Cognitive Computation (2019) 11:331–346
https://doi.org/10.1007/s12559-018-9617-2

# A Grammar-Guided Genetic Programing Algorithm for Associative Classification in Big Data

**F. Padillo[1] · J. M. Luna[1,3] · S. Ventura[1,2,3]**

Received: 18 April 2018 / Accepted: 12 November 2018 / Published online: 16 January 2019


**Abstract**

The state-of-the-art in associative classification includes interesting approaches for building accurate and interpretable classifiers. These approaches generally work on four different phases (data discretization, pattern mining, rule mining, and classifier building), some of them being computational expensive. The aim of this work is to propose a novel evolutionary algorithm for efficiently building associative classifiers in Big Data. The proposed model works in only two phases (a grammar-guided genetic programming framework is performed in each phase): (1) mining reliable association rules; (2) building an accurate classifier by ranking and combining the previously mined rules. The proposal has been implemented on different architectures (multi-thread, Apache Spark and Apache Flink) to take advantage of the distributed computing. The experimental results have been obtained on 40 well-known datasets and analyzed through non-parametric tests. Results were compared to multiple approaches in the field and analyzed on three ways: quality of the predictions, level of interpretability, and efficiency. The proposed method obtained accurate and interpretable classifiers in an efficient way even on high-dimensional data, outperforming the state-of-the-art algorithms on three different levels: quality of the predictions, interpretability, and efficiency.

**Keywords** Big data · Associative classification · Evolutionary computation

## Introduction

As time goes by, data storage is getting cheaper and cheaper what implies an increment in the efforts for analyzing and extracting valuable information from such ever larger datasets [1]. This issue has motivated that recent research studies is being focused on high-performance techniques for data analysis, giving rise to the new buzzword Big Data [2]. This term encompasses a set of techniques to face up problems derived from the management and analysis of huge quantities of data [3]. In data analysis, two different tasks are considered: descriptive tasks, which depict intrinsic and important properties of data [4]; and predictive tasks, which predict output variables for unseen data [5] by learning a mapping between a set of input variables and the output variable. Focusing on predictive tasks, different methodologies can be considered to build accurate models that predict the output variable: rule-based systems [5], decision trees [6], and support vector machines [7], just to list a few. From all these methodologies, rule-based classifiers provide a high-level of interpretability and, therefore, classification results can be explained since rules tend to be easily understood and interpreted by the end-user. Additionally, different research studies [8] have demonstrated that rule-based classification systems are competitive with other methodologies [9]. Such systems are mainly divided into two main groups: rule induction; and classification based on association rule mining.

✉ S. Ventura
sventura@uco.es

F. Padillo
fpadillo@uco.es

J. M. Luna
jmluna@uco.es

[1] Department of Computer Science and Numerical Analysis, University of Cordoba, Cordoba, Spain

[2] Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

[3] Knowledge Discovery and Intelligent Systems in Biomedicine Laboratory, Maimonides Biomedical Research Institute of Cordoba, Cordoba, Spain

Classification based on association rule mining, generally known as associative classification (AC), integrates a descriptive task (association rule mining [4]) in the process of inferring a new classifier [10]. Recent studies have shown that AC has specific advantages over other traditional classification approaches. In this sense, Bechini et al. [11] described that the resulting models in AC are often capable of building efficient and accurate classification systems, since in the training phase they leverage association rule discovery methods that find all possible relationships among the attribute values in the training dataset. This in turn leads to extract all hidden rules possibly missed by other classification algorithms. Bechini et al. [11] also describe that a major advantage of AC with regard to decision tree approaches is its ability to update and tune rules without affecting the complete rule set; whereas in the decision tree approach, the same task requires redrawing the whole tree [5]. Last but not least, an advantage of AC with regard to approaches not based on rules (neural networks [5], support vector machines [7], etc.) is the final model representation, which considers simple rules that allow the end-user to easily understand and interpret the results.

AC algorithms generally operate in four phases. First, those datasets including continuous attributes are required to be preprocessed so any variable is finally defined in a discrete domain. Second, the attribute values are combined and characterized by an occurrence value beyond a given threshold. Third, any feasible association rule (the consequent is fixed to the class variable) is obtained. Finally, rules are ranked and post-processed to build an accurate classifier. These numerous phases, specially when working on Big Data, become unfeasible to be addressed by existing methodologies even when advanced techniques in distributed computing [12] are considered. At this point, a reduction in the number of required phases has been recently addressed by considering evolutionary algorithms (EAs) [13]. Here, rules were directly mined without a previous step of extracting patterns (combination of attribute values). Nevertheless, even for a lower number of phases, the computational complexity in Big Data is still a handicap since it exponentially increases with the number of variables ($2^k - 1$ solutions can be found from $k$ variables). Recent approaches (MRAC [11], MRAC+ [11], DAC [14], etc.) have dealt with the problem through current advances in distributed computing. These approaches, however, were based on classical algorithms and only provided an improvement in runtime. The Big Data problem is much more complex (the increment of data also increases fake correlations among variables hampering both interpretability and accuracy [11]) and it therefore requires new methodologies specifically designed to be run on Big Data [15].

The aim of this paper is therefore to improve the state-of-the-art algorithms in AC. Here, a new grammar-guided genetic programming (G3P) algorithm for AC in Big Data is proposed. G3P has been already studied in mining association rules [16], and it has proved to obtain excellent results in both introducing subjective knowledge into the mining process and constraining the search space by including syntax constraints. Special interest has been paid on the complexity of the obtained rules with the aim of easing the interpretability of the results. Another major feature of the proposed algorithm, which really improves the state-of-the-art, is the running on just two phases: (1) mining reliable association rules; and (2) building an accurate classifier. First, the best rules for each class are obtained by means of multiple and independent evolutionary processes (no discretization step is required since the use of a grammar enables continuous features to be encoded). Second, the set of the previously mined rules are ranked and combined to form an accurate classifier. Since rules for each class is obtained, it is guaranteed that minority/majority classes are equally considered. This is an additional major feature of the proposal since many AC approaches are focused on improvements of classification accuracy, not paying attention to the imbalance problem. Finally, it is important to remark that, even when the computational complexity is reduced (only two phases are required now), the analysis of truly Big Data still slows the process down, hampering their applicability in real-world scenarios [1]. To this end, our proposal has been implemented on different architectures where the unique difference among them is the parallelism, all of them return the very same results. These implementations include recent advantages of distributed computing by means of platforms such as Spark and Flink, or more classic approaches as sequential and multi-thread solutions. This work aims at solving some of the problems related to cognitive computation. First, it deals with very large datasets on an efficient way [17]. Among all the possible solutions [18], an evolutionary algorithm has been selected since it has proved to obtain excellent results in many different fields as well as AC [19, 20]. Second, interpretability of the solutions is a dare, so the use of grammars (in the form desired by the end-user) to encode solutions and restrict the search space is a major feature.

In an experimental analysis, the proposal has been compared to multiple AC approaches as well as traditional classification algorithms. In this study, both sequential and trending MapReduce AC algorithms are considered. Experiments were performed on a total of 40 datasets (10 of them are Big Datasets) and results were validated by non-parametric statistical tests. Results were analyzed on three ways: quality of the predictions, level of interpretability, and efficiency (ability to scale up).

The rest of the paper is organized as follows. Section "Preliminaries" presents the most relevant definitions and

related work; "Methods" describes the proposed algorithm; "Results" presents the datasets used in the experiments and the results; finally, some concluding remarks are outlined in "Conclusions".

## Preliminaries

In this section, the associative classification task is formally defined first. Then, different frameworks for distributed computing are described.

### Associative Classification

Let $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ be the set of items, features, or attributes in a dataset comprising a set of transactions $\mathcal{T} = \{t_1, t_2, ..., t_m\}$. Here, each transaction $t_j$ comprises a subset of items $\{i_k, ..., i_l\}$, $1 \leq k, l \leq n$. An association rule is formally defined [4] as an implication of the form $X \rightarrow Y$ where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. The meaning of an association rule is that if the antecedent $X$ is satisfied for a specific transaction $t_j$, i.e., $X \subset t_j$, then it is highly probable that the consequent $Y$ is also satisfied for that transaction, i.e. ,$Y \subset t_j$. The frequency of an itemset $X \subset \mathcal{I}$, denoted as $support(X)$, is defined as the number of transactions from $\mathcal{T}$ that satisfies $X \subset t_j$, i.e., $|\{\forall t_j \in \mathcal{T} : X \subseteq t_j; t_j \subseteq \mathcal{I}\}|$. In the same way, the support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both $X$ and $Y$, i.e., $|\{\forall t_j \in \mathcal{T} : X \subset t_j, Y \subset t_j; t_j \subseteq \mathcal{I}\}|$. Additionally, the strength of implication of the rule, also known as confidence, is defined as the proportion of transactions that satisfy both $X$ and $Y$ among those transactions that contain only the antecedent $X$, i.e., $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Finally, it is noteworthy to mention that association rules can also be used to describe a specific target variable or class label, giving rise to the concept of class association rules [16]. These are implications of the form $X \rightarrow y$, where $X \subseteq \mathcal{I}$ and $y \in Y$, $Y$ being the set of class labels.

