



UNIVERSIDAD DE CÓRDOBA

PROGRAMA DE DOCTORADO EN COMPUTACIÓN
AVANZADA, ENERGÍA Y PLASMAS

Nuevos retos en clasificación asociativa: Big Data y aplicaciones

*New challenges on associative classification: Big Data and
applications*

MEMORIA DE TESIS PRESENTADA POR
Francisco Padillo Ruz

COMO REQUISITO PARA OPTAR AL
GRADO DE DOCTOR EN INFORMÁTICA

DIRECTORES

Sebastián Ventura Soto

José María Luna Ariza

14 de mayo de 2020

TITULO: *New challenges on associative classification: Big Data and applications*

AUTOR: *Francisco Solano Padillo Ruz*

© Edita: UCOPress. 2020
Campus de Rabanales
Ctra. Nacional IV, Km. 396 A
14071 Córdoba

<https://www.uco.es/ucopress/index.php/es/>
ucopress@uco.es

La memoria de Tesis Doctoral titulada "*Nuevos retos en clasificación asociativa: Big Data y aplicaciones*", que presenta Francisco Padillo Ruz para optar al grado de Doctor, ha sido realizada dentro del Programa de Doctorado en Computación, Avanzada, Energía y plasmas, bajo la dirección de los doctores Sebastián Ventura Soto y José María Luna Ariza, cumpliendo con los requisitos exigidos a este tipo de trabajos, y respetando los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Córdoba, 13 de mayo de 2020

El Doctorando

Fdo: Francisco Padillo Ruz

Directores

Fdo: Dr. Sebastián Ventura Soto

Fdo: Dr. José María Luna Ariza



TÍTULO DE LA TESIS: Nuevos retos en clasificación asociativa: Big Data y aplicaciones

DOCTORANDO/A: Francisco Solano Padillo Ruz

INFORME RAZONADO DEL/DE LOS DIRECTOR/ES DE LA TESIS

(se hará mención a la evolución y desarrollo de la tesis, así como a trabajos y publicaciones derivados de la misma).

El trabajo realizado por el doctorando durante todo el periodo de investigación ha sido muy satisfactorio, como bien justifican los resultados obtenidos. Se comenzó trabajando en un modelo evolutivo basado en MapReduce para extraer reglas de asociación en Big Data, primer paso en clasificación asociativa. Como resultado, se publicó el modelo en la revista *Integrated Computer-Aided Engineering*. Posteriormente, se comenzó a trabajar tanto en la integración de dicho modelo en un entorno de clasificación asociativa, como en el desarrollo de modelos existentes con los que realizar comparativas. El nuevo modelo dio lugar a la publicación en la revista *Cognitive Computation*. Además, el conjunto de modelos existentes y que representaban el estado del arte en clasificación asociativa fue unificado en formato librería y publicado en la revista *Knowledge-Based Systems*. Por último, destacar que algunas modificaciones de modelos existentes fueron también publicados en la revista *Big Data Analytics*, si bien esta no está incluida dentro del JCR. Además, se han presentado avances de nuestro trabajo en un congreso internacional de Big Data, como el *4th International Conference on Internet of Things, Big Data and Security*, y una propuesta evolutiva en un congreso internacional como *2017 IEEE Congress on Evolutionary Computation*.

Por todo ello, se autoriza la presentación de la tesis doctoral.

Córdoba, 13 de mayo de 2020

Firma del/de los director/es

Fdo.: José María Luna Ariza

Fdo.: Sebastián Ventura Soto

Tesis Doctoral subvencionada por la Comisión Interministerial de
Ciencia y Tecnología (CICYT) con los proyectos **TIN2014-55252-P**
y **TIN-2017-83445-P**.



Resumen

La clasificación asociativa surge como resultado de la unión de dos importantes ámbitos del aprendizaje automático. Por un lado la tarea descriptiva de extracción de reglas de asociación, como mecanismo para obtener información previamente desconocida e interesante de un conjunto de datos, combinado con una tarea predictiva, como es la clasificación, que permite en base a un conjunto de variables explicativas y previamente conocidas realizar una predicción sobre una variable de interés o predictiva. Los objetivos de esta tesis doctoral son los siguientes: 1) El estudio y el análisis del estado del arte de tanto la extracción de reglas de asociación como de la clasificación asociativa; 2) La propuesta de nuevos modelos de clasificación asociativa así como de extracción de reglas de asociación teniendo en cuenta la obtención de modelos que sean precisos, interpretables, eficientes así como flexibles para poder introducir conocimiento subjetivo en éstos. 3) Adicionalmente, y dado la gran cantidad de datos que cada día se genera en las últimas décadas, se prestará especial atención al tratamiento de grandes cantidades de datos, también conocido como Big Data.

En primer lugar, se ha analizado el estado del arte tanto de clasificación asociativa como de la extracción de reglas de asociación. En este sentido, se ha realizado un estudio y análisis exhaustivo de la bibliografía de los trabajos relacionados para poder conocer con gran nivel de detalle el estado del arte. Como resultado, se ha permitido sentar las bases para la consecución de los demás objetivos así como detectar que dentro de la clasificación asociativa se requería de algún mecanismo que facilitara la unificación de comparativas así como que fueran lo más completas posibles. Para tal fin, se ha propuesto una herramienta de software que cuenta con al menos un algoritmo de todas las categorías que componen la taxonomía actual. Esto permitirá dentro de las investigaciones del área, realizar comparaciones más diversas y completas que hasta el momento se consideraba una tarea en el mejor de los casos muy ardua, al no estar disponibles muchos de los algoritmos en un formato ejecutable ni mucho menos como código abierto. Además, esta herramienta tam-

bién dispone de un conjunto muy diverso de métricas que permite cuantificar la calidad de los resultados desde diferentes perspectivas. Esto permite conseguir clasificadores lo más completos posibles, así como para unificar futuras comparaciones con otras propuestas.

En segundo lugar, y como resultado del análisis previo, se ha detectado que las propuestas actuales no permiten escalar, ni horizontalmente, ni verticalmente, las metodologías sobre conjuntos de datos relativamente grandes. Dado el creciente interés, tanto del mundo académico como del industrial, de aumentar la capacidad de cómputo a ingentes cantidades de datos, se ha considerado interesante continuar esta tesis doctoral realizando un análisis de diferentes propuestas sobre Big Data. Para tal fin, se ha comenzado realizando un análisis pormenorizado de los últimos avances para el tratamiento de tal cantidad de datos. En este respecto, se ha prestado especial atención a la computación distribuida ya que ha demostrado ser el único procedimiento que permite el tratamiento de grandes cantidades de datos sin la realización de técnicas de muestreo. En concreto, se ha prestado especial atención a las metodologías basadas en MapReduce que permite la descomposición de problemas complejos en fracciones divisibles y paralelizables, que posteriormente pueden ser agrupadas para obtener el resultado final. Como resultado de este objetivo se han propuesto diferentes algoritmos que permiten el tratamiento de grandes cantidades de datos, sin la pérdida de precisión ni interpretabilidad. Todos los algoritmos propuestos se han diseñado para que puedan funcionar sobre las implementaciones de código abierto más conocidas de MapReduce.

En tercer y último lugar, se ha considerado interesante realizar una propuesta que mejore el estado del arte de la clasificación asociativa. Para tal fin, y dado que las reglas de asociación son la base y factores determinantes para los clasificadores asociativos, se ha comenzado realizando una nueva propuesta para la extracción de reglas de asociación. En este aspecto, se ha combinado el uso de los últimos avances en computación distribuida, como MapReduce, con los algoritmos evolutivos que han demostrado obtener excelentes resultados en el área. En particular, se ha hecho uso de programación genética gramatical por su flexibilidad para codificar las soluciones, así como introducir conocimiento subjetivo en el proceso de búsqueda a la vez que permiten aliviar los requisitos computacionales y de memoria. Este nuevo algoritmo, supone una mejora significativa de la extracción de reglas de asociación ya que ha demostrado obtener mejores resultados que las propuestas existentes sobre diferentes tipos de datos así como sobre diferentes métricas de interés, es decir, no sólo obtiene mejores resultados sobre Big Data, sino que se ha comparado en su versión secuencial con los algoritmos existentes. Una vez que se ha conseguido este algoritmo que permite extraer excelentes reglas de asociación, se ha adaptado para la obtención de reglas de asociación de clase así como para obtener un clasificador a partir de tales reglas. De nuevo, se ha hecho uso de programación genética

gramatical para la obtención del clasificador de forma que se permite al usuario no sólo introducir conocimiento subjetivo en las propia formas de las reglas, sino también en la forma final del clasificador. Esta nueva propuesta también se ha comparado con los algoritmos existentes de clasificación asociativa forma secuencial para garantizar que consigue diferencias significativas respecto a éstos en términos de exactitud, interpretabilidad y eficiencia. Adicionalmente, también se ha comparado con otras propuestas específicas de Big Data demostrado obtener excelentes resultados a la vez que mantiene un compromiso entre los objetivos conflictivos de interpretabilidad, exactitud y eficiencia.

