A Distributed Approach to Multi-Objective Evolutionary Generation of Fuzzy Rule-based Classifiers from Big Data

Andrea Ferranti[a], Francesco Marcelloni[a,*], Armando Segatori[a], Michela Antonelli[b], Pietro Ducange[c]

[a] *Dipartimento di Ingegneria dell'Informazione: University of Pisa, Pisa, Italy*
[b] *Translational Imaging Group, University College of London, London, UK*
[c] *SMART Engineering Solutions & Technologies (SMARTEST) Research Centre, eCAMPUS University, Novedrate, Italy*

Abstract

In the last years, multi-objective evolutionary algorithms (MOEAs) have been extensively used to generate sets of fuzzy rule-based classifiers (FRBCs) with different trade-offs between accuracy and interpretability. Since the computation of the accuracy for each chromosome evaluation requires the scan of the overall training set, these approaches have proved to be very expensive in terms of execution time and memory occupation. For this reason, they have not been applied to very large datasets yet. On the other hand, just for these datasets, interpretablity of classifiers would be very desirable. In the last years the advent of a number of open source cluster computing frameworks has however opened new interesting perspectives. In this paper, we exploit one of these frameworks, namely Apache Spark, and propose the first distributed multi-objective evolutionary approach to learn concurrently the rule and data bases of FRBCs by maximizing accuracy and minimizing complexity. During the evolutionary process, the computation of the fitness is divided among the cluster nodes, thus allowing the designer to distribute both the computational complexity and the dataset storing. We have performed a number of experiments on ten real-world big datasets, evaluating our distributed approach in terms of both classification rate and scalability, and comparing it with two well-known state-of-art distributed classifiers. Finally, we have evaluated the achievable speedup on a small computer cluster. We present that the distributed version can efficiently extract compact rule bases with high accuracy, preserving the interpretability of the rule base, and can manage big datasets even with modest hardware support.

Keywords: Multi-objective Evolutionary Fuzzy Systems, Big Data, Fuzzy Rule-based Classifiers, Apache Spark

*Corresponding author, email: f.marcelloni@iet.unipi.it, Tel: +39 0502217678 Fax: +39 0502217600

Email addresses: andreaferranti@hotmail.it (Andrea Ferranti), f.marcelloni@iet.unipi.it (Francesco Marcelloni), armando.segatori@for.unipi.it (Armando Segatori), m.antonelli@ucl.ac.uk (Michela Antonelli), pietro.ducange@uniecampus.it (Pietro Ducange)

1. Introduction

In the last decades, Fuzzy Rule-Based Classifiers (FRBCs) have been widely employed in several engineering applications. This success is mainly due to three factors. First, FRBCs generally achieve accuracy comparable with other types of state-of-the-art classifiers. Second, the intrinsic nature of FRBCs allows them to deal with vague and noisy data. Third, the structure of FRBCs permits to explain how the classification task is performed. This factor is particularly relevant when the classifier is automatically generated from data rather than from the human experience on the specific application domain: the interpretability of the classifier increases the confidence on the classifier outputs and is considered to be of the utmost importance in contexts such as finance, fault detection and medicine.

The structure of an FRBC consists of a rule base (RB) and a data base (DB): the DB contains the definition in terms of fuzzy sets of the linguistic values used in the fuzzy rules.

Since interpretability is a subjective concept, it is hard to find a worldwide agreed definition and consequently a universal measure of interpretability. In [27] a taxonomy of the interpretability measures for fuzzy rule-based models has been proposed by considering two different dimensions, namely semantic and complexity, at RB and DB levels. As regards the DB, the semantic interpretability is usually evaluated in terms of integrity of the fuzzy partitions, whereas the complexity is evaluated in terms of number of fuzzy sets. As regards the RB, the interpretability is mostly analyzed in terms of complexity and one of the most used metrics is the total rule length (TRL) [13, 31], that is, the total number of conditions used in the RB.

A number of methods have been proposed in the literature to generate the DB and RB of an FRBC from data [11]. At the beginning, these methods have mainly focused on maximizing the accuracy, thus typically generating FRBCs characterized by a high number of rules and low interpretability of the DB [14]. However, in the last years, multi-objective evolutionary algorithms (MOEAs) have been widely used with the aim of generating FRBCs characterized by good trade-offs between accuracy and interpretability [4, 14, 21]

Independently of the approach used to produce the DB and the RB of the fuzzy rule-based systems, the computation of the accuracy of each individual generated in the evolutionary process requires the scan of the overall training set. When the size of the dataset is very large, the application of MOEA-based approaches to the fuzzy rule-based system generation is very critical. The solutions proposed so far in the literature have mainly focused on reducing the number of instances in the training set [5], by using some instance selection method, and on adopting techniques for

speeding-up the convergence of the MOEA [7]. The first type of solutions, although effective, rises the problem of determining how much the dataset can be reduced without loosing information relevant to the evolutionary generation of the FRBCs. The second type reduces the number of evolutions and therefore the number of evaluations, but does not solve the problem of managing very large datasets. Indeed, these datasets could be so large that it would be impossible to store them in a single machine.

Thus, the natural way for managing very large datasets would be to adopt the second type of solution in order to speed up the computation, but exploiting a distributed implementation on a computer cluster. In this paper, we adopt this approach and propose the first distributed implementation of an MOEA to learn concurrently the rule and data bases of FRBCs, by maximizing accuracy and minimizing complexity. In particular, we propose DPAES-RCS, a distributed version of PAES-RCS [7]. PAES-RCS has proven to be very efficient in obtaining satisfactory approximations of the Pareto front using a limited number of iterations [7]. This result has been obtained by learning the RB through a rule and condition selection strategy, which selects a reduced number of rules from a heuristically generated set of candidate rules and a reduced number of conditions, for each selected rule, during the evolutionary process. We implemented DPAES-RCS on Apache Spark.

We would like to point out that DPAES-RCS introduces a design model for making the distributed evolutionary FRBC generation independent of how the dataset is split into chunks. Indeed, DPAES-RCS distributes the computation of the accuracy among the computing units, but runs the evolutionary scheme sequentially on a single unit, thus making it independent of the data distribution, although still efficiently generating FRBCs characterized by good trade-offs between accuracy and complexity. We show the effectiveness of DPAES-RCS by evaluating the generated FRBCs in terms of classification rate, interpretability and scalability on ten real-world big datasets. We compare the results achieved by DPAES-RCS with the ones obtained by two state-of-art approaches to big data classification, namely a distributed decision tree learning algorithm and a distributed implementation of the well-known Chi et al. algorithm for generating FRBCs [17]. The comparison shows that DPAES-RCS outperforms the other two algorithms in terms of interpretability. As regards the accuracy, DPAES-RCS outperforms the distributed Chi et al. algorithm and is statistically equivalent to the distributed decision tree learning algorithm. Furthermore, results achieved by DPAES-RCS are independent of the number of computing units, data chunks and how the dataset is distributed among the machines of the cluster. Finally, we highlight that the proposed approach allows handling big datasets even with modest hardware support.

The paper is organized as follows. In Section 2, we briefly describe some contributions regarding

the use of MOEAs for generating FRBCs and regarding recent distributed implementations of evolutionary algorithms. Section 3 introduces some preliminaries on FRBCs and on the distributed computing framework for big data. Section 4 describes PAES-RCS in short and therefore DPAES-RCS. In Section 5, we illustrate the experimental results and in Section 6 we draw some final conclusion.

2. Related Works

In the last years, the generation of FRBCs from data has been modeled as a multi-objective optimization problem, taking accuracy and interpretability as the conflicting objectives to be optimized. To this aim, MOEAs have been widely used as optimization technique and the term Multi-Objective Evolutionary Fuzzy Classifiers (MOEFCs) has been coined [4, 14, 21].

In the first works proposed in the literature for generating fuzzy rule-based systems, MOEAs have been used to select [32] or learn [13, 18] fuzzy rules, and to perform the tuning [10] of the DB with prefixed DB and RB, respectively. The most recent works perform the learning [1, 3, 4, 6] or the selection [20, 26, 39] of the rules concurrently with the learning of some elements of the DB, namely the granularity and the membership function parameters of the fuzzy partitions. While in rule selection, rules are selected from an initial set of candidate rules generated from data by some heuristic, in rule learning, rules are created during the evolutionary process.

In the last years, different classifiers have been proposed for dealing with a huge amount of data [9, 12, 16, 44]. As regards FRBCs for big data, as discussed in a recent contribution published in [22], some approaches, which investigate performance in terms of accuracy and scalability without considering interpretability, have been recently proposed in the literature [17, 19, 23, 37]. These approaches mainly focus on the accuracy of the models and employ a high number of rules, thus making them not interpretable. To the best of our knowledge, in the framework of evolutionary fuzzy systems (EFSs), that is, fuzzy rule-based systems generated by means of evolutionary algorithms, only two papers address big datasets [24, 40]. In [24] the authors propose a distributed implementation of the 2-tuple lateral tuning for FRBCs by using the CHC algorithm. The work in [40] discusses the results achieved by a distributed implementation of a rule learning method for subgroup discovery based on an MOEA, namely NSGA-II. Both the implementations use the MapReduce programming model, which was introduced by Google in 2004 for simplifying the distribution of the computation flow across large-scale clusters of machines [15]. However, the results of the two implementations depend on how the data are distributed among the computing units. Indeed, the entire dataset is split into data chunks: each mapper of the MapReduce model, which

is executed on a specific data chunk, generates an independent set of rules. Then, a reduce stage is used for merging all the generated rules. Thus, the results of the two approaches depend on the number of data chunks and the actual distribution of the data among them: each mapper generates a different set of rules depending on the specific data chunk.

As regards evolutionary algorithms applied to Big Data, several approaches have been proposed for parallelizing and distributing their execution [8, 30], by studying and experimenting different models [8, 30], such as master-slaves, island, cellular, hierarchy, pool, co-evolution and multi-agent models.

A recent survey [30] has shown and discussed some different research challenges of distributed evolutionary algorithms. Several algorithms have been reviewed and classified according to the parallelism level, the adopted model and the infrastructure employed in their implementation (MPI, grid computing, P2P network, cloud computing and MapReduce, GPU and CUDA).

The most popular execution environment for the distributed implementation of evolutionary algorithms has been so far Apache Hadoop [45], which supports the MapReduce programming paradigm. A number of distributed implementations on Hadoop of genetic algorithms [29, 36, 43], ant colony optimization [46] and differential evolution [48] have been proposed and applied to different domains, achieving good performance in terms of scalability and proving the effectiveness of Hadoop in dealing with big data [36].

As shown in [48], however, the extra costs of the Hadoop distributed file system I/O operations and of the system bookkeeping overhead significantly reduce the benefits of parallelism. Thus, new distributed cluster computing frameworks, such as Apache Spark [47], which implement the concept of in-memory cluster computing, should be employed. Indeed, in [41], the authors have proposed a pairwise test generation based on parallel genetic algorithm and Apache Spark.

3. Preliminaries

In this section, we first introduce some basic concepts and notations regarding FRBCs. Then, we briefly introduce the distributed computing frameworks for big data.

3.1. Fuzzy Rule-Based Classifiers

Instance classification consists of assigning a class C_m from a predefined set $C = \{C_1, \ldots, C_K\}$ of K 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 $U_f \subset \Re$. Let $P_f = \{A_{f,1}, \ldots, A_{f,T_f}\}$, $f = 1, \ldots, F$, be a fuzzy partition with T_f fuzzy sets of the universe U_f. We associate a label $L_{f,j}$ with each fuzzy set

$A_{f,j}$. 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. The m-th rule R_m, $m = 1, \ldots, M$, of an FRBC is typically expressed as:

$$R_m : \textbf{If } X_1 \textbf{ is } L_{1,j_{m,1}} \textbf{ and } \ldots \textbf{ and } X_F \textbf{ is } L_{F,j_{m,F}} \textbf{ then } Y \textbf{ is } C_{k_m} \tag{1}$$

where Y is the classifier output, C_{k_m} is the class label associated with rule R_m, $j_{m,f} \in [1, T_f]$ identifies the index of the label, which has been selected for X_f in rule R_m. The label identifies one among the T_f fuzzy sets of the partition P_f of a continuous attribute or among the set of values of a categorical attribute. To take the "don't care" condition into account, a fictitious label $L_{f,0}, f = 1, \ldots, F$, is added as possible value of each attribute X_f. This label identifies a set characterized by a membership function equal to 1 on the overall universe, thus making the attribute irrelevant in the inference process. The label $L_{f,0}$ allows therefore generating rules that contain only a subset of the attributes.