In 1998, Liu et al. [10] connected association rule mining (ARM) and classification rule mining to give rise to the task known as associative classification (AC). The aim of this task is to build an accurate and high interpretable classifier by means of rules obtained from ARM techniques. In order to obtain this kind of classifiers, many methods have been proposed along the years [8], almost all of them being based on exhaustive search ARM algorithms [4]. In general, existing AC approaches work on four different steps, which is a real handicap when computationally expensive problems are addressed. For instance, the extraction of patterns (item sets) and association rules from them implies two different and computationally hard problems (the number of solutions exponentially increases with the number of items in data) [15]. To overcome these and other problems (working on continuous domains), some researchers have focused on the application of evolutionary algorithms (EAs) for performing the AC task. Nevertheless, although really efficient and accurate AC algorithms have been proposed, the analysis of truly Big Data slows down the process [17]. Under these circumstances, new forms of building this type of classifiers from a Big Data perspective is an interesting and emerging topic [21], which has not received yet the needed attention. At this point, the combination of EAs with emerging paradigms like MapReduce to process high volumes of data in an accurate and efficient fashion is a trending topic [15].

### Big Data Architectures: Apache Spark, Apache Flink, and its Origins

MapReduce [12] is a recent paradigm of distributed computing in which programs are composed of two main stages, that is, map and reduce. In the map phase, each mapper processes a subset of input data and produces a set of $\langle k, v\rangle$ pairs. Finally, the reducer takes this new list to produce the final values. A typical problem in which this framework is the word count. In this example, mappers produce a set of $\langle k, v\rangle$ pairs, where $k$ is each word in the sentence and $v$ is its frequency of occurrence, which takes the value 1 by default. Then, an intermediate step is carried out, known as shuffle phase, which merges all the values associated with the same key $k$. For example, given three different pairs with the same key, i.e. ,$\langle k, v_1\rangle$ $\langle k, v_2\rangle$ $\langle k, v_3\rangle$, the merging process will return $\langle k, \langle v_1, v_2, v_3\rangle\rangle$. Finally, the reducer takes this new list as input to produce the final values. It should be noted that all the map and reduce operations are run on a distributed way.

Hadoop [22] is the de facto standard for MapReduce applications. Even when the paradigm is efficiently implemented by Hadoop, its major drawback is it imposes an acyclic data flow graph, and there are applications (iterative or interactive analysis [23]) that cannot be efficiently modeled through this kind of graph. Besides, MapReduce is not aware of the total pipeline so it cannot keep intermediate data in memory for faster performance. Instead, it flushes intermediate data to disk between each step. To solve these downsides, Apache Spark has risen up for solving all the deficiencies of Hadoop, introducing an abstraction called RDD (resilient distributed datasets) to store data in main memory as well as a new approach that considers micro-batch technology. Unfortunately, Spark does not support native iterations, meaning that its engine does not directly handle iterative algorithms. In order to implement

an iterative algorithm, a loop needs to repeatedly instruct Spark to execute the step function and manually check the termination criterion, significantly increasing overhead for large-scale iterative jobs. This issue is unlikely to have any practical significance on operations unless the use case requires low latency. In this sense, Apache Flink has been proposed to face the problems of Spark. By the time Flink came along, Apache Spark was already the most suitable framework for fast, in-memory Big Data analytic requirements for a number of organizations around the world. This made Flink appears superfluous, but in the recent years the attention on this new platform has risen up considerably [24, 25].

## Methods

The proposed algorithm, named G3P-ACBD (grammar-guided genetic programming algorithm for associative classification in Big Data), has been eminently designed to tackle Big Data problems and, by this reason, its evolutionary process can be performed in a parallel way without affecting the final accuracy. Unlike existing AC approaches, G3P-ACBD just requires two stages: (1) mining reliable association rules; (2) building an accurate classifier by ranking and combining the previously mined rules. Additionally, due to the growing interest in data gathering, a unique and universal implementation of the proposed algorithm is not useful different adaptations are required depending on the data size. It is noteworthy to mention that all these adaptations return the exact same classifier, being the unique difference the level of parallelism. They all require two stages to obtain the final classifier:

1. Rule extraction. An evolutionary process is performed for each different class to extract interesting rules from the training dataset.
2. Rule selection. Using the previously mined rules, a new evolutionary process aims at sorting and selecting the best rules to build the final classifier.

These two stages are described in the following subsections by considering different implementations, that is, a sequential version (baseline) as well as different parallel versions (multi-thread, Spark, and Flink).

### Baseline Approach

The baseline algorithm is based on a grammar-guided genetic programming methodology and only requires two steps to perform the AC task. Each of these steps are described in the following subsections.

#### Step 1: Rule Extraction

The first step is responsible for extracting the best set of rules for each class by means of several evolutionary processes (one per class). In each of these processes, individuals (representing class association rules) are encoded by a context-free grammar. Thanks to this grammar, an expert in the domain may determine the maximum or minimum length of the rules, and the kind of conditions each rule should include [16].

**Encoding** This first phase of G3P-ACBD represents each solution as a derivation syntax tree encoded by means of a set of production rules from the context-free grammar shown in Fig. 1. A context-free grammar is formally defined as a four-tuple $(\Sigma_N, \Sigma_T, P, S)$ where $\Sigma_T$ represents the alphabet of terminal symbols, $\Sigma_N$ the alphabet of non-terminal symbols, and they have no common elements, i.e., $\Sigma_N \cap \Sigma_T = \emptyset$. Additional $P$ represents the set of productions rules, and $S$ the start symbol. A production rule is defined as $\alpha \rightarrow \beta$ where $\alpha \in \Sigma_N$, and $\beta \in \{\Sigma_T \cup \Sigma_N\}^*$. To encode a solution, a number of production rules from $P$ is applied, resulting in a derivation syntax tree (internal nodes contain only non-terminal symbols, and leaves contain only terminal symbols). Considering the grammar $G$ (see Fig. 1), the following language is obtained $L(G) = \{condition\ (AND\ condition)^*\ consequent\}$, where $\{condition\ (AND\ condition)^*\}$ represents the antecedent of the rule. Finally, it is important to highlight that, in order to avoid bloating, the maximum number of derivations (maximum length of the rules) is determined by a predefined parameter.

Let us finally consider a sample individual (see Fig. 2) encoded through the grammar illustrated in Fig. 1. This sample individual has been generated from the set of production rules $P$ and start from the start symbol $S$ (root of the syntax tree). As shown, leafs represent terminal symbols. In order to obtain the phenotype of this individual, non-terminal symbols are removed giving rise to the

$G$ = $(\Sigma_N, \Sigma_T, P, S)$ with:
$S$ = <Rule>
$\Sigma_N$ = {<Rule>, <Antecedent>, <Consequent>, <Condition>, <Nominal>, <Numerical>}
$\Sigma_T$ = {attribute, value, AND, =, IN, Min_value, Max_value, class}
$P$ = {<Rule> ⇐ <Antecedent>, <Consequent> ;
<Antecedent> ⇐ <Condition> (AND <Condition>)* ;
<Consequent> ⇐ class = value ;
<Condition> ⇐ <Numerical> | <Nominal> ;
<Numerical> ⇐ name IN Min_value, Max_value ;
<Nominal> ⇐ name = value ;
}

**Fig. 1** Context-free grammar defined for the rule extraction stage of the G3P-ACBD algorithm. The grammar is expressed in extended BNF notation

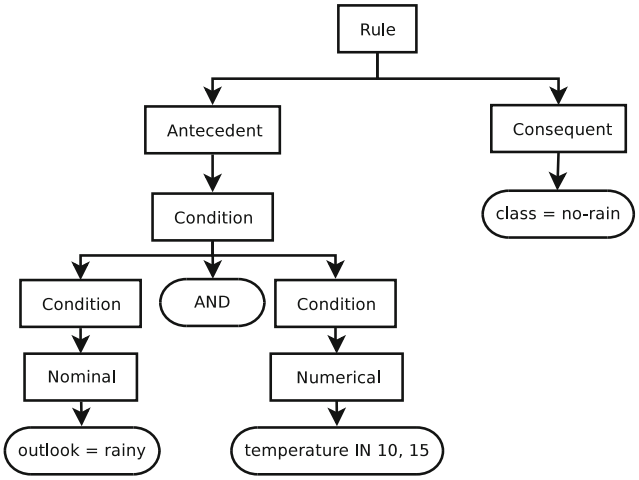


**Fig. 2** Sample rule encoded through the grammar defined in Fig. 1

following rule: **If** $outlook = rainy\ AND\ temperature\ IN\ 10, 15$ **THEN** $no\text{-}rain$.