Esta tesis doctoral se ha desarrollado bajo un entorno experimental apropiado, haciendo uso de diversos conjunto de datos incluyendo tanto datos de pequeña dimensionalidad como Big Data. Además, todos los conjuntos de datos usados están publicados libremente y conforman un conglomerado de diversas dimensionalidades, número de instancias y de clases. Todos los resultados obtenidos se han comparado con el estado de arte correspondiente, y se ha hecho uso de tests estadísticos no paramétricos para comprobar que las diferencias encontradas son significativas desde un punto de vista estadístico, y no son fruto del azar. Adicionalmente, todas las comparaciones realizadas consideran diferentes perspectivas, es decir, se ha analizado rendimiento, eficiencia, precisión así como interpretabilidad en cada uno de los estudios.

Abstract

This Doctoral Thesis aims at solving the challenging problem of associative classification and its application on very large datasets. First, associative classification state-of-art has been studied and analyzed, and a new tool covering the whole taxonomy of algorithms as well as providing many different measures has been proposed. The goal of this tool is two-fold: 1) unification of comparisons, since existing works compare with very different measures; 2) providing a unique tool which has at least one algorithm of each category forming the taxonomy. This tool is a very important advancement in the field, since until the moment the whole taxonomy has not been covered due to that many algorithms have not been released as open source nor they were available to be run.

Second, AC has been analyzed on very large quantities of data. In this regard, many different platforms for distributed computing have been studied and different proposals have been developed on them. These proposals enable to deal with very large data in a efficient way scaling up the load on very different compute nodes.

Third, as one of the most important part of the associative classification is to extract high quality rules, it has been proposed a novel grammar-guided genetic programming algorithm which enables to obtain interesting association rules with regard to different metrics and in different kinds of data, including truly Big Data datasets. This proposal has proved to obtain very good results in terms of both quality and interpretability, at the same time of providing a very flexible way of representing the solutions and enabling to introduce subjective knowledge in the search process. Then, a novel algorithm has been proposed for associative classification using a non-trivial adaptation of the aforementioned algorithm to obtain the rules forming the classifier. This methodology is also based on grammar-guided genetic programming enabling user not only to constrain the form of the rules, but the final form of the classifier. Results have proved that this algorithm obtains very accurate classifiers at the same time of maintaining a good level of interpretability.

All the methodologies proposed along this Thesis has been evaluated using a proper experimental framework, using a varied set of datasets including both classical and Big Data dataset, and analyzing different metrics to quantify the quality of the algorithms with regard to different perspectives. Results have been compared with state-of-the-art and they have been verified by means of non-parametric statistical tests proving that the proposed methods overcome to existing approaches.

Contents

Part I: Ph.D. Dissertation	1
1. Introduction	1
1.1. Unification: facilitating adoption and comparison with existing approaches	2
1.2. Scalability issues on Big Data problems	3
1.3. Building accurate, efficient, interpretable and flexible classifiers	5
2. Objectives	7
3. Methodology	9
4. Results	11
4.1. Unification: facilitating adoption and comparison with existing approaches	11
4.2. Scalability issues on Big Data problems	12
4.3. Building accurate, efficient, interpretable and flexible classifiers	13
5. Conclusions and future works	17
5.1. Conclusions	17
5.1.1. Unification: facilitating adoption and comparison with existing ap- proaches	17
5.1.2. Scalability issues on Big Data problems	18
5.1.3. Building accurate, efficient, interpretable and flexible classifiers . . .	18
5.2. Future works	19
Bibliography	21
Part II: Journal Publications	25
LAC: Library for Associative Classification	25

Evaluating associative classification algorithms for Big Data	32
A Grammar-Guided Genetic Programming Algorithm for Associative Classifica- tion in Big Data	62
Mining Association Rules on Big Data through MapReduce Genetic Programming	81

Publications in conferences	101
------------------------------------	------------

Introduction

The increasing innovation in technology over the last decades has provoked an exponential growth on both the quantity of data being generated, and its complexity [14]. The discovery of high level information and knowledge from these complex large quantities of data has become significantly ambitious and challenging. Knowledge Discovery in Databases (KDD) is the non-trivial process of extracting implicit, previously unknown and potentially useful information from a collection of raw data [12]. Additionally, as technology advances and hardware is improved, more and more data are being able to be stored, thus, the quantity of data to deal with have increased like never before [10]. Big Data is the term more and more used to comprise a subset of these techniques focused on facing up the problems derived from the management and analysis from very huge quantities of data [5].

Aiming at extracting hidden, interesting and previously unknown information from large quantities of data, many different techniques have been proposed along the years [12]. Nevertheless, all of them could be categorized in two main groups: descriptive tasks, which depict intrinsic and important properties of data; and predictive tasks, which predict an output variable for unseen data. Focusing on the last one, a diverse set of methodologies could be considered to build accurate models for predicting the output variable: rule-based systems [19], decision trees [27] and support vector machines [7], just to list a few of them. Even when all these methodologies have obtained very good results, rule-based classifiers have stand out thanks to provide a high-level interpretability, while maintaining a good trade-off in both efficiency and accuracy [20]. Such systems are mainly divided into two main groups: rule induction [6]; and classification based on Association Rule Mining (ARM) [2]. While rule induction approaches locally derive rules, AC aims at constructing global classifiers using techniques from ARM. Very briefly, ARM could be described as the process of identifying an antecedent (or condition) and a consequent (or output) that

have a conditional connection, i.e. if a condition or set of conditions take place, a result will occur with a large probability.

Classification based on ARM, generally known as Associative Classification (AC), integrates a descriptive task (ARM) in the process of generating a classifier [19]. Several researches have proved that AC has specific advantages over other classification approaches [31]. AC algorithms are able to obtain accurate and interpretable results in an efficient way thanks to leveraging association rule discovery methods in the training phase. This enables to obtain all the possible hidden relationships among the attribute values which possibly may be missed by other lesser exhaustive methodologies [6]. Furthermore, AC also enables to update and tune a subset of rules without having to redraw the whole tree as happens in decision tree approaches [20]. Last but not least, the main advantage of AC with regard to other techniques is the final model representation, which is formed by simple and easy to interpret rules that enables end-user to understand and interpret the results.

Even when AC has received attention from the research community in the last years [4, 28, 34], to the best of our knowledge there is not a universal tool with enough AC algorithms and even less covering all the diverse taxonomy of types of algorithms [31]. Furthermore, although the state-of-art in AC is very rich and is formed by very efficient algorithms, there are very few researches on proposes exclusively aimed at solving this challenging problem on Big Data. Finally, existing approaches have not yet incorporated both the Big Data branch and new advances on ARM algorithms to obtain flexible, interpretable and accurate classifiers. The goal of this dissertation is to solve the aforementioned problems. Next, each issue is analyzed in more detail and the motivation and justification behind them are also provided.

1.1. Unification: facilitating adoption and comparison with existing approaches

Nowadays there are a very varied set of techniques of AC proposed obtaining very good results [31], however there has not been almost any effort on standardizing it in one unique tool or programming language. Until the moment, researchers have focused on proposing new approaches without performing enough comparison with state-of-art or performing comparisons which may not be fair enough [1, 16]. Given that very few AC algorithms are distributed as open source, researchers have compared runtime from other algorithms using those results published on papers without having into account that the hardware between themselves and other researchers may be very different [3, 13]. Furthermore,

comparisons between algorithms implemented on very different programming languages have also been performed ignoring the implementation details, and the obtained small differences may be provoked by this fact and not by differences on the runtime complexity of the algorithms [35, 38]. Similarly, other kind of researchers have performed comparisons with some open source algorithms but they are not enough representative because existing open source tools do not cover the whole taxonomy of algorithms [31]. In this sense, very well-known existing tools such as KEEL [32] or WEKA [11] provides a subset of these algorithms but they do not cover the whole taxonomy. Even what is much more important, KDD community has not been able to adopt many of these algorithms because they are not released as open source or they use its own complex data format hampering the adoption.