Let $TR = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$ be the training set of N instances, where ($\mathbf{x}_n$, y_n) is the n-th input-output pair, with $\mathbf{x}_n$ a vector of F values, which can be both real and categorical, and $y_n \in C$. The *matching degree* of the rule R_m with the input $\mathbf{x}_n$ represents the strength of activation of the rule and is calculated as:

$$w_m(\mathbf{x}_n) = \prod_{f=1}^{F} \mu_{L_{f,j_{m,f}}}(x_{n,f}) \tag{2}$$

where $\mu_{L_{f,j_{m,f}}}(x_{n,f})$ is either the membership value of $x_{n,f}$ to the fuzzy set $A_{f,j_{m,f}}$ represented by label $L_{f,j_{m,f}}$ in case of continuous attributes, or 1 if $x_{n,f} = L_{f,j_{m,f}}$ in case of categorical attributes.

We adopt the maximum matching method as *reasoning method*: an instance is classified into the class corresponding to the rule with the maximum matching degree calculated for the instance. In the case of tie, we classify the instance with the class of the first rule in the chromosome that has the maximum association degree. If no rule is fired, we classify the instance with the most frequent class.

3.2. Big Data and cluster computing frameworks

The term *Big Data* refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage and analyze [34]. With the aim of dealing with big data effectively and timely, different ad-hoc solutions at different levels have been proposed [25]:

- Distributed file systems (for instance, the Apache Hadoop distributed file system (HDFS) [45]) and databases for massive data management and storage (such as MongoDB [1] and Apache

[1] www.mongodb.com

Hive [2]);

- Programming models such as *MapReduce* [15] and *Pregel* [38] for the parallelization of computational flow on a cluster of machines;
- cluster computing frameworks such as Apache Hadoop [45] and Apache Spark [47] for integrating data storage, data processing and system management.

Currently, the most popular cluster computing framework is Apache Hadoop, an open-source implementation designed directly upon the MapReduce programming paradigm and framework proposed by Google in 2004 [15]. The success of this framework is mainly due to the MapReduce paradigm simplicity [44]. Hadoop takes care of communication, network bandwidth, disk usage and possible failures. Hadoop, however, is optimized for one-pass batch processing of on-disk data, which makes it slow for interactive data exploration and more complex multi-pass analytics algorithms. Moreover, due to a poor inter-communication capability and inadequacy for in-memory computation [35, 47], Hadoop is not suitable for those applications that require iterative and/or online computations.

Recently, Apache Spark [47], an open-source framework originally developed in the AMPLab at UC Berkley, has emerged as the next generation big data processing tool due to its enhanced flexibility and efficiency. Indeed, Spark allows employing different distributed programming models, such as MapReduce and Pregel, and has proved to perform faster than Hadoop [47], especially in case of iterative and online applications. Unlike the disk-based MapReduce paradigm supported by Hadoop, Spark employs the concept of in-memory cluster computing, where datasets are cached in memory to reduce their access latency.

At high level, a Spark application runs as a set of independent processes on the top of the dataset distributed across the machines of the cluster and consists of one *driver program* and several *executors* [47]. The driver program, hosted in the master machine, runs the user's main function and distributes *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.

The main abstraction provided by Spark is the *resilient distributed dataset* (RDD) [47]. RDD is a fault-tolerant collection of elements partitioned across the machines of the cluster that can be processed in parallel. These collections are resilient, because they can be rebuilt if a portion of the dataset is lost. RDDs support two types of operations: *transformations*, which create a new

[2] hive.apache.org

RDD from an existing one, and *actions*, which return a value to the driver program after running a computation on the RDD. The applications developed using the Spark framework are totally independent of the file system or the database management system used for storing data. Indeed, there exist connectors for reading data, creating the RDD and writing back results on files or on databases.

Another abstraction provided by Spark is the *shared variable*, which is a variable shared across tasks, or between tasks and the driver program. shared variables can be used in parallel operations. To reduce the communication cost, Spark supports two types of shared variables for the two common usage patterns: *broadcast variables*, which can be used to cache a read-only variable in memory on each machine, and *accumulators*, which are variables for associative operations such as counting, addition and multiplication.

4. The PAES-RCS algorithm

In this section, first we introduce the PAES-RCS algorithm we proposed in [7]. Then, we describe in detail its distributed implementation, denoted as DPAES-RCS, on the Apache Spark framework.

4.1. PAES-RCS

The PAES-RCS algorithm employs an MOEA for generating a set of FRBCs with different trade-offs between accuracy and complexity measured in terms of classification rate and TRL, respectively. The FRBC generation process selects rules and conditions from a set of candidate rules, and concurrently learns the parameters that define the fuzzy sets associated with the labels used in the conditions of the rules. To this aim, PAES-RCS adopts a chromosome C composed of two parts (C_{RB}, C_{DB}), which define the RB and the membership function parameters of the continuous attributes, respectively. We apply both crossover and mutation operators to each part of the chromosome independently. The set of candidate rules is extracted from a decision tree obtained by applying the well-known C4.5 algorithm to the training set.

Before applying the C4.5 algorithm, each continuous attribute X_f is transformed into a categorical and ordered variable by using a fuzzy uniform partition P_f of T_f triangular fuzzy sets: the categorical values of the variable are the labels corresponding to the fuzzy sets in P_f. The training set is therefore discretized by associating a categorical value with each continuous value $x_{n,f}$. The categorical value is determined by the index of the fuzzy set of the partition P_f to which $x_{n,f}$ belongs at maximum grade; in case of tie, we choose randomly.

Let $RB_{C4.5}$ and $M_{C4.5}$ be the set of candidate rules extracted from the decision tree generated by the C4.5 algorithm and the cardinality of this set, respectively. Especially when dealing with large and high dimensional datasets, the C4.5 algorithm could generate RBs composed of a high number of rules. Since we are interested in compact and interpretable RBs, as proposed in [7], we permit that the RBs of the solutions contain $\widehat{M} = min(M_{C4.5}, M_{MAX})$ rules, where M_{MAX} is a user parameter. In the experiments, we have set $M_{MAX} = 100$. We have verified that this value allows us to achieve a reasonable accuracy maintaining the complexity at an adequate level.

The C_{RB} part of the chromosome is a vector of $\widehat{M}$ pairs $\mathbf{p}_m = (k_m, \mathbf{v}_m)$, where $k_m \in [0, \ldots, M_{C4.5}]$ identifies the index of the rule in $RB_{C4.5}$ selected for the current RB and $\mathbf{v}_m = [v_{m,1}, \ldots, v_{m,F}]$ is a binary vector which indicates, for each condition in the rule, if the condition is present or corresponds to a "don't care". In particular, if $k_m = 0$, the m^{th} rule is not included in the RB. Thus, we can generate RBs with a lower number of rules than $\widehat{M}$. Further, if $v_{m,f} = 0$, the f^{th} condition of the m^{th} rule is replaced by a "don't care" condition; otherwise it remains unchanged.

The C_{DB} part of the chromosome consists of $\widehat{F}$ vectors of real numbers, where $\widehat{F} <= F$ is the number of continuous attributes (we recall that the C4.5 algorithm can perform an attribute selection): the f^{th} vector contains the $[b_{f,2}, ..., b_{f,T_f-1}]$ cores, which define the positions of the membership functions for the continuous attribute X_f. We adopt triangular fuzzy sets $A_{f,j}$ defined by the tuple ($a_{f,j}$,$b_{f,j}$,$c_{f,j}$), where $a_{f,j}$ and $c_{f,j}$ correspond to the left and right extremes of the support of $A_{f,j}$, and $b_{f,j}$ to the core. Since we adopt strong fuzzy partitions with, for $j = 2, ..., T_f - 1$, $b_{f,j} = c_{f,j-1}$ and $b_{f,j} = a_{f,j+1}$, in order to define each fuzzy set of the partition it is sufficient to fix the positions of the cores $b_{f,j}$ throughout the universe U_f of the f^{th} attribute. Since $b_{f,1}$ and b_{f,T_f} coincide with the extremes of the universe, the partition of each attribute X_f is completely defined by $T_f - 2$ parameters. To ensure a good integrity level of the fuzzy partitions in terms of order, coverage and distinguishability, we force $b_{f,j}$ to vary in $\left[b_{f,j} - \frac{b_{f,j}-b_{f,j-1}}{2}, b_{f,j} + \frac{b_{f,j}-b_{f,j-1}}{2}\right]$, $\forall j \in [2, T_f - 1]$. The integrity can be measured by using the paritition integrity index I_{int} proposed in [3]. Index I_{int} is defined as:

$$I_{int} = 1 - \frac{1}{\widehat{F}+1} \sum_{f=1}^{\widehat{F}} \frac{D_f}{D_f^{MAX}} \tag{3}$$

where $D_f = \sum_{j=2}^{T_f-1} \left| b_{f,j} - \widetilde{b}_{f,j} \right|$ is a dissimilarity measure which expresses how much, on average, the partitions generated by the evolutionary membership function parameter learning are different from the initial partitions, $D_f^{MAX} = \frac{T_f-2}{2}$ is the possible maximum value of D_f, and $\widehat{F}$ is the number of attributes used actually in the rule base of the solution. The value of I_{int} ranges in [0,1]. If the partitions are equal to the initial uniform partitions, then $I_{int} = 1$. The higher the

difference between the initial uniform partitions and the final partitions, the lower the value of I_{int} (at the minimum, $I_{int} = 0$). Since the uniform partition is considered among the most interpretable partitions, values of I_{int} close to 1 indicate a high level of partition interpretability [3].

We apply the two-point crossover to the C_{RB} part and the BLX-α crossover, with $\alpha = 0.5$, to the C_{DB} part with probabilities $P_{C_{RB}}$ and $P_{C_{DB}}$, respectively. As regards the mutation, for the C_{RB} part, we apply two well-known operators, namely, random mutation and flip-flop mutation with probabilities $P_{M_{RB1}}$ and $P_{M_{RB2}}$, respectively. The first step of each mutation operator randomly selects a pair $\mathbf{p}_m$, i.e. a rule, in the chromosome. The random mutation operator replaces the value of k_m in the selected pair with an integer value randomly generated in $[1, \ldots, M_{C4.5}]$. If k_m was equal to 0, the values of $\mathbf{v}_m$ are randomly chosen. The flip-flop mutation operator modifies the antecedent $\mathbf{v}_m$ of the selected rule by complementing each gene $v_{m,f}$ with a probability equal to P_{cond} ($P_{cond} = 2/F$ in the experiments). After applying the two mutation operators, we remove the duplicate rules in the RB. Random mutation is also applied to the C_{DB} part with probability $P_{M_{DB}}$. The operator, first, randomly chooses an attribute X_f, $f \in [1, F]$, and a fuzzy set $j \in [2, T_f - 1]$ and then replaces the value of $b_{f,j}$ with a value randomly chosen within the definition interval of $b_{f,j}$.

As MOEA we use the (2+2)M-PAES algorithm that has been successfully employed in our previous works [3, 5]. (2+2)M-PAES, which is a modified version of the well-known (2+2)PAES introduced in [33], is a steady state MOEA, which uses two current solutions and stores the non-dominated solutions in an archive. Unlike the classical (2+2)PAES, which maintains the current solutions until they are not replaced by solutions with particular characteristics, we randomly extract, at each iteration, the current solutions.

