold Pedrycz, *Fellow, IEEE*

***Abstract*—Fuzzy decision trees (FDTs) have shown to be an effective solution in the framework of fuzzy classification. The approaches proposed so far to FDT learning, however, have generally neglected time and space requirements. In this paper, we propose a distributed FDT learning scheme shaped according to the MapReduce programming model for generating both binary and multi-way FDTs from big data. The scheme relies on a novel distributed fuzzy discretizer that generates a strong fuzzy partition for each continuous attribute based on fuzzy information entropy. The fuzzy partitions are therefore used as input to the FDT learning algorithm, which employs fuzzy information gain for selecting the attributes at the decision nodes. We have implemented the FDT learning scheme on the Apache Spark framework. We have used ten real-world publicly available big datasets for evaluating the behavior of the scheme along three dimensions: i) performance in terms of classification accuracy, model complexity and execution time, ii) scalability varying the number of computing units and iii) ability to efficiently accommodate an increasing dataset size. We have demonstrated that the proposed scheme turns out to be suitable for managing big datasets even with modest commodity hardware support. Finally, we have used the distributed decision tree learning algorithm implemented in the MLLib library and the Chi-FRBCS-BigData algorithm, a MapReduce distributed fuzzy rule-based classification system, for comparative analysis.**

***Keywords*—Fuzzy Decision Trees, Big Data, Fuzzy Entropy, Fuzzy Discretizer, Apache Spark, MapReduce, Fuzzy Partitioning**

I. Introduction

Decision trees are widely used classifiers, successfully employed in many application domains such as security assessment [1], health system [2] and road traffic congestion [3]. The popularity of decision trees is mainly due to the simplicity of their learning schema. Further, decision trees are considered among the most interpretable classifiers [4], [5], that is, they can explain how an output is inferred from the inputs. Finally, the tree learning process usually requires only a few parameters that must be adjusted. A large number of algorithms have been proposed in the last decades for generating decision trees: most of them are extensions or improvements of the well-known ID3 proposed by Quinlan et al. [6] and CART proposed by Brieman et al. [7]. In a decision tree, each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of the test, and each leaf (or terminal) node holds a class label.

Several works have exploited the possibility of integrating decision trees with the fuzzy set theory to deal with uncertainty [8], [9], leading to the so-called fuzzy decision trees (FDTs). Unlike Boolean decision trees, each node in FDTs is characterized by a fuzzy set rather than a set. Thus, each instance can activate different branches and reach multiple leaves. Both Boolean and fuzzy decision trees are generated by applying a top-down approach that partitions the training data into homogeneous subsets, that is, subsets of instances belonging to the same class [10].

Like classical decision trees, FDTs can be categorized into two main groups, depending on the splitting method used in generating child nodes from a parent node [11]: *binary* (or two-way) split trees and *multi-way* split trees. Binary split trees are characterized by recursively partitioning the attribute space into two subspaces so that each parent node is connected exactly with two child nodes. On the other hand, multi-way split trees partition the space into a number of subspaces so that each parent node generates in general more than two child nodes. Since a tree with multi-way splits can be always redrawn as a binary tree [12], apparently the use of multi-way split seems to offer no advantage. We have to consider, however, that binary split implies that an attribute can be used several times in the same path from the root to a leaf. Thus, a binary split tree is generally deeper and sometimes harder to interpret than a multi-way split tree [13], [14]. Further, in some domain [13], multi-way splits seem to lead to more accurate trees but, since multi-way splits tend to fragment the training data very quickly [12], they generally need larger data size in order to work effectively.

Typically, FDT learning algorithms require that a fuzzy partition has been already defined upon each continuous attribute. For this reason, continuous attributes are usually discretized by optimizing purposely-defined indexes [15], [16]. Discretization can drastically affect the accuracy of classifiers [17], [18], [19] and therefore should be realized with great care. In [17], authors performed an interesting analysis by investigating how different discretization approaches influence the accuracy and the complexity (in terms of number of nodes) of the generated FDTs: they employed several well-known fuzzy partitioning methods and different approaches for, given a Boolean partition generated by well-known discretization algorithms [18] [19], defining different types of membership functions. The experimental results reported on 111 different combinations highlight that seven of them outperform the others in both accuracy and number of nodes.

FDTs have been mainly used in the literature for classifying small datasets. Thus, FDT learning approaches have focused on increasing classification accuracy, often neglecting time

A. Segatori and F. Marcelloni are with the Dipartimento di Ingegneria dell'Informazione, University of Pisa, Pisa, Italy 56122 (e-mail: armando.segatori@for.unipi.it and francesco.marcelloni@unipi.it)

W. Pedrycz is with the Department of Electrical & Computer Engineering, University of Alberta, Edmonton AB Canada T6R 2V4, with the Department of Electrical and Computer Engineering, Faculty of Engineering, King Abdulaziz University, Jeddah, 21589, Saudi Arabia and also with the Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland (e-mail: wpedrycz@ualberta.ca)

and space requirements, by adopting several heavy tasks such as pruning steps, genetic algorithms, and computation of the optimal split among all points at each node [20] [21] [22] [23]. Thus, these approaches are not generally suitable for dealing with a huge amount of data. A possible simple solution for applying these approaches would be to select only a subset of data objects by applying some downsampling technique. However, these techniques may ignore some useful knowledge, making FDT learning approaches purposely designed for managing the overall dataset more desirable and effective. In our context, this means explicitly addressing Big Data.

Big Data is a term which identifies datasets so large and complex that traditional data processing approaches are inadequate. Big Data requires specific technologies to support semi-structured or unstructured data and scale out with commodity hardware in parallel to cope with ever-growing data volumes. To address these challenges several solutions have been proposed in the last years [24], such as (*i*) *cloud computing*, an infrastructure layer for big data systems to meet requirements on cost-effectiveness, elasticity, and ability to scale up/out; (*ii*) *distributed file systems* and *NoSQL databases*, for persistent storage and management of massive scheme-free datasets; (*iii*) *MapReduce* [25] and *Pregel* [26], two programming models proposed by Google for simplifying the distribution of the computation flow across large-scale clusters of machines; (*iv*) *cluster computing frameworks*, powerful system-level solutions, like Apache Hadoop [27], [28] and Apache Spark [29], [30], for distributed data storage and processing, system and failure management, and efficient network bandwidth and disk usage.

Most of the studies recently proposed in the literature for mining big data combine the MapReduce model with the Apache Hadoop and Apache Spark cluster computing frameworks. With regard to classification problems, some recent works have proposed several distributed MapReduce versions of classical algorithms, such as SVM [31], [32], prototype reduction [33], kNN [34], associative classifiers [35], [36], boosting [37], decision trees [38] [39] [40], naive Bayes classifiers and neural networks [41], investigating performance in terms of speedup [41]. Although with the increase of the number and size of big data, researchers are continuously investigating new algorithms, taking into account not only the accuracy of the classifiers, but also the scalability of the proposed approaches, only few works have integrated fuzzy set theory [36], [42], [43].

In this paper, we propose a distributed fuzzy discretizer and a distributed FDT (DFDT) learning scheme upon the MapReduce programming model for managing big data. To the best of our knowledge, in the context of big data, no distributed discretizer for generating fuzzy partitions and no DFDT have been proposed in the literature. Our novel discretizer generates a strong fuzzy partition for each continuous attribute by using a purposely adapted distributed version of the well-known method proposed by Fayyad and Irani in [44]. The fuzzy partitions computed by the discretizer are used as input to the DFDT learning algorithm. We adopt and compare two different versions of the learning algorithm based on binary and multi-way splits, respectively. Both the versions employ the information gain computed in terms of fuzzy entropy for selecting the attribute to be adopted at each decision node.

We have implemented both the discretizer and the learning scheme on Apache Spark[1]. We have used 10 real-world big datasets characterized by a different number of instances (up to 11 millions) and class labels (from 2 to 50). We have compared the results obtained by our approach with those achieved by two state-of-the-art distributed classifiers, namely the distributed decision tree (DDT) learning algorithm implemented in the MLLib on Spark and the Chi-FRBCS-BigData algorithm [45], a MapReduce distributed fuzzy rule-based classification system, with respect to accuracy, complexity, and scalability.

The paper is organized as follows. Section II discusses some related works in the framework of distributed decision trees and distributed fuzzy classifiers.Section III provides some preliminaries on FDTs, the MapReduce programming model and the Apache Spark framework. Section IV first introduces the fuzzy discretizer and the FDT learning algorithm and then discusses their distributed implementation, detailing each single MapReduce job. Section V presents and discusses the experimental results comparing the proposed approach with the state-of-the-art DDT in terms of accuracy, complexity, and scalability. Finally, in Section VI we draw some final conclusion.

II. Related Work

A number of works have discussed on how a decision tree can be generated efficiently from very large datasets. The various techniques proposed in the literature can be roughly grouped into two categories, which are characterized by performing a pre-sorting of the data or by adopting approximate representations of the data such as samples and/or histograms [46]. While pre-sorting techniques are more accurate, they cannot accommodate very large datasets or streaming data [46].

One of the oldest approaches in the first category is SLIQ, proposed in [47]. SLIQ reduces decision tree learning time without loss in accuracy by exploiting a pre-sorting technique in the tree-growth phase. This technique is integrated with a breadth-first tree growing strategy to enable classification of disk-resident datasets. SLIQ also uses a tree-pruning algorithm, based on the Minimum Description Length principle, which is inexpensive and results in compact and accurate trees. However, SLIQ requires that some data per record reside in memory all the time. Since the size of this in-memory data structure grows in direct proportion to the number of input records, this limits the amount of data, which can be classified by SLIQ. SPRINT, proposed in [48], removes these memory restrictions. Further, it has also been designed to be easily parallelized, achieving good scalability.

As regards the second category, the BOAT algorithm proposed in [49] exploits a novel optimistic approach to tree construction, which generates an initial tree using a small subset of the data and refines it to arrive at the final tree. The authors guarantee that any difference with respect to the

[1]The code is publicly available on GitHub at the following link: http://github.com/BigDataMiningUnipi/FuzzyDecisionTreeSpark

real tree (i.e., the tree that would have been constructed by examining all the data in a traditional way) is detected and corrected. The several levels of the tree are built in only two scans over the training dataset. In [50] a decision tree construction algorithm called SPIES is proposed. SPIES limits the number of possible split points by taking a sample from the data set, partitions the values into intervals and computes the class histograms for candidate split points. This reduces the space complexity of the algorithm and the communication cost between processors.

The different ways to parallelize decision tree learning can be grouped into 4 main categories [46], [51]: i) horizontal, or data-based, parallelism partitions the data so that different processors work on different examples; ii) vertical, or feature-based, parallelism enables different processors to consider different attributes; iii) task, or tree node-based, parallelism distributes the tree nodes to the slave processors and iv) hybrid parallelism combines horizontal or vertical parallelism in the first stages of tree construction with task parallelism towards the end. In [51], the authors show how to parallelize two different decision tree learning algorithms, namely C4.5 and the univariate linear discriminant tree proposed in [52], exploiting horizontal, vertical and task parallelism. Experimental results show that performance of the parallelization highly depends on the dataset, although node-based parallelization shows generally good speed-ups. In [46], the authors exploit an approximate representation and horizontal parallelism. The core of the algorithm is an on-line method for building histograms from streaming data at the processors. The histograms are compressed representations of the data, which can be transmitted to a master processor with low communication complexity. The master processor integrates the information received from all processors and determines which terminal nodes to split and how. A short review on decision tree learning algorithms proposed to handle very large datasets has been presented in [53]. Following the previous works on distributed decision trees, the FDT learning algorithms proposed in this paper exploit both horizontal and task parallelism.

In the last years, some decision tree learning algorithms have been proposed for managing big data by adopting the MapReduce paradigm on the top of Apache Hadoop [38] [39] [40]. MapReduce is based on functional programming and divides the computational flow into two main phases, namely Map and Reduce, which communicate by $\langle key, value\rangle$ pairs. The MapReduce implementation of a distributed decision tree proposed in [40] employs, for instance, four map-reduce stages. The first stage scans the dataset for creating the initial data structures employed in the other three stages. These stages are executed iteratively for, respectively, (i) selecting the best attribute, (ii) updating the statistics for the new nodes and (iii) growing the tree. The experimental results discussed in the paper are limited only to the scalability analysis by varying the number of nodes and the dataset size (up to 3 millions of instances). The effectiveness of the decision trees for managing big data has been demonstrated in real application domains such as stock futures prediction [54] and clinical decision support [55]. Other works [56] [57] [58] exploit decision trees for generating ensemble of classifiers such as random forest. To deal with big data, the proposed algorithms first build concurrently multiple trees from different chunks of data and then group all of them for generating the forest. However, the generation of each tree is not distributed on the cluster but is performed sequentially on a single chunk of the entire dataset.

To the best of our knowledge, only a few classifiers proposed for managing big data employ fuzzy sets. In [42], [45], the authors describe Chi-FRBCS-BigData, a fuzzy rule-based classification system based on the Chi et al.'s approach [59]. This approach has been modified to deal with big data by employing two map-reduce stages. The first stage builds the model from chunks of the training set: a group of fuzzy rules is generated from each chunk. Then, these groups are fused together in the reduce phase. The second stage estimates the class using the model learned in the first stage. The authors have investigated different approaches for fusing the fuzzy rules by developing two different versions, named Chi-FRBCS-BigData-Max and Chi-FRBCS-BigData-Ave. Moreover, an improved version, called Chi-FRBCS-BigDataCS, has been proposed in [43] for handling imbalanced big datasets.

Since, to the best of our knowledge, there do not exist works that have discussed FDT for cloud computing environments, taking into account accuracy, complexity and scalability, we think that this paper can represent a significant contribution for future researches on how to handle big datasets using FDTs.