Finally, related to this very fact of lacking a standardized tool, comparisons among algorithms are not universally performed, but almost each researcher uses a different set of metrics. Whereas some researchers perform comparisons having into account accuracy, they disregard classification on not balanced datasets. In this same regard, the similar problems happen with interpretability which is differently measured in each work [1].

1.2. Scalability issues on Big Data problems

Algorithms for AC commonly consist of four parts to generate the final classifier. First, continuous features are discretized. Second, patterns of associations among attributes and the class are generated by means of exhaustive search algorithms [2], and only those whose frequency of occurrence is greater than a threshold are selected for the next phase. Third, association rules are obtained using the previously mined frequent patterns. Fourth and last, rules are ranked and post-processed to build the final classifier. This methodology is very well-known and it has been widely used for almost all the state-of-art algorithms. For instance, CBA [19] and its improved version CBA2 [20] are two examples of AC algorithms that mine association rules by means of exhaustive search algorithms. However, this is a major drawback when large datasets are required to be analysed since for a dataset comprising k single items, a total of $3^k - 2^{k+1} + 1$ rules can be computed and saved in memory. To solve this issue, additional approaches, e.g. CMAR [18], have been proposed which are mainly based on novel data structures that avoid the generation of any candidate. Another example is CPAR [38], which combines the advantages of AC with traditional rule induction algorithms. While these algorithms work well for not so big datasets, they cannot be used in Big Data due to the associated complexity [22]. Under these circumstances, new forms of building this type of classifiers from a Big Data

perspective is an interesting and emerging topic [30], which has not received the needed attention yet.

Recent advances on distributed computing has been more and more used to solve the aforementioned problems [28]. In this sense, MapReduce [8] is a recent paradigm of distributed computing in which programs are composed of two main stages, 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. Among the many open source implementations available, Hadoop [17] is the *de facto* standard for MapReduce applications. Even when Hadoop implements these paradigms efficiently, its major drawback is it imposes an acyclic data flow graph, and there are applications that cannot be modelled efficiently using this kind of graph such as iterative or interactive analysis [39]. To solve these downsides, Apache Spark has risen up for solving all the deficiencies of Hadoop, introducing an abstraction called Resilient Distributed Datasets (RDD) to store data in main memory and a new approach using micro-batch technology. Unfortunately, Spark does not support native iterations, which means 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 where delay of the order of milliseconds can cause significant impact. Furthermore, Flink includes its own memory manager reducing the time required by garbage collector, whereas Spark addresses this issue later with Tungsten optimization project [37]. In this sense, Apache Flink has been proposed to face the problems of Spark and to address the problem of streaming applications differently and in a more native way (vs the micro-batch methodology of Spark). Apache Spark was commonly being used as the most suitable in-memory Big Data analytic tool when Apache Flink came along. This fact may have hampered the appearance of Flink, although in the last years the attention on this platform has risen up.

In conclusion, although there have been some advances on distributed computing and even some of them have been incorporated into the AC field [34], existing approaches for AC in Big Data are not able to scale up for truly Big Data, or if they do, they completely sacrifice both accuracy and interpretability of the classifiers.

1.3. Building accurate, efficient, interpretable and flexible classifiers

Even when AC is a mature and studied field, and many different methodologies have been proposed along the years, almost all of them have completely ignored the interpretability and the flexibility of the classifiers. Existing researches have been mainly focused on optimizing the accuracy of the classifiers at the cost of reducing interpretability. In this sense, current approaches in AC obtain very complex classifiers formed by many rules and complicated rules, hampering the ability of the user to follow the logic behind the prediction or to extract interesting and previously unknown information from the classifiers. In order to solve these problems, many studies have proved that a trade-off between these conflicting goals are required to guarantee their applicability on many different real-world problems [20]. This challenge is even worse when considering Big Data problems where there are tons of available data, and where their properties are even more complex. All these issues have hampered the use of AC and specially of exhaustive search algorithms on these kind of data.

Aiming at solving these challenges, and specially on Big Data, two main issues should be addressed:

- Optimizing the phase of obtaining rules and improving its quality. As it was previously aforementioned, this phase is hindered by the use of exhaustive search methodologies which take a very long time to obtain all the rules. In this sense, Evolutionary Algorithms (EAs) have been successfully used on ARM to alleviate the runtime and memory requirements [33]. In concrete terms, a significant extension of genetic programming, known as Grammar-Guided Genetic Programming (GGGP or G3P), has proved to reduce the memory and computational complexity of exhaustive methodologies while obtaining both very interesting and interpretable rules [21]. G3P thanks to its flexible representation enables to remove the first step of discretizing the numeric attributes followed in classical AC algorithms. Finally but not least, G3P also makes use of a context-free grammar to encode the solutions allowing to restrict the search space by adding some syntax constraints, i.e., it enables expert's knowledge to be introduced into the mining process. This prior knowledge is highly important since it provides domain consistency and it drastically reduces the number of admissible solutions. Additionally, the obtained rules form the foundations for the classifier, so more attention has to be given to this key part of the algorithms to guarantee more accurate and interpretable classifiers.
- Improving scalability of the classifiers while obtaining a good trade-off between ac-

curacy and interpretability. Existing approaches for AC do not incorporate the last trends on ARM, but they are based on basic algorithms which have been improved or surpassed on more recent works [31]. This very fact is even worse on current Big Data methodologies for AC where rules are obtained following naive or older methodologies. Furthermore, existing AC methodologies for Big Data have disregarded interpretability or flexibility despite being one of the main reasons while using AC [19]. Finally, existing methodologies for AC are not flexible and experts are not able to constrain or to guide the final form of the classifiers hampering the applicability of very large search spaces or directly its complete applicability on some real-world domains. Each one of the aforementioned problems have proved to be interesting, and no enough attention has been given due to its complexity and the challenge that it involves.

Objectives

The main objective of this dissertation is to propose new, accurate, efficient and interpretable AC algorithms which are able to tackle very high-dimensional data by means of G3P algorithms and distributed computing. To achieve this aim, the following objectives were pursued.

- Analysis of the start-of-art in the AC field to detect drawbacks of existing methodologies, and to pinpoint new and exciting open challenges. Analyzing flexibility, comprehensibility and accuracy of the existing algorithms.
- Analysis and study of the new and appealing distributed platforms to speed up and scale up the performance of the algorithms. Analysis on the flexibility, efficiency and accuracy of current algorithms when they are adapted to work on those distributed platforms.
- Design and implementation of novel ARM algorithms to obtain very interesting rules considering EAs and G3P methodologies to overcome the computational and memory requirements of other methodologies. Although the focus is on Big Data datasets, it should be also able to obtain very good results on classical datasets.
- Making use of the previously obtained algorithm to mine association rules, to propose new AC algorithms which are able to solve the conflicting goal of performing accurate predictions while maintaining interpretable rules. Furthermore, special focus is given on the flexibility of the solutions and the possibility of introducing subjective knowledge on the search process. Finally, special attention would be given to comprehensibility, accuracy, complexity, flexibility and human understanding of the final classifiers.

Methodology

The goal of this chapter is to provide a detailed explanation on the methods and tools used in the development in this dissertation. A more detailed description of the employed methodologies for each of the experimental studies performed along this thesis is provided in each respective article.

Datasets

All the datasets used in this dissertation are publicly available and are free to be downloaded and used. Datasets have been obtained mainly from two repositories, UCI machine learning repository¹ and the KEEL repository². At least 40 different datasets have been considered in each work performed in this dissertation and, in some cases, this number is even larger (around 75 different datasets). They contain a varied set of properties such as, the number of attributes ranges from 3 up to 2000, the number of instances ranges from 40 up to 34,890,838, and file sizes up to 800 GBytes have been analyzed. This variability on the properties enables to evaluate the performance, accuracy and soundness of the methodologies under very different problems.

¹<https://archive.ics.uci.edu/ml/datasets.php>

²<https://sci2s.ugr.es/keel/datasets.php>

Software

In this work different well-known distributed platforms have been used (Apache Hadoop³, Apache Spark⁴ and Apache Flink⁵). Furthermore, another type of framework as Remote Method Invocation (RMI) for Java has also been considered. Depending on the platform two different programming languages have been used. Java has been mainly used to work with Apache Hadoop, RMI and in a sequential fashion. When considering more functional-oriented platforms as Apache Spark or Apache Flink, Scala has been considered for being a much better fit for this kind of work. Both languages run on the Java Virtual Machine so there is not much many differences regarding performance between them. Operating system was Linux CentOS 6.3.