4.2. The Distributed Approach

In this section we describe DPAES-RCS, the distributed implementation of the PAES-RCS algorithm [7] employed for dealing with Big Data. For the sake of clarity, we directly refer to its implementation on Apache Spark; as stated before, since Spark is particularly suitable for managing iterative computations, we exploit such framework as distributed data processing environment. DPAES-RCS consists of the following two phases, carefully designed to take advantages of the potentialities of the distributed approach:

- *distributed candidate rule generation*: the candidate rule base $RB_{C4.5}$ is generated by exploiting both a distributed discretization of the training set and a distributed implementation of the C4.5 algorithm;

- *distributed evolutionary optimization*: a set of FRBCs with different trade-offs between accuracy and complexity is determined by using a distributed MOEA.

In the Apache Spark environment, the training set is split into V *chunks* and distributed on the cluster. We assume that the dataset is uniformly partitioned so that each chunk $chunk_v$, with $v \in [1 \dots V]$, contains a subset of $N_v = |N/V|$ instances $(\mathbf{x}_t^v, y_t^v)$ of the training set.

Figure 1 shows the overall *distributed candidate rule generation* process, which consists of three steps. In the figure, we have highlighted the operations performed on a single machine (the master) by the driver program and on the cluster of slave machines by the executors.

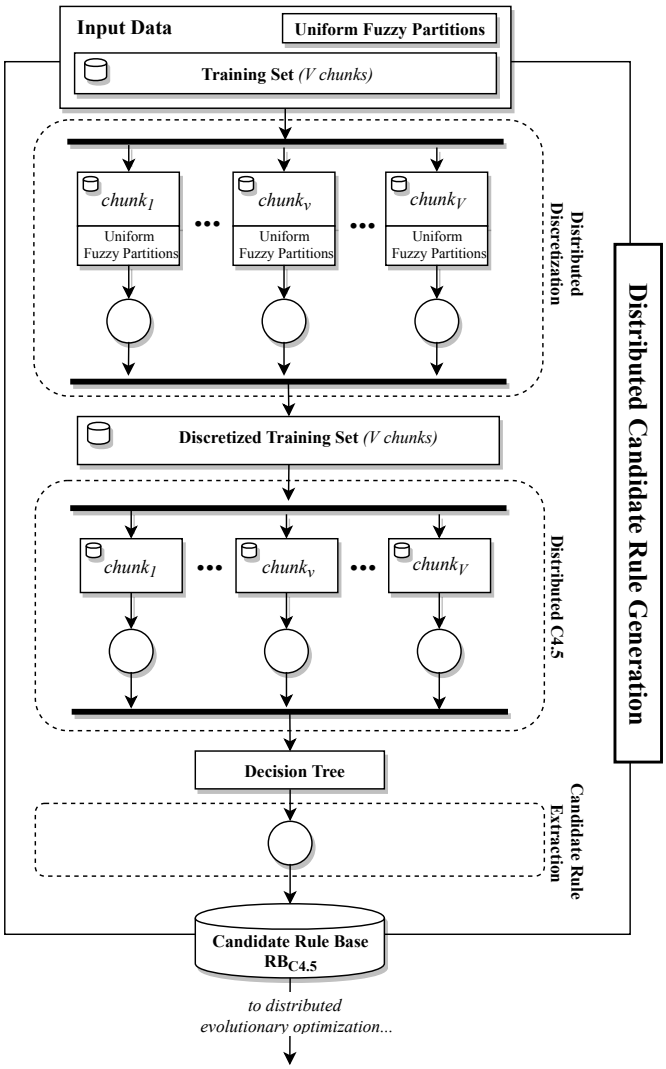

Figure 1: The distributed candidate rule generation phase.

Algorithm 1 details the pseudo-code of the three steps. Given a uniform fuzzy partition with

T_f fuzzy sets on each continuous attribute (line 3), first of all, each pattern $x^v_{t,f}$ in $chunk_v$ is labeled with the categorical value corresponding to the fuzzy set with the highest membership degree (line 13). Note that the discretization is applied in parallel on each chunk of the RDD. Then, a distributed C4.5 algorithm is executed on the discretized training set (line 6). Finally, the candidate rule base $RB_{C4.5}$ is extracted from the decision tree generated by the C4.5 algorithm.

As regards the distributed C4.5 algorithm, we have exploited the distributed decision tree (DDT) implementation provided by MLlib[3].

Algorithm 1 Distributed Candidate Rule Generation.

Require: TR split into V chunk, M_{MAX}

1: **procedure** CANDIDATERULEGENERATION(**in**: TR, M_{MAX})
2: **for** each continuous attribute X_f in **X do**
3: $P[f] \leftarrow$ define fuzzy partition of T_f fuzzy sets on X_f
4: **end for**
5: $disTR \leftarrow$ DISCRETIZE(TR, **P**)
6: $tree_{C4.5} \leftarrow$ EXECUTEC4.5($disTR$, **P**)
7: $RB_{C4.5} \leftarrow$ EXTRACTRULES($tree_{C4.5}$)
8: **return** $RB_{C4.5}$
9: **end procedure**
10: **procedure** DISCRETIZE(**in**: TR, **P**)
11: **for** each $chunk_v$ in TR **do**
12: **for** each continuous value $x^v_{t,f}$ in $chunk_v$ **do**
13: $disTR^v_{t,f} \leftarrow$ label $L_{f,j}$ of fuzzy set $A_{f,j} \in P_f$ with $j = \arg(\max_{1 \le j \le T_f} \mu_{A_{f,j}}(x^v_{t,f}))$
14: **end for**
15: **end for**
16: **return** $disTR$
17: **end procedure**

Similar to the CART algorithm, DDT performs a recursive binary partitioning of the attribute space, where each partition is chosen greedily by selecting the best split from a set of possible splits, which maximizes the information gain. Such approach is applied to both continuous and categorical attributes. We have modified the DDT implementation so as to manage categorical attributes as in the original C4.5. In particular, each parent node generates as many child nodes as the number of categorical values of the selected attribute. Thus, in case of a discretized continuous attribute X_f, the parent node generates T_f child nodes. The attribute used in the parent decision node is selected by exploiting the entropy as impurity measure: the information gain is computed as difference between the parent node impurity and the weighted sum of the child nodes impurities.

[3]http://spark.apache.org/mllib/

Unlike classical C4.5 algorithm [42], no pruning step is performed. Since we aim to ensure a sufficiently large search space for allowing a good exploration to the MOEA, we decided to consider all the rules generated by the C4.5 algorithm. Further, during the evolutionary optimization some conditions in the antecedent of the rules can be replaced by "don't care" conditions: this is similar to perform during the evolutionary optimization a pruning step guided by the increase of the accuracy and the decrease of the TRL. The complexity of the *distributed candidate rule generation* is mainly affected by the complexity of the DDT learning, which is approximately linear in the number of training instances and in the number of attributes[4].

Figure 2 shows the overall *distributed evolutionary optimization* process. We have distributed the computation of the accuracy by employing a master-slave model. We have exploited such approach because it benefits from two advantages. First, it is easy to implement since the master-slave model represents the most straightforward approach for master-worker architectures, which are typical of cluster computing frameworks such as Apache Spark and Apache Hadoop. Second the classification rate and the goodness of the generated solutions are independent of the number of computing units, chunks and how data has been distributed among the machines of the cluster. Algorithm 2 shows the pseudo-code of the *distributed evolutionary optimization* phase. First of all, we initialize the (2+2)M-PAES archive by generating at least two non-dominated solutions consisting of $\widehat{M}$ rules randomly selected from $RB_{C4.5}$. In these solutions, all the conditions in the antecedents of the rules are maintained. We generate pairs of solutions until at least two solutions are not present in the archive. A solution is added to the archive only if it is dominated by no solution contained in the archive; possible solutions in the archive dominated by the solution are removed. The generation of the pair of solutions and the evaluation of the TRLs of these solutions are performed by the *driver program* in a single machine, while the accuracy of the solutions is calculated by exploiting the cluster.

More specifically, let $count_1$ and $count_2$ be the counters associated with s_1 and s_2, respectively. Each *executor* classifies all the $\mathbf{x}_t^v$ instances belonging to its $chunk_v$ of the training set: if $\mathbf{x}_t^v$ is correctly labeled with y_t^v, then the corresponding counter is incremented by 1. Note that counters have been implemented as *accumulators*, which are efficiently managed by Spark. The distributed evolutionary optimization consists of three main steps. These steps are iteratively executed until the maximum number of fitness evaluations is achieved.

More in detail, the first step generates two new candidate offspring solutions $[o_1, o_2]$ by applying

[4]For a complete description of this implementation and scaling issues, please refer to `https://spark.apache.org/docs/1.4.1/mllib-decision-tree.html`

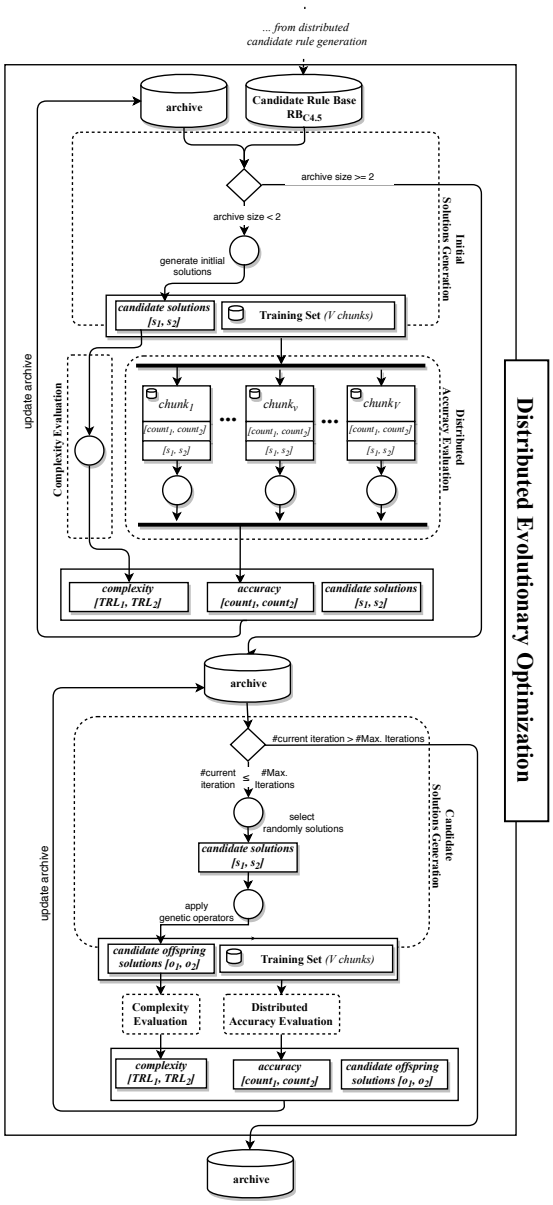

Figure 2: The distributed evolutionary optimization phase.

Algorithm 2 Distributed Evolutionary Optimization.