III. Background

In this section, we first introduce the FDT and the necessary notations used in the paper and then we describe both the MapReduce programming model and the Apache Spark cluster computing framework. This framework is exploited in our DFDT.

A. *Fuzzy Decision Tree*

Instance classification consists of assigning a class C_m from a predefined set $C = \{C_1, \ldots, C_M\}$ of M classes to an unlabeled instance. Each instance can be described by both numerical and categorical attributes. Let $\mathbf{X} = \{X_1, \ldots, X_F\}$ be the set of attributes. In case of numerical attributes, X_f is defined on a universe $U_f \subset \Re$. In case of categorical attributes, X_f is defined on a set $L_f = \{L_{f,1}, \ldots, L_{f,T_f}\}$ of categorical values. An FDT is a directed acyclic graph, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of the test, and each leaf (or terminal) node holds one or more class labels. The topmost node is the *root* node. In general, each leaf node is labeled with one or more classes C_m with an associated weight w_m: weight w_m determines the strength of class C_m in the leaf node.

Let $TR = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$ be the training set, where, for each instance $(\mathbf{x}_i, y_i)$, with $i = 1, ..., N$, $y_i \in C$ and $x_{i,f} \in U_f$ in case of continuous attribute and $x_{i,f} \in L_f$ in case of categorical attribute, with $f = 1, ..., F$. FDTs are generated in a top-down way by performing recursive partitions of the attribute space.

Algorithm 1 shows the scheme of a generic FDT learning process. The $SelectAttribute$ procedure selects the attribute used in the decision node and determines the splits generated from the values of this attribute. The selection of the attribute is carried out by using appropriate metrics, which measure the difference between the levels of homogeneity of the class labels in the parent node and in the child nodes generated by the splits. The commonly used metrics are the fuzzy information gain [17], fuzzy Gini index [21], minimal ambiguity of a possibility distribution [15], maximum classification importance of attribute contributing to its consequent [60] and normalized fuzzy Kolmogorov-Smirnov discrimination quality measure [61]. In this paper, we adopt the fuzzy information gain, which will be defined in Section IV-B. The splitting method adopted in the $SelectAttribute$ procedure determines the attribute to be selected and also the number of child nodes. In the literature, both multi-way and binary splits are used. We have implemented both the approaches and evaluated their pros and cons.

Once the tree has been generated, a given unlabeled instance $\widehat{\mathbf{x}}$ is assigned to a class $C_m \in C$ by following the activation of nodes from the root to one or more leaves. In classical decision trees, each node represents a crisp set and each leaf is labeled with a unique class label. It follows that $\widehat{\mathbf{x}}$ activates a unique path and is assigned to a unique class. In FDT, each node represents a fuzzy subset. Thus, $\widehat{\mathbf{x}}$ can activate multiple paths in the tree, reaching more than one leaf with different strengths of activation, named *matching degrees*. Given a current node CN, the matching degree $md^{CN}(\widehat{\mathbf{x}})$ of $\widehat{\mathbf{x}}$ with CN is calculated as:

$$md^{CN}(\widehat{\mathbf{x}}) = TN(\mu^{CN}(\widehat{x}_f), md^{PN}(\widehat{\mathbf{x}})) \tag{1}$$

where TN is a T-norm, $\mu^{CN}(\widehat{x}_f)$ is the membership degree of $\widehat{x}_f$ to the current node CN, which considers X_f as splitting attribute, and $md^{PN}(\widehat{\mathbf{x}})$ is the matching degree of $\widehat{\mathbf{x}}$ with the parent node PN.

The *association degree* $AD_m^{LN}(\widehat{\mathbf{x}})$ of $\widehat{\mathbf{x}}$ with the class C_m at leaf node LN is calculated as:

$$AD_m^{LN}(\widehat{\mathbf{x}}) = md^{LN}(\widehat{\mathbf{x}}) \cdot w_m^{LN} \tag{2}$$

where $md^{LN}(\widehat{\mathbf{x}})$ is the matching degree of $\widehat{\mathbf{x}}$ with LN and w_m^{LN} is the class weight associated with C_m at leaf node LN. In the literature, different definitions have been proposed for weight w_m^{LN} [62]. Further, it has been proved that the use of class weights can increase the performance of fuzzy classifiers [63]. To determine the output class label of the unlabeled instance $\widehat{\mathbf{x}}$, two different approaches are often adopted in the literature:

- *maximum matching*: the class corresponds to the maximum association degree calculated for the instance;
- *weighed vote*: the class corresponds to the maximum total strength of vote. The total strength of vote for each class is computed by summing all the activation degrees in each leaf for the class. If no leaf has been reached, the instance $\widehat{\mathbf{x}}$ is classified as unknown.

B. MapReduce and Apache Spark

In 2004, Google proposed the MapReduce programming framework [25] for distributing the computation flow across large-scale clusters of machines, taking care of communication, network bandwidth, disk usage and possible failures.

Algorithm 1 Pseudo code of a generic FDT learning process.

Require: training set TR, set X of attributes, splitting method $SplitMet$, stopping method $StopMet$

procedure FDTLEARNING(**in**: TR, X, $SplitMet$, $StopMet$)
  $root \leftarrow$ create a new node
  $tree \leftarrow$ TREEGROWING($root$, TR, X, $SplitMet$, $StopMet$)
  return $tree$
end procedure
procedure TREEGROWING(**in**: $node$, S, X, $SplitMet$, $StopMet$)
  if STOPMET($node$) **then**
    $node \leftarrow$ mark $node$ as leaf
  else
    $splits \leftarrow$ SELECTATTRIBUTE(X, S, $SplitMet$)
    for each $split_z$ in $splits$ **do**
      $S_z \leftarrow$ get the set of instances from S determined by $split_z$
      $child_z \leftarrow$ create one node by using $split_z$ and S_z
      $node \leftarrow$ connect the node with TREEGROWING($child_z$, S_z, X_z, $SplitMet$, $StopMet$)
    end for
  end if
  return $node$
end procedure

At high level, the framework, which is based on functional programming, divides the computational flow into two main phases, namely Map and Reduce, organized around $\langle key, value\rangle$ pairs.

When the MapReduce execution environment runs a user program, the framework automatically partitions the data into a set of independent *chunks*, that can be processed in parallel by different machines. Each machine can host several Map and Reduce tasks. In the Map phase, each Map task is fed by one chunk of data and, for each $\langle key, value\rangle$ pair as input, it generates a list of intermediate $\langle key, value\rangle$ pairs as output. In the Reduce phase, all the intermediate results are grouped together according to a key-partitioning scheme, so that each Reduce task processes a list of values associated with a specific key as input for generating a new list of values as output. In general, developers are able to implement parallel algorithms that can be executed across the cluster by simply defining Map and Reduce functions.

In the last years, several open source projects have been developed to deal with big data [64]. So far, the most popular execution environment for the MapReduce programming model is Apache Hadoop [27] [28], that allows the execution of custom applications for processing big datasets stored in its distributed file system, called Hadoop Distributed FileSystem (HDFS). However, due to a poor inter-communication capability and inadequacy for in-memory computation [29] [65], Hadoop is not suitable for specific types of applications such as the ones that need iterative or online computations. Recently, different projects have been implemented to overcome these drawbacks. Apache Spark is certainly the most popular among these projects, thanks to its flexibility and efficiency.

Indeed, it allows implementing several distributed models like MapReduce and Pregel [26]. Further, it has proved to perform faster than Hadoop [29], especially for iterative and online processing.

The main abstraction provided by Spark [29] is the *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements, partitioned across the machines of the cluster, that can be processed in parallel. At high level, a Spark application runs as an independent set of processes on the top of the RDDs and consists of a *driver program* and a number of *executors*. The driver program, hosted in the master machine, is in charge to both run the user's main function and distribute *operations* on the cluster by sending several units of work, called *tasks*, to the executors. Each executor, hosted in a slave machine, runs tasks in parallel and keeps data in memory or disk storage across them.

Regarding data mining tools for big data, the MLlib library [66] is the most popular machine learning library running on top of Spark. It implements a wide range of machine learning and data mining algorithms for Extract, Transform, Load (ETL) operations, attribute selection, clustering, recommendation systems, frequent pattern mining, classification and regression problems.

IV. The Proposed Distributed Fuzzy Decision Tree for Big Data

In this section, we introduce the DFDT learning algorithm for handling Big Data. We aim to propose an approach that is easy to implement, is computationally light and guarantees to achieve accuracy values and execution times comparable with other distributed classifiers. We discuss two distinct versions, which differ from each other on the nature of the splitting mechanism.

The workflow of the DFDT learning process consists of the two following main steps:

1) *Fuzzy Partitioning*: a strong fuzzy partition is determined on each continuous attribute by using a novel discretizer based on fuzzy entropy;
2) *FDT Learning*: an FDT is induced from data by using either a multi-way or a binary splitting mechanism based on the concept of fuzzy information gain.

In the following, we first discuss the two steps in detail and then we describe the adopted distributed implementation for handling Big Data, by specifying how the execution can be parallelized and distributed among the Computing Units (CUs) available on the cluster.

A. Fuzzy Partitioning

Partitioning of continuous attributes is a crucial aspect in the generation of FDTs and therefore should be performed carefully. An interesting study proposed in [17] has investigated 111 different approaches for generating fuzzy partitions and has analyzed how these approaches can influence the accuracy and the complexity (in terms of number of nodes) of the generated FDTs. Among them, Fuzzy Partitioning based on Fuzzy Entropy (FPFE) has proved to be very effective. In this section, we propose an FPFE for generating strong triangular fuzzy partitions when handling big data.

The proposed FPFE is a recursive supervised method, which generates *candidate fuzzy partitions* and evaluates these partitions employing the fuzzy entropy. The algorithm selects the candidate fuzzy partition that minimizes the fuzzy entropy and then splits the continuous attribute domain into two subsets. Similar to the Entropy Minimization method proposed by Fayyad and Irani in [44], the process is repeated for each generated subset until a stopping condition is met. The candidate fuzzy partitions are generated for each value of the attribute in the training set: the values are sorted in increasing order. Since both the sorting process and the evaluation of this huge amount of candidate fuzzy partitions are computationally very heavy when dealing with big data, we will discuss in Section IV-C an approximated version of the fuzzy partitioning approach, which exploits equi-frequency bins.

In the following, we first recall some definition and then we describe the method.

Let $TR_f = [x_{1,f}, ..., x_{N,f}]^T$ be the projection of the training set TR along attribute X_f. We assume that the values $x_{i,f}$ are sorted in increasing order. Let I_f be an interval defined on the universe of attribute X_f. Let l_f and u_f be the lower and upper bounds of I_f. Let S_f be the set of values $x_{i,f} \in TR_f$ contained in I_f. Let us assume that a fuzzy partition $P_{I_f} = \{B_{f,1}, \ldots, B_{f,K_{P_{I_f}}}\}$, where $K_{P_{I_f}}$ is the number of fuzzy sets in P_{I_f}, is defined on I_f. Let $S_{f,1}, \ldots, S_{f,K_{P_{I_f}}}$ be the subsets of points in S_f, contained in the supports of $B_{f,1}, \ldots, B_{f,K_{P_{I_f}}}$, respectively. The weighted fuzzy entropy $WFEnt(P_{I_f}, I_f)$ of partition P_{I_f} is defined as:

$$WFEnt(P_{I_f}; I_f) = \sum_{j=1}^{K_{P_{I_f}}} \frac{|B_{f,j}|}{|S_f|} FEnt(B_{f,j}) \tag{3}$$

where $|B_{f,j}|$ is the fuzzy cardinality of fuzzy set $B_{f,j}$, $|S_f|$ is the cardinality of set S_f and $FEnt(B_{f,j})$ is the fuzzy entropy of $B_{f,j}$.

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

$$|B_{f,j}| = \sum_{i=1}^{N_{f,j}} \mu_{B_{f,j}}(x_{i,f}) \tag{4}$$

where $N_{f,j}$ is the number of points in $S_{f,j}$ and $\mu_{B_{f,j}}(x_{i,f})$ is the membership degree of $\mathbf{x}_i$ to fuzzy set $B_{f,j}$. The fuzzy entropy of $B_{f,j}$ is defined as

$$FEnt(B_{f,j}) = \sum_{m=1}^{M} -\frac{|B_{f,j,C_m}|}{|B_{f,j}|} log_2(\frac{|B_{f,j,C_m}|}{|B_{f,j}|}) \tag{5}$$

where fuzzy cardinality $|B_{f,j,C_m}|$ is computed on the set of instances in $S_{f,j}$ with class label C_m.

At the beginning, I_f coincides with the universe of X_f and $S_f = TR_f$. For each value $x_{i,f}$ between l_f and u_f (at the beginning of the partitioning procedure, $i = 1, \ldots, N$), we define a strong fuzzy partition $P_{x_{i,f}}$ on I_f by using three triangular fuzzy sets, namely $B_{f,1}$, $B_{f,2}$ and $B_{f,3}$, as shown in Fig. 1. The cores of $B_{f,1}$, $B_{f,2}$ and $B_{f,3}$ coincide with l_f, $x_{i,f}$ and u_f, respectively. Let $S_{f,1}$, $S_{f,2}$ and $S_{f,3}$ be the subsets of points in S_f, contained in the supports of $B_{f,1}$, $B_{f,2}$ and $B_{f,3}$, respectively.