Hardware

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.

Performance evaluation

The evaluation framework employed in the experimentation includes very well-known measures to quantify the quality of the solutions both with regard to accuracy and interpretability [31]. A 10-fold stratified cross-validation has been used [15]. Stochastic algorithms have been run at least 10 times with different seeds to avoid randomness on the result. Statistical tools as non-parametric tests, such as Wilcoxon signed-rank [36], Friedman and Holland tests [29], have also been used to prove the significance of the conclusions [9].

³<https://hadoop.apache.org>

⁴<https://spark.apache.org>

⁵<https://flink.apache.org>

Results

This chapter presents a discussion of the results achieved while pursuing the objectives aimed at this thesis.

4.1. Unification: facilitating adoption and comparison with existing approaches

As it was previously stated, even when there are many interesting approaches for AC, there are not any kind of tools or libraries which contain a varied set of AC algorithms. In this regard, existing tools as KEEL [32] or WEKA [11] have some algorithms, but they do not cover the whole taxonomy of algorithms and they are not exclusively limited to AC [31]. Moreover, the community lacks a suitable software tool that can integrate the major works in the field as others have previously done. In [26] we have proposed, to the best of our knowledge, the very first AC library known as LAC (Library for Associative Classification). Unlike existing tools, LAC covers the whole taxonomy containing at least one algorithm for each category of algorithms. In this sense, LAC even includes algorithms which have not been publicly released as open source (nor as a private software). Nevertheless LAC is not exclusively limited to algorithms, but it also includes several quality measures to quantify different perspective of the results. 6 well-known metrics have been included regarding accuracy of the solutions (Accuracy, Kappa, Recall, Precision, F-Measure micro, F-Measure macro), and confusion matrix has also been included. With regard to interpretability, LAC provides 2 measures to quantify the interpretability of the solutions (number of rules forming the classifier and the average number of attributes in those rules). Additionally, LAC includes three different input formats (CSV, KEEL and ARFF) to guarantee its adoption on existing datasets. While considering output, LAC

provides different reports to satisfy all the requirements of the user.

LAC is configured by means of YAML file easing the work. YAML enables to share configuration for the algorithm in all the datasets or only in the specified one, facilitating future changes in the parameters. Unlike existing approaches, LAC does not require specifying all the parameters, but if it is not specified, default values will be used reducing the complexity of configuration for the majority of users. No restriction is imposed with regard to datasets, that is, configuration file could be created on a computer where datasets are not present, and then the configuration file could be moved to a remote server where the datasets are present. This reduces the disk usage, since no datasets is duplicated nor personal computer has to have the dataset but only the remote server where the execution will be run and it also reduces the network use since a YML file is a text-file which may take some KB in very heavy scenarios being bytes in the majority of cases. LAC enables to use one big file with all the experimental study to run, or to split that file in as many as you want, and run all of them at the same time.

Finally, LAC aims at being the standardized tool in AC, thus special attention has been given to the hierarchy of classes and its software design. In this sense, each algorithm in LAC has to implements an interface to guarantee that each algorithm has to follow the same conventions. LAC provides several utilities classes to perform common operations of AC algorithms. New algorithms are completely abstracted from the input type being used, since this logic is covered by the layer of inputs. In the same regard, output neither has not to be implemented in each algorithm but this responsibility is covered by the layer of outputs. Reading of the configuration or parsing are also completely transparent to developers adding new algorithms, since they only have to specify which parameters requires its algorithm and their types. These facts, facilitate the creation of new algorithm in LAC, since developers of new algorithms only have to know how to implement their algorithms and can be completely focused on only that task. The abstraction is done up to that level that developers do not need to register their algorithms, but LAC is able to automatically detect, load and run them. Finally, it should be highlighted that to ease the labor of adding new algorithms the manual covers a complete example, and provide a guided and easy to follow steps (<https://github.com/kdis-lab/lac/blob/main/doc/manual.pdf>).

4.2. Scalability issues on Big Data problems

Even when there has been recent advances on distributed computing, these have not been fully incorporated in AC. Some works have been initiated on the adoption on these

distributed platforms but the approach followed have been mainly on sampling techniques or generating classifiers for subsets of the classifiers, and subsequently joining them on only one final classifier. These new methodologies are not able to scale up for truly Big Data, or if they do, they completely sacrifice both accuracy and interpretability of the classifiers. In this sense, in [25] we have proposed two different methods based on traditional algorithms for AC (CBA and CPAR) through emerging paradigms of distributed computing (Spark and Flink). In this regard, CBA and CPAR were selected since, according to some authors [18, 19, 38], these are the most interesting ones based on accuracy and interpretability. CBA is considered as the algorithm that obtains the most interpretable classifiers, whereas CPAR is able to obtain very accurate classifiers with accuracy values, in average, greater than those obtained by CBA [38]. The new algorithms based on distributed platforms return the same results as those obtained on the sequential approaches.

An experimental study has been performed on 40 datasets. Results have been analyzed by means of non-parametric tests, and they proved that CBA-Spark/Flink obtained interpretable classifiers but it was more time consuming than CPAR-Spark/Flink. Thus in this Ph.D. dissertation, it has been demonstrated that the proposals have been able to run on Big Data (file sizes up to 200 GBytes). The analysis of different quality metrics revealed that no statistical difference can be found for these two approaches.

Once of the goal of this Ph.D. dissertation is to determine if it is required to develop new algorithms, or it is enough if existing algorithms are adapted to be run on distributed platforms. Even when we have been able to run on large datasets, the process was time-consuming and require many main memory. Both points were caused because these algorithms generate many different rules which are redundant and hence discarded on following phases (but in the meantime they are considered, and therefore they has to be saved on memory). Regarding scalability, three different metrics (speed-up, scale-up and size-up) have also been analyzed to determine the behavior of the algorithms on Big Data. For our cluster we have found that at one particular level more compute nodes do not improve as much as expected, meaning that scalability of these approaches may be limited to certain point (250 GBytes). Notwithstanding, both approaches have been able to tackle larger datasets than those reported by other AC algorithms until the moment [31].

4.3. Building accurate, efficient, interpretable and flexible classifiers

Even when AC is a mature and studied field, and many different methodologies have been proposed along the years, almost all of them have mainly focused on maximizing

the accuracy. However, one of the main reasons behind using AC approaches are obtaining both interpretable and flexible classifiers, being those objective until a certain point conflicting goals with maximizing accuracy. In this sense, current approaches in AC obtain very complex classifiers formed by many rules and complicated rules, hampering the ability of the user to follow the logic behind the prediction or to extract interesting and previously unknown information from the classifiers. In order to solve these problems, two different actions have been taken in this thesis:

- Optimizing the phase of obtaining rules and improving its quality. In [23] we have proposed an ARM algorithm to obtain very interesting rules on Big Data. In this sense, we have proposed a new efficient EA to extract association rules in Big Data. The baseline of this work is a new Grammar-Guided Genetic Programming algorithm to optimize Leverage, Support and Confidence, known as G3P-LSC. The proposed model makes use of a context-free grammar to encode the solutions and it allows to restrict the search space by adding some syntax constraints, i.e. it enables expert's knowledge to be introduced into the mining process. Furthermore, our proposal is eminently designed to be as parallel as possible so Big Data can be tackled, and its operators have been specifically designed to avoid the loss in large search spaces as well as to maintain diversity in the solutions. In this regard, its genetic operators provide a reduced set of rules with high values for many different quality measures and few attributes, making it easier to understand from a user's perspective. Due to the growing interest in data gathering, a unique implementation of the proposed algorithm is not useful. In this sense, during this Ph.D. thesis different implementations (considering different architectures such as RMI, Hadoop and Spark) have been developed depending on the data size. All these adaptations obtain exactly the same solutions as those of the original algorithm since they only differ on the software architectures. When comparing the obtained results with other 14 algorithms, using 12 different metrics and more than 75 datasets, it is obtained that the work performed along this Ph.D. thesis mines rules with better values for interesting metrics and few attributes, providing the user with high quality rules. In order to analyze whether exists any statistical difference, several non-parametric tests were carried out, proving that the results obtained in this dissertation are statistically significant. Finally, the scalability is also analyzed by considering the three parallel implementations on high dimensional datasets (3,000 millions of instances) and file sizes up to 800 GB. This fact proves that the work developed along this Ph.D presents a good computational cost and a promising scalability when the size of the problem increases.