Require: TR split into V chunk, RB_{C45}, $prob$ (probabilities for genetic operators), $\widehat{M}$, $\#maxIterations$

procedure EVOLUTIONARYOPTIMIZATION(**in**: TR, RB_{C45}, $prob$, $\widehat{M}$, $\#maxIteratons$)
  $archive \leftarrow$ create empty archive
  while archive does not contain at least two solutions **do**
    $[s_1, s_2] \leftarrow$ GENERATEINITIALSOLUTIONS(RB_{C45},$\widehat{M}$)
    $[count_1, count_2] \leftarrow$ EVALUATEACCURACY(TR, $[s_1, s_2]$)
    $[TRL_1, TRL_2] \leftarrow$ CALCULATECOMPLEXITY($[s_1, s_2]$)
    $archive \leftarrow$ UPDATEARCHIVE($[s_1, s_2]$,$[count_1, count_2]$,$[TRL_1, TRL_2]$)
  end while
  $i \leftarrow 0$
  while $i < \#maxIteratons$ **do**
    $[s_1, s_2] \leftarrow$ SELECTRANDOMLYSOLUTIONS($archive$)
    $[o_1, o_2] \leftarrow$ APPLYGENETICOPERATORS($[s_1, s_2]$, $prob$)
    $[count_1, count_2] \leftarrow$ EVALUATEACCURACY(TR, $[o_1, o_2]$)
    $[TRL_1, TRL_2] \leftarrow$ CALCULATECOMPLEXITY($[o_1, o_2]$)
    $archive \leftarrow$ UPDATEARCHIVE($[o_1, o_2]$,$[count_1, count_2]$,$[TRL_1, TRL_2]$)
    $i \leftarrow i + 1$
  end while
  return $archive$
end procedure
procedure EVALUATEACCURACY(**in**: TR, $[sol_1, sol_2]$, N)
  $[count_1, count_2] \leftarrow [0, 0]$
  for each $chunk_v$ in TR **do**
    for each $(\mathbf{x}_t^v, y_t^v)$ in $chunk_v$ **do**
      $[y_1, y_2] \leftarrow$ classify $\mathbf{x}_t^v$ with $[sol_1, sol_2]$
      if $y_1 == y_t^v$ **then**
        $count_1 \leftarrow count_1 + 1$
      end if
      if $y_2 == y_t^v$ **then**
        $count_2 \leftarrow count_2 + 1$
      end if
    end for
  end for
  return $[count_1, count_2]$
end procedure

the genetic operators (line 12) to two solutions $[s_1, s_2]$ randomly extracted from the archive (line 11). This step is performed by the *driver program* in a single machine. The second step evaluates the accuracy and the TRL of each offspring solution. The last step updates the archive according to the new candidate solutions and is performed by the *driver program* in a single machine. If o_1 and o_2 are dominated by no solution contained in the archive, then they are added to the archive. Obviously, possible solutions in the archive dominated by the candidate solutions are removed. When the archive is full (the size of the archive is fixed before starting the execution of the (2+2)M-PAES) and a new solution z has to be added to the archive, if z dominates no solution in the archive, then we insert z into the archive and remove the solution that belongs to the region with the highest crowding degree (note that even z itself can be removed). In case the region contains more than one solution, then, we randomly choose the solution to be removed.

Time complexity of the *distributed evolutionary optimization* is mainly affected by the second step, which represents the most time-consuming part of DPAES-RCS since a scanning of the overall training set for each iteration is required. Since the process is carried out in parallel by each *executor* and requires the evaluation of two candidate solutions, time complexity is, in the worst case, equal to $O(\lceil \frac{V}{Q} \rceil \cdot 2 \cdot \#MaxIterations \cdot N_v \cdot \widehat{M})$, where Q is the number of Computing Units (CUs) available in the cluster, i.e the number of cores in our experiments, and coincides with the number of *tasks* that can be executed concurrently. Obviously, if $V \leq Q$, then all tasks can be run simultaneously and the global runtime practically corresponds to the longest of the task runtimes. In case $V > Q$, only Q tasks can be executed in parallel and the remaining $(V - Q)$ tasks are queued, waiting for being executed as soon as one of the running Q tasks terminates. Thus, in the ideal case where the execution time is the same for all tasks, each core executes at most $\lceil \frac{V}{Q} \rceil$ tasks.

5. Experimental Study

In this section, we show the experimental study we performed for evaluating the effectiveness of DPAES-RCS. The study has been focused on three aspects: i) performance in terms of classification accuracy, model complexity and interpretability of the solutions contained in the Pareto front; ii) comparison of these solutions with two state-of-the-art classification models purposely proposed for managing Big Data; and iii) scalability varying the number of CUs.

As shown in Table 1, we have employed ten real-world big datasets extracted from the UCI[5] and

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

Table 1: Datasets used in the experiments.

Datasets				
Name	**# Instances**	**# Attributes**	**# Classes**	**# Size**
Covertype 2 (COV_2)	581,012	54 (n:10, c:44)	2	75.2 MB
Covertype 7 (COV_7)	581,012	54 (n:10, c:44)	7	75.2 MB
eCO (ECO)	4,178,504	16 (n:16)	10	534 MB
eME (EME)	4,178,504	16 (n:16)	10	535.2 MB
Higgs (HIG)	11,000,000	28 (n:28)	2	8.04 GB
Kddcup 2 (KDD_2)	4,856,151	41 (n:26, c:15)	2	476 MB
Kddcup 5 (KDD_5)	4,898,431	41 (n:26, c:15)	5	480 MB
Kddcup 23 (KDD_23)	4,898,431	41 (n:26, c:15)	23	484 MB
PokerHand (POK)	1,025,010	10 (c:10)	10	24.5 MB
Susy (SUS)	5,000,000	18 (n:18)	2	2.4 GB

the LIBSVM[6] repositories. These datasets are characterized by different numbers of input/output instances (up to 11 millions), attributes (from 10 to 54) and classes (from 2 to 23). Furthermore, for each dataset, the numbers of numeric (n) and categorical (c) attributes, and the size in terms of memory occupancy are also reported.

For each dataset, we performed a five-fold cross-validation. DPAES-RCS has been executed on Apache Spark 1.4.1, using a small computer cluster, i.e., 5 nodes connected by an 1 Gigabit Ethernet (1 Gbps) that run Ubuntu 12.04. The master node, which hosts the *driver program*, has a 4-core CPU (Intel Core i5 CPU 750 x 2.67 GHz), 8 GB of RAM and a 500GB Hard Drive. Each slave node, which runs an *executor*, is equipped by a 4-core CPU with Hyperthreading (Intel Core i7-2600K CPU x 3.40 GHz, 8 threads), 16GB of RAM and 1 TB Hard Drive. The training sets are stored in the Hadoop Distributed File System.

Table 2 shows the values of the parameters used in the experiments. These values have been obtained by performing different trials and, for each trial, by evaluating a different configuration of the parameters, starting from the values used in [7], where the sequential PAES-RCS has been proposed. Actually, most of the values in Table 2 coincide with the ones employed in [7]. The only differences relate to the archive size AS, the maximum number M_{MAX} of rules and the probability $P_{M_{DB}}$, which were increased from 32, 50 and 0.2 to 64, 100 and 0.6, respectively. These increases have been necessary for dealing with a size of the datasets larger than that of the datasets used in [7]. Regarding the number of evaluations, for most of the datasets, we experimentally verified that the evolutionary optimization process has a similar behavior as the one discussed in [7], where we have shown that 50,000 fitness evaluations allow obtaining Pareto fronts statistically equivalent to the ones achieved after 1 million evaluations. For the sake of brevity, we do not report this analysis in the paper. Since each iteration of the (2+2)M-PAES requires two fitness evaluations,

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

it follows that 50,000 fitness evaluations correspond to 25,000 iterations. The number T_f of fuzzy sets has been chosen equal to 5 for all the attributes X_f. Obviously, the value of T_f affects the results achieved by DPAES-RCS. We chose to fix it to 5 for the following reasons. First of all, in our previous works, we compared different possible values for the number of fuzzy sets and, in general, the value of 5 has guaranteed the best performance. Second, we aim to generate interpretable FRBCs. In the literature, it is well-known that the number of linguistic values (and consequently of fuzzy sets) used for a linguistic variable should be lower than 7 for allowing human comprehension. Obviously, the lower this number, the higher the interpretability. Thus, we chose the value of 5, which guarantees good interpretability of the partitions and high modeling capability. Third, interpretability of an FRBC is also related to the number of rules in the RB: a high number of fuzzy sets brings to generate a high number of rules. Indeed, since the tuning of the fuzzy sets during the evolutionary process is constrained for preserving interpretability of the partitions, high accuracies can only be achieved by using a high number of rules. Finally, we performed at the beginning of our work on DPAES-RCS some experiment with different numbers of fuzzy sets for the partitions and we verified that 5 allowed us to have the best trade-off between modeling capability and complexity.

Table 2: Values of the parameters used in the experiments for DPAES-RCS.

# Fitness Evaluations	Maximum number of fitness evaluations	50,000
AS	(2+2)M-PAES archive size	64
T_f	Number of fuzzy sets for each continuous attribute X_f	5
M_{MAX}	Maximum number of rules in an RB	100
$P_{C_{RB}}$	Probability crossover operator for C_{RB}	0.1
$P_{C_{DB}}$	Probability crossover operator for C_{DB}	0.5
$P_{M_{RB1}}$	Probability first mutation operator for C_{RB}	0.1
$P_{M_{RB2}}$	Probability second mutation operator for C_{RB}	0.7
$P_{M_{DB}}$	Probability mutation operator for C_{DB}	0.6

Table 3 shows, for each dataset, the values of the parameters used for executing the C4.5 algorithm for generating the initial set of candidate rules, the average number of rules generated and the average number of attributes selected at the end of the execution. For two datasets, namely Susy and Higgs, we observed that the C4.5 algorithm generated trees with a large number of leaves and consequently a large number of rules. Thus, with the aim of limiting the number of candidate rules and therefore the search space, we have increased the minimum number of instances per leaf to 0.01% of the number of instances of the whole dataset. On the other hand, for PokerHand, we have not changed the value of the "minimum number of instances per leaf". Such dataset contains only categorical attributes and represents a special application field for DPAES-RCS. Indeed, the search space is smaller than that of other datasets because no data base learning is performed. Indeed, in this case, DPAES-RCS is employed only for selecting rules and conditions during the

optimization process.

Table 3: Values of the parameters used for executing the C4.5 algorithm, and average numbers of rules and attributes in the rule bases extracted from the generated trees.

Dataset	min # instances per leaf	$\overline{Rules}$	$\overline{Attributes}$
COV_2	1	290.4	12.0
COV_7	1	18,176.2	53.0
ECO	1	591.4	12.0
EME	1	1,244.0	16.0
HIG	1100	5,270.6	26.4
KDD_2	1	494.6	26.4
KDD_5	1	1,311.0	33.4
KDD_23	1	1,342.4	33.8
POK	1	323,428.2	10.0
SUS	500	2,286.0	18.0

By comparing Table 3 with Table 1, we can observe that, for some dataset, the C4.5 algorithm performs a significant attribute selection. Indeed, for COV_2, KDD_2, KDD_5 and KDD_23, percentages of, respectively, 22.22%, 64.39%, 81.66% and 82.43% of the original attributes were selected.

5.1. Performance analysis of DPAES-RCS

In this section, we analyze the performance of DPAES-RCS in terms of accuracy, model complexity, and interpretability of the different solutions generated during the optimization process. The analysis is performed by adopting the method used in previous papers [2, 3]. For each of the five folds, we sort the solutions of the Pareto front approximation obtained by the DPAES-RCS execution for decreasing accuracy. Then, for each fold, we select the first (the most accurate), the last (the least accurate) and the median between the first and last solutions. In the following, we denote these representative solutions as FIRST, MEDIAN and LAST, respectively. The three solutions allow evaluating the width of the Pareto front and performing statistical tests.

Table 4 shows, for each dataset and for each representative solution, the average values and the standard deviations of the accuracy achieved on both the training (Acc_{Tra}) and test (Acc_{Tst}) sets, the average values and the standard deviations of the TRL and of the number (N_{DS}) of non-dominated solutions contained in the archive at the end of the evolutionary process. Furthermore, to allow the reader visually evaluate the width of the Pareto front approximations obtained by DPAES-RCS, in Figure 3 we plot on the classification rate/TRL plane the average values achieved by the three representative solutions, for all the datasets, on both training and test sets.

We observe that, for each representative solution, the average values of the accuracies achieved on the training set are very close to the ones obtained on the test set. Thus, we can conclude that the solutions generated by DPAES-RCS are not affected from overtraining. As regards the interpretability of the solutions, we note that the average values of TRL are not excessively high,

Table 4: Average values and standard deviations of the accuracy on the training and test sets and of TRL of the FIRST, MEDIAN and LAST solutions and average values and standard deviations of the number of non-dominated solutions generated by DPAES-RCS.