For each partition $P_{x_{i,f}}$ induced by $x_{i,f}$, we compute the weighted fuzzy entropy $WFEnt(P_{x_{i,f}}, I_f)$ using Eq. 3. The optimal value $x^0_{i,f}$, which minimizes $WFEnt(P_{x_{i,f}}, I_f)$ over all possible candidate fuzzy partitions, is then selected. This value identifies the fuzzy partition $P_{x^0_{i,f}} = \{B^0_{f,1}, B^0_{f,2}, B^0_{f,3}\}$. Let $S^0_{f,1}$, $S^0_{f,2}$ and $S^0_{f,3}$ be the subsets of points in S_f, contained in the supports of the three fuzzy sets, respectively. Then, we apply recursively the procedure for determining the optimal strong fuzzy partition to the intervals $I^1_f = [l_f, x^0_{i,f}]$ and $I^2_f = (x^0_{i,f}, u_f]$ identified by $x^0_{i,f}$, by considering $S_f = S^0_{f,1}$ and $S_f = S^0_{f,3}$, respectively.

As an example, let us consider $I_f = I^1_f$. We have an initial partition $P^0_{I_f}$ on I^1_f, which consists of the fuzzy set $\widehat{B}^0_{f,1} = B^0_{f,1}$ and of fuzzy set $\widehat{B}^0_{f,2}$, which coincides from l_f to $x^0_{i,f}$ with fuzzy set $B^0_{f,2}$. For each value $x_{i,f}$ in I^1_f, we define a strong fuzzy partition $P_{x_{i,f}}$ on $I_f = I^1_f$ and compute the corresponding fuzzy entropy $WFEnt(P_{x_{i,f}}, I_f)$ as explained above. Let $x^1_{i,f}$ be the value, which minimizes $WFEnt(P_{x_{i,f}}, I_f)$. This value identifies the optimal fuzzy partition $P_{x^1_{i,f}} = \{B^1_{f,1}, B^1_{f,2}, B^1_{f,3}\}$. Let $S^1_{f,1}$, $S^1_{f,2}$ and $S^1_{f,3}$ be the subsets of points in S_f, contained in the supports of the three fuzzy sets, respectively.

The partitioning process continues until the following stopping condition proposed in [17] has been met:

$$FGain(x^1_{i,f}; I_f) < \frac{\log_2(|S_f|-1)}{|S_f|} + \frac{\Delta(x^1_{i,f}; I_f)}{|S_f|} \quad (6)$$

where

$$FGain(x^1_{i,f}; I_f) = WFEnt(P^0_{I_f}, I_f) - WFEnt(P_{x^1_{i,f}}; I_f) \quad (7)$$

$$\Delta(x^1_{i,f}; I_f) = \log_2(3^{k_f} - 2) - \left[\sum_{t=1}^{2} k_f \cdot FEnt(\widehat{B}^0_{f,t}) - \sum_{j=1}^{3} k^1_{f,j} \cdot FEnt(B^1_{f,j})\right] \quad (8)$$

and k_f and $k^1_{f,j}$ are the numbers of class labels represented in the sets S_f and $S^1_{f,j}$, respectively.

If no initial partition exists on I_f (this occurs when I_f coincides with the universe of X_f and $S_f = TR_f$), we assume that only a fuzzy set $\widehat{B}^0_f$ is defined on I_f with membership function equal to 1 for the overall interval I_f. Thus, $WFEnt(P^0_{I_f}, I_f) = FEnt(\widehat{B}^0_f)$. In this case, if the stopping condition is satisfied and therefore no partition is possible for attribute X_f, then X_f is discarded and not employed in the FDT learning.

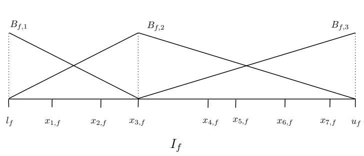


Fig. 1. An example of fuzzy partition defined on $x_{3,f}$.

Fig. 2 shows an example of application of the recursive procedure to the fuzzy partition shown in Fig. 1. We can observe that the partitioning of both I^1_f and I^2_f generates three fuzzy sets in both $[l_f, x^0_{i,f}]$ and in $(x^0_{i,f}, u_f]$. Actually, the two fuzzy sets, which have the core in $x^0_{i,f}$, are fused for generating a unique fuzzy set. Thus, the resulting partition is a strong partition with five fuzzy sets. This fusion can be applied at each level of the recursion. The final result is a strong fuzzy partition $P_f = \{A_{f,1}, ..., A_{f,T_f}\}$ on U_f, where $A_{f,j}$, with $j = 1, ..., T_f$, is the j^{th} triangular fuzzy set.

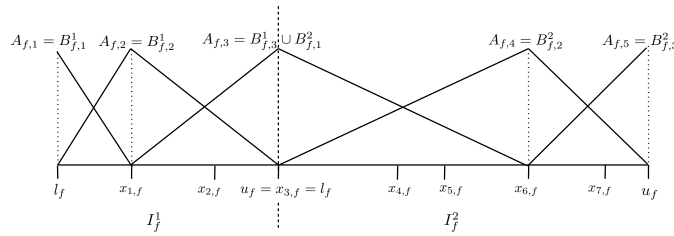


Fig. 2. An example of application of the recursive procedure to the fuzzy partition shown in Fig. 1 ($x^0_{i,f} = x_{3,f}$).

The procedure adopted for the fuzzy partition generation is simple, although computationally quite heavy. Further, it generates strong fuzzy partitions, which are widely assumed to have a high interpretability [67]. Finally, it allows performing an attribute selection because it may lead to the elimination of attributes, speeding up the FDT learning process.

B. FDT Learning

In this section, we introduce the FDT learning algorithm. We describe two distinct approaches, which differ from each other for the splitting mechanism used in the decision nodes. We adopt the FDT learning scheme described in Algorithm 1. The $SelectAttribute$ procedure selects the attribute, which maximizes the fuzzy information gain. Then Z child nodes are created. The number of child nodes as well as the computation of the fuzzy information gain depend on the employed splitting method. We have experimented two different methods: binary and multi-way splitting. The two methods generate Fuzzy Binary Decision Trees (FBDTs) and Fuzzy Multi-way Decision Trees (FMDTs), respectively. Both the trees use fuzzy linguistic terms to specify recursively branching condition of nodes until one of the following termination conditions ($StopMethod$ in Algorithm 1) is met:

1) the node contains only instances of the same class;
2) the node contains a number of instances lower than a fixed threshold λ;
3) the tree has reached a maximum fixed depth β;
4) the value of the fuzzy information gain is lower than a fixed threshold ϵ. In our experiments, we set $\epsilon = 10^{-6}$.

In case of multi-way splitting, for each parent node PN, FMDT generates as many child nodes CN_j as the number T_f of linguistic values defined on the splitting attribute X_f: each child node CN_j contains only the instances belonging to the support of the fuzzy set $A_{f,j}$ corresponding to the linguistic value. Let S_f be the set of instances in the parent node and $S_{f,j}$ be the set of instances in child node CN_j. Set $S_{f,j}$ contains the instances that belong to the support of $A_{f,j}$. Each node

CN_j is characterized by a fuzzy set G_j, whose cardinality is defined as

$$|G_j| = \sum_{i=1}^{N_j} \mu_{G_j}(\mathbf{x}_i) = \sum_{i=1}^{N_j} TN(\mu_{A_{f,j}}(x_{f,i}), \mu_G(\mathbf{x}_i)) \quad (9)$$

where N_j is the number of instances (crisp cardinality) in set $S_{f,j}$, $\mu_G(\mathbf{x}_i)$ is the membership degree of instance $\mathbf{x}_i$ to parent node PN (for the root of the decision tree, $\mu_G(\mathbf{x}_i) = 1$) and the operator TN is a T-norm.

In the $SelectAttribute$ procedure in Algorithm 1, we adopt the fuzzy information gain $FGain$, computed for a generic attribute X_f as:

$$FGain(P_f; I_G) = FEnt(G) - WFEnt(P_f; I_G) \quad (10)$$

where I_G is the support of fuzzy set G, and $FEnt(G)$ and $WFEnt(P_f; I_G)$ are computed as in Eq. 5 and Eq. 3, respectively.

In case of categorical attributes, we split the parent node into a number of child nodes CN_j equal to the number of possible values for the attribute. Each node CN_j is characterized by a fuzzy set G_j, whose cardinality is

$$|G_j| = \sum_{i=1}^{N_j} \mu_{G_j}(\mathbf{x}_i) = \sum_{i=1}^{N_j} TN(1, \mu_G(\mathbf{x}_i)) \quad (11)$$

Note that an attribute can be considered only once in the same path from the root to the leaf.

Figure 3 illustrates an example of how multi-way splitting is performed. Let us suppose that a fuzzy partition P_f with five triangular fuzzy sets has been defined on a continuous attribute X_f. For a given parent node, the method generates exactly five child nodes, one for each fuzzy set. Let us suppose that, a given instance, represented as a blue circle in Figure 3, belongs to $A_{f,1}$ and $A_{f,2}$ with membership values equal to 0.3 and 0.7, respectively. Thus, the instance belongs to only the child nodes corresponding to $A_{f,1}$ and $A_{f,2}$ and contributes to $|G_1|$ and $|G_2|$ with $TN(0.3, \mu_G)$ and $TN(0.7, \mu_G)$, respectively.

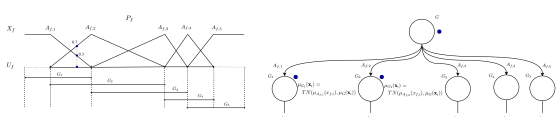


Fig. 3. An example of multiple splitting of a continuous attribute with five triangular fuzzy sets. The blue circle shows an example of how a given instance contributes to the cardinality computation.

Unlike FMDT, FBDT performs binary splitting at each node. As shown in Figure 4, the algorithm generates exactly 2 child nodes. To calculate the split with the maximum $FGain$, we exploit all possible candidates, by grouping together adjacent fuzzy sets into two disjoint groups Z_1 and Z_2. The two subsets G_1 and G_2 of instances contain the points that belong to the supports of the fuzzy sets contained in Z_1 and Z_2, respectively. A fuzzy partition with T_f fuzzy sets generates $T_f - 1$ candidates. Starting with $Z_1 = \{A_{f,1}\}$ and $Z_2 = \{A_{f,2}, ..., A_{f,T_f}\}$, we compute the fuzzy information gain by applying Eq. 10, with $P_f = \{Z_1, Z_2\}$ and cardinality $|G_1| = \sum_{i=1}^{N_1} TN(\mu_{A_{f,1}}(x_{f,i}), \mu_G(\mathbf{x}_i))$ and $|G_2| = \sum_{i=1}^{N_2} TN(\mu_{A_{f,2}}(x_{f,i}) + \ldots + \mu_{A_{f,T_f}}(x_{f,i}), \mu_G(\mathbf{x}_i))$, where N_1 and N_2 are the numbers of instances in the supports of the fuzzy sets in Z_1 and Z_2, respectively, and $\mu_G(\mathbf{x}_i)$ is the membership degree of instance $\mathbf{x}_i$ to the parent node. Iteratively, the algorithm investigates all candidates by moving the first fuzzy set in Z_2 to Z_1 and computing the corresponding $FGain$, until $Z_2 = \{A_{f,T_f}\}$. The pair (Z_1,Z_2), which obtains the highest $FGain$, is used for creating the two child nodes. The two nodes contain, respectively, the examples that belong to the support of the fuzzy sets in Z_1 and Z_2.

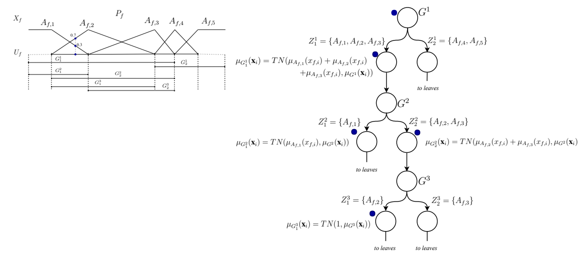


Fig. 4. An example of binary split performed by FBDT on a continuous attribute partitioned by five triangular fuzzy sets.

In case of categorical attributes, FBDT still performs binary splits. However, since a categorical attribute with L values generates $2^{L-1} - 1$ candidates, the computational cost can become very prohibitive for a large number of values. In case of binary classification, we can reduce the number of candidates to $L-1$ by sorting the categorical values according to the probability of membership to the positive class. As proved in [7] and [68], this approach gives the optimal split in terms of entropy. In case of multiclass classification, we adopt the heuristic method proposed in [69] to approximate the best split: the number of candidates is reduced to $L-1$ by sorting the categorical values according to their impurity.

In FBDT, both categorical and continuous attributes can be considered in several fuzzy decision nodes in the same path from the root to a leaf. In each node, we apply the same binary splitting approach described above but restricted only to the categorical values or fuzzy sets considered in the node. Figure 4 shows the splitting approach performed by FBDT, considering the same fuzzy partition used for FMDT. Let us suppose that, at the root, the attribute X_f is selected. Further, let us assume that the two child nodes of the root node contain instances belonging to the supports of $Z_1^1 = \{A_{f,1}, A_{f,2}, A_{f,3}\}$ and $Z_2^1 = \{A_{f,4}, A_{f,5}\}$, respectively. If X_f is selected again in the path starting from Z_1, then the two child nodes are created by considering only the three fuzzy sets in Z_1^1 and the instances contained in $[l_{f,1}, u_{f,3}]$, where $l_{f,1}$ and $u_{f,3}$ are the lower and upper bounds of the supports of $A_{f,1}$ and $A_{f,3}$, respectively. If the highest fuzzy information gain is obtained by splitting the three fuzzy sets into $\{A_{f,1}\}$ and $\{A_{f,2}, A_{f,3}\}$, then the two child nodes contain the instances belonging to the intervals $[l_{f,1}, u_{f,1}]$ and $[l_{f,2}, u_{f,3}]$, respectively.