**Fitness Evaluation** It is the responsible for determining how promising each individual is to achieve the aim. This fitness function has been specifically designed to obtain a good trade-off between reliability (confidence of the rule) and frequency with regard to the class. Given a rule $R \equiv X \rightarrow Y$, the fitness function is mathematically defined as $F(R) = confidence(R) \times support(R)/support(Y)$, taking values in the range [0, 1] (the higher the value, the better the solution). Support values are calculated taken into account the weights of the transactions so individuals have a greater chance for covering new areas of the search space. Analyzing each of the metrics that appear in $F$, it should be noted that the use of confidence in isolation may provoke overfitting in the classifier [8]—infrequent rules may produce high confidence values due to they represent very concrete examples from the training dataset and, therefore, they are not representative to generate predictions. In order to avoid this drawback, the frequency with regard to the class is also considered by the fitness function.

**Genetic Operators** A crossover genetic operator is first applied with a certain probability, generating two offspring. Individuals are then independently mutated with a certain probability. Here, genetic operators widely used in grammar-guided genetic programming have been considered [26]. The crossover operator works by interchanging random sub-trees between two parents (conditions within the rules), whereas the mutation operator applies changes to attributes (changing the whole attribute or just its value). It is worth mentioning that results obtained after applying these operators will fulfill the context-free grammar and also maintain the same class value.

**Algorithm 1** G3P-ACBD baseline algorithm - Rule extraction stage

1: **for all** $c$ **in** $classes$ **do**
2: $P_{0,c} \leftarrow$ Generate a random population of $n$ rules following the grammar with class $c$
3: $auxiliary\_population_c \leftarrow \emptyset$
4: **for** i = 0 **to** NumberOfGenerations **do**
5: evaluate_rules($P_{i,c}$)
6: $auxiliary\_population_c \leftarrow$ Maintain elitism using $P_{i,c} \cup auxiliary\_population_c$
7: Stop if mean($auxiliary\_population_c$) has not changed in a number of generations specified by the user
8: $selected\_invididuals \leftarrow$ Apply tournament selector **to** $P_{i,c}$
9: **for all** $pair$ **in** $selected\_individuals$ **do**
10: $offspring \leftarrow pair$
11: **if** $Rand\_number(0, 1) > Prob_{cro}$ **then**
12: $offspring \leftarrow$ Apply crossover operator (offspring)
13: **end if**
14: **for all** $individual$ **of the** $offspring$ **do**
15: **if** $Rand\_number(0, 1) > Prob_{mut}$ **then**
16: $offspring \leftarrow$ Apply mutation operator ($individual$)
17: **end if**
18: **end for**
19: **end for**
20: $P_{i+1,c} \leftarrow offspring \cup$ best $n$ individuals from $auxiliary\_population_c$
21: **end for**
22: $pool\_rules \leftarrow pool\_rules \cup auxiliary\_population_c$
23: **end for**

**Algorithm** The pseudo-code of the baseline approach is shown in Algorithm 1. It starts by encoding a set of individuals (see line 2, Algorithm 1) through a number of production rules from the previously defined context-free grammar. In this process, each individual represents a unique rule for a specific class, so the population $P_c$ comprises a set of rules related to the $c$-th class value within the dataset. The algorithm includes an elite population ($auxiliary\_population$) in which the best solutions found until that moment are kept unaltered (see line 6, Algorithm 1). Unlike the main population $P_c$, which size is fixed to $n$ beforehand, the size of $auxiliary\_population$ will vary along the evolutionary process in order to guarantee that enough rules from each class can be mined (classes that are hardly described by general rules may require a higher number of really specific rules).

The first phase of the G3P-ACBD algorithm works in an iterative fashion through a number of generations previously fixed by the end-user (see lines 4 to 21, Algorithm 1). This iterative process may finish without reaching the maximum

$G$ = ($\Sigma_N$, $\Sigma_T$, $P$, $S$) with:
$S$ = Classifier
$\Sigma_N$ = {<Classifier>, <Rules>, <DefaultClass>}
$\Sigma_T$ = {rule, class, =, value}
$P$ = {<Classifier> ⇐ <Rules> <DefaultClass> ;
<Rules> ⇐ rule (rule)* ;
<DefaultClass> ⇐ class = value ;
}

**Fig. 3** Context-free grammar defined for the rule selection stage of the G3P-ACBD algorithm. The grammar is expressed in extended BNF notation. It should be noted that the terminal symbol *rule* represents an individual from the previous phase and encoded through the grammar shown in Fig. 1

number of generations in situations where the average fitness value of individuals within $auxiliary\_population$ does not change for a number of generations (see line 7). In order to guarantee the quality of the solutions within $auxiliary\_population$ and to avoid redundant solutions, a pattern weighting scheme is used [27]. For each data instance, a count $j$ is considered to represent the number of rules that covers this example. The weight of each instance decreases based on the formula $w(e_i, j) = \frac{1}{j+1}$. The default weight is 1 for all the instances and, in the following iterations, the contributions of covered instances are inversely proportional to their coverage by previously selected rules. In this way, the examples already covered by one or more selected rules decrease their weights while uncovered instances will have a greater chance of being covered in the following iterations.

In order to produce new individuals, a tournament selector is applied in each iteration of the evolutionary process (see line 8, Algorithm 1). Such new individuals are obtained by the previously defined genetic operators, giving rise to a new population named $offspring$ (see lines 9 to 19, Algorithm 1). Finally, the general population $P_{i+1,c}$ is updated by considering the sets $offspring$ and $auxiliary\_population$ (see lines 20, Algorithm 1). Once the whole evolutionary process has been performed for each class value $c$, results within $auxiliary\_population$ are kept into a major population of rules that will be used in the next stage of the G3P-ACBD algorithm (see line 22), that is, the rule selection step to form a classifier.

### Step 2: Rule Selection

In this second phase of G3P-ACBD, the resulting set of rules previously mined are filtered, sorted, and arranged to form the final classifier. Here, a completely different evolutionary process is run. Unlike the previous step (a single individual represents a single rule), each individual represents now a set of rules that will form the final classifier. It is important to remark that no rule can be produced in this phase and it only works with those previously obtained in the first phase.

**Encoding** This second phase considers a grammar (see Fig. 3) to customize the shape of the final classifer. For instance, an expert could determine that those rules that describe the class $c_1$ should appear first in the final classifier, or whatever restriction the domain problem requires. The process of producing new individuals is similar to the one considered in the first phase of G3P-ACBD (see "Step 1: Rule Extraction"). A maximum number of derivations is considered, determining the number of rules that will be included in the final classifier. For a matter of clarification, a sample individual is illustrated in Fig. 4. This sample individual has been generated from the set of production rules $P$ of the proposed grammar, where the leafs represent terminal symbols (each one is replaced by a rule from the previous stage). Focusing on the sample tree depicted, the phenotype represents a classifier including three different rules and a default class.

**Fitness Evaluation** The quality of each individual (set of rules that form a classifier) is calculated as the average accuracy in training for each class. Given an individual $ind$, the fitness function $F$ is defined as $F(ind) = \frac{1}{l} * \frac{\sum_{n=1}^{m} accuracy_c}{m}$. Here, $m$ is the number of classes, $l$ the number of rules included in $ind$, and $accuracy_c$ is the accuracy in the training set for the class $c$. The aim of considering $l$ is to avoid classifiers including a large number of rules and, therefore, hardly understandable. Additionally,

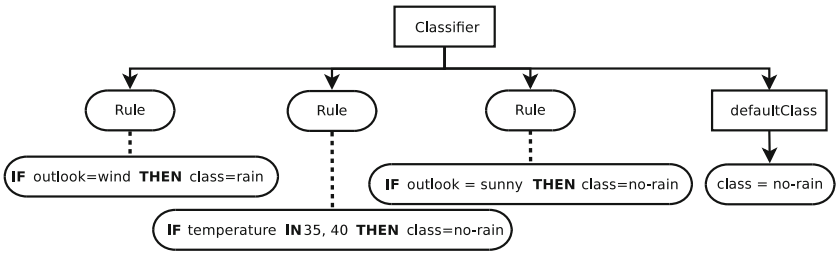


**Fig. 4** Sample rule encoded through the grammar defined in Fig. 3. Rule is a terminal symbol that represents a rule encoded through the grammar previously illustrated in Fig. 1

it is specifically designed to penalize classifiers that ignore minority class.

**Genetic Operators** A crossover genetic operator is first applied with a certain probability, and producing two offspring. Then, individuals are independently mutated with a certain probability. The crossover genetic operator works by interchanging the best rules (considering the fitness value assigned in the first stage) within two individuals, whereas the mutation operator applies changes to the set of rules (adding, subtracting or reordering the rules). These are two well-known genetic operators that have proved to obtain promising results [26].