	FIRST			**MEDIAN**			**LAST**			
Dataset	Acc_{Tra}	Acc_{Tst}	TRL	Acc_{Tra}	Acc_{Tst}	TRL	Acc_{Tra}	Acc_{Tst}	TRL	N_{DS}
COV_2	75.753 ± 0.004	75.732 ± 0.003	74.4 ± 23.0	74.968 ± 0.005	74.909 ± 0.005	38.7 ± 17.3	72.708 ± 0.007	72.681 ± 0.006	10.0 ± 3.2	27.8 ± 3.4
COV_7	72.383 ± 0.003	72.374 ± 0.003	145.0 ± 37.0	71.940 ± 0.004	71.924 ± 0.004	84.2 ± 25.1	57.921 ± 0.106	57.907 ± 0.106	58.2 ± 19.9	32.2 ± 1.5
ECO	77.133 ± 0.004	77.115 ± 0.004	168.4 ± 79.6	74.995 ± 0.011	74.984 ± 0.011	117.7 ± 73.4	56.228 ± 0.078	56.244 ± 0.078	54.4 ± 24.2	30.2 ± 3.9
EME	80.600 ± 0.008	80.570 ± 0.008	187.4 ± 39.8	78.221 ± 0.010	78.201 ± 0.010	112.0 ± 27.2	61.407 ± 0.061	61.391 ± 0.061	75.2 ± 17.3	35.4 ± 9.4
HIG	65.008 ± 0.012	64.998 ± 0.012	125.2 ± 40.2	64.389 ± 0.008	64.370 ± 0.008	78.7 ± 28.6	59.825 ± 0.017	59.849 ± 0.017	48.6 ± 21.4	35.2 ± 9.0
KDD_2	99.948 ± 0.012	99.947 ± 0.012	35.4 ± 8.0	99.933 ± 0.008	99.934 ± 0.008	19.5 ± 4.3	98.508 ± 0.017	98.514 ± 0.017	8.2 ± 1.3	18.8 ± 2.9
KDD_5	99.740 ± 0.012	99.734 ± 0.012	80.0 ± 17.8	99.717 ± 0.008	99.711 ± 0.008	50.4 ± 13.2	82.920 ± 0.017	82.925 ± 0.017	30.2 ± 12.4	29.8 ± 3.8
KDD_23	99.802 ± 0.000	99.803 ± 0.000	77.8 ± 21	99.735 ± 0.000	99.735 ± 0.000	42.2 ± 10.9	63.384 ± 0.354	63.374 ± 0.354	23.6 ± 6.3	28.2 ± 5.2
POK	60.233 ± 0.006	60.221 ± 0.006	113.2 ± 13.3	58.423 ± 0.008	58.430 ± 0.009	68.1 ± 11.8	48.772 ± 0.031	48.749 ± 0.032	34.2 ± 8.4	51.2 ± 6.7
SUS	78.123 ± 0.001	78.110 ± 0.001	80.4 ± 33.4	77.658 ± 0.003	77.659 ± 0.003	45.6 ± 25.5	68.131 ± 0.083	68.128 ± 0.082	22.0 ± 14.0	29.4 ± 8.6

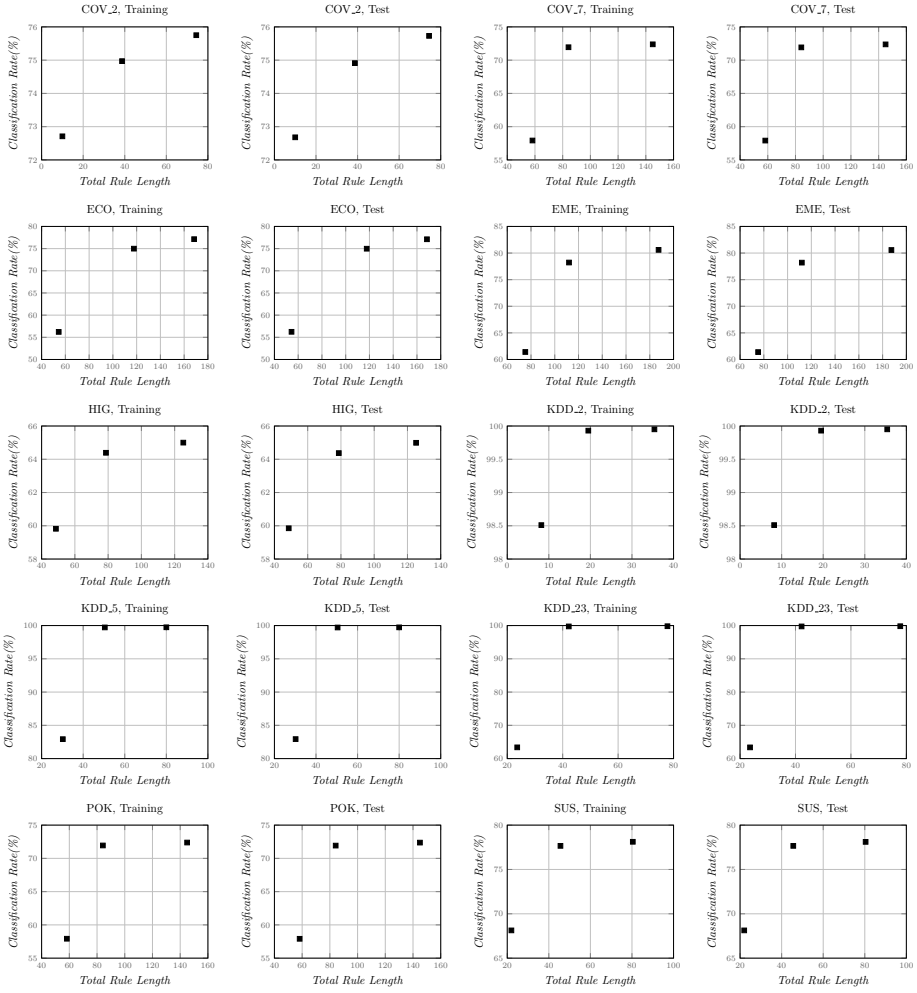

Figure 3: Plots of the average accuracy on the training and test sets and average TRL of the FIRST, MEDIAN and LAST solutions generated by DPAES-RCS.

also considering the number of instances and attributes of the datasets. To evaluate more precisely the interpretability of the solutions, we computed the average number M of rules, the average number $\widehat{F}$ of attributes used in the rule base, the average rule length (ARL), that is, the average number of conditions in the rule antecedents, and the average value of the partition integrity index I_{int}.

Table 5 shows the average values and the standard deviations of M, $\widehat{F}$, ARL and I_{int} for the FIRST, MEDIAN and LAST solutions generated by DPAES-RCS. We can observe that the number of rules is quite low, also for the FIRST solutions, making the RB very interpretable. Further, the number of attributes used in the RB is much lower than the original number of attributes, thus testifying the effectiveness of DPAES-RCS in selecting attributes during the evolutionary process. Further, since ARL is always very low, we can deduce that DPAES-RCS generates RBs mostly composed by generic rules. Finally, the values of I_{int} are higher than 0.92 and do not vary too much among the three representative solutions. Hence, we can conclude that the evolutionary DB learning preserves the partition interpretability. We note that for PokerHand $I_{int} = 1$ for all the three representative solutions. We recall that such dataset contains only categorical attributes and therefore no DB learning is performed by DPAES-RCS during the evolutionary optimization process.

Table 5: Average values and standard deviations of the number (M) of rules, the number ($\widehat{F}$) of attributes, the average rule length (ARL) and the partition integrity index (I_{int}) for the FIRST, MEDIAN and LAST solutions generated by DPAES-RCS.

	FIRST				**MEDIAN**				**LAST**			
Dataset	M	$\widehat{F}$	ARL	I_{int}	M	$\widehat{F}$	ARL	I_{int}	M	$\widehat{F}$	ARL	I_{int}
COV_2	33.6 ± 8.4	9.0 ± 1.7	2.2 ± 0.1	0.940 ± 0.004	21.7 ± 7.3	7.7 ± 1.3	1.8 ± 0.1	0.934 ± 0.007	9.2 ± 2.6	4.2 ± 1.2	1.1 ± 0.0	0.936 ± 0.006
COV_7	36.2 ± 7.3	32.0 ± 2.6	4.0 ± 0.2	0.945 ± 0.003	29.4 ± 6.8	25.9 ± 2.5	2.9 ± 0.1	0.945 ± 0.003	28.0 ± 6.4	24.0 ± 3.9	2.1 ± 0.1	0.945 ± 0.002
ECO	54.0 ± 16.5	12.0 ± 0.0	3.1 ± 0.3	0.940 ± 0.007	45.4 ± 17.3	11.8 ± 0.4	2.6 ± 0.3	0.941 ± 0.006	35.2 ± 10.9	11.4 ± 1.2	1.5 ± 0.1	0.935 ± 0.007
EME	58.6 ± 5.7	15.8 ± 0.4	3.2 ± 0.3	0.945 ± 0.005	48.1 ± 5.9	15.4 ± 0.5	2.3 ± 0.1	0.941 ± 0.003	44.6 ± 4.6	14.8 ± 0.4	1.7 ± 0.1	0.943 ± 0.004
HIG	30.2 ± 8.2	19.0 ± 0.6	4.1 ± 0.1	0.928 ± 0.004	25.8 ± 6.8	15.2 ± 3.0	3.1 ± 0.3	0.927 ± 0.004	23.2 ± 7.2	14.4 ± 4.2	2.1 ± 0.3	0.929 ± 0.003
KDD_2	21.8 ± 4.1	12.6 ± 1.5	1.6 ± 0.0	0.956 ± 0.004	13.2 ± 2.5	8.8 ± 1.5	1.5 ± 0.0	0.958 ± 0.004	8.0 ± 1.4	5.4 ± 1.0	1.0 ± 0.0	0.955 ± 0.005
KDD_5	34.8 ± 6.9	17.8 ± 1.8	2.3 ± 0.1	0.935 ± 0.003	26.5 ± 5.0	14.1 ± 2.2	1.9 ± 0.0	0.934 ± 0.003	23.4 ± 6.4	13.2 ± 2.7	1.3 ± 0.0	0.937 ± 0.001
KDD_23	33.2 ± 6.2	20.0 ± 1.4	2.3 ± 0.0	0.937 ± 0.004	23.5 ± 5.1	15.7 ± 2.1	1.8 ± 0.1	0.936 ± 0.006	20.0 ± 4.7	11.2 ± 1.9	1.2 ± 0.0	0.936 ± 0.005
POK	50.0 ± 4.6	5.0 ± 0.0	2.3 ± 0.0	1.000 ± 0.000	35.2 ± 6.3	5.0 ± 0.0	1.9 ± 0.0	1.000 ± 0.000	25.4 ± 3.1	5.0 ± 0.0	1.3 ± 0.0	1.000 ± 0.000
SUS	28.0 ± 8.6	13.6 ± 1.9	2.9 ± 0.2	0.923 ± 0.005	19.9 ± 7.7	12.1 ± 2.5	2.3 ± 0.1	0.927 ± 0.009	15.0 ± 6.9	9.6 ± 3.0	1.5 ± 0.1	0.925 ± 0.008

Table 6 shows, for each dataset, the average execution time (in seconds) and the standard deviation spent by DPAES-RCS on a cluster of 4 slaves with 8 cores per slave (32 cores in total). For each dataset, we report also the execution time of the distributed evolutionary optimization phase.

As expected, the distributed evolutionary optimization represents the most time consuming part of DPAES-RCS. In particular, the runtime is driven by both the number of instances and the complexity of the two solutions evaluated at each iteration. For instance, ECO and EME have

Table 6: Average computation times (in seconds) and standard deviations for the distributed evolutionary optimization (DEO) phase and the overall algorithm (Tot).

	Execution Time (s)	
Datasets	DEO	Tot
COV_2	$3,602 \pm 234.41$	$3,665 \pm 233.69$
COV_7	$4,804 \pm 1,066.74$	$4,854 \pm 1,065.91$
ECO	$28,188 \pm 3,832.29$	$28,248 \pm 3,832.17$
EME	$39,722 \pm 2,410.13$	$39,775 \pm 2,410.15$
HIG	$76,811 \pm 8,797.67$	$77,703 \pm 8,800.06$
KDD_2	$6,314 \pm 1,974.74$	$6,464 \pm 1,977.02$
KDD_5	$12,400 \pm 4,324.83$	$12,480 \pm 4,322.73$
KDD_23	$11,305 \pm 2,272.03$	$11,392 \pm 2,272.02$
POK	$2,400 \pm 1,874.44$	$2,428 \pm 1,878.47$
SUS	$32,063 \pm 3,082.04$	$32,230 \pm 3,057.74$

the same number of instances. However, since rules in EME tend to be characterized by a higher complexity than the ones in ECO, as we can realize by inspecting Tables 4 and 5, the execution time for EME is longer than that for ECO.