Due to the use of the T-norm, and in particular of the product employed in our experiments, the binary splitting approach tends to penalize the cardinality of continuous attributes that are repeatedly selected along a same path. To limit this effect, we use a strategy that keeps track of the fuzzy sets, which have been activated by an instance in the path from the root to the leaves: we consider the membership value to a fuzzy set only the first time the fuzzy set is met. The subsequent times the membership value is set to 1 in the computation of the T-norm. For example, let us suppose that an instance $\mathbf{x}_i$ belongs to fuzzy sets $A_{f,1}$ and $A_{f,2}$ with membership values 0.3 and 0.7, respectively, as shown in Figure 4 (see blue circle). When splitting G^2, the instance contributes to the cardinality computation of G_1^2 and G_2^2 with $\mu_{G_1^2}(\mathbf{x}_i) = TN(\mu_{A_{f,1}}(x_{i,f}), \mu_{G^2}(\mathbf{x}_i))$ and $\mu_{G_2^2}(\mathbf{x}_i) = TN((\mu_{A_{f,2}}(x_{i,f})), \mu_{G^2}(\mathbf{x}_i))$, respectively. When splitting G^3, the membership degree $\mu_{A_{f,2}}(x_{i,f})$ of the instance $\mathbf{x}_i$ to $A_{f,2}$ is considered equal to 1 and the instance contributes to the cardinality computation of the subset G_1^3 with $\mu_{G_1^3}(\mathbf{x}_i) = TN(1, \mu_{G^3}(\mathbf{x}_i))$. On the other hand, the actual fuzzy membership value $\mu_{A_{f,2}}(x_{i,f})$ of instance $\mathbf{x}_i$ to $A_{f,2}$ has been already considered in the computation of $\mu_{G^3}(\mathbf{x}_i)$. Unlike crisp decision trees, for both FMDT and FBDT, we label each leaf node LN with all the classes that have at least one example in the leaf node. Each class C_m has an associated weight w_m^{LN} proportional to the fuzzy cardinality of training instances of that m^{th} class in the node. More formally, $w_m^{LN} = \frac{|G_{C_m}|}{|G|}$, where G_{C_m} is the set of instances in G with class label equal to C_m.

Both FMDT and FBDT adopt the weighed vote for deciding the class to be output for the unlabeled instance. For each class, the vote is computed as sum of the association degrees determined by any leaf node of the tree for that class, where the association degree is calculated by Eq. 2 . In case of FBDT, the fuzzy cardinality used in the computation of the matching degree is determined by considering the membership value to a specific fuzzy set only one time, also if the fuzzy set is met more times in the path from the root to the leaf, as explained above. Each activated leaf produces a list of class association degrees, which are summed up to compute the strength of vote for that class. The unlabeled pattern $\widehat{x}$ is associated with the class with the highest strength of vote.

C. The Distributed Approach

In Section I we have pointed out that the current implementations of FDTs are not suitable for managing big data. In this section we introduce our DFDT learning approach by describing in detail the distributed implementation of the two main steps, namely *Fuzzy Partitioning* and *FDT Learning*. We highlight that our approach is based on the Map-Reduce paradigm and can be easily deployed on several cloud-computing environments such as Hadoop and Spark.

Let V be the number of chunks used for splitting the training set and Q the number of CUs available in the cluster. Each chunk fed only one Map task, while one CU can process several tasks, both Map and Reduce. Obviously, only Q tasks can be executed in parallel.

The distributed implementation of the fuzzy partitioning approach described in Section IV-A is similar to the one we have proposed in [35]. In particular, the approach described in Section IV-A is not suitable for dealing with a huge amount of data because both the sorting of the values and the computation of the fuzzy information gain for each possible candidate fuzzy partition are computationally expensive in case of datasets with millions of instances. To overcome this drawback, we adopt an approximation of FPFE by limiting the number of possible candidate partitions to be analyzed. In particular, for each single chunk of the training set, independently of the others, we apply the sorting of the values and split the domain of the continuous attributes into a fixed number L of equi-frequency bins. Then, we aggregate the lists of the bin boundaries generated for each chunk and, for each pair of consecutive bin boundaries, we generate a new bin and compute the distribution of the classes among the instances belonging to the bin. Finally, we generate candidate fuzzy partitions for each bin boundary and exploit the class distribution in each bin for computing the fuzzy entropy and fuzzy information gain at each iteration of the algorithm. Obviously, the lower the number of bins used for splitting the domain of the attribute is, the coarser the approximation in determining the fuzzy partition is. As regards the computation of the fuzzy entropy, we consider each bin $b_{f,l}$ represented by its central value $\bar{b}_{f,l}$. Thus, the cardinality of a fuzzy set $B_{f,j}$ is computed as:

$$|B_{f,j}| = \sum_{l=1}^{L_{f,j}} \mu_{B_{f,j}}(\bar{b}_{f,l}) \tag{12}$$

where $L_{f,j}$ is the number of bins in $S_{f,j}$ and $\mu_{B_{f,j}}(\bar{b}_{f,l})$ is the membership degree of the central value $\bar{b}_{f,l}$ of bin $b_{f,l}$ to fuzzy set $B_{f,j}$. The fuzzy entropy of $B_{f,j}$ is computed as

$$FEnt(B_{f,j}) = \sum_{m=1}^{M} -\frac{|B_{f,j,C_m}|}{|B_{f,j}|} log_2(\frac{|B_{f,j,C_m}|}{|B_{f,j}|}) \tag{13}$$

where the fuzzy cardinality $|B_{f,j,C_m}|$ is calculated by considering the distribution of class C_m in each bin contained in the support of $B_{f,j}$.

Figure 5 shows the overall Fuzzy Partitioning process, which consists of two Map-Reduce steps. The first Map-Reduce step scans the training set to compute at most $\Omega = V \cdot (L+1)$ bin boundaries, where L is equal to the percentage γ of the chunk size. In our experiments, we set $\gamma = 0.1\%$. Algorithm 2 details the pseudo code of the first Map-Reduce step. Each Map-Task, first, loads the v^{th} chunk of the training set, and then for each continuous attribute X_f, sorts the values of X_f, and computes and outputs the bin boundaries of equi-frequency bins, where each bin contains a number of instances equal to the percentage γ of the data chunk. Let $BB_{v,f} = \{b_{v,f}^{(1)}, ..., b_{v,f}^{(L)}\}$ be the sorted list of bin boundaries for the f^{th} attribute extracted from the v^{th} chunk. The Map-Task outputs a key-value pair $\langle key = f, value = BB_{v,f}\rangle$, where f is the index of the f^{th} attribute. Each Reduce-Task is fed by V lists $List(BB_{v,f})$ and, for the f^{th} attribute, outputs $\langle key = f, value = BB_f\rangle$, where $BB_f = \{b_f^{(1)}, ..., b_f^{(\Omega)}\}$ with, $\forall w \in [1, ..., \Omega - 1]$, $b_f^{(w)} < b_f^{(w+1)}$ is the sorted list of the bin boundaries for attribute X_f. Space and time complexities, for the Map phase, are $O(\lceil \frac{V}{Q} \rceil \cdot N/V)$ and $O(\lceil \frac{V}{Q} \rceil \cdot (F \cdot N \cdot (log(N/V))/V))$, respectively, and, for the Reduce phase, are $O(F \cdot \Omega/Q)$ and $O(F \cdot (\Omega \cdot log(\Omega))/Q)$, respectively.

Algorithm 3 details the pseudo code of the second Map-Reduce step. Each Map-Task, first, loads the v^{th} chunk of the training set and, for each attribute X_f, initializes a vector $W_{v,f}$ of $\Omega - 1$ elements. Each element $W_{v,f}^{(r)}$ corresponds to the

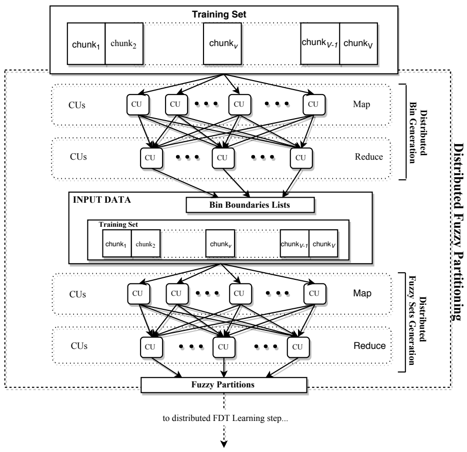


Fig. 5. The overall distributed Fuzzy Partitioning of the FDT.

Algorithm 2 Distributed Bin Generation.

Require: TR split into V $chunk_v$.
procedure MAP-TASK(**in**: $chunk_v$, γ)
  for each continuous attribute X_f in **X** **do**
    sort values of X_f
    $BB_{v,f} \leftarrow$ compute boundaries of equi-frequency bins according to γ
    output $\langle key = f, value = BB_{v,f}\rangle$
  end for
end procedure
procedure REDUCE-TASK(**in**: f, $List(BB_{v,f})$)
  $BB_f \leftarrow$ sort elements of $List(BB_{v,f})$
  output $\langle key = f, value = BB_f\rangle$
end procedure

Algorithm 3 Distributed Fuzzy Sets Generation.

Require: TR split into V $chunk_v$. Matrix BB where the f^{th} row contains BB_f.
procedure MAP-TASK(**in**: $chunk_v$, BB, M)
  $W_{v,f} \leftarrow$ create F arrays according to BB and M
  for each instance $(\mathbf{x}_n, y_n)$ in $chunk_v$ **do**
    for each continuous attribute X_f in **X** **do**
      $W_{v,f}^{(r)} \leftarrow$ update number of instances of y_n
    end for
  end for
  for each continuous attribute X_f in **X** **do**
    output $\langle key = f, value = W_{v,f}\rangle$
  end for
end procedure
procedure REDUCE-TASK(**in**: f, $List(W_{v,f})$, BB_f)
  $W_f \leftarrow$ element-wise addition of $List(W_{v,f})$
  $P_f \leftarrow$ FUZZYPARTITIONING(W_f, B_f)
  output $\langle key = f, value = P_f\rangle$
end procedure

---

bin $(b_f^r, b_f^{(r+1)}]$ and contains a vector of M elements, which stores, for each of the M classes, the number of instances of the class belonging to the r^{th} bin in the v^{th} chunk. Then, for each instance of the chunk, the Map-Task updates $W_{v,f}$ and finally outputs a key-value pair $\langle key = f, value = W_{v,f}\rangle$. Each Reduce-Task is fed by a list $List(W_{v,f})$ of V vectors. For each attribute X_f, it first creates a vector W_f of $\Omega - 1$ elements by performing an element-wise addition of all V vectors $W_{v,f}$. Thus, W_f stores the number of instances for each class in each bin along the overall training set. Then, the Reduce-Task applies the Fuzzy Partitioning as described in Section IV-A, where *candidate fuzzy partitions* are defined upon bin boundaries and the fuzzy mutual information is computed according to W_f. Finally, it outputs the key-pair $\langle key = f, value = P_f\rangle$, where P_f is the strong fuzzy partition defined on the f^{th} attribute. Space and time complexities of the Map phase are $O(\lceil\frac{V}{Q}\rceil \cdot N/V)$ and $O(\lceil\frac{V}{Q}\rceil \cdot (N \cdot log(\Omega)/V))$, respectively. For the Reduce phase, space and time complexities are $O(F \cdot (\Omega-1)/Q)$ and $O(F \cdot (2 \cdot max(T_f)-3) \cdot (\Omega-1)^2)/Q)$, respectively, where $max(T_f)$ is the maximum number T_f of fuzzy sets generated for an attribute.

The proposed distributed approach can manage a large number of instances: the bin boundaries allow reducing the number of candidate fuzzy partitions to be explored. Obviously, the number of equi-frequency bins is a parameter of the approach, which affects both the fuzzy partitioning of the continuous attributes and the results of the FDT. However, this parameter is not particularly critical. Indeed, we have to consider that we are managing millions of data. Thus, a difference of a few instances in determining the best fuzzy partition is generally negligible in terms of the accuracy achieved by the FDTs.

As regards the DFDT learning, we distribute the computation of the best split for each node across the CUs. Figure 6 illustrates the overall DFDT learning algorithm, which executes iteratively a Map-Reduce step. Algorithms 4 and 5 detail the pseudo code of the DFDT learning.

Let H be the number of iterations performed by the algorithm and h be the index of the h^{th} iteration. Let R be the list of nodes to be split, initialized with only one element consisting of the root of the tree. The algorithm iteratively retrieves a group R_h of Y nodes from R, where $Y = min(size(R), maxY)$ is computed according to the number of nodes in R and a fixed threshold $maxY$, which defines the maximum number of nodes processed at most at each iteration. Finally, it performs a Map-Reduce step for distributing the growing process of the tree. The v^{th} Map-Task, first, loads the v^{th} chunk of the training set and then, for each node NT_y in R_h, initializes a vector $D_{v,y}$ of $|D| = \sum_{\forall f \in F} T_f$ instances. For each attribute of each instance of the chunk, the Map-Task updates all $D_{v,y}$ vectors by exploiting Eq. 9 or Eq. 11 in case the attribute is continuous or categorical, respectively, and then, for each node, outputs the key-value pair $\langle key = y, value = D_{v,y}\rangle$, where y is the index of the y^{th} node in R_h. At the end of the map phase, each element of $D_{v,y}$ stores the cardinality of each attribute value from the root to NT_y only for the instances in the v^{th} chunk. Each Reduce-Task is fed by a list, say $List(D_{v,y})$, of vectors

$D_{v,y}$ and creates a vector D_y by performing an element-wise addition of all V vectors in $List(D_{v,y})$. Thus, D_y stores the cardinality of each attribute value from the root to NT_y along the overall training set. Then, the Reduce-Task generates and outputs the child nodes by employing multi-way or binary splitting methods, respectively. The children generated from each NT_y are finally used to update the tree and R: if a child node is not labeled as leaf, then it is inserted into the list and employed at the next iterations. The algorithm repeats all the steps until R is empty. Space and time complexities of the Map phase are $O(\lceil\frac{V}{Q}\rceil \cdot N/V)$ and $O(\lceil\frac{V}{Q}\rceil \cdot (N \cdot Y \cdot log(|D|)/V))$, respectively. For the Reduce phase, space and time complexities are $O(Y/Q)$ and $O(Y \cdot |allSplits|/Q)$, respectively, where $|allSplits|$ is the number of splits that have to be investigated for computing the best split among all attributes for the node. Note that $|allSplits| = F$ and $|allSplits| = |D|$ for multi-way and binary splitting approaches, respectively. Since time complexity of the Map phase represents the heaviest part of the computational cost, the time complexity of Algorithm 5 is $O(H \cdot (\lceil\frac{V}{Q}\rceil \cdot (N \cdot log(|D|)/V)))$.