**Algorithm** The pseudo-code of the second stage of G3P-ACBD is illustrated in Algorithm 2. This algorithm starts by initializing individuals by randomly selecting rules from the pool of rules previously mined ($pool_rules$ returned in the first phase of G3P-ACBD). Like any evolutionary process, the algorithm is repeated a number of generations (see lines 2 to 19, Algorithm 2) previously specified by the end-user. This iterative process may finish without reaching the maximum number of generations in situations where the average fitness value of individuals within $auxiliary\_population$ does not change for a number of generations (see line 5). In order to produce new individuals along the evolutionary process, a tournament selector is considered (see line 6, Algorithm 2) and different genetic operators are applied (see lines 7 to 17, Algorithm 2). Finally, a new population $P_{i+1}$ is generated by also considering the best individual discovered along the generations (see lines 18, Algorithm 2). It is therefore guaranteed that the improvement of the final classifier along the evolutionary process. Last but not least, the most frequent class is taken as default class and it would be only used when no rule covers an example (see line 21, Algorithm 2).

## Parallel and Distributed Computing to Scale G3P-ACBD Up

The runtime of the baseline approach can be improved through parallel and distributed computing architectures (multi-thread, Spark, and Flink), enabling Big Data environments to be also considered. It is important to highlight that all these implementations produce the same results and the only difference lies in the runtime. To improve the runtime, the process to be parallelized is the evaluation process that is the most time-consuming process [28]. In the following subsections, each of three parallel and distributed computing versions of G3P-ACBD are described.

**Algorithm 2** G3P-ACBD baseline algorithm - rule selection stage

1: $P_0 \leftarrow$ Initialize a random population of $n$ individuals (classifiers) including rules from $pool\_rules$
2: **for** i = 0 **to** NumberOfGenerations **do**
3: evaluate($P_i$)
4: $best\_individual \leftarrow$ Best individual from $P_i$
5: Stop if $best\_individual(P_i)$ has not improved in a number of generations specified by the user
6: $selected\_invididuals \leftarrow$ Apply tournament selector **to** $P_i$
7: **for all** $pair$ **in** $selected\_individuals$ **do**
8: $offspring \leftarrow pair$
9: **if** $Rand\_number(0, 1) > Prob_{cro}$ **then**
10: $offspring \leftarrow$ Apply crossover operator (offspring)
11: **end if**
12: **for all** $individual$ **of the** $offspring$ **do**
13: **if** $Rand\_number(0, 1) > Prob_{mut}$ **then**
14: $offspring \leftarrow$ Apply mutation operator ($individual$)
15: **end if**
16: **end for**
17: **end for**
18: $P_{i+1} \leftarrow offspring \cup best\_invididual$
19: **end for**
20: $default\_class \leftarrow$ Class whose frequency of occurrence is the highest
21: Classify using $best\_individual$ and $default\_class$

### Multi-thread Implementation

In this parallel version of G3P-ACBD, any available core in the computer is used by means of threads. The two phases (rule extraction and rule selection) of G3P-ACBD are described as follows:

**Algorithm 3** G3P-ACBD Multi-thread algorithm - Rule extraction stage

1: **for all** $c$ **in** $classes$ **do**
2: **new** Thread() **do**
3: $P_{0,c} \leftarrow$ Generate a random population of $n$ rules following the grammar with class $c$
4: $auxiliary\_population_c \leftarrow \emptyset$
5: **for** i = 0 **to** NumberOfGenerations **do**
6: **if** Thread.avalailables() > 0 **then**
7: creates evaluation threads to evaluate ($P_{i,c}$)
8: **else**
9: evaluate rules in current thread ($P_{i,c}$)
10: **end if**
11: evolve population as baseline. Algorithm 1 (Line 6-20)
12: **end for**
13: $pool\_rules \leftarrow pool\_rules \cup auxiliary\_population_c$
14: **end**
15: **end for**

**Algorithm 4** G3P-ACBD Multi-thread algorithm - Rule selection stage

```
procedure evaluate(population)
 1: subpopulations ← population.groups(Thread.availables())
    // Split population in as many subsets as free threads
 2: for all subpopulation in subpopulations do
 3:     new Thread() do
 4:         evaluate subpopulation
 5:     end
 6: end for
end procedure
```

- **Rule extraction.** In this first phase, two different types of threads are considered. The first one is an evolutionary thread in which the number of threads to be created is equal to the number of classes in data. Here, in each thread, a whole evolutionary process is performed for each class (see Algorithm 3). Neither communication nor synchronization is required among the threads since the processes are totally independent. Results for each thread are gathered by the main process. The second type of thread is responsible for evaluating the population of individuals and it is only performance if there are available threads. Otherwise, the evaluation is performance in the main thread.
- **Rule selection.** In this second phase, the parallelization is carried out on the evaluation process (see Algorithm 4). In this stage, as many threads as resources exists in the computer are created, and the population is split into such number (one chunk per thread). The evaluation process is not fully performed until all the threads end their execution.

### Spark and Flink Implementations

These two implementations follow the exact same philosophy between them except for some minor adaptations depending on the platform (Spark or Flink). These implementations have been designed to be run on a cluster of computers that includes a central computer (driver program) acting as point of coordination, and several additional nodes that collaboratively work with the driver. The two phases (rule extraction and rule selection) are described as follows:

- **Rule extraction.** Similarly to the multi-thread implementation, the driver program starts by creating as many threads as classes exist in data (see $driver\ procedure$ of Algorithm 5). After that, each thread enqueues several MapReduce jobs (which will run on several compute nodes) to evaluate its population. In the *mapper* procedure (see Algorithm 5), the input for each mapper is a chunk of data and the population. A group of pairs $\langle key, value \rangle$ is generated by each mapper, $key$ being the rule, $value$ representing a tuple of support values (antecedent, consequent, and whole rule). Reducers (see $reducer$ procedure, Algorithm 6), on the contrary, receive the previously created $\langle key, value \rangle$ pairs as input. Here, the global support values for antecedent ($support(X)$), consequent ($support(Y)$), and rule ($support(R)$) are obtained for each individual. Once, these three measures have being calculated, the fitness function is obtained as $F(R) = confidence(R) \times support(R)/support(Y)$ and the rules (individuals) are returned to their respective threads. Each thread continues its evolutionary process until a new population is required to be evaluated and the previous process repeated. $\sum_{i=0}^{m} numberGenerations(c_i)$ represents the number of MapReduce jobs, where $m$ is the number of classes, and $numberGenerations(c_i)$ is the number of generations for the $i$-th class. Once all the threads end, a pool of rules is obtained by gathering rules for each class (obtained by different threads). This pool of rules is saved on distributed structures of storage as RDD for Spark and ataset for Flink, enabling a distributed fast access as well as a large quantity of results to be saved.

**Algorithm 5** G3P-ACBD Spark-Flink algorithm - Rule extraction stage

```
procedure driver
 1: for all c in classes do
 2:     new Thread() do
 3:     P_{0,c} ← Generate a random population of n rules following
        the grammar with class c
 4:     auxiliary_population_c ← ∅
 5:     for i = 0 to NumberOfGenerations do
 6:         MapReduce to evaluate rules (P_{i,c})
 7:         evolve population as baseline. Algorithm 1 (Line 6-20)
 8:     end for
 9:     pool_rules ← pool_rules ∪ auxiliary_population_c
10:     end
11: end for
end procedure
procedure mapper(instance, P_{i,c})
 1: for all rule in P_{i,c} do
 2:     measures ← rule.evaluate(instance)
 3:     emit(rule, measures)
 4: end for
end procedure
procedure reducer(rule, measures[])
 1: finalMeasures ← (0, 0, 0) // Support antecedent, consequent and rule
 2: for all measure in measures[] do
 3:     for i = 0 to 2 do
 4:         finalMeasures[i] ← finalMeasures[i] + measure[i]
 5:     end for
 6: end for
 7: fitness ← calculateFitness(finalMeasures)
 8: emit(rule, fitness)
end procedure
```

– **Rule selection.** In this second phase, the evaluation process is the only procedure to be parallelized. In this regard, Algorithm 6 shows pseudo-code for the evaluation process through a MapReduce Job. The *mapper* procedure (see Algorithm 6) receives two elements as input: a subset of the dataset, and the population. A group of pairs $\langle key, value \rangle$ is generated by each mapper, where the $key$ is the rule set, and the value is a tuple with the accuracy values per class. The reducer procedure (see Algorithm 6), on the contrary, receives the previously created $\langle key, value \rangle$ pairs as input. Its goal is to calculate the total accuracy values per class (considering the whole dataset). After that, the fitness function is calculated as $F(rule - set) = \frac{1}{l} * \frac{\sum_{n=1}^{m} accuracy_c}{m}$ and the rule set is returned. The output of the reducer is the evaluated population considering the whole dataset.