As an example of how much DBs and RBs generated by DPAES-RCS are interpretable, in the following we show the MEDIAN solution obtained on the first fold of the COV_2 dataset. We decided to present a MEDIAN solution because it represents the average trade-off between accuracy and complexity. Indeed, as shown in Table 4, the MEDIAN solution can achieve considerable values of accuracy (not very far from those obtained by the FIRST solutions) with a quite low TRL. The COV_2 dataset is derived from a binary classification problem aimed at predicting forest cover type from cartographic variables only, including four wilderness areas. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are a result of ecological processes rather than of forest management practices. The goal is to classify if the primary species of the forest in a given area is "Lodgepole Pine" ($Type\_2$) or not ($Type\_1$). Table 7 describes the attributes that characterize the COV_2 dataset[7].

Table 7: Description of the attributes of the COV_2 dataset (n and c stands for numerical and categorical, respectively).

Name	**Data Type (Unit)**	**Description**
Elevation	n (meters)	Elevation in meters
Aspect	n (azimuth)	Aspect in degrees azimuth
Slope	n (degrees)	Slope in degrees
Hor_Dist_To_Hyd	n (meters)	Horizontal distance to the nearest surface water
Ver_Dist_To_Hyd	n (meters)	Vertical distance to the nearest surface water
Hor_Dist_To_Roa	n (meters)	Horizontal distance to the nearest roadway
Hillshade_9am	n (0-255)	Hill shade index at 9am, summer solstice
Hillshade_Noon	n (0-255)	Hill shade index at Noon, summer solstice
Hillshade_3pm	n (0-255)	Hill shade index at 3pm, summer solstice
Hor_Dist_To_Fire_Pts	n (meters)	Horz. dist. to the nearest wildfire ignition points
Wilderness_Area (4 binary columns)	c (0-1)	Wilderness area designation
Soil_Type (40 binary columns)	c (0-1)	Soil type area designation

For each continuous attribute of the COV_2 dataset, Figure 4 shows the uniform fuzzy partition (dashed line) and the fuzzy partition (solid line) of the MEDIAN solution obtained at the end of the

[7]More details on the COV_2 dataset are available at `https://archive.ics.uci.edu/ml/datasets/Covertype`.

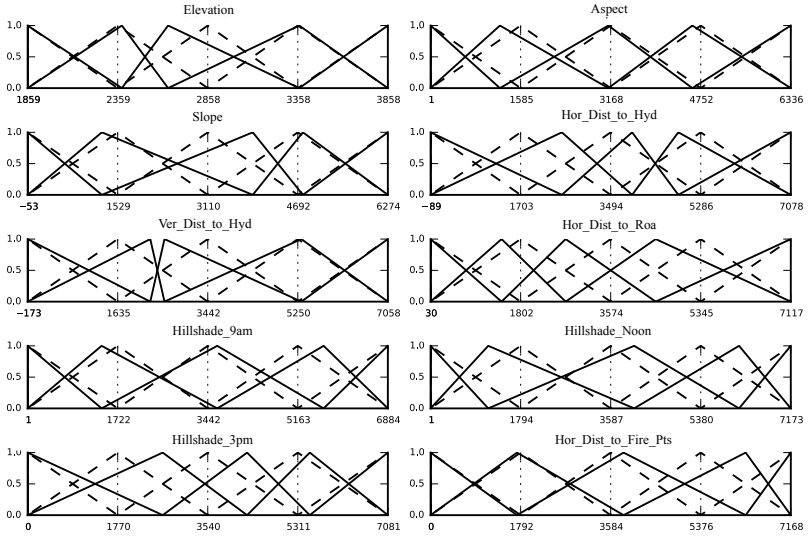

Figure 4: Uniform fuzzy partitions (dashed line) and learned fuzzy partitions (solid line) of the attributes of the MEDIAN solution obtained at the end of the evolutionary process on the COV_2 dataset.

evolutionary process. We can appreciate how the final partitions result to be still very interpretable. We assume that the five fuzzy sets are labeled, from the leftmost to the rightmost, as *very_low*, *low*, *medium*, *high* and *very_high*, respectively. Figure 5 describes the final RB of the FRBC. We can realize that the RB generated by DPAES-RCS is highly interpretable. Indeed, the RB consists of only 20 rules and each of these rules contains at most three conditions in the antecedent. If we consider the number of instances and of attributes, which characterize the COV_2 dataset, the complexity of the FRBC generated by DPAES-RCS is actually very low.

IF Elevation **IS** 'low' **THEN** *Y* **IS** Type_1
IF Elevation **IS** 'very_low' **THEN** *Y* **IS** Type_1
IF Hor_Dist_To_Hyd **IS** 'very_low' **AND** Hor_Dist_To_Roa **IS** 'low' **THEN** *Y* **IS** Type_1
IF Hor_Dist_To_Roa **IS** 'low' **AND** Hor_Dist_To_Fire_Pts **IS** 'medium' **THEN** *Y* **IS** Type_2
IF Elevation **IS** 'medium' **THEN** *Y* **IS** Type_2
IF Hor_Dist_To_Hyd **IS** 'high' **AND** Hor_Dist_To_Roa **IS** 'low' **THEN** *Y* **IS** Type_1
IF Hor_Dist_To_Hyd **IS** 'very_high' **THEN** *Y* **IS** Type_2
IF Hor_Dist_To_Roa **IS** 'very_high' **THEN** *Y* **IS** Type_2
IF Elevation **IS** 'high' **AND** Aspect **IS** 'low' **AND** Hor_Dist_To_Hyd **IS** 'low' **AND** Ver_Dist_To_Hyd **IS** 'low' **THEN** *Y* **IS** Type_1
IF Elevation **IS** 'high' **AND** Slope **IS** 'low' **AND** Hor_Dist_To_Hyd **IS** 'low' **AND** Hillshade_Noon **IS** 'low' **THEN** *Y* **IS** Type_1
IF Hor_Dist_To_Hyd **IS** 'low' **AND** Ver_Dist_To_Hyd **IS** 'high' **AND** Hillshade_3pm **IS** 'very_low' **THEN** *Y* **IS** Type_1
IF Ver_Dist_To_Hyd **IS** 'very_high' **AND** Hillshade_3pm **IS** 'low' **THEN** *Y* **IS** Type_1
IF Elevation **IS** 'high' **AND** Hor_Dist_To_Fire_Pts **IS** 'low' AND **THEN** *Y* **IS** Type_1
IF Elevation **IS** 'high' **AND** Hor_Dist_To_Roa **IS** 'low' AND **THEN** *Y* **IS** Type_2
IF Hor_Dist_To_Roa **IS** 'medium' **AND** Hor_Dist_To_Fire_Pts **IS** 'low' **THEN** *Y* **IS** Type_1
IF Elevation **IS** 'high' **AND** Hor_Dist_To_Fire_Pts **IS** 'medium' **THEN** *Y* **IS** Type_1
IF Hor_Dist_To_Hyd **IS** 'very_low' **AND** Hor_Dist_To_Roa **IS** 'medium' **AND** Hor_Dist_To_Fire_Pts **IS** 'very_low' **THEN** *Y* **IS** Type_2
IF Hor_Dist_To_Hyd **IS** 'very_low' **AND** Hor_Dist_To_Roa **IS** 'high' **THEN** *Y* **IS** Type_2
IF Elevation **IS** 'high' **AND** Hor_Dist_To_Fire_Pts **IS** 'high' **THEN** *Y* **IS** Type_1
IF Elevation **IS** 'very_high' **THEN** *Y* **IS** Type_1

Figure 5: RB of the MEDIAN solution obtained on the COV_2 dataset.

5.2. Comparison with two state-of-the-art classifiers for big data

In this section, we evaluate the performance of DPAES-RCS in comparison with the DDT implementation available in MLlib and with a distributed version of the Chi et al. algorithm [17], denoted as Chi-FRBCS-BigData in the following, for generating FRBCs. These two algorithms can be considered as two recent state-of-the-art classification models purposely designed for handling Big Data.

As described in Section 4.2, DDT splits the attribute space by performing a recursive binary partitioning. We used the default parameters suggested in the guidelines provided by MLlib for executing DDT: the *maximum depth* of the tree is set to 5 and the *number of bins* used to discretize continuous attributes is set to 32. As regards the Chi-FRBCS-BigData, we exploited the Hadoop implementation available on Gitub [8]. We set the values of the parameters as suggested in [17]. We recall that Chi-FRBCS-BigData generates a set of rules for each chunk of the training set and then adopts a reduce stage for fusing the rules by using two different methods. Once all the rules generated by each mapper are grouped together, both methods search for the rules with the same antecedent. Among these rules, the first method selects only the rule with the highest weight and maintains it in the final RB. The second method computes the average weight of the rules, which have the same consequent, and keep in the final RB the rule with the highest average weight. We have chosen the last method because it achieves the best results.

Table 8 shows the average values and the standard deviations of the accuracy on the training (Acc_{Tra}) and test (Acc_{Tst}) sets for the FIRST solution generated by DPAES-RCS, for DDT and for the FRBC generated by Chi-FRBCS-BigData. Further, we show the number M of rules and the TRL for the FIRST solution generated by DPAES-RCS and for the FRBC generated by Chi-FRBCS-BigData. For DDT, we present the number of nodes (NN) and of leaves (NL). We observe that the values of accuracy achieved by DDT are comparable with the FIRST solution obtained by DPAES-RCS. As regards the comparison with Chi-FRBCS-BigData, we highlight that the accuracies achieved by the FIRST solution of DPAES-RCS are almost always higher.

To statistically validate this observation, we generate, for each comparison algorithm, a distribution consisting of the average values of accuracy obtained on the test set on all datasets. Then, we apply non-parametric statistical tests. First, we perform the Friedman test in order to compute a ranking among the distributions and the Iman and Davenport test 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 $\alpha = 0.05$), we can reject the null hypothesis

[8] https://github.com/saradelrio/Chi-FRBCS-BigData-AVE

and affirm that there exist statistical differences among 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. This test allows detecting effective statistical differences between the control approach, i.e. the one with the lowest Friedman rank, and the remaining approaches. Details on the aforementioned tests may be found in [28].

Table 9 summarizes the results of the statistical tests. We observe that the null hypothesis of the Iman and Davenport test is rejected for the accuracy on the test set. Thus, we performed the Holm post-hoc procedure by considering DPAES-RCS (FIRST) as control approach. By analyzing Table 10 we can conclude that the accuracies achieved by the FIRST solution generated by DPAES-RCS are statistically equivalent to the ones achieved by DDT. On the other hand, the tests confirm that DPAES-RCS outperforms Chi-FRBCS-BigData in terms of accuracy. We performed the same statistical tests using the MEDIAN solution instead of the FIRST solution and we got the same results. Indeed, the accuracy achieved by the MEDIAN solutions are statistically equivalent to the ones achieved by DDT and are statistically higher than the ones achieved by Chi-FRBCS-BigData.

Regarding the complexity, as shown in Table 8, we would like to point out that the complexity of the rule bases generated by Chi-FRBCS-BigData is at least one order of magnitude higher than that of the solutions generated by DPAES-RCS.