Algorithm 4 Distributed Fuzzy Decision Tree Learning.

Require: stopping method $stopMet$.
procedure FDTLEARNING(**in**:$stopMet$, $maxY$)
    $tree \leftarrow$ create $root$
    $R \leftarrow$ create list and insert $root$
    repeat
        $R_h \leftarrow$ get nodes from R
        $children \leftarrow$ DISTRIBUTEDNODESPLITTING
        for each $child$ in $children$ **do**
            $tree \leftarrow$ update model with $child$
            if ISNOTLEAF($child$, $stopMet$) **then**
                $R \leftarrow$ insert $child$
            end if
        end for
    until R is not empty
    return $tree$
end procedure

The proposed distributed approach allows managing a large amount of data: performing the splitting on a group of nodes significantly reduces the number of scans over the training set, but also requires a larger quantity of memory and a longer computation time for each iteration (the computational cost is limited by collecting and aggregating the necessary statistics). Thus, the maximum number $maxY$ of nodes, which can be processed in parallel at each iteration, depends on the memory availability on the cluster. Obviously, the higher the number of categorical values and fuzzy sets defined by the fuzzy partitioning, the higher the memory used for collecting the statistics and the lower the number of nodes that can be processed in parallel at each iteration.

V. EXPERIMENTAL STUDY

We performed several experiments for investigating the behavior of the proposed approach, focusing on the following three crucial aspects: i) performance in terms of classification accuracy, model complexity, and execution time; ii) scalability with a complete dataset, varying the number of CUs; iii) ability to efficiently accommodate an increasing dataset size.

Algorithm 5 Distributed Node Splitting.

Require: TR split into V $chunk_v$, splitting method $splitMet$.
procedure MAP-TASK(**in**: $chunk_v$, R_h)
    for each node NT_y in R_h **do**
        $D_{v,y} \leftarrow$ create a vector of $|D|$ elements
        for each instance $\mathbf{x}_n$ in $chunk_v$ **do**
            $D_{v,y} \leftarrow$ update statistics with $x_{f,n}$ according to Eq. 9 or Eq. 11
        end for
        output $\langle key = y, value = D_{v,y}\rangle$
    end for
end procedure
procedure REDUCE-TASK(**in**: y, $List(D_{v,y})$)
    $D_y \leftarrow$ element-wise addition of $List(D_{v,y})$
    if $splitMet$ is multiple splitting **then**
        $children \leftarrow$ MULTISPITTING(NT_y, D_y)
    else
        $children \leftarrow$ BINARYSPITTING(NT_y, D_y)
    end if
    output $\langle key = y, value = children\rangle$
end procedure

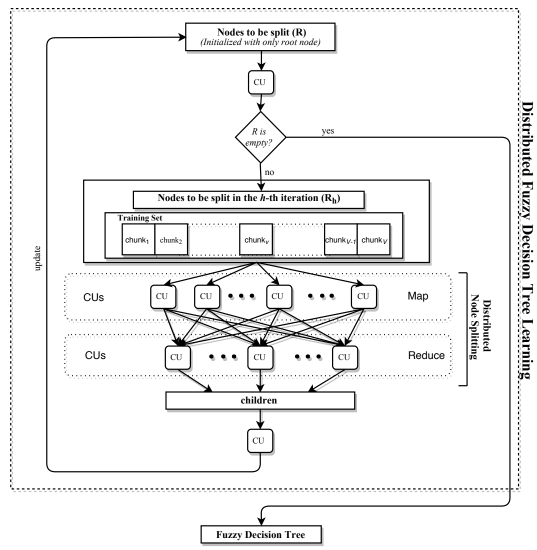


Fig. 6. The overall DFDT Learning approach.

As shown in Table I, we employed 10 well-known big datasets freely available from the UCI[2] repository. The datasets are characterized by different numbers of input/output instances (from 1 million to 11 millions), classes (from 2 to 50), and attributes (from 10 to 41). For each dataset, we also report the number of numeric (*num*) and categorical (*cat*) attributes.

All the experiments have been executed on a cluster consisting of one master equipped with a 4-core CPU (Intel Core i5 CPU 750 x 2.67 GHz), 8 GB of RAM and a 500GB Hard Drive, and three slave nodes equipped with a 4-core CPU with Hyperthreading (Intel Core i7-2600K CPU x 3.40 GHz, 8 threads), 16GB of RAM and a 1 TB Hard Drive. All nodes

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

TABLE I. BIG DATASETS USED IN THE EXPERIMENTS.

Dataset	# Instances	# Attributes	# Classes
ECO_E (ECO_E)	4,178,504	16 (num:16)	10
ECO_CO (ECO_CO)	4,178,504	16 (num:16)	21
EM_E (EM_E)	4,178,504	16 (num:16)	10
EM_M (EM_M)	4,178,504	16 (num:16)	50
Higgs (HIG)	11,000,000	28 (num:28)	2
KDDCup 1999 2 Classes (KDD99_2)	4,856,151	41 (num:26, cat:15)	2
KDDCup 1999 5 Classes (KDD99_5)	4,898,431	41 (num:26, cat:15)	5
KDDCup 1999 (KDD99)	4,898,431	41 (num:26, cat:15)	23
Poker-Hand (POK)	1,025,010	10 (cat:10)	10
Susy (SUS)	5,000,000	18 (num: 18)	2

are connected by a Gigabit Ethernet (1 Gbps) and run Ubuntu 12.04. The algorithm has been deployed upon Apache Spark 1.5.2 as data-processing framework: the master hosts the *driver program*, while each slave runs an *executor*. The training sets are stored in the HDFS.

A. Performance analysis

In this section, we analyze the performance of both FMDT and FBDT in terms of accuracy, model complexity, and execution time and compare both of them with the Distributed Decision Tree (DDT) available in MLlib [66] and with Chi-FRBCS-BigData[3] [45]. DDT performs a recursive binary partitioning of the attribute space. The partitions of the continuous attributes are generated by dividing each attribute into equi-frequency bins (at most $maxBins$) over a sampled fraction of the data. Then, at each decision node, the best split is chosen by selecting the one that maximizes the information gain. Entropy or Gini index can be used for computing impurity of the node. Further, a maximum depth $maxDepth$ of the tree can be fixed by the user. Chi-FRBCS-BigData was described in Section II.

Table II summarizes, for each algorithm, the parameters used in the experiments. For FMDT, we limit the number of fuzzy sets defined for each attribute during the fuzzy partitioning process by forcing that the support of each fuzzy set contains at least $\phi = 0.02 \cdot N$ and the number of instances belonging to each node is at least $\lambda = 10^{-4} \cdot N$. We have performed several experiments varying ϕ from $0.01 \cdot N$ to $0.1 \cdot N$ with step $0.01 \cdot N$, and λ from $10^{-5} \cdot N$ to $10^{-3} \cdot N$, with step $10^{-5} \cdot N$. We have observed that the best accuracy is just achieved with $\phi = 0.02 \cdot N$ and $\lambda = 10^{-4} \cdot N$. In practice, we have verified that smaller supports tend to fragment the data too quickly, leaving insufficient instances at the deepest nodes of the tree. After a limited number of levels, it is unlikely to execute further splits. On the other hand, wider supports do not allow obtaining satisfactory fuzzy partitions. Further, higher and lower values of λ lead to a classifier, respectively, excessively general and specialized on the training set, penalizing the performance on the test set. Also, lower values for ϕ and λ increase the overall run-time with no real advantage.

Binary splitting overcomes the previous discussed drawbacks. Thus, for FBDT no specific limitation is imposed and we set $\phi = \lambda = 1$. For both FMDT and FBDT, we used $\gamma = 0.1\%$ as suggested by authors in [35] and product as T-norm. As regards DDT, we adopted the values suggested in the guidelines provided with the library. As regards Chi-FRBCS-BigData, we used the Chi-FRBCS-BigData-Ave version, which, in the reduce phase, fuses the fuzzy rules generated in the map phase by computing the average of the rule weights. In detail, this version first searches for rules with the same antecedent. Then, it computes the average weight of the rules that have the same consequent. Finally, it keeps in the final rule base the rule with the largest average weight. As shown in [45] the Chi-FRBCS-BigData-Ave version achieves more accurate results than the Chi-FRBCS-BigData-Max version. Further, as suggested in [45], we adopted three fuzzy labels for each attribute, the product as T-norm, the winning rule as fuzzy reasoning method, the Penalized Certain Factor (PCF) [63] as rule weight, and 128 mappers.

[3] Code available at https://github.com/saradelrio/Chi-FRBCS-BigData-Ave

TABLE II. VALUES OF THE PARAMETERS FOR EACH ALGORITHM USED IN THE EXPERIMENTS.

Method	Parameters
FMDT	$\gamma = 0.1\%, \phi = 0.02 \cdot N, \lambda = 10^{-4} \cdot N, TN = product$
FBDT	$\gamma = 0.1\%, \phi = 1, \lambda = 1, TN = product$
DDT	$maxBins = 32, Impurity = Entropy$
Chi-FRBCS-BigData	$NumFuzzyLabels = 3, TN = product,$ $reasoningMethod = winningRule, ruleWeight = PCF$

For each dataset and for each algorithm, we performed a five-fold cross-validation by using the same folds for all the datasets and, for the three decision trees, varying the maximum depth β of the tree. Table III shows, for each dataset and for each algorithm, the average values $\pm$ standard deviation of the accuracy, both on the training ($AccTr$) and test ($AccTs$) sets obtained by the algorithms. The highest accuracy values for each dataset are shown in bold. Table IV shows the complexity of each algorithm. For each dataset and for each decision tree learning algorithm, we report the average values of the number of nodes ($\#Nodes$), number of leaves ($\#Leaves$) and minimum, maximum and average depths ($Depth$) of the trees, expressed as (min,max,avg) in the table. For the Chi-FRBCS-BigData dataset, we report the average value of the number ($\#Rules$) of rules.

The analysis of the three tables highlights that, on average, both FMDT and FBDT outperform DDT and Chi-FRBCS-BigData. With respect to the FDTs, we can observe that, when comparing trees of the same depth, the multi-way splitting tends to achieve higher accuracy because it is able to investigate a higher number of correlations between attributes by generating a higher number of nodes at each level. On the other hand, as shown in Table IV, the trees are characterized by a significantly higher number of nodes and therefore are more complex. For instance, for ECO_CO, ECO_E, EM_E, EM_M, HIG and SUS, FMDT employs more than 160,000 leaves with only five levels of depth. For higher values of β, the algorithm generates too many nodes and the overall process takes an unreasonable amount of time. For this reason, no result for higher values of β has been reported in Table III. However, we can observe that for $\beta = 5$, FMDT achieves accuracy comparable to the one obtained by the other algorithms. On the other hand, FBDT and DDT are able to generate deeper trees. Note that deeper trees are more expressive and achieve higher accuracy on the training set, but are typically also affected by a higher probability of over-training. However, FBDT tends to be more tolerant to over-training than DDT. In particular, unlike DDT, for $\beta = 15$, FBDT achieves results

TABLE III. AVERAGE ACCURACY ± STANDARD DEVIATION ACHIEVED BY FMDT, FBDT, DDT AND CHI-FRBCS-BIGDATA.