- Combining all the state-of-art ARM, distributed computing and ideas from ARM in AC. The state-of-the-art in associative classification includes interesting approaches for building accurate and interpretable classifiers [31]. These approaches generally work on four different phases (data discretization, pattern mining, rule mining, and classifier building), some of them being computational expensive. In [24] we have proposed a novel evolutionary algorithm for efficiently building associative classifiers in Big Data. The work proposed in this thesis makes use of 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. 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). Thanks to using the key ideas of our previous work [23] in the first phase, we could guarantee that the obtained rules were very interesting, and we were able to tackle very large datasets using this philosophy. In the second phase, 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 work of this thesis since many AC approaches have been focused on improvements of classification accuracy, not paying attention to the imbalance problem. Additionally, we are also adapted a grammar-guided genetic programming framework to be able to introduce subjective knowledge in the final classifier. As it was previously aforementioned, in this Ph.D. thesis each algorithm 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. Finally, in an experimental analysis, the results obtained during this thesis has been compared to multiple AC approaches as well as traditional classification algorithms. Both sequential and trending MapReduce AC algorithms have been considered. Experiments were performed on a total of 40 datasets (including large datasets of 250 GBytes) and results were validated by non-parametric statistical tests regarding three different levels: quality of the predictions, level of interpretability, and efficiency (ability to scale up). Hence, we conclude that during this Ph.D. thesis we have obtained accurate and interpretable classifiers in an efficient way even on high-dimensional data, outperforming the state-of-art algorithms.

Conclusions and future works

The goal of this chapter is to summarize the concluding remarks obtained during this Ph.D. dissertation and to provide some future works.

5.1. Conclusions

In this Ph.D. thesis we have explored the use of G3P on AC. We have exhaustively reviewed existing bibliography, identified open issues and proposed solutions for them. In more concrete terms, different objectives have been pursued, namely the unification of algorithms, the scalability of the algorithms and the building of interpretable, flexible and accurate associative classifiers.

5.1.1. Unification: facilitating adoption and comparison with existing approaches

First, we have carefully studied and analyzed the state-of-the-art in AC. As a result of this process, we have identified an open issue, there was not any specific tool for AC, many algorithms were not available to be run and even less as open source. Furthermore, existing works use very different metrics to quantify the quality of the solutions or they were not quantifying all the perspective while considering AC (interpretability, accuracy and efficiency). In this sense, in [26] we have proposed a tool, known as LAC, which covers the whole taxonomy of AC. To the best of our knowledge this is the very first tool for AC, and while some tools exists with some algorithms they were very limited (having very few algorithms) or they imposed very specific data formats. LAC provides support for different input formats to guarantee compatibility with existing tools (CSV, KEEL and ARFF). Including 10 well-known algorithms from the AC field, and covering the whole

taxonomy. With regard to results, LAC provides different kind of outputs, being the most basic one a summary containing accuracy and runtime, but it also supports more extensive outputs containing many different measures to quantify both interpretability and accuracy. Additionally, LAC also provides an experimental framework which enables to automate and parallel the experimentation. In this sense, LAC is able to run each algorithm contained in the experimental study in one different thread to speed-up the overall time required for experimentation. This tool has been released as open source and distributed under GPLv3.

5.1.2. Scalability issues on Big Data problems

Next, we have continued analyzing the associative classification field and we have discovered that existing proposals do not scale with large quantities of data. Concretely, these algorithms were not able to neither scale horizontally nor vertically. This problem has proved to be very challenging specially on exponential search spaces as those shown in pattern mining. Thus, we follow a complete different approach to scale up. In [25] we have proposed two different algorithms to deal with Big Data based on different platforms for distributed computing. In this sense, Apache Spark and Apache Flink have been used to scale up two well-known algorithms. This process of scalability was achieved without altering the accuracy of the final classifiers neither its interpretability, but both the sequential approach as the distributed returns the very same results. The work performed during this Ph.D. thesis has behaved very well on large quantities of data, and no statistical significant difference could be found with regard to accuracy. Although these results were very interesting and achieved a very good performance on Big Data, they were not enough when dealing with very complex data or different types of attributes.

5.1.3. Building accurate, efficient, interpretable and flexible classifiers

Finally, we have approached the challenge of improving existing state-of-the-art for AC, including but not being exclusively limited to Big Data. In this sense, as the most important part for associative classifiers are the rules, we have opened the problem proposing a very efficient algorithm for obtaining association rules in [23]. This algorithm aims at obtaining very interesting association rules as well as maintaining the interpretability. Thanks to its flexible representation, it is able to encode many different kinds of data as well as enabling to introduce subjective knowledge in the search space thanks to its context-free grammar. The experimental study developed in this Ph.D. thesis for this

work includes more than 14 different algorithms using more than 75 datasets and analyzing the behavior of 12 quality measures. Different implementations have been presented depending on the data size. Finally, in terms of scalability has obtained excellent results being able to tackle up file size up to 800 GBytes.

Once, we have obtained very interesting rules with the aforementioned algorithm. We have adapted the proposal to obtain class association rules, and in [24] we have proposed a novel algorithm for associative classification. This work includes the philosophy of the first for mining association rules, including the grammar to constrain the form of the rules, and it also includes a grammar for the second phase of the algorithm, that is, the post-processing of the rules. This second grammar enables to constrain the form of the classifier, and to introduce subjective knowledge in the form of the final classifier. This approach has also been developed on different levels of parallelism depending on the data size. Results have been compared to 13 different algorithms, with different measures (accuracy, interpretability and efficiency) on many different datasets. Non-parametric tests have been performed, and they have proved that the results obtained along this Ph.D. thesis have statistical significant differences with regard to the state-of-art algorithms.

5.2. Future works

The goal of this section is to provide future lines of research that have given rise along the development of this dissertation.

First, there are methodologies of ARM which aim to extract rare association rules. These kinds of rules enables to obtain interesting and unknown information for very infrequent patterns. Thus, the synergy of rare association rules with AC may be beneficial to obtain classifiers for identifying outliers or rare patterns.

Second, multi-objective algorithms have proved to obtain excellent results for optimizing multiple and conflicting objectives. The combination of multi-objective with distributed computing may be interesting to analyze in future works, since the distribution of the objective along with the data may achieve very good results.

Bibliography

- [1] N. Abdelhamid, A. Ayesh, F. Thabtah, S. Ahmadi, and W. Hadi, “Mac: A multiclass associative classification algorithm,” *Journal of Information Knowledge Management*, vol. 11, 06 2012.
- [2] R. Agrawal, T. Imieliński, and A. Swami, “Mining association rules between sets of items in large databases,” *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, 1993.
- [3] E. Baralis and P. Garza, “A lazy approach to pruning classification rules,” in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, Dec 2002, pp. 35–42.
- [4] A. Bechini, F. Marcelloni, and A. Segatori, “A mapreduce solution for associative classification of big data,” *Information Sciences*, vol. 332, pp. 33 – 55, 2016.
- [5] H. Chen, R. Chiang, and V. Storey, “Business intelligence and analytics: From big data to big impact,” *MIS Quarterly: Management Information Systems*, vol. 36, no. 4, pp. 1165–1188, 2012.
- [6] P. Clark and T. Niblett, “The cn2 induction algorithm,” *Machine Learning Journal*, vol. 3, no. 4, pp. 261–283, 1989.
- [7] C. Cortes and V. Vapnik, “Support vector networks,” *Machine Learning*, vol. 20, pp. 273–297, 1995.
- [8] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, vol. 51, no. 1, pp. 107–113, 2008.
- [9] J. Derrac, S. García, D. Molina, and F. Herrera, “A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing

- evolutionary and swarm intelligence algorithms,” *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3 – 18, 2011. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2210650211000034>
- [10] A. Fernández, S. del Río, N. V. Chawla, and F. Herrera, “An insight into imbalanced big data classification: outcomes and challenges,” *Complex & Intelligent Systems*, vol. 3, no. 2, pp. 105–120, Jun 2017.
- [11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: An update,” *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.
- [12] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.
- [13] Z. Huang, Z. Zhou, T. He, and X. Wang, “Acac: Associative classification based on all-confidence,” 11 2011, pp. 289–293.
- [14] N. Khan, I. Yaqoob, I. A. T. Hashem, Z. Inayat, W. K. Mahmoud Ali, M. Alam, M. Shiraz, and A. Gani, “Big Data: Survey, Technologies, Opportunities, and Challenges,” *The Scientific World Journal*, vol. 2014, pp. 1–18, 2014. [Online]. Available: <http://www.hindawi.com/journals/tswj/2014/712826/>
- [15] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 1137–1143.
- [16] G. Kundu, M. M. Islam, S. Munir, and M. F. Bari, “Acn: An associative classifier with negative rules,” in *2008 11th IEEE International Conference on Computational Science and Engineering*, July 2008, pp. 369–375.
- [17] C. Lam, *Hadoop in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2010.
- [18] W. Li, J. Han, and J. Pei, “Cmar: Accurate and efficient classification based on multiple class-association rules,” in *2001 IEEE International Conference on Data Mining(ICDM01)*, 2001, pp. 369–376.
- [19] B. Liu, W. Hsu, and Y. Ma, “Integrating classification and association rule mining,” in *4th International Conference on Knowledge Discovery and Data Mining(KDD98)*, 1998, pp. 80–86.