As regards DDT, with the aim of comparing the complexity of the generated solutions with the ones associated to DPAES-RCS, we can extract two-valued logic rules from the decision tree generated by the DDT learning algorithm: the number of rules is equal to the number of leaves. The number of rules of the FIRST solutions (the most complex) generated by DPAES-RCS is comparable with the number of rules extracted from the decision tree, with the exception of ECO, EME and POK datasets. We recall that the DDT learning, however, performs a binary split at each node. Thus, the rules extracted from the decision tree are different from those in the FRBCs generated by DPAES-RCS. Indeed, they have conditions that are expressed by using intervals in case of continuous attributes and "or" of linguistic values in case of categorical attributes. Figure 6 shows an example of rules generated by the DDT learning algorithm. Here, X_1 and X_F are numerical and categorical attributes, respectively. Furthermore, $l_{1,m}$ and $u_{1,m}$ are the lower and the upper bounds of the interval defined in U_1 for the m-th rule and $L_{F,1}$, $L_{F,2}$ and $L_{F,3}$ are the first three categorical values defined on X_F. Thus, a comparison considering TRL is not meaningful because the rules exploited in the solutions of DPAES-RCS are different from those extracted from the decision tree generated from the DDT learning algorithm. It is worth noting that these rules are more complex and hardly interpretable than the ones in the FRBCs generated by DPAES-RCS.

In Table 11, we show, for each dataset, the average computation times (in seconds) and stan-

Table 8: Average accuracies ± standard deviations achieved by the FIRST solution generated by DPAES (DPAES-RCS(FIRST)), by DDT and by the FRBC generated by Chi-FRBCS-BigData.

	DPAES-RCS (FIRST)				**Decision Tree**				**Chi-FRBCS-BigData**			
Datasets	Acc_{Tra}	Acc_{Tst}	M	TRL	Acc_{Tra}	Acc_{Tst}	NN	NL	Acc_{Tra}	Acc_{Tst}	M	TRL
COV_2	75.753 ± 0.004	75.732 ± 0.003	33.6 ± 8.4	74.4 ± 23.0	73.890 ± 0.018	73.810 ± 0.088	62.2 ± 1.095	31.6 ± 0.548	51.261 ± 0.029	51.255 ± 0.022	72 ± 0.472	3888 ± 25.488
COV_7	72.383 ± 0.003	72.374 ± 0.003	36.2 ± 7.3	145.0 ± 37.0	69.880 ± 0.148	69.990 ± 0.059	63.0 ± 0.000	32.0 ± 0.000	81.692 ± 1.108	79.153 ± 1.169	$81,527 \pm 4,743.073$	$4,402,458 \pm 256,125.942$
ECO	77.133 ± 0.004	77.115 ± 0.004	54.0 ± 16.5	168.4 ± 79.6	77.907 ± 0.017	77.895 ± 0.035	63.0 ± 0.000	32.0 ± 0.000	54.454 ± 7.813	54.485 ± 7.806	$4,148 \pm 79.938$	$66,368 \pm 1,279.008$
EME	80.600 ± 0.008	80.570 ± 0.008	58.6 ± 5.7	187.4 ± 39.8	78.085 ± 0.244	78.048 ± 0.303	62.2 ± 1.095	31.6 ± 0.548	67.285 ± 8.962	67.277 ± 8.952	$13,711 \pm 106.714$	$219,376 \pm 1,707.424$
HIG	65.008 ± 0.012	64.998 ± 0.012	30.2 ± 8.2	125.2 ± 40.2	66.338 ± 0.011	66.336 ± 0.027	63.0 ± 0.000	32.0 ± 0.000	55.933 ± 0.080	55.897 ± 0.119	$24,058 \pm 325.440$	$673,624 \pm 9,112.32$
KDD99_2	99.948 ± 0.012	99.947 ± 0.012	21.8 ± 4.1	35.4 ± 8.0	99.950 ± 0.000	99.950 ± 0.000	43.0 ± 1.095	22.0 ± 0.548	99.934 ± 0.001	99.933 ± 0.002	$1,020 \pm 5.034$	$41,820 \pm 206.394$
KDD99_5	99.740 ± 0.012	99.734 ± 0.012	34.8 ± 6.9	80.0 ± 17.8	99.693 ± 0.001	99.694 ± 0.001	58.2 ± 0.894	29.6 ± 0.447	96.395 ± 0.043	96.302 ± 0.044	$11,585 \pm 31.998$	$474,985 \pm 1,311.918$
KDD99_23	99.802 ± 0.000	99.803 ± 0.000	33.2 ± 6.2	77.8 ± 21	99.730 ± 0.007	99.730 ± 0.007	48.2 ± 7.014	24.6 ± 3.507	99.988 ± 0.001	99.610 ± 0.010	$102,014 \pm 315.773$	$4,182,574 \pm 12,946.693$
POK	60.233 ± 0.006	60.221 ± 0.006	50.0 ± 4.6	113.2 ± 13.3	54.708 ± 0.412	54.696 ± 0.430	63.0 ± 0.000	32.0 ± 0.000	99.711 ± 0.002	5.178 ± 0.049	$813,193 \pm 2,830.838$	$8,131,930 \pm 28,308.380$
SUS	78.123 ± 0.001	78.110 ± 0.001	28.0 ± 8.6	80.4 ± 33.4	77.016 ± 0.060	76.965 ± 0.056	63.0 ± 0.000	32.0 ± 0.000	55.747 ± 0.110	55.751 ± 0.157	678 ± 4.396	$12,204 \pm 79.128$

Table 9: Results of the Friedman and of the Iman and Davenport tests on the accuracy computed on the test set

Algorithm	*Friedman rank*	*Iman and Davenport p-value*	*Hypothesis*
DPAES-RCS (FIRST)	1.4		
DDT	1.8	0.0013526	Rejected
Chi-FRBCS-BigData	2.8		

Table 10: Results of the Holm post hoc procedure for $\alpha = 0.05$

i	algorithm	z-value	p-value	alpha/i	Hypothesis
2	Chi-FRBCS-BigData	3.130495	0.001745	0.025	Rejected
1	DDT	0.894427	0.371093	0.05	Not Rejected

dard deviations spent by the DDT learning algorithm and the Chi-FRBCS-BigData algorithm for generating the classification models. As expected, the computation times of the DDT learning algorithm are considerably shorter than the ones of the DPAES-RCS and Chi-FRBCS-BigData algorithms. On the other hand, DPAES-RCS performs 50,000 evaluations by using the overall dataset during the evolutionary optimization, which allows us to generate FRBCs with a limited number of rules and conditions, thus making them interpretable.

Table 11: Average computation times (in seconds) and standard deviations for the DDT learning algorithm and the Chi-FRBCS-BigData algorithm.

	Execution Time (s)	
Datasets	DDT	Chi-FRBCS-BigData
COV_2	9 ± 2.998	110 ± 7.968
COV_7	13 ± 6.628	$2,810 \pm 1,030.255$
ECO	16 ± 4.038	$1,263 \pm 320.382$
EME	15 ± 3.683	$1,276 \pm 433.667$
HIG	126 ± 4.272	$19,889 \pm 4,327.076$
KDD_2	19 ± 4.869	$2,756 \pm 440.825$
KDD_5	23 ± 6.744	$3,615 \pm 135.106$
KDD_23	23 ± 7.294	$6,551 \pm 226.477$
POK	7 ± 1.750	$18,918 \pm 5,919.047$
SUS	49 ± 8.114	$1,444 \pm 375.832$

Finally we want to highlight that, unlike DPAES-RCS, both the accuracy and the complexity of the FRBCs generated by the Chi-FRBCS-BigData depend on how the training set is split into the chunks.

5.3. Scalability

In this section, we investigate the behavior of the proposed approach by employing an increasing number of CUs. To this aim, we use the *speedup* σ, which is the main metric used in parallel computing for evaluating scalability. The speedup calculates the efficiency of a program, which uses multiple CUs, comparing the execution time of the parallel implementation with the corresponding sequential version. Due to the high number of instances in the datasets considered in our

R_m^{DDT} : **IF** X_1 **is in** $[l_{1,m}\ ,\ u_{1,m}]$ **and** $\ldots$ **and** X_F **is** $L_{F,1}$ **or** $L_{F,2}$ **or** $L_{F,3}$ **then** Y **is** $C_{k,m}$

Figure 6: Example of the m-th rule extracted by the DDT.

experiments, the sequential implementation of the algorithm would take an unreasonable amount of time. To overcome this drawback, we adopt a slightly different definition for the speedup taking as reference a run over Q identical CUs, with $Q > 1$. Thus, we redefine the speedup on Q CUs as follows:

$$\sigma_{Q^*}(Q) = \frac{Q^* \cdot \tau(Q^*)}{\tau(Q)} \tag{4}$$

where $\tau(Q)$ is the runtime using Q CUs and Q^* is the number of CUs used to run the reference execution. In practice, $Q^* \cdot \tau(Q^*)$ allows us to estimate an ideal single-CU runtime. Note that $\tau(Q^*)$ accounts also for the basic overhead due to the Apache Spark platform.

In our experiments, we assumed $Q^* = 8$ so as to have 1 working slave available in the cluster and thus considering in $\sigma_8(Q)$ also 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 4, 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. Obviously, due to the overhead from the Spark procedures and the contention of shared resources among cores, we expect that the speedup is sub-linear.

Table 12 presents the average execution time, speedup $\sigma_8(Q)$ and utilization $\sigma_8(Q)/Q$ obtained on the COV_2 dataset in five trials. Figures 7 and 8 show, respectively, the average execution time and speedup against the number of CUs. Considering the structure of DPAES-RCS, we split the RDD into a number of partitions equal to the total number of cores available on the slaves. We observe that the speedup does not excessively differ from the ideal linear trend. The overhead is mainly due to the time required to send the solutions (C_{RB}, C_{DB}) from the master to each slave node and, of course, to the necessary sequential parts performed on the master.

Table 12: Average execution time, speedup $\sigma_8(Q)$ and utilization $\sigma_8(Q)/Q$ obtained on the COV_2 dataset in five trials

# Cores	Time (s)	$\sigma_8(Q)$	$\sigma_8(Q)/Q$
8	13,297.264	8.0	1.0
16	6,901.795	15.4131	0.9633
24	4,858.456	21.8955	0.9123
32	3,704.587	28.7152	0.8974

As described in Section 4.2, the *distributed evolutionary optimization* phase is the most time-consuming part of the algorithm since it involves the scanning of the overall training set at each

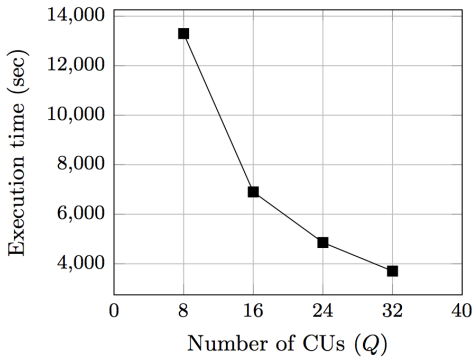

Figure 7: Average execution time on the COV_2 dataset for increasing number of cores

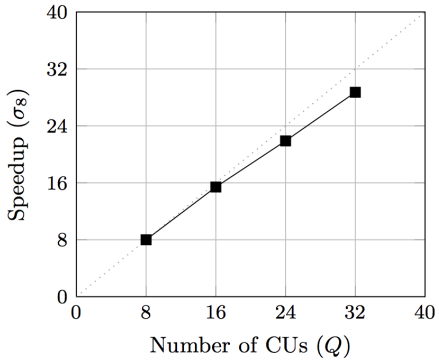

Figure 8: Average speedup on the COV_2 dataset for increasing number of cores

iteration. The execution time of this phase is strongly affected by the complexity of the solutions since, for each instance in the training set, the matching degree for each rule in the RB of the solution has to be computed. With the increase of the number of iterations, we expect that the complexity of the current solutions of DPAES-RCS decreases and therefore the execution time of the iterations at the end of the evolutionary process is lower than that at the beginning. As an example, every 1,000 iterations we computed the sum of the TRLs of the two current solutions and the execution time for computing the accuracy of the two solutions on the first fold of the COV_2 dataset. Figures 9 and 10 show the trend of the sum of the TRLs and of the execution time against the number of iterations, respectively. As expected, the execution time is proportional to the complexity and tends to decrease during the evolutionary optimization.

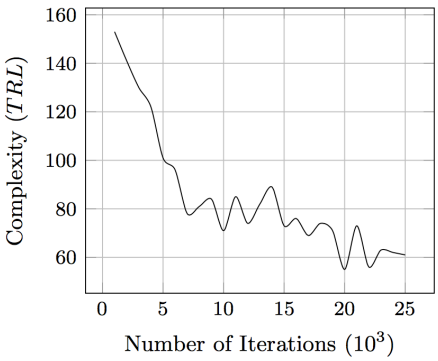

Figure 9: Sum of the TRLs of the two current solutions against the number of iterations.