		FMDT		FBDT		DDT		Chi-FRBCS-BigData	
Dataset	β	Acc_{Tr}	Acc_{Ts}	Acc_{Tr}	Acc_{Ts}	Acc_{Tr}	Acc_{Ts}	Acc_{Tr}	Acc_{Ts}
ECO_E	5	**97.641** ± 0.019	**97.585** ± 0.041	78.244 ± 0.015	78.242 ± 0.037	77.718 ± 0.765	77.721 ± 0.729		
	10	–	–	89.347 ± 0.105	89.335 ± 0.142	88.099 ± 0.164	88.082 ± 0.179	54.454 ± 7.813	54.485 ± 7.806
	15	–	–	97.315 ± 0.025	97.262 ± 0.045	95.874 ± 0.269	95.756 ± 0.286		
ECO_CO	5	97.559 ± 0.001	97.526 ± 0.023	68.049 ± 0.006	68.030 ± 0.032	68.902 ± 0.154	68.892 ± 0.173		
	10	–	–	89.375 ± 0.139	89.374 ± 0.147	88.046 ± 0.476	88.039 ± 0.475	73.639 ± 6.415	73.604 ± 6.406
	15	–	–	**97.847** ± 0.027	**97.795** ± 0.022	96.670 ± 0.164	96.563 ± 0.161		
EM_E	5	96.962 ± 0.008	96.913 ± 0.018	77.381 ± 0.245	77.354 ± 0.303	77.270 ± 1.569	77.254 ± 1.592		
	10	–	–	90.751 ± 0.051	90.705 ± 0.051	89.756 ± 0.171	89.729 ± 0.186	67.285 ± 8.962	67.277 ± 8.952
	15	–	–	**96.991** ± 0.021	**96.928** ± 0.032	95.856 ± 0.203	95.726 ± 0.194		
EM_M	5	96.078 ± 0.008	96.001 ± 0.026	74.311 ± 0.045	74.319 ± 0.011	72.484 ± 0.315	72.493 ± 0.312		
	10	–	–	91.044 ± 0.077	91.036 ± 0.085	90.061 ± 0.243	90.062 ± 0.251	93.179 ± 7.629	92.707 ± 7.601
	15	–	–	**96.879** ± 0.024	**96.746** ± 0.023	95.894 ± 0.040	95.669 ± 0.058		
HIG	5	72.638 ± 0.018	71.253 ± 0.029	66.451 ± 0.013	66.441 ± 0.025	66.344 ± 0.080	66.335 ± 0.106		
	10	–	–	70.723 ± 0.013	70.697 ± 0.022	70.481 ± 0.040	70.403 ± 0.063	55.933 ± 0.080	55.897 ± 0.119
	15	–	–	72.631 ± 0.019	**72.266** ± 0.008	**73.073** ± 0.031	71.871 ± 0.013		
KDD99_2	5	99.986 ± 0.006	99.986 ± 0.005	99.989 ± 0.000	99.987 ± 0.000	99.980 ± 0.008	99.979 ± 0.008		
	10	–	–	99.999 ± 0.000	**99.999** ± 0.000	99.999 ± 0.001	**99.999** ± 0.001	99.934 ± 0.001	99.933 ± 0.002
	15	–	–	99.999 ± 0.000	**99.999** ± 0.000	**100.000** ± 0.000	**99.999** ± 0.000		
KDD99_5	5	99.976 ± 0.002	99.973 ± 0.003	99.893 ± 0.000	99.894 ± 0.002	99.669 ± 0.010	99.882 ± 0.010		
	10	–	–	99.995 ± 0.000	99.992 ± 0.001	99.991 ± 0.001	99.989 ± 0.001	96.395 ± 0.043	96.302 ± 0.044
	15	–	–	**99.999** ± 0.000	**99.995** ± 0.000	**99.999** ± 0.001	99.994 ± 0.001		
KDD99	5	99.950 ± 0.001	99.948 ± 0.002	99.597 ± 0.008	99.598 ± 0.009	99.669 ± 0.104	99.669 ± 0.103		
	10	–	–	99.990 ± 0.000	99.971 ± 0.001	99.991 ± 0.001	99.989 ± 0.001	99.988 ± 0.001	99.610 ± 0.010
	15	–	–	99.997 ± 0.000	**99.994** ± 0.001	**99.999** ± 0.000	99.993 ± 0.001		
POK	5	78.479 ± 0.031	**77.176** ± 0.068	54.708 ± 0.405	54.696 ± 0.432	54.708 ± 0.405	54.696 ± 0.432		
	10	–	–	58.806 ± 0.508	58.490 ± 0.599	58.806 ± 0.508	58.490 ± 0.599	**99.711** ± 0.002	5.178 ± 0.049
	15	–	–	67.553 ± 0.422	62.479 ± 0.504	67.553 ± 0.422	62.479 ± 0.504		
SUS	5	**80.962** ± 0.007	79.639 ± 0.016	77.312 ± 0.060	77.230 ± 0.057	77.023 ± 0.025	77.018 ± 0.038		
	10	–	–	79.118 ± 0.016	79.091 ± 0.024	79.022 ± 0.043	78.940 ± 0.052	55.747 ± 0.110	55.751 ± 0.157
	15	–	–	79.969 ± 0.030	**79.722** ± 0.043	80.393 ± 0.026	79.304 ± 0.032		

TABLE IV. COMPLEXITIES OF FMDT, FBDT, DDT AND CHI-FRBCS-BIGDATA.

		FMDT			FBDT			DDT			Chi-FRBCS-BigData
Dataset	β	$\#Nodes$	$\#Leaves$	$Depth$	$\#Nodes$	$\#Leaves$	$Depth$	$\#Nodes$	$\#Leaves$	$Depth$	$\#Rules$
ECO_E	5	222,694	200,048	(2,5,2.73)	63	32	(5,5,5)	63	32	(5,5,5)	
	10	-	-	-	1,695	849	(6,10,9.87)	1,530	765	(6,10,9.78)	4,148
	15	-	-	-	17,532	8,741	(6,15,14.23)	12,323	6,162	(6,15,13.99)	
ECO_CO	5	190,637	169,621	(2,5,2.38)	63	32	(5,5,5)	63	32	(5,5,5)	
	10	-	-	-	1,746	872	(6,10,9.88)	1,552	777	(5,10,9.80)	25,717
	15	-	-	-	18,785	9,370	(6,15,14.26)	13,827	6,914	(5,15,14.06)	
EM_E	5	240,406	218,557	(5,5,5)	63	32	(5,5,5)	63	32	(5,5,5)	
	10	-	-	-	1,694	847	(5,10,9.88)	1,702	851	(6,10,9.86)	13,711
	15	-	-	-	20,996	10,477	(5,15,14.38)	14,515	7,258	(5,15,14.11)	
EM_M	5	218,562	196,344	(2,5,2.76)	63	32	(5,5,5)	63	32	(5,5,5)	
	10	-	-	-	1,792	897	(6,10,9.90)	1,521	761	(5,10,9.81)	665,160
	15	-	-	-	23,022	11,495	(6,15,14.40)	18,900	9,451	(5,15,14.25)	
HIG	5	972.779	920,942	(2,5,3.30)	63	32	(5,5,5)	63	32	(5,5,5)	
	10	-	-	-	1,686	844	(5,10,9.89)	2,045	1,023	(9,10,9.99)	24,058
	15	-	-	-	34,444	17,209	(5,15,14.79)	49,822	24,911	(9,15,14.80)	
KDD99_2	5	703	630	(2,5,2.54)	41	21	(3,5,4.62)	37	19	(3,5,4.50)	
	10	-	-	-	131	66	(3,10,7.76)	95	48	(3,10,7.12)	1,020
	15	-	-	-	222	112	(3,15,10.18)	121	61	(3,15,8.18)	
KDD99_5	5	2,716	2,351	(2,5,2.60)	46	24	(2,5,4.83)	49	25	(2,5,4.83)	
	10	-	-	-	335	168	(2,10,8.78)	356	179	(2,10,8.70)	11,585
	15	-	-	-	779	389	(2,15,11.68)	544	272	(2,15,10.65)	
KDD99	5	2,164	1,875	(2,5,2.79)	37	19	(2,5,4.63)	40	20	(2,5,4.83)	
	10	-	-	-	369	185	(2,10,9.03)	303	152	(2,10,8.58)	102,014
	15	-	-	-	972	485	(2,15,12.11)	581	291	(2,15,10.94)	
POK	5	30,940	28,561	(4,4,4)	63	32	(5,5,5)	63	32	(5,5,5)	
	10	-	-	-	2,024	1,012	(9,10,9.99)	2,024	1,012	(9,10,9.99)	813,193
	15	-	-	-	44,297	22.149	(9,15,14.75)	44,297	22,149	(9,15,14.75)	
SUS	5	805,076	758,064	(2,5,3.46)	63	32	(5,5,5)	63	32	(5,5,5)	
	10	-	-	-	1,360	681	(5,10,9.76)	1,984	993	(8,10,9.98)	678
	15	-	-	-	21,452	10,723	(5,15,14.62)	35,133	17,567	(8,15,14.59)	

comparable to FMDT on both training and test sets, with the only exception for POK. We have to consider that the attributes in POK are categorical. As explained in Section IV-B, FBDT employs the method proposed in [69] for limiting the number of candidate splits when managing categorical attributes. The method determines an approximation of the optimal split and the error generated by such approximation is propagated to the child nodes. Thus, deeper trees are more affected by this problem. In general, however, both FMDT and FBDT do not particularly suffer from over-training. Indeed, the difference between the classification rates obtained on the training and test sets is quite limited for all the datasets, except for HIG, POK and SUS for FMDT, and POK for FBDT. Actually, for HIG and SUS, FMDT generates trees with a very high number of nodes and leaves, and therefore particularly specialized on the training set.

As regards Chi-FRBCS-BigData, we can observe that the accuracy is strongly dependent on the specific dataset, but in general much lower than the one obtained by the three decision trees. In particular, we note that, for the POK dataset, Chi-FRBCS-BigData suffers from a strong over-training, probably due to the high number (13) of possible different categorical values for five attributes, which brings the learning algorithm to generate a very high number of rules and therefore to specialize very much the classifier on the training set. In all datasets, except for the three KDD datasets, the classification rates both on the training and test sets are much lower that the ones obtained by the other three approaches. As regards the complexity, it is well-known in the literature that a decision tree can be expressed as a rule base, with the number of rules equal to the number of leaves. We have to consider however that the rules extracted from the binary decision trees are different from those extracted from the multi-way decision tree and from those in the rule base generated by Chi-FRBCS-BigData. Indeed, the rules extracted from the binary decision trees can have conditions that are expressed by using “or” of categorical values rather than a unique value. Although considering this difference, we note that the complexity of the fuzzy rule-based classifiers generated by Chi-FRBCS-BigData is much higher than the one of the decision trees, except for three datasets, namely ECO_E, HIG and SUS.

To statistically compare the four approaches, for each algorithm, we generate a distribution consisting of the mean values of the accuracy of solutions on the test set by using all the datasets. Then, we apply the Friedman test in order to compute a ranking among the distributions [70], and the Iman and Davenport test [71] to evaluate whether there exists a statistical difference among the distributions. If the Iman and Davenport p-value is lower than the level of significance α (in the experiments α = 0.05), we can reject the null hypothesis and affirm that there exist statistical differences between the multiple distributions associated with each approach. Otherwise, no statistical difference exists. If there exists a statistical difference, we apply a post-hoc procedure, namely the Holm test [72]. This test allows detecting effective statistical differences between the control approach, i.e. the one with the lowest Friedman rank, and the remaining approaches.

In Table V we show the Friedman rank and the Iman and Davenport p-value for each algorithm (we consider the results for $\beta = 15$ for both FBDT and DDT). We observe that the statistical hypothesis of equivalence is rejected. Thus, we apply the Holm post-hoc procedure considering FBDT as control algorithm (associated with the lowest rank and in bold in the Table). As shown in Table VI, we observe that the FBDT statistically outperforms FMDT, DDT and Chi-FRBCS-BigData.

For the sake of completeness, we mention that the classification rates of both FMDT and FBDT are also higher than the ones reported in [33] [35]. In [33], the authors investigate several prototype reduction techniques on Apache Hadoop with the aim of improving the classification rates of the nearest neighbor classifier. The experimental results on three big datasets have proven that these methods are very competitive in reducing the computational cost and high storage requirements of the nearest neighbor classifier, improving its classification performance. In [35], the authors have proposed MRAC+, a fast MapReduce associative classifier based on frequent pattern mining on Apache Hadoop. The experimental results performed on seven big datasets show that MRAC+ obtains comparable performance in terms of accuracy to DDT and is able to achieve speedup and scalability close to the ideal ones. Due to the limited number of datasets adopted by the authors, we have not shown the results in Table III, but however, we highlight that the average accuracy achieved by FMDT and FBDT in the common datasets is higher than the one obtained by the algorithms proposed in [33] [35]. The unique exception is the POK dataset, where MRAC+ achieves an average accuracy of 94.480%. We recall that POK contains only categorical attributes and associative classifiers have proved to perform particularly well on this type of datasets [35].

Table VII shows the main characteristics of the partitions obtained by applying the fuzzy partitioning approach. In particular, the table reports the average number ($\overline{NFS}$) of fuzzy sets determined for the continuous attributes, the number of fuzzy sets for the attributes with the lowest (min_{NFS}) and highest (max_{NFS}) numbers of fuzzy sets, and the number DA of attributes discarded by the fuzzy partitioning process. Obviously, for POK, which is characterized by only categorical attributes, fuzzy partitioning is not performed.

TABLE VII. COMPLEXITIES OF FUZZY PARTITIONING FOR BOTH FMDT AND FBDT.

	FMDT				FBDT			
Dataset	$\overline{NFS}$	min_{NFS}	max_{NFS}	DA	$\overline{NFS}$	min_{NFS}	max_{NFS}	DA
ECO_E	36.625	35	41	0	180.05	91	257	0
ECO_CO	35.613	32	41	0	184.75	93	273	0
EM_E	36.875	35	42	0	176.225	98	245	0
EM_M	34.863	35	39	0	165.55	96	206	0
HIG	8.229	3	32	6	10.136	3	42	6
KDD99_2	2.654	3	15	4	9.315	3	31	0
KDD99_5	3.3	3	15	4	15.131	3	42	0
KDD99	3.269	3	15	4	14.962	3	41	0
SUS	13.989	5	25	3	18.9	5	45	3

As shown in Table VII, for ECO_CO, ECO_E, EM_E, EM_M, HIG and SUS, fuzzy partitioning generates a high number of fuzzy sets, making the partitions hardly interpretable. To limit the number of fuzzy sets, a possible solution is to increment the value of ϕ as exploited for FMDT. On the other hand, the parameter can affect the number of attributes discarded from the fuzzy partitioning. For instance, unlike FBDT, for KDD99_2, KDD99_5 and KDD99, the algorithm removes 4 attributes that will be not employed by the FMDT.