- [20] B. Liu, Y. Ma, and C. Wong, *Classification Using Association Rules: Weaknesses and Enhancements*. Kluwer Academic Publishers, 2001, pp. 591–601.
- [21] J. M. Luna, J. R. Romero, and S. Ventura, “G3PARM: A grammar guided genetic programming algorithm for mining association rules,” in *Proceedings of the IEEE Congress on Evolutionary Computation*, ser. IEEE CEC 2010, Barcelona, Spain, 2010, pp. 2586–2593. [Online]. Available: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5586504>
- [22] L. Oneto, F. Bisio, E. Cambria, and D. Anguita, “Slt-based elm for big social data analysis,” *Cognitive Computation*, vol. 9, no. 2, pp. 259–274, Apr 2017.
- [23] F. Padillo, J. M. Luna, H. F., and S. Ventura, “Mining association rules on big data through mapreduce genetic programming,” *Integrated Computer-Aided Engineering*, vol. 25, no. 1, pp. 31–48, 2018. [Online]. Available: <https://doi.org/10.3233/ICA-170555>
- [24] F. Padillo, J. M. Luna, and S. Ventura, “A grammar-guided genetic programming algorithm for associative classification in big data,” *Cognitive Computation*, vol. 11, no. 3, pp. 331–346, 2019. [Online]. Available: <https://doi.org/10.1007/s12559-018-9617-2>
- [25] —, “Evaluating associative classification algorithms for big data,” *Big Data Analytics*, vol. 4, no. 1, p. 2, 2019. [Online]. Available: <https://doi.org/10.1186/s41044-018-0039-7>
- [26] —, “Lac: Library for associative classification,” *Knowledge-Based Systems*, p. 105432, 2019. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0950705119306586>
- [27] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.
- [28] A. Segatori, A. Bechini, P. Ducange, and F. Marcelloni, “A distributed fuzzy associative classifier for big data,” *IEEE Transactions on Cybernetics*, vol. 48, no. 9, pp. 2656–2669, Sept 2018.
- [29] D. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. Chapman and Hall/CRC, 2003.
- [30] N. Siddique and H. Adeli, “Nature inspired computing: An overview and some future directions,” *Cognitive Computation*, vol. 7, no. 6, pp. 706–714, Dec 2015.

- [31] F. A. Thabtah, “A review of associative classification mining,” *Knowledge Engineering Review*, vol. 22, no. 1, pp. 37–65, March 2007.
- [32] I. Triguero, S. González, J. M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M. J. del Jesús, L. Sánchez, and F. Herrera, “Keel 3.0: an open source software for multi-stage analysis in data mining,” *International Journal of Computational Intelligence Systems*, vol. 10, no. 1, pp. 1238–1249, 2017.
- [33] S. Ventura and J. M. Luna, *Pattern Mining with Evolutionary Algorithms*. Springer International Publishing, 2016.
- [34] L. Venturini, E. Baralis, and P. Garza, “Scaling associative classification for very large datasets,” *Journal of Big Data*, vol. 4, no. 1, p. 44, Dec 2017.
- [35] K. Wang, S. Zhou, and Y. He, “Growing decision trees on support-less association rules,” in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, 2000, pp. 265–269. [Online]. Available: <https://doi.org/10.1145/347090.347147>
- [36] F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics*, vol. 1, pp. 80–83, 1945.
- [37] R. Xin and J. Rose, “Project tungsten: Bringing apache spark closer to bare metal,” url<https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html>, 2015.
- [38] X. Yin and J. Han, “Cpar: Classification based on predictive association rules,” in *3rd SIAM International Conference on Data Mining(SDM03)*, 2003, pp. 331–335.
- [39] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud’10, Berkeley, CA, USA, 2010.

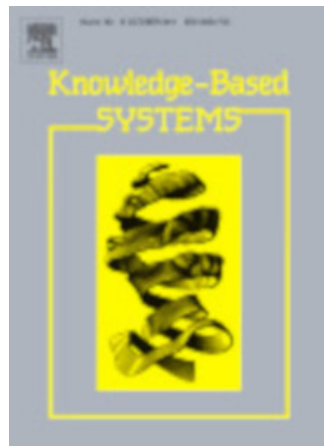
Part II: Journal Publications

TITLE:

LAC: Library for Associative Classification

AUTHORS:

F. Padillo, J.M. Luna and S. Ventura



Knowledge-Based Systems, pp. 105432, 2019

RANKING:

Impact factor (2018 JCR): 5.101

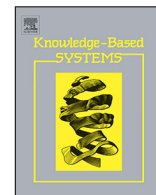
Knowledge area: Computer Science.

DOI: 10.1016/j.knosys.2019.105432



Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Original software publication

LAC: Library for associative classification[☆]Francisco Padillo^a, Jose Maria Luna^{a,c}, Sebastian Ventura^{a,b,c,*}^a Department of Computer Science and Numerical Analysis, University of Cordoba, Spain^b Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia^c Knowledge Discovery and Intelligent Systems in Biomedicine Laboratory, Maimonides Institute of Biomedicine, Cordoba, Spain

ARTICLE INFO

Article history:

Received 5 September 2019

Received in revised form 21 November 2019

Accepted 23 December 2019

Available online xxxx

Keywords:

Associative classification

Association rule mining

Java class library

Classification software

ABSTRACT

The goal of this paper is to introduce LAC, a new Java Library for Associative Classification. LAC is the first tool that covers the full taxonomy of this classification paradigm through 10 well-known proposals in the field. Furthermore, it includes several measures to quantify the quality of the solutions as well as different input/output data formats. Last but not least, the library also provides a framework to automate experimental studies, supporting both sequential and parallel executions. Thanks to the GPLv3 license, LAC is totally free, open-source and publicly available.

© 2019 Elsevier B.V. All rights reserved.

1. Introduction

Classification based on association rule mining, generally known as Associative Classification (AC), integrates a descriptive task (association rule mining [1]) in the process of inferring a new classifier [2]. Recent studies [3] have shown that AC has the following advantages over traditional classification approaches: (1) Accuracy [2], 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; (2) Usability [4], unlike decision tree approaches, AC does not require to redraw the whole model when the rule set is updated and tuned; (3) Readability [3], the final model of AC comprises a simple set of rules that allow the end-user to easily understand and interpret the results. Nonetheless, it is important to remark that AC models are not often as accurate as black-box models [5].

AC could be seen as a special case of Association Rule Mining (ARM) in which only the class attribute is considered in the consequent part of the rule, aiming at obtaining this kind of classifiers many different approaches have been proposed [3]. Almost all of them share a two-steps process to obtain the final classifier [2]. First, rules are obtained using adapted versions of algorithm from ARM. Second, once the rules have been mined,

they are post-processed to remove redundant rules, sorted to improve accuracy and to form the final classifier. While in the former many algorithms share the same philosophy with small differences, in the latter many more differences exist since each algorithm use a different step to perform the post-processing of the rules. By this last reason, existing work has centered on classifying algorithms in a taxonomy based on the first step used to obtain the rules [3]. In this regard, Abdelhamid *et al.* proposed the following criteria: apriori-like (as CBA), FP-Growth-like (as CMAR), greedy-like (as CPAR), charm-like (as ACCF), multiple supports (as CBA2), emerging patterns (as ADT), TID list intersection (as MAC).

However, in spite of AC is really interesting for the research community, to the best of our knowledge there is not any available library or tool [6,7] covering the full taxonomy of AC [3]. Existing tools have its own restrictions with regard to input format, quality metrics, and the inability to automate and parallel experimental studies. Moreover, the community lacks a suitable software tool that can integrate the major works in the field as others have previously done [8]. Unlike existing tools, LAC includes algorithms which have not been published released as open source until now (nor as a private software). Specific measures for AC has also been included to better quantify the interpretability of the models, and many different but well-known metrics have also been added to gauge the quality of the obtained solutions. Thus, the main contributions of LAC could be summarized in four points: (1) It covers the whole AC taxonomy; (2) It includes plenty of quality measures to quantify not only the quality but also the interpretability; (3) It is easy to be used with multiple input data formats; (4) It enables to fully automate and parallelize the experimental studies.