**Algorithm 6** G3P-ACBD Spark/Flink algorithm - Rule selection stage

**procedure** evaluate($population$)
1: **for all** $ruleset$ **in** $population$ **do**
2: MapReduce to evaluate $ruleset$
3: **end for**
**end procedure**

**procedure** mapper($chunk, ruleset$)
1: $accuracies\_per\_class \leftarrow (0, ..., 0)$ // As many as classes exist in dataset
2: **for all** $instance$ **in** $chunk$ **do**
3: $accuracies\_per\_class[instance.class] \leftarrow accuracies\_per\_class[instance.class]$ + Accuracy of $ruleset$ in $instance$
4: **end for**
5: emit($ruleset, accuracies\_per\_class$)
**end procedure**

**procedure** reducer($ruleset, accuracies\_per\_class[]$)
1: $final\_accuracy\_per\_class \leftarrow (0, ..., 0)$ // As many as classes exist in dataset
2: **for all** $accuracy\_per\_class$ **in** $accuracies\_per\_class[]$ **do**
3: **for** $i = 0$ **until** $number\_classes$ **do**
4: $final\_accuracy\_per\_class[i] \leftarrow accuracy\_per\_class[i]$
5: **end for**
6: **end for**
7: $fitness \leftarrow ruleset$.calculateFitness($final\_accuracy\_per\_class$)
8: emit($ruleset, fitness$)
**end procedure**

## Results

The aim of this section is to study the results of multiple approaches on three ways: quality of the predictions, level of interpretability, and efficiency. This analysis is carried out through non-parametric tests and considering more than 40 well-known datasets. The goal of this experimental section is therefore summarized as follows:

1. To compare the quality of the predictions with other well-known algorithms taken from the associative classification field, considering classical approaches, bio-inspired algorithms, and Big Data methods.
2. To analyze the interpretability of the results with regard to other methodologies.
3. To compare the efficiency of these approaches which obtain the best possible results.
4. To analyze the scalability in Big Data environments when different parallel implementations are considered.

All the experiments have been run on a HPC cluster comprising 12 compute nodes, with two Intel E5-2620 microprocessors at 2-GHz and 24-GB DDR memory. Cluster operating system was Linux CentOS 6.3. As for the specific details of the used software, the experiments have been run on Spark 2.0.0 and Flink 1.3.0. To quantify the usefulness of the solutions in this experimental analysis, both accuracy rate [5] and Cohen's kappa rate [29] are considered. The accuracy rate (number of successful predictions relative to the total number of examples in data) has been taken since it is the most well-known metric. On the contrary, and due to it may achieve unfair results with imbalanced data, Cohen's kappa rate [29] has been considered since it evaluates the merit of the classifier, i.e., the actual hits that can be attributed to the classifier and not by mere chance. It takes values in the range $[-1, 1]$, where a value of $-1$ means a total disagreement, a value of 0 may be assumed as a random classification, and a value of 1 is a total agreement. This metric is calculated as $Kappa = \frac{N \sum_{i=1}^{k} x_{ii} - \sum_{i=1}^{k} x_{i.} x_{.i}}{N^2 - \sum_{i=1}^{k} x_{i.} x_{.i}}$, where $x_{ii}$ is the count of cases in the main diagonal of the confusion matrix, $N$ is the number of instances and, finally, $x_{.i}$ and $x_{i.}$ are the column and row total counts respectively.

### Experimental Setup

For the sake of analyzing the behavior of our proposal, 40 well-known datasets are considered (see Table 1)—all of them are available at KEEL [30] repository. In these datasets, the number of attributes ranges from 2 to 60, the number of classes varies between 2 and 23, and the number of instances ranges from 87 to $11 \cdot 10^6$. A 10-fold stratified cross-validation has been used, and each algorithm has been executed 5 times. Thus, the results for each dataset are the average result of 50 different runs. An additional experimental study was previously performed to setup the parameters of our proposal. Here, different datasets and

**Table 1** List of datasets (in alphabetical order) used for the experimental study

| Datasets | # attributes | # instances | # classes |
|---|---|---|---|
| Classical datasets | | | |
| Appendicitis | 7 | 106 | 2 |
| Australian | 14 | 690 | 2 |
| Banana | 2 | 5300 | 2 |
| Breast | 9 | 277 | 2 |
| Cleveland | 13 | 297 | 5 |
| Contraceptive | 9 | 1473 | 3 |
| Flare | 11 | 1066 | 6 |
| German | 20 | 1000 | 2 |
| Hayes-Roth | 4 | 160 | 3 |
| Heart | 13 | 270 | 2 |
| Iris | 4 | 150 | 3 |
| Lymphography | 18 | 148 | 4 |
| Magic | 10 | 19,020 | 2 |
| Mammographic | 5 | 830 | 2 |
| Monk-2 | 6 | 432 | 2 |
| Mushroom | 22 | 5644 | 2 |
| Page-blocks | 10 | 5472 | 5 |
| Phoneme | 5 | 5404 | 2 |
| Pima | 8 | 768 | 2 |
| Post-operative | 8 | 87 | 3 |
| Saheart | 9 | 462 | 2 |
| Spectfheart | 44 | 267 | 2 |
| Splice | 60 | 3190 | 3 |
| Tae | 5 | 151 | 3 |
| Tic-tac-toe | 9 | 958 | 2 |
| Titanic | 3 | 2201 | 2 |
| Vehicle | 18 | 846 | 4 |
| Wine | 13 | 178 | 3 |
| Winequality-white | 11 | 4898 | 7 |
| Wisconsin | 9 | 683 | 2 |
| Big Data datasets | | | |
| Census | 40 | 299,285 | 2 |
| CoverType | 54 | 581,012 | 2 |
| Hepmass | 28 | 10,500,000 | 2 |
| Higgs | 28 | 11,000,000 | 2 |
| Poker | 10 | 1,025,010 | 11 |
| Kddcup1999 | 41 | 4,898,431 | 23 |
| KDD99_2 | 41 | 4,856,151 | 2 |
| KDD99_5 | 41 | 4,856,151 | 5 |
| Record-linkage | 12 | 5,749,132 | 2 |
| Sussy | 18 | 5,000,000 | 2 |

They have been categorized into two categories: classical datasets and Big Data datasets

combination of parameters were considered and results are publicly available at http://www.uco.es/kdis/g3p-acbd/. Additionally, 14 different algorithms have been considered in the experimental study. Some classic methodologies for predictive tasks were also taking into account since they are generally considered in many related works [10, 31, 32]. Furthermore, due to this work is mainly designed to be run on large quantities of data, algorithms for Big Data have also been included in the experiments. All the algorithms have been selected according to their efficiency and significance within AC field. It is important to note that the parameters for these algorithms are those provided by the original authors since they have proven to obtain the best results. The algorithms used in this experimental analysis are divided into two main groups, that is, classical and Big Data algorithms.

- **Classical algorithms**
  - CBA [10]. The very first AC algorithm. It is composed of two parts: first, it obtains class association rules and, then, rules are sorted according to their precedence relation.
  - CBA2 [33]. It is an improvement of CBA that considers multiple class minimum support in rule generation.
  - CMAR [32]. It uses a recognized algorithm (FP-Growth [34]) from ARM to obtain rules without candidate generation.
  - CPAR [31]. It adopts a greedy algorithm to generate interval association rules directly. In this process, this algorithm selects multiple literals with similar gains to build multiple rules simultaneously in order to avoid missing important rules.
  - FARCHD [13]. It is a fuzzy association rule–based classification method for high-dimensional problems based on three stages to obtain an accurate and compact fuzzy rule based classifier.
  - C4.5 [6]. One of the most well-known algorithms to generate a decision tree in the same way as ID3 algorithm [5], which uses the concept of information entropy.
  - RIPPER [35]. It is a rule-based learner that builds a set of rules to identify the classes while minimizing the amount of error (the number of training examples misclassified by the rules).
  - CORE [36]. It is a coevolutionary algorithm for rules induction. It coevolves rules and rule sets concurrently in two cooperative populations.
  - OneR [37]. It is a simple, yet accurate, classification algorithm that generates one rule

for each predictor in the data. Then, it selects the rule with the smallest total error as its one rule.

- **Big Data algorithms**
  - MRAC [11]. Distributed association rule–based classification scheme shaped according to the MapReduce programming model.
  - MRAC+ [11]. Improved version of MRAC where some time-consuming operations were removed.
  - DAC [14]. Ensemble learning which distributes the training of an associative classifier among parallel workers.
  - DFAC-FFP [38]. An efficient distributed fuzzy associative classification approach based on the MapReduce paradigm.

## Comparative Analysis

The main goal of this experimental study is to statistically determine which algorithm performs better in three ways: quality of the solutions, interpretability, and efficiency. The experimental study has been divided into three steps (quality of the solutions, interpretability, and efficiency). In each step, the best algorithms are selected and used in the next step so the final step will provide the best algorithms for all the three criteria. Each of these three analyses are carried out from two different perspectives: classical algorithms and datasets (aiming at proving that our proposal outperforms the state-of-the-art even when a small quantity of data is considered); Big Data algorithms and datasets (aiming at comparing to current state-of-the-art in Big Data). Finally, it is important to remark that classical approaches cannot be run on Big Data datasets.