Table VIII summarizes the execution times (in seconds) of each approach. For all approaches, we show the execution time of the learning process (*Learning*), and only for FMDT and FBDT, the execution time of the fuzzy partitioning process (*FP*) and the overall execution time (*Tot*). Here, the datasets have been split into a number of chunks equal to the number of cores available in the cluster, so that each core processes more or less the same number of instances. DDT is much faster than the two DFDTs: the execution time of DDT is more than one order of magnitude lower than the one of the two DFDTs. This is mainly due to two factors. First, the total execution time of FMDT and FBDT is affected by the fuzzy partitioning process. Such process is not performed by DDT. Second, the amount of information managed by the FDT learning is higher than the one managed by the DDT learning. Indeed, since each value $x_{f,n} \in U_f$ belongs to two fuzzy sets, space complexity of FDT learning step is, in the worst case, twice than the one of DDT. The overall execution time of FBDT is comparable with the one of FMDT. In particular, as shown in Table VII, although FBDT employs a lower number of nodes than FMDT, it evaluates different binary splits for each attribute. However, the choice of the best split is bounded by the number of fuzzy sets defined on the attribute, which is

TABLE V. FRIEDMAN RANK AND IMAN AND DAVENPORT P-VALUE FOR FMDT, FBDT, DDT AND CHI-FRBCS-BIGDATA

Algorithm	*Friedman rank*	*Iman and Davenport p-value*	*Hypothesis*
FBDT	1.3		
FMDT	2.2	0.000047	Rejected
DDT	2.5		
Chi-FRBCS-BigData	4		

TABLE VI. HOLM POST HOC PROCEDURE FOR $\alpha = 0.05$

i	algorithm	z-value	p-value	alpha/i	Hypothesis
3	Chi-FRBCS-BigData	4.676537	0.000003	0.016667	Rejected
2	DDT	2.683282	0.00729	0.025	Rejected
1	FMDT	2.012461	0.044171	0.05	Rejected

significantly lower than the number of instances. On the other hand, FMDT can perform only one split for each attribute for a given node, thus speeding up the computation of the splitting procedure. Chi-FRBCS-BigData is much slower than the other approaches. We have to consider however that Chi-FRBCS-BigData is implemented on Apache Hadoop, which has proved to be less efficient than Apache Spark, although the operations performed by Chi-FRBCS-BigData should not particularly suffer from the inefficiencies of Apache Hadoop.

B. Scalability analysis

In this section, we investigate the scalability of the proposed approaches by employing an increasing number of CUs. To this aim, we measure the values assumed by the *speedup* σ that represents the main metrics used in parallel computing. According to the speedup definition, the efficiency of a program using multiple CUs is calculated comparing the execution time of the parallel implementation against the corresponding sequential version. Unfortunately, due to the large size of the involved datasets, the sequential version of the overall algorithm would take an unreasonable amount of time. Thus, for the scalability analysis we refer to a run over Q^* identical CUs, with $Q^* > 1$. With this aim, we adopt the following slightly different definition for the speedup on n identical CUs:

$$\sigma_{Q^*}(n) = \frac{Q^* \cdot \tau(Q^*)}{\tau(n)} \tag{14}$$

where $\tau(n)$ is the run-time using n CUs, and Q^* is the number of CUs used to run the reference execution, which lets us estimate a fictitious, ideal single-core run-time as $Q^* \cdot \tau(Q^*)$. Of course, $\sigma_{Q^*}(n)$ makes sense only for $n \geq Q^*$. Note that $\tau(Q^*)$ accounts also for the basic overhead due to the Apache Spark platform. Obviously, for $n > Q^*$ the speedup is expected to be sub-linear due to the increasing overhead from the Spark tasks, the behavior of the algorithm (considering also the granularity of the necessary sequential parts) and the contention for shared resources. In our tests, we assumed $Q^* = 8$ so as to have 1 working slave available in the cluster and thus accounting in σ_8 also for the basic overhead due to thread interference. Horizontal scalability has been studied by varying the number of switched-on CUs: we vary the number of slaves from 1 to 3, each with one executor with 8 cores. Considering the structure of our approach, we split the RDD into a number of partitions equal to the total number of cores available on the cluster.

Table IX summarizes the results obtained on the Susy dataset by FBDT with $\beta = 15$. For the sake of brevity, we considered only one dataset and FBDT. However, similar results can be obtained on the other datasets and/or using FMDT.

TABLE IX. RUN-TIME, SPEEDUP (σ_8), AND UTILIZATION ($\sigma_8(Q)/Q$) OF BOTH FUZZY PARTITIONING AND FBDT LEARNING PROCESSES FOR THE SUSY DATASET.

	Fuzzy Partitioning			**Learning**		
# Cores	**Time (s)**	$\sigma_8(Q)$	$\sigma_8(Q)/Q$	**Time (s)**	$\sigma_8(Q)$	$\sigma_8(Q)/Q$
8	185	8	1.00	636	8	1.00
16	141	10.50	0.66	324	15.70	0.98
24	153	9.67	0.40	230	22.12	0.92

The actual speedup shows a different behavior depending on the algorithm. As regards Fuzzy Partitioning, σ_8 rapidly decreases and using 24 cores does not produce a real advantage; indeed the execution time with 24 cores is higher than the one obtained by using 16 cores. The result is mainly affected by two factors. First, the number of bins, namely $\Omega = V \cdot \gamma$, used to split the domain of each attribute is equal to 8,000, 16,000 and 24,000 for 8, 16 and 24 cores, respectively. Thus, in case of 24 cores, the amount of information handled by the algorithm is higher than the one handled for the other experiments, affecting the overall execution time. Second, the fuzzy partitioning of each continuous attribute is distributed among the cores available in the cluster so that each attribute is assigned to one core. Since Susy is characterized by 18 continuous attributes, each core processes approximately 3, 2 and 1 attributes in case of 8, 16 and 24 cores, respectively. However, as shown in Table VII, three attributes are discarded by the fuzzy partitioning process, thus for such attributes the overall process is performed in a few milliseconds (it requires exactly one scan for the exploration of candidate fuzzy partitions). Considering this result, the overall execution time can be roughly approximated with the same time required for 18-3=15 continuous attributes, thus each core processes approximately 2, 1 and 1 attributes in case of 8, 16 and 24 cores, respectively. The result highlights that, as regards the distribution of the computational flow, using a number of cores higher than 16 does not produce a real advantage and in such cases the execution time is only affected by the number of bins employed to explore the candidate fuzzy partitions.

As regard FBDT learning, σ_8 does not excessively diverge from the linear trend, i.e. the number of CUs: $\sigma_8(16)/16 = 0.98$ and $\sigma_8(24)/24 = 0.92$. The overhead is mainly due to higher number of executors handled by the Spark frameworks and the communication cost required to send the nodes that must be split from the master to the slaves.

C. Dealing the dataset size

From a practical point of view, it is crucial to understand how the proposed algorithms behave as the size of the input dataset increases. To evaluate this aspect, we have performed several experiments using different dataset sizes. We have employed the Susy dataset and have used different percentages of this dataset. We indicate with the notation $Susy_x$ the dataset

TABLE VIII. THE EXECUTION TIMES (IN SECONDS) FOR FMDT, FBDT, DDT AND CHI-FRBCS-BIGDATA

		FMDT			FBDT			DDT	Chi-FRBCS-BigData
Dataset	β	FP	$Learning$	Tot	FP	$Learning$	Tot	$Learning$	$Learning$
ECO_E	5	28	364	392	29	64	93	11	1,263
	10	-	-	-	29	215	244	13	
	15	-	-	-	29	691	720	16	
ECO_CO	5	88	691	779	36	47	83	17	1,491
	10	-	-	-	36	180	216	18	
	15	-	-	-	36	1,034	1070	29	
EM_E	5	23	349	372	24	42	66	11	1,276
	10	-	-	-	24	138	162	13	
	15	-	-	-	24	579	603	17	
EM_M	5	58	3,947	4,005	48	39	87	15	62,175
	10	-	-	-	48	174	222	21	
	15	-	-	-	48	3,868	3,916	67	
HIG	5	180	706	886	180	131	311	130	19889
	10	-	-	-	180	224	404	132	
	15	-	-	-	180	424	604	149	
KDD99_2	5	15	17	32	21	26	47	15	2,756
	10	-	-	-	21	49	70	16	
	15	-	-	-	21	68	89	17	
KDD99_5	5	16	24	40	30	41	71	17	3,615
	10	-	-	-	30	67	97	20	
	15	-	-	-	30	86	116	21	
KDD99	5	16	22	38	46	41	87	17	6,551
	10	-	-	-	46	67	113	19	
	15	-	-	-	46	78	124	20	
POK	5	-	3	3	-	4	4	4	18,918
	10	-	-	-	-	6	6	6	
	15	-	-	-	-	11	11	11	
SUS	5	122	133	255	126	22	148	47	1,444
	10	-	-	-	126	66	192	49	
	15	-	-	-	126	130	255	54	

composed with x% of instances of the Susy dataset (the complete dataset is $Susy_{100}$). Moreover, we limit the experiments only to FBDT with $\beta = 15$ but similar considerations can be applied to FMDT.

Table X shows the run-time (in seconds) for building the tree (including the fuzzy partitioning), according to different dataset sizes. We report also the total number of instances N and the total number of instances in each chunk $N_v = N/V$. Like in the previous experiments, we distribute uniformly the entire dataset upon the number of available cores, i.e. $V = Q = 24$ in our tests. Note that for $Susy_{50}$, the average run-time of three different experiments executed over three distinct subsets of Susy (with instances randomly sampled) is reported.

TABLE X. RUN-TIME (IN SECONDS) OF FBDT ON THE SUSY DATASET, VARYING THE DATASET SIZE.

Dataset			FBDT		
Size (%)	N	N_v	Fuzzy Partitioning	Learning	Tot
50 ($Susy_{50}$)	2,500,000	104,167	124	111	238
100 ($Susy_{100}$)	5,000,000	208,333	153	230	383
200 ($Susy_{200}$)	10,000,000	416,667	204	477	681
300 ($Susy_{300}$)	15,000,000	625,000	255	800	1055

The execution time of the two algorithms increases with different trends. However, the results are consistent with the time complexity analysis described in Section IV-C. As regards Fuzzy Partitioning, the computational cost is mainly driven by the number of bins Ω employed to explore the candidate fuzzy partitions. Since such value is constant in all tests, i.e. $\Omega = 24,000$, the execution time of the two reduce phases of fuzzy partitioning is more or less the same in all experiments. On the other hand, both map phases depend on the number of instances processed by each Map-Task. We recall that the first Map-Task performs a sorting of the instances for retrieving the equi-frequency bins and the second Map-Task computes for each bin the number of instances belonging to the different classes. Such operations are performed in $O(F \cdot N_v \cdot log(N_v))$ and $O(F \cdot N_v \cdot log(\Omega))$, respectively. However, considering the experiments and the number of instances involved, Ω and F are constants and $log(N_v)$ assumes more or less the same values (i.e. $log(N_v)$ ranges from about 5.02 to 5.8). Thus, we can expect that the run-time trend for both Map-Tasks is slightly higher than the linear one. These observations can be used to get a very rough estimation of the run-time expected for different dataset sizes. For instance, if adding 2,500,000 instances (from $Susy_{50}$ to $Susy_{100}$), the run-time increases of $153 - 129 = 24$ seconds, in the ideal case, we expect that adding 5,000,000 instances the execution time is slightly longer than twice. Thus we should obtain about $153+24\times2 = 201$ and $153+24\times4 = 249$ seconds for $Susy_{200}$ and $Susy_{300}$, respectively. As it can be noted, such values do not excessively differ from the measured ones. Of course, the actual run-times are necessarily higher due to the logarithmic factor $log(N_v)$ of the first Map-Task and the overheads for the sharing of memory resources.

As regards FBDT learning, we can perform the same observations exploited for Fuzzy Partitioning. In particular, as described in Section IV-C, time complexity of Reduce-Task depends only on the number of splits, which have to be evaluated for computing the best splits among all attributes for the node, and is not affected by the number of instances. On the other hand, time complexity of Map-Task is equal to $O(N_v \cdot Y \cdot log(|D|))$. Since the number of nodes to split Y and the total number of fuzzy sets $|D|$ defined by Fuzzy Partitioning are more or less the same in all experiments, the overall run-time is mainly affected by N_v. Thus, increasing the number of instances, we expect that in the ideal case the execution time trend is linear, i.e. $111\times2 = 222$, $111\times4 = 444$ and $111 \times 6 = 666$ for $Susy_{100}$, $Susy_{200}$ and $Susy_{300}$, respectively. As it can be noted, such values do not excessively differ from the measured ones. Of course, the actual run-times are necessarily higher due to the overheads for the sharing of memory resources.

VI. Conclusions

We have proposed a distributed fuzzy decision tree (FDT) learning scheme shaped according to the MapReduce programming model for generating both binary (FBDT) and multi-way (FMDT) FDTs from big data. We have first introduced a novel distributed fuzzy discretizer, which generates strong fuzzy partitions for each continuous attribute based on fuzzy information entropy. Then, we have discussed a distributed implementation of an FDT learning algorithm, which employs the fuzzy information gain for selecting the attributes to be used in the decision nodes. We have implemented the FDT learning scheme on the Apache Spark framework.

Experimental results performed on ten real-world big datasets show that our scheme is able to achieve speedup and scalability figures close to the ideal ones. It is worth highlighting that such results can be obtained without adopting any specific dedicated hardware, but rather by using just a few personal computers connected by a Gigabit Ethernet. The results have been compared with the ones obtained by the distributed decision tree (DDT) implemented in the MLlib library on the Apache Spark framework and by the Chi-FRBCS-BigData algorithm, a MapReduce distributed fuzzy rule-based classification system. In the comparison we have considered accuracy, complexity and execution time. We have shown that FBDT statistically outperforms FMDT, DDT and Chi-FRBCS-BigData in terms of accuracy. In terms of complexity, FBDT and DDT employ a lower (generally one order of magnitude) number of nodes than FMDT. Further, the number of rules, which can be extracted from the three decision trees, is on average lower than the one of Chi-FRBCS-BigData. From the run-time perspective, FBDT and FMDT require comparable computation times, but both of them are slower than DDT (not surprisingly, considering that FBDT and FMDT perform a fuzzy partitioning step and manage more information, due to fuzzy logic), but faster than Chi-FRBCS-BigData. Finally, computation time scales approximately linear with the number of computational units and instances.