[☆] No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to <https://doi.org/10.1016/j.knosys.2019.105432>.

* Corresponding author at: Department of Computer Science and Numerical Analysis, University of Cordoba, Spain.

E-mail address: sventura@uco.es (S. Ventura).

<https://doi.org/10.1016/j.knosys.2019.105432>

0950-7051/© 2019 Elsevier B.V. All rights reserved.

Table 1

Software metadata.

Nr.	(Executable) software metadata description	Please fill in this column
S1	Current software version	0.2.0
S2	Permanent link to executables of this version	https://github.com/kdis-lab/lac/releases/tag/v0.2.0
S3	Legal Software License	GPLv3
S4	Computing platforms/Operating Systems	GNU Linux, OS X, Microsoft Windows
S5	Installation requirements & dependencies	Java version 1.8 or higher.
S6	If available, link to user manual—if formally published include a reference to the publication in the reference list	https://github.com/kdis-lab/lac/tree/v0.2.0/doc/manual.pdf
S7	Support email for questions	sventura@uco.es

Table 2

Code metadata.

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.2.0
C2	Permanent link to code/repository used for this code version	https://github.com/kdis-lab/lac/tree/v0.2.0
C3	Legal Code License	GPLv3
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Java version 1.8 or higher. Maven 3.3.9 or higher.
C6	Compilation requirements, operating environments & dependencies	SnakeYAML 1.24. maven-jar-plugin 3.1.2. maven-assembly-plugin 3.1.1. junit 4.12. mockito 3.1.0. system-rules 1.19.0
C8	If available Link to developer documentation/manual	https://github.com/kdis-lab/lac/tree/v0.2.0/doc/manual.pdf
C8	Support email for questions	sventura@uco.es

The rest of the manuscript is organized as follows: Section 2 presents the software framework, Section 3 shows some illustrative examples and finally Section 4 presents some conclusions.

2. LAC software

The LAC code and its documentation are publicly available under GPLv3 license at Github (<https://github.com/kdis-lab/lac/>)

2.1. Software architecture

LAC has been developed using Java. It only has one external dependency, called SnakeYaml, used to read yaml files (used for configuration and the automation framework). Next, each package is briefly described.

- *lac.algorithms* includes the base classes for all the algorithms. Each algorithm is contained in one package with the same name as the algorithm. For example, considering the CBA [2] algorithm, the name of the package must be *lac.algorithms.cba*.
- *lac.data* includes several classes to represent the data, and one class for each format of input file. At the current version, the following formats are supported: arff [7], keel dat [6] and csv [9].
- *lac.metrics* includes all the metrics to quantify the quality of the solutions. The following measures are supported: *accuracy*, *cohen's kappa*, *recall*, *precision*, *f-measure*, number of rules and average number of attributes per rule.
- *lac.reports* provides reports to show different results of the algorithms.

- *lac.runner* provides a framework to automate and parallelize the experimental studies.
- *lac.utils* includes some extra and useful methods to work with rule sets.

2.2. Software functionalities

LAC provides the following functionalities:

- 10 well-known AC algorithms. LAC covers the whole taxonomy of AC [3], including at least one algorithm per taxonomy.
- Easy to add new algorithms thanks to its base classes.
- Three input formats are supported (arff [7], keel dat [6] and csv [9]).
- Several measures to quantify the quality of solutions, including both traditional and specific quality measures *accuracy*, *cohen's kappa*, *recall*, *precision*, *f-measure*, number of rules and average number of attributes per rule.
- Automation framework for experimental studies, which facilitates the comparison of many algorithms and datasets. The level of parallelism could be configured to be both sequential or parallel. The parallelism of each algorithm is not changed, since they were designed to be sequentially run, but each independent execution is parallelized, that is, if one config file has 5 different executions, each one will be performed on a different thread, where each thread will run the sequential algorithm.

3. Illustrative examples

In this section, a brief example of LAC is described. In this example, the well-known CBA [2] algorithm is configured and run on *weather.nominal*. In order to use LAC, a YAML file needs to be created (see Listing ??). This file is responsible for selecting the algorithms to be executed, the datasets, the parameters and the type of report. These config files need to follow a set of conventions and they are fully explained on Section 4.3 of user manual available at <https://github.com/kdis-lab/lac/tree/v0.2.0/doc/manual.pdf>.

The configuration file needs to have a root called *executions* of kind array. Each element for this array in YAML is represented by each line starting with the '-' symbol and tabulated below the parent element. The algorithm to be run is specified using *name_algorithm*. Then, the configuration of this specific algorithm should be described using *configuration*, or default configuration will be used in case of omission. For this specific case, two parameters are specified. Then, both training and test file have to be specified using *train* and *test* directives. Finally, reports could be also specified. First, the path to save all the produced files should be specified using *reports*. Second, the type of report could also be specified using *reports_type*. In this particular case, *ClassifierReport* is being used to save the final classifier and *MetricsReport* to calculate all the possible measures included in LAC.

Finally, it is also interesting to point that based on the same schema of the previous config file, more advanced experimental studies could be designed and they even could be run on parallel by means of the automation framework. Aforementioned manual contains richer examples considering more complex cases of both sequential and parallel executions.

```
executions:
```

```
- name_algorithm: 'CBA'
  configuration: { min_sup: 0.01, min_conf: 0.9 }
  train: 'weather-training.arff'
  test: 'weather-test.arff'
  reports: 'results/weather.nominal'
  reports_type: ['ClassifierReport', 'MetricsReport']
```

Listing 1: Configuration file using YAML to run CBA on weather.nominal.arff

4. Conclusions

LAC provides 10 state-of-art algorithms for AC and covering its whole taxonomy. Several measures have been included together with a framework to carry out both parallel and sequential experimental studies. Therefore, this library is a very interesting option to compare the performance of new proposals with the well-known methods available in LAC.

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund, Project TIN-2017-83445-P.

Appendix. Required metadata

Current executable software version

See [Table 1](#).

Current code version

See [Table 2](#).

References

- [1] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, *SIGMOD Rec.* 22 (2) (1993) 207–216.
- [2] B. Liu, W. Hsu, Y. Ma, Integrating classification and association rule mining, in: 4th International Conference on Knowledge Discovery and Data Mining, KDD98, New York, 1998, pp. 80–86.
- [3] N. Abdelhamid, F. Thabtah, Associative classification approaches: Review and comparison, *J. Inf. Knowl. Manag.* 13 (03) (2014) 1450027.
- [4] F. Padillo, J.M. Luna, S. Ventura, A grammar-guided genetic programming algorithm for associative classification in big data, *Cogn. Comput.* (2019) <http://dx.doi.org/10.1007/s12559-018-9617-2>.
- [5] J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2011.
- [6] I. Triguero, S. González, J.M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M.J. del Jesús, L. Sánchez, F. Herrera, KEEL 3.0: an open source software for multi-stage analysis in data mining, *Int. J. Comput. Intell. Syst.* 10 (1) (2017).
- [7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: An update, *SIGKDD Explor. Newsl.* 11 (1) (2009) 10–18.
- [8] C. Zhang, J. Bi, S. Xu, E. Ramentol, G. Fan, B. Qiao, H. Fujita, Multi-imbalance: An open-source software for multi-class imbalance learning, *Knowl.-Based Syst.* 174 (2019) 137–143, <http://dx.doi.org/10.1016/j.knosys.2019.03.001>.
- [9] RFC, Common format for comma-separated values (CSV) files, 2019, <https://tools.ietf.org/html/rfc4180>.

TITLE:

Evaluating associative classification algorithms for Big Data

AUTHORS:

F. Padillo, J.M. Luna and S. Ventura



Big Data Analytics, *Volume 4, Issue 1, pp. 2, 2019*

RANKING:

Impact factor (2018 JCR): -

Knowledge area: Big Data, Data Mining and Knowledge Discovery


DOI: 10.1186/s41044-018-0039-7

RESEARCH

Open Access



Evaluating associative classification algorithms for Big Data

Francisco Padillo¹, José María Luna^{1,3} and Sebastián Ventura^{1,2,3*} 

*Correspondence: sventura@uco.es

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

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

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

Abstract

Background: Associative Classification, a combination of two important and different fields (classification and association rule mining), aims at building accurate and interpretable classifiers by means of association rules. A major problem in this field is that existing proposals do not scale well when Big Data are considered. In this regard, the aim of this work is to propose adaptations of well-known associative classification algorithms (CBA and CPAR) by considering different Big Data platforms (Spark and Flink).