### Analysis of the Quality of the Solutions

The goal of this study is to analyze the quality of the solutions obtained by G3P-ACBD and other well-known algorithms in the field.

#### Classical State-of-the-Art

Table 2 shows average ranking for both accuracy and kappa measures. Analyzing accuracy (see Table 2a), it is obtained that CMAR and OneR have obtained the worst results. It is mainly caused by two different facts. First, CMAR is based on exhaustive search algorithms that cannot be directly run on numeric attributes, requiring a discretization step that implies data loss [39]. Second, CMAR optimizes the confidence measure in isolation, generating very specific classifiers that are not able to correctly predict unseen examples. Results obtained by rule induction algorithms (Ripper and CORE) are not very interesting at all. Considering CBA and its improved version CBA2, very good results in accuracy have been obtained, being even similar to the results obtained by C4.5. Additional AC algorithms such as FARCHD, CPAR, and G3P-ACBD obtained the best results with really small differences among them. Finally, focusing on the Kappa metric (see Table 2b), very similar results have been obtained and the three best algorithms in ranking were FARCHD, CPAR, and G3P-ACBD.

**Table 2** Average ranking for each algorithm (sorted in descending order) according to the Friedman test when 30 datasets are considered

| Algorithm | Ranking |
|---|---|
| (a) Ranking for the accuracy measure | |
| CMAR | 7.916 |
| OneR | 7.350 |
| CORE | 7.150 |
| CBA | 6.233 |
| Ripper | 5.950 |
| CBA2 | 4.816 |
| C4.5 | 4.233 |
| FARCHD | 3.983 |
| *CPAR* | *3.683* |
| *G3P-ACBD* | *3.683* |
| (b) Ranking for the kappa measure | |
| OneR | 7.850 |
| CMAR | 7.400 |
| CORE | 7.383 |
| CBA | 6.100 |
| Ripper | 4.966 |
| CBA2 | 4.766 |
| C4.5 | 4.383 |
| CPAR | 4.283 |
| FARCHD | 4.150 |
| *G3P-ACBD* | *3.716* |

Italic typeface denotes the algorithm that achieves the best ranking

Aiming at analyzing whether there exist any statistical difference in the aforementioned results, several non-parametric tests were carried out. First, a Friedman test has been performed on the accuracy measure, obtaining a $X_F^2 = 77.55$ with a critical value of 21.66 and a $p$ value $= 4.93^{-13}$. In the same way, a $X_F^2 = 70.66$ with a critical value of 21.66 and a $p$ value $= 1.12^{-11}$ has been obtained for the Kappa metric. In both cases, and considering a value $\alpha = 0.01$, it is not possible to assert that all the algorithm equally behave for both measures. Thus, a post-hoc test is performed (see Table 3) and considering $\alpha = 0.01$. Focusing on the accuracy measure, those

**Table 3** $p$ values for the Holland test with $\alpha = 0.01$

| | CBA | CBA2 | CMAR | CPAR | FARCHD | C4.5 | Ripper | CORE | OneR |
|---|---|---|---|---|---|---|---|---|---|
| (a) Results for the accuracy measure | | | | | | | | | |
| CPAR | 0.035 | 0.933 | *0.000* | | 0.999 | 0.998 | 0.100 | *0.000* | *0.000* |
| G3P-ACBD | 0.035 | 0.933 | *0.000* | | 0.999 | 0.998 | 0.100 | *0.000* | *0.000* |
| (b) Results for the kappa measure | | | | | | | | | |
| G3P-ACBD | 0.060 | 0.965 | *0.000* | 0.999 | 1.000 | 0.999 | 0.890 | *0.000* | *0.000* |

algorithms that achieved the best ranking in the previous analysis have been taken as control, that is, CPAR and G3P-ACBD. Results of this post-hoc test (see Table 3a) denote some statistical differences with regard to CMAR, CORE, and OneR. Additionally, it is also interesting to study whether there are any statistical difference between CPAR and G3P-ACBD (those algorithms that achieved the best ranking). A Wilcoxon signed-rank test has been carried out in this regard, obtaining a $Z$ value = $-0.6582$ with $p$ value = 0.50926. It is therefore not possible to assert that, at a significance level of $\alpha = 0.01$, there is a significant difference between CPAR and G3P-ACBD. Finally, the same process is carried out for the kappa measure, taking the algorithm that achieved the best ranking as control, that is, G3P-ACBD. Results of a post-hoc test (see Table 3b) revealed that there are statistical differences with regard to CMAR, CORE, and OneR. To sum up, among the ten selected algorithms, seven of them equally behave according to the quality of their solutions (accuracy and kappa) and, therefore, additional criteria need to be used to select the best approach.

**Table 4** Average ranking for each algorithm (sorted in descending order) according to the Friedman test when 10 Big Data datasets are considered

| Algorithm | Ranking |
|---|---|
| (a) Ranking for accuracy measure | |
| DAC | 4.350 |
| MRAC | 4.150 |
| MRAC+ | 2.700 |
| DFAC-FFP | 2.150 |
| *G3P-ACBD* | *1.650* |
| (b) Ranking for kappa measure | |
| DAC | 4.600 |
| MRAC | 4.100 |
| MRAC+ | 2.600 |
| DFAC-FFP | 1.900 |
| *G3P-ACBD* | *1.800* |

Italic typeface denotes the algorithm that achieves the best ranking

### Big Data State-of-the-Art

Table 4 shows the ranking for both accuracy and kappa measure, G3P-ACBD achieving the best results. DFAC-FPP obtained almost the same results as G3P-ACBD for both quality measures. In order to analyze whether exists any statistical difference, several non-parametric tests are carried out. Focusing on accuracy, the Friedman test revealed a $X_F^2 = 23.12$ with a critical value of 13.277 and a $p$ value = 0.0001. Considering the Kappa measure, results for Friedman was $X_F^2 = 26.32$ with a critical value of 13.277 and a $p$ value $=2.72 \cdot 10^{-5}$. For both measures, with $\alpha = 0.01$, it is possible to assert not all the algorithms equally behave. A post-hoc test is therefore performed to determine the algorithms that present some differences. Table 5 shows the $p$ values for Holland test with $\alpha = 0.01$. According to the results, DAC and MRAC behave statistically different (worse) with regard to the rest of algorithms. Considering G3P-ACBD, DFAC-FFP, and MRAC+, there are no statistical difference in terms of quality and all of them obtained very good results.

### Analysis of the Interpretability of the Solutions

In the previous section, several algorithms obtained statistically significant differences in terms of accuracy and kappa measures. Only those best algorithms have been considered in this next study of interpretability of the classifiers. This is a key analysis since a major reason to use AC algorithms is the interpretability of the final classifier. In this sense, the number of variables per rule and the number of rules per classifier are analyzed (few variables and rules ease the understanding from the

**Table 5** $p$ values for Holland test with $\alpha = 0.01$

| | MRAC | MRAC+ | DAC | DFAC-FFP |
|---|---|---|---|---|
| (a) Results for accuracy measure | | | | |
| G3P-ACBD | *0.004* | 0.447 | *0.001* | 0.821 |
| (b) Results for kappa measure | | | | |
| G3P-ACBD | *0.009* | 0.697 | *0.001* | 0.888 |

Italicized values represent the best ranking value

**Table 6** Ranking (sorted in descending order) for the complexity of the classifiers

| Algorithm | Ranking |
|---|---|
| Ripper | 5.900 |
| CPAR | 5.750 |
| C4.5 | 5.650 |
| CBA2 | 2.967 |
| G3P-ACBD | 2.900 |
| CBA | 2.766 |
| *FARCHD* | *2.066* |

Italic typeface denotes the algorithm that achieves the best ranking

expert's point of view). Given a classifier $C$ including a set of rules $C = \{R_0, ..., R_n\}$, then $complexity(C) = n \sum_{i=0}^{n} attributes(R_i)$ represents the intepretability or complexity of $C$, where $n$ is the number of rules used by the classifier $C$, $R_i$ is a specific rule of the form $R_i = X \rightarrow y$ in the position $i$ of the classifier $C$, and $attributes(R_i)$ is defined as the number of variables, i.e. $|X|$, that $R_i$ includes.

### Classical State-of-the-Art

Table 6 shows the average ranking for complexity measure. As it is illustrated, the best results have been obtained by FARCHD and closely followed by CBA, G3P-ACBD and CBA2. At this point, it is important to remark that the size of the rules in FARCHD is limited, by definition, to 3 variables or conditions. Hence, any rule discovered by this algorithm is extremely short. To prove whether exist some kind of statistical differences among the results, a Friedman test has been performed, obtaining a value of $X_F^2 = 108.850$ with a critical value of 16.812 and a $p\text{-}value = 2.2^{-16}$, meaning that there are some statistically differences among the algorithms for $\alpha = 0.01$. Next, a post-hoc test has been performed to state among which algorithms there are any differences. In this regard, Table 7 shows the $p$ values for the Holland test with $\alpha = 0.01$. Focusing on complexity, the algorithm selected as control is FARCHD. Results of this post-hoc test (see Table 7) denote that there are statistical differences with regard to CPAR, C4.5, and Ripper. Focusing on the proposed G3P-ACBD algorithm, a good interpretability behavior has been achieved since no statistical differences have been found with regard to the control algorithm for interpretability. To sum up, it is possible to state that, in terms of both quality of predictions and interpretability, the best algorithms are FARCHD, CBA, CBA2, and G3P-ACBD. As there are not statistical differences among them, different criteria are required to choose the best algorithm.