As highlighted in the overall paper, the main reason for proposing FBDT and FMDT is to generate effective and efficient classifiers when managing big data. Obviously, the distribution of the dataset along the computer cluster implies the parallelization of the fuzzy decision tree learning and therefore a faster tree generation. Thus, FBDT and FMDT find a natural use in all the application domains where decision trees have to be generated very quickly from a large amount of data. As an example, the increase of the number of sensors deployed everywhere and the subsequent intent to extract useful knowledge from the data collected by these sensors, has given rise to a growing interest in data mining approaches for streaming data, possibly able to manage concept drift. Most strategies used in this context use sliding windows of fixed or variable sizes and a retraining learning mode. A window is maintained that keeps the most recently acquired examples, and from which older examples are dropped according to some set of rules. Periodically, the retraining learning mode discards the current model and builds a new model from scratch using the buffered data in the windows. An interesting survey of streaming data analysis and concept drift adaptation can be found in [73]. Our fuzzy decision tree learning algorithms result to be particularly suitable for the retraining learning mode, especially when the size of the window is particularly large.

Concluding, we believe that the work presented in this paper is the first extensive study on the application of FDTs to big data, considering both binary and multi-way splits. We expect that the experimental results can be used as baseline for future research in this field.

References

[1] R. Diao, K. Sun, V. Vittal, R. J. O'Keefe, M. R. Richardson, N. Bhatt, D. Stradford, and S. K. Sarawgi, "Decision tree-based online voltage security assessment using PMU measurements," *IEEE Transactions on Power Systems*, vol. 24, no. 2, pp. 832–839, 2009.

[2] T. Goetz, *The decision tree: Taking control of your health in the new era of personalized medicine*. Rodale Inc., 2010.

[3] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 247–256.

[4] J. Han, M. Kamber, and J. Pei, *Data mining: Concepts and techniques*. Elsevier, 2011.

[5] L. Rokach and O. Maimon, *Data mining with decision trees: Theory and applications*. World scientific, 2014.

[6] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[7] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[8] C. Z. Janikow, "Fuzzy decision trees: Issues and methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 28, no. 1, pp. 1–14, 1998.

[9] Y.-l. Chen, T. Wang, B.-s. Wang, and Z.-j. Li, "A survey of fuzzy decision tree classifier," *Fuzzy Information and Engineering*, vol. 1, no. 2, pp. 149–159, 2009.

[10] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[11] X. Liu, X. Feng, and W. Pedrycz, "Extraction of fuzzy rules from fuzzy decision trees: An axiomatic fuzzy sets (AFS) approach," *Data & Knowledge Engineering*, vol. 84, pp. 1–25, 2013.

[12] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: Data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[13] H. Kim and W.-Y. Loh, "Classification trees with unbiased multiway splits," *Journal of the American Statistical Association*, pp. 589–604, 2011.

[14] F. Berzal, J.-C. Cubero, N. Marın, and D. Sánchez, "Building multi-way decision trees with numerical attributes," *Information Sciences*, vol. 165, no. 1, pp. 73–90, 2004.

[15] Y. Yuan and M. J. Shaw, "Induction of fuzzy decision trees," *Fuzzy Sets and systems*, vol. 69, no. 2, pp. 125–139, 1995.

[16] R. Weber, "Fuzzy-ID3: A class of methods for automatic knowledge acquisition," in *Int. Conf. on Fuzzy Logic & Neural Networks*, 1992, pp. 265–268.

[17] M. Zeinalkhani and M. Eftekhari, "Fuzzy partitioning of continuous attributes through discretization methods to construct fuzzy decision tree classifiers," *Information Sciences*, vol. 278, pp. 715–735, 2014.

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

[19] S. Kotsiantis and D. Kanellopoulos, "Discretization techniques: A recent survey," *GESTS International Transactions on Computer Science and Engineering*, vol. 32, no. 1, pp. 47–58, 2006.

[20] A. D. D. Matteis, F. Marcelloni, and A. Segatori, "A new approach to fuzzy random forest generation," in *IEEE International Conference on Fuzzy Systems*, 2015, pp. 1–8.

[21] B. Chandra and P. P. Varghese, "Fuzzy SLIQ decision tree algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 5, pp. 1294–1301, 2008.

[22] C. Z. Janikow, "A genetic algorithm method for optimizing fuzzy decision trees," *Information Sciences*, vol. 89, no. 3, pp. 275–296, 1996.

[23] A. Myles and S. Brown, "Induction of decision trees using fuzzy partitions," *Journal of chemometrics*, vol. 17, no. 10, pp. 531–536, 2003.

[24] X. Cheng, X. Jin, Y. Wang, J. Guo, T. Zhang, and G. Li, "Survey on big data system and analytic technology," *J. Softw*, vol. 25, no. 9, pp. 1889–1908, 2014.

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

[26] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.

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

[28] "Apache Hadoop," https://hadoop.apache.org/, accessed: March 2016.

[29] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[30] "Apache Spark," http://spark.apache.org/, accessed: March 2016.

[31] N. K. Alham, M. Li, Y. Liu, and S. Hammoud, "A MapReduce-based distributed SVM algorithm for automatic image annotation," *Computers & Mathematics with Applications*, vol. 62, no. 7, pp. 2801–2811, 2011.

[32] G. Caruana, M. Li, and M. Qi, "A MapReduce based parallel SVM for large scale spam filtering," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, vol. 4, 2011, pp. 2659–2662.

[33] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera, "MRPR: A MapReduce solution for prototype reduction in big data classification," *Neurocomputing*, vol. 150, pp. 331–345, 2015.

[34] C. Zhang, F. Li, and J. Jestes, "Efficient parallel kNN joins for large data in MapReduce," in *Proceedings of the 15th International Conference on Extending Database Technology*, 2012, pp. 38–49.

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

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

[37] I. Palit and C. K. Reddy, "Scalable and parallel boosting with MapReduce," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 10, pp. 1904–1916, 2012.

[38] R. Wang, Y.-L. He, C.-Y. Chow, F.-F. Ou, and J. Zhang, "Learning ELM-Tree from big data based on uncertainty reduction," *Fuzzy Sets and Systems*, vol. 258, pp. 79–100, 2015.

[39] S. Wang, J. Zhai, H. Zhu, and X. Wang, "Parallel ordinal decision tree algorithm and its implementation in framework of MapReduce," in *Machine Learning and Cybernetics*, 2014, pp. 241–251.

[40] W. Dai and W. Ji, "A MapReduce implementation of C4.5 decision tree algorithm," *International Journal of Database Theory and Application*, vol. 7, no. 1, pp. 49–60, 2014.

[41] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," *Advances in neural information processing systems*, vol. 19, p. 281, 2007.

[42] V. Lopez, S. del Rio, J. M. Benitez, and F. Herrera, "On the use of MapReduce to build linguistic fuzzy rule based classification systems for big data," in *IEEE International Conference on Fuzzy Systems*, 2014, pp. 1905–1912.

[43] 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.

[44] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proceedings of the International Joint Conference on Uncertainty in AI*, 1993, pp. 1022–1027.

[45] 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," *International Journal of Computational Intelligence Systems*, vol. 8, no. 3, pp. 422–437, 2015.

[46] Y. Ben-Haim and E. Tom-Tov, "A streaming parallel decision tree algorithm," *J. Mach. Learn. Res.*, vol. 11, pp. 849–872, Mar. 2010.

[47] M. Mehta, R. Agrawal, and J. Rissanen, *SLIQ: A fast scalable classifier for data mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 18–32.

[48] J. C. Shafer, R. Agrawal, and M. Mehta, "Sprint: A scalable parallel classifier for data mining," in *Proceedings of the 22th International Conference on Very Large Data Bases*, ser. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 544–555.

[49] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh, "Boat—optimistic decision tree construction," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 169–180.

[50] R. Jin and G. Agrawal, "Communication and memory efficient parallel decision tree construction," in *Proceedings of the Third SIAM International Conference on Data Mining*, 2003, pp. 119–129.

[51] O. Yildiz and O. Dikmen, "Parallel univariate decision trees," *Pattern Recognition Letters*, vol. 28, no. 7, pp. 825–832, 2007.

[52] O. Yildiz and E. Alpaydan, "Linear discriminant trees," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 03, pp. 323–353, 2005.

[53] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, 2013.

[54] D. Wang, X. Liu, and M. Wang, "A DT-SVM strategy for stock futures prediction with big data," in *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, 2013, pp. 1005–1012.

[55] K. S. Mann and N. Kaur, "Cloud-deployable health data mining using secured framework for clinical decision support system," in *Computing and Communication (IEMCON), 2015 International Conference and Workshop on*, Oct 2015, pp. 1–6.

[56] I. Triguero, S. del Río, V. López, J. Bacardit, J. M. Benítez, and F. Herrera, "ROSEFW-RF: The winner algorithm for the ECBDL14 big data competition: An extremely imbalanced big data bioinformatics problem," *Knowledge-Based Systems*, vol. 87, pp. 69–79, 2015.

[57] S. del Río, V. López, J. M. Benítez, and F. Herrera, "On the use of MapReduce for imbalanced big data using random forest," *Information Sciences*, vol. 285, pp. 112–137, 2014.

[58] B. Li, X. Chen, M. J. Li, J. Z. Huang, and S. Feng, "Scalable random forests for massive data," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2012, pp. 135–146.

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

[60] X. Wang, D. S. Yeung, and E. C. C. Tsang, "A comparative study on heuristic algorithms for generating fuzzy decision trees," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 31, no. 2, pp. 215–226, 2001.

[61] X. Boyen and L. Wehenkel, “Automatic induction of fuzzy decision trees and its application to power system security assessment,” *Fuzzy Sets and Systems*, vol. 102, no. 1, pp. 3–19, 1999.

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

[63] H. Ishibuchi and T. Yamamoto, “Rule weight specification in fuzzy rule-based classification systems,” *IEEE Transactions on Fuzzy Systems*, vol. 13, no. 4, pp. 428–435, 2005.

[64] A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesus, J. M. Benítez, and F. Herrera, “Big data with cloud computing: An insight on the computing environment, MapReduce, and programming frameworks,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 380–409, 2014.

[65] J. Lin, “MapReduce is good enough? If all you have is a hammer, throw away everything that’s not a nail!” *Big Data*, vol. 1, no. 1, pp. 28–37, 2013.

[66] “Apache Mllib,” http://spark.apache.org/mllib/, accessed: March 2016.

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

[68] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge university press, 1996.

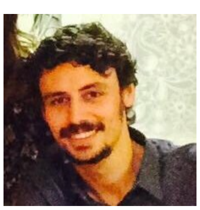
[69] W.-Y. Loh and N. Vanichsetakul, “Tree-structured classification via generalized discriminant analysis,” *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 715–725, 1988.

[70] M. Friedman, “The use of ranks to avoid the assumption of normality implicit in the analysis of variance,” *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, 1937.

[71] R. L. Iman and J. M. Davenport, “Approximations of the critical region of the fbietkan statistic,” *Communications in Statistics-Theory and Methods*, vol. 9, no. 6, pp. 571–595, 1980.

[72] S. Holm, “A simple sequentially rejective multiple test procedure,” *Scandinavian journal of statistics*, pp. 65–70, 1979.

[73] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 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.

Currently, he is a PostDoc researcher with the Department of Information Engineering, University of Pisa. 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.

Francesco Marcelloni received the Laurea degree in Electronics Engineering and the Ph.D. 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. He has co-founded the Computational Intelligence Group at the Department of Information Engineering of the University of Pisa in 2002. Further, he is the founder and head of the Competence Centre on MObile Value Added Services (MOVAS). He has been a member of the Academic Senate of the University of Pisa from 2012 to 2016.

His main research interests include fuzzy classifiers for big data, multi-objective evolutionary algorithms, genetic fuzzy systems, fuzzy clustering algorithms, pattern recognition, signal analysis, neural networks, 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.

Witold Pedrycz (F98) is Professor and Canada Research Chair (CRC) in Computational Intelligence in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada. He received MSc, PhD and DSci, all from the Silesian University of Technology, Gliwice, Poland. He is also with the Systems Research Institute of the Polish Academy of Sciences, Warsaw, Poland. In 2009 Dr. Pedrycz was elected a foreign member of the Polish Academy of Sciences. In 2012 he was elected a Fellow of the Royal Society of Canada. Witold Pedrycz has been a member of numerous program committees of IEEE conferences in the area of fuzzy sets and neurocomputing. In 2007 he received a prestigious Norbert Wiener award from the IEEE Systems, Man, and Cybernetics Society. He is a recipient of the IEEE Canada Computer Engineering Medal, a Cajastur Prize for Soft Computing from the European Centre for Soft Computing, a Killam Prize, and a Fuzzy Pioneer Award from the IEEE Computational Intelligence Society.

His main research directions involve Computational Intelligence, fuzzy modeling and Granular Computing, knowledge discovery and data mining, fuzzy control, pattern recognition, knowledge-based neural networks, relational computing, and Software Engineering. He has published numerous papers in this area. He is also an author of 15 research monographs covering various aspects of Computational Intelligence, data mining, and Software Engineering.

Dr. Pedrycz is intensively involved in editorial activities. He is an Editor-in-Chief of ***Information Sciences***, Editor-in-Chief of ***WIREs Data Mining and Knowledge Discovery*** (Wiley), and ***Int. J. of Granular Computing*** (Springer). He currently serves on the Advisory Board of ***IEEE Transactions on Fuzzy Systems*** and is a member of a number of editorial boards of other international journals.