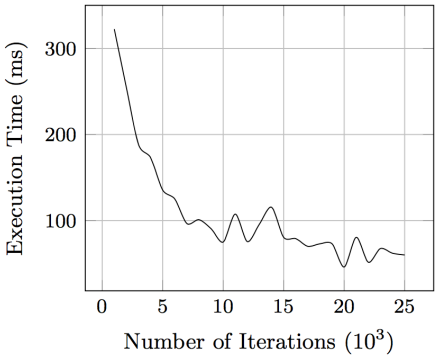

Figure 10: Execution time for computing the accuracy of the two current solutions against the number of iterations.

6. Conclusions

The objective of developing classifiers, which are both accurate and interpretable, is certainly very appealing, especially in some application domains. Fuzzy rule-based classifiers (FRBCs) have proved to be both effective and interpretable models for classification problems. However, the desire of increasing the accuracy has often led to generate FRBCs with a high number of rules and fuzzy sets in the attribute partitions that did not correspond to the intuitive meaning of the related linguistic values. With the aim of generating accurate and interpretable FRBCs, multi-objective evolutionary algorithms (MOEAs) have been extensively employed in the last years. Although the MOEA-based approach has proved to be very effective in determining sets of FRBCs with different trade-offs between accuracy and interpretability, the evaluation of each solution, however, requires the scan of the overall training set and typically a large number of solutions are generated and evaluated before achieving the convergence. When dealing with big datasets, the effort can be too high for a unique computational unit. In this paper, we have proposed the first distributed, scalable and effective implementation of an MOEA which learns concurrently the rule and data bases of FRBCs by maximizing accuracy and minimizing complexity. We have adopted Apache Spark, which is one of the most promising open source cluster computing frameworks for iterative and on-line applications, thanks to its in-memory computation. Spark has permitted us to both efficiently parallelize the computation flow across a computer cluster and provide a robust and transparent environment, taking care of communications and possible failures.

In the experimental study, we have adopted ten real-world big datasets and have compared the solutions generated by our approach with the decision tree available in the MLlib library on Spark and with a state-of-the-art algorithm for generating FRBCs for big data. We have highlighted that our approach generates accurate classifiers with very low complexity. In order to evaluate the scalability performance of the proposed algorithm, we have tested the speedup on one of the big datasets, by using personal computers connected by a Gigabit Ethernet. The experiments highlight that the speedup is close to the ideal achievable targets.

As future work, we intend to investigate the use of distributed fuzzy discretizers for generating the initial fuzzy partition, instead of using a uniform partition with a pre-fixed number of fuzzy sets. Further, we aim to experiment distributed fuzzy decision trees for generating the initial set of candidate rules. Finally, we would like to integrate into DPAES-RCS a set of strategies that will allow us to learn concurrently the rule base, the fuzzy set parameters and also the granularity of the fuzzy partitions.

Acknowledgments

This work is partially supported by the "IoT e Big Data: metodologie e tecnologie per la raccolta e l'elaborazione di grosse moli di dati" project funded by "Progetti di Ricerca di Ateneo - PRA 2017" of the University of Pisa.

References

[1] R. Alcalá, P. Ducange, F. Herrera, B. Lazzerini, F. Marcelloni, A multiobjective evolutionary approach to concurrently learn rule and data bases of linguistic fuzzy-rule-based systems, IEEE Transactions on Fuzzy Systems 17 (5) (2009) 1106–1122.

[2] M. Antonelli, P. Ducange, B. Lazzerini, F. Marcelloni, Learning concurrently data and rule bases of mamdani fuzzy rule-based systems by exploiting a novel interpretability index, Soft Computing 15 (10) (2011) 1981–1998.

[3] M. Antonelli, P. Ducange, B. Lazzerini, F. Marcelloni, Learning knowledge bases of multi-objective evolutionary fuzzy systems by simultaneously optimizing accuracy, complexity and partition integrity, Soft Computing 15 (12) (2011) 2335–2354.

[4] M. Antonelli, P. Ducange, B. Lazzerini, F. Marcelloni, Multi-objective evolutionary design of granular rule-based classifiers, Granular Computing 1 (1) (2016) 37–58.

[5] M. Antonelli, P. Ducange, F. Marcelloni, Genetic training instance selection in multi-objective evolutionary fuzzy systems: A co-evolutionary approach, IEEE Transactions on Fuzzy Systems 20 (2) (2012) 276–290.

[6] M. Antonelli, P. Ducange, F. Marcelloni, An experimental study on evolutionary fuzzy classifiers designed for managing imbalanced datasets, Neurocomputing 146 (2014) 125–136.

[7] M. Antonelli, P. Ducange, F. Marcelloni, A fast and efficient multi-objective evolutionary learning scheme for fuzzy rule-based classifiers, Information Sciences 283 (2014) 36–54.

[8] J. Bacardit, X. Llorà, Large-scale data mining using genetics-based machine learning, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 3 (1) (2013) 37–61.

[9] A. Bechini, F. Marcelloni, A. Segatori, A MapReduce solution for associative classification of big data, Information Sciences 332 (2016) 33–55.

[10] A. Botta, B. Lazzerini, F. Marcelloni, D. C. Stefanescu, Context adaptation of fuzzy systems through a multi-objective evolutionary approach based on a novel interpretability index, Soft Computing 13 (5) (2009) 437–449.

[11] Y.-C. Chen, N. Pal, I.-F. Chung, An integrated mechanism for feature selection and fuzzy rule extraction for classification, IEEE Transactions on Fuzzy Systems 20 (4) (2012) 683–698.

[12] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, K. Olukotun, Map-Reduce for Machine Learning on Multicore, Advances in neural information processing systems 19 (2007) 281.

[13] M. Cococcioni, P. Ducange, B. Lazzerini, F. Marcelloni, A Pareto-based multi-objective evolutionary approach to the identification of Mamdani fuzzy systems, Soft Computing 11 (11) (2007) 1013–1031.

[14] O. Cordón, A historical review of evolutionary learning methods for Mamdani-type fuzzy rule-based systems: Designing interpretable genetic fuzzy systems, International Journal of Approximate Reasoning 52 (6) (2011) 894–913.

[15] J. Dean, S. Ghemawat, MapReduce: a flexible data processing tool, Communications of the ACM 53 (1) (2010) 72–77.

[16] S. del Río, V. Løpez, J. M. Benítez, F. Herrera, On the use of mapreduce for imbalanced big data using random forest, Information Sciences 285 (2014) 112–137.

[17] S. del Río, V. López, J. M. Benítez, 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 8 (3) (2015) 422–437.

[18] P. Ducange, B. Lazzerini, F. Marcelloni, Multi-objective genetic fuzzy classifiers for imbalanced and cost-sensitive datasets, Soft Computing 14 (7) (2010) 713–728.

[19] P. Ducange, F. Marcelloni, A. Segatori, A MapReduce-based fuzzy associative classifier for big data, in: Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2015, pp. 1–8.

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

[21] M. Fazzolari, R. Alcalá, Y. Nojima, H. Ishibuchi, F. Herrera, A review of the application of multi-objective evolutionary fuzzy systems: Current status and further directions, IEEE Transactions on Fuzzy Systems 21 (1) (2013) 45–65.

[22] A. Fernández, C. J. Carmona, M. J. del Jesus, F. Herrera, A view on fuzzy systems for big data: progress and opportunities, International Journal of Computational Intelligence Systems 9 (sup1) (2016) 69–80.

[23] A. Fernández, S. del Río, A. Bawakid, F. Herrera, Fuzzy rule based classification systems for big data with mapreduce: granularity analysis, Advances in Data Analysis and Classification (2016) 1–20.

[24] A. Fernández, S. del Río, F. Herrera, A first approach in evolutionary fuzzy systems based on the lateral tuning of the linguistic labels for big data classification, in: Fuzzy Systems (FUZZ-IEEE), 2016 IEEE International Conference on, IEEE, 2016, pp. 1437–1444.

[25] A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesus, J. M. Benítez, 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 4 (5) (2014) 380–409.

[26] M. J. Gacto, R. Alcalá, F. Herrera, Integration of an index to preserve the semantic interpretability in the multiobjective evolutionary rule selection and tuning of linguistic fuzzy systems, IEEE Transactions on Fuzzy Systems 18 (3) (2010) 515–531.

[27] M. J. Gacto, R. Alcalá, F. Herrera, Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures, Information Sciences 181 (20) (2011) 4340–4360.

[28] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: a case study on the cec2005 special session on real parameter optimization, Journal of Heuristics 15 (6) (2009) 617–644.

[29] L. D. Geronimo, F. Ferrucci, A. Murolo, F. Sarro, A Parallel Genetic Algorithm based on Hadoop MapReduce for the Automatic Generation of JUnit Test Suites, in: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), 2012, pp. 785–793.

[30] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, J.-J. Li, Distributed evolutionary algorithms and their models: A survey of the state-of-the-art, Applied Soft Computing Journal 34 (2015) 286–300.

[31] H. Ishibuchi, Y. Nojima, Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning, International Journal of Approximate Reasoning 44 (1) (2007) 4–31.

[32] H. Ishibuchi, T. Yamamoto, Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining, Fuzzy Sets Syst. 141 (1) (2004) 59–88.

[33] J. D. Knowles, D. W. Corne, Approximating the nondominated front using the pareto archived evolution strategy, Evolutionary computation 8 (2) (2000) 149–172.

[34] D. Laney, 3D Data Management: Controlling Data Volume, Velocity and Variety, META Group Research Note 6 (2001) 70.

[35] J. Lin, MapReduce is good enough? If all you have is a hammer, throw away everything that's not a nail!, Big Data 1 (1) (2013) 28–37.

[36] X. Llora, A. Verma, R. H. Campbell, D. E. Goldberg, When huge is routine: scaling genetic algorithms and estimation of distribution algorithms via data-intensive computing, in: Parallel and Distributed Computational Intelligence, Springer, 2010, pp. 11–41.

[37] V. López, S. del Río, J. M. Benítez, F. Herrera, Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data, Fuzzy Sets and Systems 258 (2015) 5–38.

[38] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, 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.

[39] C. H. Nguyen, W. Pedrycz, T. L. Duong, T. S. Tran, A genetic design of linguistic terms for fuzzy rule based classifiers, International Journal of Approximate Reasoning 54 (1) (2013) 1–21.

[40] F. Pulgar-Rubio, A. Rivera-Rivas, M. Pérez-Godoy, P. González, C. Carmona, M. del Jesus, Mefasd-bd: Multi-objective evolutionary fuzzy algorithm for subgroup discovery in big data environments-a mapreduce solution, Knowledge-Based Systems 117 (2017) 70–78.

[41] R. Qi, Z. Wang, S. Li, Pairwise Test Generation Based on Parallel Genetic Algorithm with Spark, in: International Conference on Computer Information Systems and Industrial Applications, Atlantis Press, 2015, pp. 67–70.

[42] J. R. Quinlan, C4.5: programs for machine learning, Elsevier, 2014.

[43] I. Triguero, M. Galar, S. Vluymans, C. Cornelis, H. Bustince, F. Herrera, Y. Saeys, Evolutionary undersampling for imbalanced big data classification, in: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2015), 2015, pp. 715–722.

[44] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, MRPR: A MapReduce solution for prototype reduction in big data classification, neurocomputing 150 (2015) 331–345.

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

[46] B. Wu, G. Wu, M. Yang, A MapReduce based Ant Colony Optimization approach to combinatorial optimization problems, in: The Eighth International Conference on Natural Computation (ICNC), 2012, pp. 728–732.

[47] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al., Apache spark: a unified engine for big data processing, Communications of the ACM 59 (11) (2016) 56–65.

[48] C. Zhou, Fast parallelization of differential evolution algorithm using MapReduce, in: Proceedings of the 12th annual conference on Genetic and evolutionary computation, 2010, pp. 1113–1114.