**Table 7** $p$ values for the Holland test with $\alpha = 0.01$ for the complexity measure

| | CBA | CBA2 | CPAR | G3P-ACBD | C4.5 | Ripper |
|---|---|---|---|---|---|---|
| FARCHD | 0.807 | 0.637 | *0.000* | 0.687 | *0.000* | *0.000* |

Italicized values represent the best ranking value

**Table 8** Ranking (sorted in descending order) for complexity of the classifiers

| Algorithm | Ranking |
|---|---|
| MRAC+ | 2.800 |
| DFAC-FFP | 2.200 |
| *G3P-ACBD* | *1.000* |

Italic typeface denotes the algorithm that achieves the best ranking

### Big Data State-of-the-Art

Table 8 shows the ranking obtained for each algorithm according to the complexity measure. G3P-ACBD obtained the best results, whereas MRAC+ obtained the worst solutions in terms of interepretability. This behavior is explained by the fact that MRAC+ aims at optimizing the confidence measure, giving rise to really specific rules (including a large number of attributes). The Friedman test was performed in order to determine whether there exist some statistical differences among the algorithms. A value of $X_F^2 = 16.800$ with a critical value of 9.210 and a $p$ value = 0.0002 were obtained, meaning that not all the algorithms equally behave with $\alpha = 0.01$. A post-hoc test was then performed (see Table 9) considering $\alpha = 0.01$. According to this test, G3P-ACBD obtained statistically significant differences with regard to MRAC+. No statistical differences were found between G3P-ACBD and DFAC-FFP, but the former obtained a better ranking.

### Analysis of the Efficiency

This third analysis is related to the runtime, and the aim is to select the faster algorithm among those that obtained good results in accuracy and interpretability.

### Classical State-of-the-Art

As a result of the previous sections, FARCHD, CBA, CBA2, and G3P-ACBD are the algorithms that have achieved

**Table 9** $p$ values for the Holland test with $\alpha = 0.01$ for the complexity measure

| | MRAC+ | DFAC-FFP |
|---|---|---|
| G3P-ACBD | *0.000* | 0.015 |

Italicized values represent the best ranking value

**Table 10** Ranking for the runtime (ordered in descending order) of the different algorithms in the original datasets

| Algorithm | Ranking |
|---|---|
| FARCHD | 6.433 |
| G3P-ACBD (Flink) | 4.766 |
| G3P-ACBD (Spark) | 4.433 |
| CBA | 4.000 |
| G3P-ACBD (Baseline) | 3.500 |
| CBA2 | 2.700 |
| *G3P-ACBD (Multi-thread)* | *2.167* |

Italic typeface denotes the algorithm that achieves the best ranking

a better trade-off between quality of the predictions and interpretability. It is therefore required to analyze such algorithms according to the efficiency. In this study, three different implementations of G3P-ACBD at different levels of parallelism are considered—all of them achieved the exact same solutions so neither the quality of the predictions nor the interpretability vary.

As it is illustrated in Table 10, the multi-thread implementation of G3P-ACBD obtained the best runtime. The baseline version of G3P-ACBD (sequential implementation) achieved the third best ranking, which is better than CBA and FARCHD. In fact, FARCHD obtained the worst performance. Finally, focusing on those G3P-ACBD versions based on distributed platforms (Spark and Flink), really bad results were obtained. These results demonstrate that such platforms are not good options for non-big datasets. The Friedman statistical test was performed to detect whether there are statistical differences, obtaining a value of $X_F^2 = 77.129$ with a critical value of 16.812 and $p$ value = $1.399^{-14}$. As a result, and considering $\alpha = 0.01$, it is possible to assert that not all the algorithm equally behave. Thus, a post-hoc test was performed to state among which algorithms there are any statistical difference. In this regard, Table 11 shows the $p$ values for the Holland test with $\alpha = 0.01$. The multi-thread of G3P-ACBD was taking as control, and the statistical test revealed some significant differences with regard to FARCHD and distributed versions of G3P-ACBD.

**Table 11** Holland test for the runtime when the original datasets are considered with $\alpha = 0.01$

| | CBA | CBA2 | FARCHD | G3P-ACBD | | |
|---|---|---|---|---|---|---|
| | | | | Baseline | Spark | Flink |
| G3P-ACBD (multi-thread) | 0.013 | 0.809 | *0.000* | 0.156 | *0.001* | *0.000* |

Italicized values represent the best ranking value

**Table 12** Ranking for the runtime (ordered in descending order) of the different algorithms in datasets with the instances duplicated

| Algorithm | Ranking |
|---|---|
| DFAC-FFP | 3.900 |
| G3P-ACBD (multi-thread) | 2.700 |
| G3P-ACBD (Flink) | 1.750 |
| *G3P-ACBD (Spark)* | *1.650* |

Italic typeface denotes the algorithm that achieves the best ranking

### Big Data State-of-the-Art

G3P-ACBD and DFAC-FFP are the only two algorithms that have achieved a good trade-off between quality of the predictions and interpretability. The aim now is to compare the runtime for both methods. Due to several implementations of G3P-ACBD for Big Data (multi-thread, Spark, and Flink) are provided in this work, all of them have been considered in this analysis. Table 12 shows the results obtained for these algorithms. G3P-ACBD Spark appears as the most efficient method, and closely followed by G3P-ACBD Flink. On the contrary, DFAC-FFP obtained the worst performance. A Friedman rank test was performed obtaining a $X_F^2 = 19.710$ with a critical value of 11.345 and a $p$ value = 0.0001. Thus, considering $\alpha = 0.01$, it is possible to assert that not all algorithms equally behave in runtime. A Holland test is therefore carried out (see Table 13) with $\alpha = 0.01$ in order to determine significant differences among the algorithms. As a result, no statistical difference can be found among the three implementations of G3P-ACBD (Spark version obtained the best results).

## Scalability of the Different Implementations in Big Data

The goal of this section is to study the behavior (see Fig. 5) of the four different implementations of G3P-ACBD proposed in this paper (sequential, multi-thread, Spark, and Flink). Authors are aware that parallel implementations obtain better results in runtime for Big Data than the baseline (sequential version). They are also aware that the

**Table 13** Holland test for runtime when Big Data datasets are considered with $\alpha = 0.01$

| | DFAC-FFP | G3P-ACBD | |
|---|---|---|---|
| | | Multi-thread | Flink |
| G3P-ACBD (Spark) | *0.001* | 0.193 | 0.862 |

Italicized values represent the best ranking value

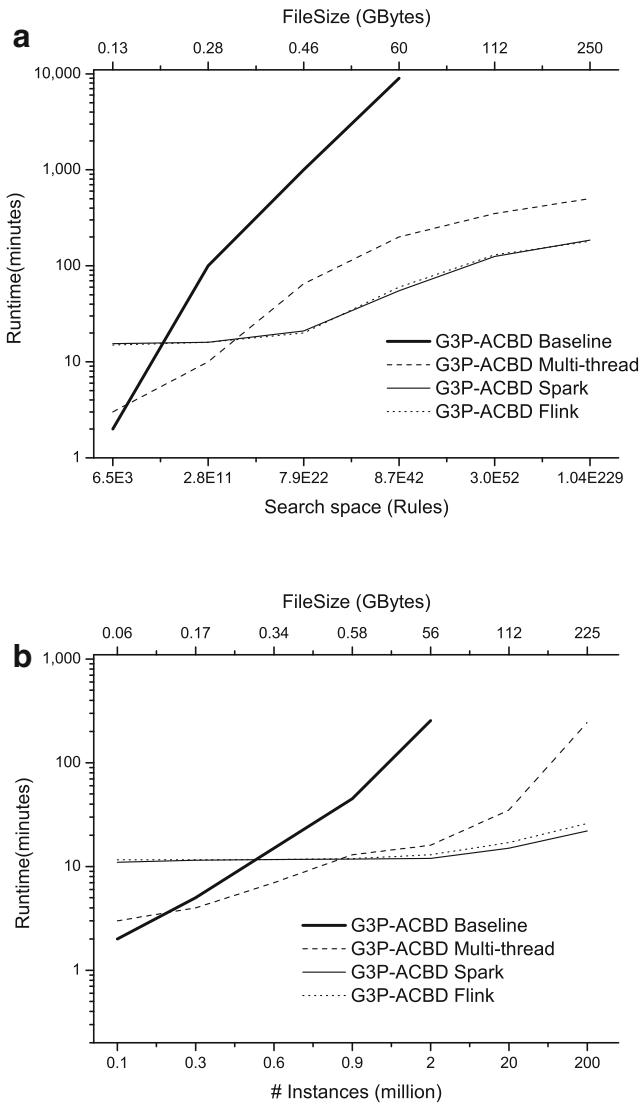


**Fig. 5** Runtime of the different implementations considering truly Big Data

multi-thread version will be worse than Spark/Flink for truly large datasets. However, the aim of this section is to know the real behavior when the search space increases and the number of instances also does. A series of synthetic datasets have been generated sampling a normal Gaussian distribution. The number of instances ranges from $1 \cdot 10^5$ to $2 \cdot 10^8$, with a search space ranging from 6500 to $1.04^{229}$, and file sizes up to 250 GB have been included. Generator is publicly available at http://www.uco.es/kdis/g3p-acbd/.

Figure 5a illustrates how the number of rules affects to the runtime. As it could be appreciated, the baseline approach has been the most efficient solution for small search spaces. When this number starts to grow, the approach based on multi-threads achieved a much better runtime than baseline version (sequential version). As for Spark and Flink implementations, they obtained good results for extremely large search spaces. For truly big search spaces, the sequential version becomes totally meaningless since it requires several days to finish. Considering both Spark and Flink versions, not high differences are found—Flink obtained a little worse runtime in some cases. Finally, it should be pointed out that Spark is more mature software than Flink and, therefore, these values might vary in future versions.

Continuing this analysis, Fig. 5b illustrates how the number of instances affects to the runtime. As previously done, different implementations of G3P-ACBD are considered and the number of instances varies in data from $1 \cdot 10^5$ to $2 \cdot 10^8$. The baseline (sequential) approach is more compelling when not so large datasets are considered, whereas parallel versions are much more appropriate when the number of instances increases. For truly large datasets (according to the number of instances), small differences in runtime are obtained between Spark and Flink.

## Conclusions

In this work, a grammar-guided genetic programming algorithm for associative classification in Big Data has been proposed. The novelty of this approach, known as G3P-ACBD, is that it is eminently designed to be as parallel as possible without affecting the accuracy and interpretability of the classifier. As a consequence of the increasing interest in data gathering, a unique and universal implementation is unfeasible and different adaptations (different levels of parallelism) are required depending on the data size. In this sense, four different versions of G3P-ACBD have been implemented including sequential, multi-thread, Apache Spark, and Apache Flink. It should be taken into account that all of them return the same classifier and the difference lies on the runtime. Additionally, a comparison with other well-known algorithms was performed by considering interpretability, efficiency, and scalability.

**Funding Information** This research was financially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund, projects TIN2017-83445-P.

## Compliance with Ethical Standards

**Conflict of interest** The authors declare that they have no conflict of interest.

**Ethical approval** This article does not contain any studies with human participants or animals performed by any of the authors.

## References


1. Fernández A, del Río S, Chawla NV, Herrera F. An insight into imbalanced big data classification: outcomes and challenges. Complex &, Intelligent Systems. 2017;3(2):105–20.
2. Chen H, Chiang R, Storey V. Business intelligence and analytics: from big data to big impact. MIS Quarterly: Management Information Systems. 2012;36(4):1165–88.
3. Cambria E, Chattopadhyay A, Linn E, Mandal B, White B. Storages are not forever. Cogn Comput. 2017;9(5):646–58.
4. Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. SIGMOD Rec. 1993;22(2):207–16.
5. Han J, Kamber M. Data mining: concepts and techniques. Morgan Kaufmann. 2011.
6. Quinlan R. C4.5: Programs for machine learning. San Mateo: Morgan Kaufmann Publishers; 1993.
7. Cortes C, Vapnik V. Support vector networks. Mach Learn. 1995;20:273–97.
8. Thabtah FA. A review of associative classification mining. Knowl Eng Rev. 2007;22(1):37–65.
9. Asghar MZ, Khan A, Bibi A, Kundi FM, Ahmad H. Sentence-level emotion detection framework using rule-based classification. Cogn Comput. 2017;9(6):868–94.
10. Liu B, Hsu W, Ma Y. Integrating classification and association rule mining. In: 4th International Conference on Knowledge Discovery and Data Mining(KDD98); 1998. p. 80–86.
11. Bechini A, Marcelloni F, Segatori A. A MapReduce solution for associative classification of big data. Inf Sci. 2016;332:33–55.
12. Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters. Communications of the ACM - 50th anniversary issue: 1958 - 2008. 2008;51(1):107–13.
13. Alcalá-Fdez J, Alcalá R, Herrera F. A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning. IEEE Trans Fuzzy Syst. 2011;19(5):857–72.
14. Venturini L, Baralis E, Garza P. Scaling associative classification for very large datasets. Journal of Big Data. 2017;4(1):44.
15. Padillo F, Luna JM, Ventura S. Exhaustive search algorithms to mine subgroups on big data using Apache spark. Progress in Artificial Intelligence. 2017;6(2):145–58.
16. Ventura S, Luna JM. Pattern mining with evolutionary algorithms. New York: Springer International Publishing; 2016.
17. Oneto L, Bisio F, Cambria E, Anguita D. SLT-based ELM for big social data analysis. Cogn Comput. 2017;9(2):259–74.
18. Kim SS, McLoone S, Byeon JH, Lee S, Liu H. Cognitively inspired artificial bee colony clustering for cognitive wireless sensor networks. Cogn Comput. 2017;9(2):207–224.
19. Al-Radaideh QA, Bataineh DQ. A hybrid approach for arabic text summarization using domain knowledge and genetic algorithms. Cogn Comput. 2018;10(4):651–69.
20. Molina D, LaTorre A, Herrera F. An insight into bio-inspired and evolutionary algorithms for global optimization: review, analysis, and lessons learnt over a decade of competitions. Cogn Comput. 2018;10(4):517–44.
21. Siddique N, Adeli H. Nature inspired computing: an overview and some future directions. Cogn Comput. 2015;7(6):706–14.
22. Lam C. Hadoop in action, 1st ed. Greenwich: Manning Publications Co.; 2010.
23. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. HotCloud'10. Berkeley, CA, USA; 2010.
24. Kumar C, Anjaiah P, Patil S, Lingappa E, Rakesh M. Mining association rules from NoSQL data bases using MapReduce fuzzy association rule mining algorithm. 2017.
25. Martín D, Martínez-Ballesteros M, García-Gil D, Alcalá-Fdez J, Herrera F, Riquelme-Santos JC. MRQAR: a generic MapReduce framework to discover quantitative association rules in big data problems. Knowl-Based Syst. 2018;153:176–92.
26. McKay RI, Hoai NX, Whigham PA, Shan Y, O'Neill M. Grammar-based genetic programming: a survey. Genet Program Evolvable Mach. 2010;11:365–96.
27. Herrera F, Carmona CJ, González P, del Jesus MJ. An overview on subgroup discovery: foundations and applications. Knowl Inf Syst. 2011;29(3):495–525.
28. Luna JM, Padillo F, Pechenizkiy M, Ventura S. Apriori versions based on MapReduce for mining frequent patterns on big data. IEEE Trans Cybern. 2017;PP(99):1–15.
29. Ben-David A. Comparison of classification accuracy using Cohen's Weighted Kappa. Expert Syst Appl. 2008;34(2):825–32.
30. Triguero I, González S, Moyano JM, Garcîa S, Alcalá-Fdez J, Luengo J, et al. KEEL 3.0: an open source software for multi-stage analysis in data mining. Int J Comput Intell Syst. 2017;10(1):1238–49.
31. Yin X, Han J. CPAR: classification based on predictive association rules. In: 3rd SIAM International Conference on Data Mining(SDM03); 2003. p. 331–5.
32. Li W, Han J, Pei J. CMAR: accurate and efficient classification based on multiple class-association rules. In: 2001 IEEE International Conference on Data Mining(ICDM01); 2001. p. 369–76.
33. Liu B, Ma Y, Wong CK. In: Classification Using Association Rules: Weaknesses and Enhancements. Kluwer Academic Publishers; 2001. p. 591–601.
34. Han J, Pei J, Yin Y, Mao R. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Disc. 2004;8(1):53–87.
35. Cohen WW. Fast effective rule induction. In: Machine Learning: Proceedings of the 12th International Conference; 1995. p. 1–10.
36. Tan KC, Yu Q, Ang JH. A coevolutionary algorithm for rules discovery in data mining. Int J Syst Sci. 2006;37(12):835–64.
37. Holte RC. Very simple classification rules perform well on most commonly used datasets. Mach Learn. 1993;11:63–91.
38. Segatori A, Bechini A, Ducange P, Marcelloni F. A distributed fuzzy associative classifier for big data. IEEE Trans Cybern. 2018;48(9):2656–69.
39. Fazzolari M, Alcalá R, Herrera F. 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. Appl Soft Comput. 2014;24:470–81.