Results: An experimental study has been performed on 40 datasets (30 classical datasets and 10 Big Data datasets). Classical data have been used to find which algorithms perform better sequentially. Big Data dataset have been used to prove the scalability of Big Data proposals. Results have been analyzed by means of non-parametric tests. Results proved that CBA-Spark and CBA-Flink obtained interpretable classifiers but it was more time consuming than CPAR-Spark or CPAR-Flink. In this study, it was demonstrated that the proposals were able to run on Big Data (file sizes up to 200 GBytes). The analysis of different quality metrics revealed that no statistical difference can be found for these two approaches. Finally, three different metrics (speed-up, scale-up and size-up) have also been analyzed to demonstrate that the proposals scale really well on Big Data.

Conclusions: The experimental study has revealed that sequential algorithms cannot be used on large quantities of data and approaches such as CBA-Spark, CBA-Flink, CPAR-Spark or CPAR-Flink are required. CBA has proved to be very useful when the main goal is to obtain highly interpretable results. However, when the runtime has to be minimized CPAR should be used. No statistical difference could be found between the two proposals in terms of quality of the results except for the interpretability of the final classifiers, CBA being statistically better than CPAR.

Keywords: Big Data, Associative classification, Flink, Spark

Introduction

Classification set of rules to form an accurate classifier [1], ARM aims at describing a dataset by means of reliable associations among patterns [2]. Associative Classification (AC) [3] come into being as the combination of the two previous fields as a way of building an interpretable and accurate classifier by means of association rules [4].

When building accurate classifiers, many different techniques have been proposed in literature such as those based on rules [5], decision trees [1] or support vector machine



© The Author(s). 2019 **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The Creative Commons Public Domain Dedication waiver (<http://creativecommons.org/publicdomain/zero/1.0/>) applies to the data made available in this article, unless otherwise stated.

[6], to list a few. Many of these techniques achieve great results, but just some of them are able to build interpretable classifiers which are essential in many fields as health-care [7] or biology [8]. In general, those approaches based on rules or trees are able to obtain highly interpretable models. However, decision trees suffer from a problem of adaptability since a small change in the input data may produce large changes in the model [5]. Other approaches do not consider the whole dataset to mine rules but small samples of data and, therefore, the final classifier could not be representative of the overall trends [9]. Unlike these approaches, in AC the training phase is about searching for hidden knowledge by means of association rule mining algorithms and then a classification model (classifier) is constructed after sorting the knowledge in regards to certain criteria as well as pruning useless and redundant knowledge [4]. AC is often capable of building robust classifiers since it obtains any association rule within the dataset that can be missed by other classification systems [10]. Moreover, the rules produced in AC are easy to understand and they even could be manually updated by the end-user, unlike neural network and probabilistic approaches, which produce classification models that are hard to understand [11].

The first algorithm proposed in the AC field is known as CBA [4]. It works in two phases and the first one generates association rules by means of an exhaustive search algorithm [2]. Then, in a second phase, it ranks the discovered rules to form the final classifier. Even when this proposal obtains very interpretable and accurate classifiers, it has some problems since the runtime could take more than expected or some minority classes could be ignored. Aiming at solving these drawbacks, the same authors proposed CBA2 [12] as an improvement of the previous proposal to avoid ignoring the minority class by using multiple class minimum support. Other approaches like CMAR [12] try to solve the same problem by considering multiple rules to predict unseen examples as well as speeding up the runtime by means of complex data structures. Even all these proposals work pretty well in terms of both accuracy and efficiency, CPAR [13] was proposed as a combination of classic approaches of rule induction with features of AC obtaining both an improvement on accuracy and runtime ignoring, in part, the interpretability.

As it is described, interesting AC algorithms have been proposed in literature, obtaining good results in interpretability, predictive power and efficiency. However, with the recent need for dealing with bigger amounts of data, these proposals are becoming insufficient. Big Data is a new buzzword used to refer to the techniques used to face up the problems arising from the management and analysis of these huge quantities of data [14]. Applying existing AC approaches on such high dimensional datasets produce some limitations in terms of both computational complexity and memory requirements [15]. Hence, it is vital to propose new approaches able to scaled out [16] and to obtain results, in a reasonable quantum of time, when they are applied to Big Data.

At this point, the goal of this work is to propose two different methods based on traditional algorithms for AC (CBA and CPAR) through emerging paradigms of distributed computing (Spark and Flink). In this regard, CBA and CPAR were selected since, according to some authors [4, 12, 13, 17], these are the most interesting ones based on accuracy and interpretability. CBA is considered as the algorithm that obtains the most interpretable classifiers, whereas CPAR is able to obtain very accurate classifiers with accuracy values, in average, greater than those obtained by CBA [13]. In the experimental stage, these two proposals have been compared to existing AC approaches to demonstrate that

both are the most interesting ones. In this work, the two proposed methods have been designed on both Apache Spark and Apache Flink, and these two platforms were selected since they are the most representative within the Big Data field [18]. Then, these two approaches have been compared to current state-of-art of associative classification in Big Data. It should be noted that the two proposals, that is, CBA (based on Spark and Flink) and CPAR (based on Spark and Flink), return exactly the same results as their sequential versions (CBA [4] and CPAR [13]) and the main difference lies in the runtime and the ability to be run on Big Data. The two proposals have been analyzed on 40 different datasets (30 classical datasets and 10 Big Data datasets), proving that they scale well even for file sizes up to 200 GBytes. Classical datasets were considered to find the algorithm or algorithms that best perform among sequential approaches. Such best algorithms were parallelized using Apache Spark and Flink, and considering 10 well-known Big Data datasets. Additionally, in order to prove the scalability of Big Data approaches three well-known metrics (scale-up, speed-up and size-up) have been considered to analyze how the algorithms behave on different number of nodes and data sizes [19]. Finally, it is important to remark that all the experimental results have been validated by non-parametric statistical tests.

The rest of the paper is organized as follows. “**Methods**” section presents the most relevant definitions and related work; “**Results**” section describes the proposed algorithm; “**Discussion**” section presents the datasets used in the experiments and the results; finally, some concluding remarks are outlined in “**Conclusion**” section.

Preliminaries

In this section, the associative classification task is first introduced in a formal way. Then, different paradigms for distributed computing are described.

Associative classification

The task of associative classification (AC) was proposed as a combination of two well-known tasks in the data mining fields, namely association rule mining and classification, as a way of building a interpretable and accurate classifiers [4]. In this regard, this section formally describes these two different tasks and, finally, it formally introduce the AC problem.

Let us first introduce association rule mining (ARM) in a formal way by considering a dataset comprising a set of transactions $\mathcal{T} = \{t_1, t_2, \dots, t_m\}$ and a set of items or features $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$. Here, each transaction t_j comprises a subset of items $\{i_k, \dots, i_l\}$, $1 \leq k, l \leq n$. An association rule is formally defined [2] 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. $|\{t_j \in \mathcal{T} : X \subset t_j; t_j \subseteq \mathcal{I}\}|$. 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. $|\{t_j \in \mathcal{T} : X \subset t_j, Y \subset t_j; t_j \subseteq \mathcal{I}\}|$. 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. $confidence(X \rightarrow Y) = support(X \rightarrow Y) / support(X)$ [20].

The classification task, on the contrary, can be formally defined by considering a set of items or features $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$ and a variable of interest or class C including a number of values. Here, a dataset is formed as a set of transactions $\mathcal{T} = \{t_1, t_2, \dots, t_m\}$ and each transaction t_j comprises a subset of items $\{i_k, \dots, i_l\}$, $1 \leq k, l \leq n$ and a specific value for the class C . The task of classification could be formally defined as predicting the class value of a l -dimensional input vector \mathcal{S} such as $\mathcal{S} = \{s_1, s_2, \dots, s_l\}$ where $\forall s \in \mathcal{S} : s \subseteq \mathcal{I}$. This task of mapping a set of input variables to an output variable is done by means of functions or rules [5].

AC is an special kind of classification where rules previously discovered by ARM are used to build an accurate classifier, that is, able to predict unseen examples. Aiming at obtaining this kind of classifiers many different methodologies have been proposed [10], and most of them obtain association rules by means of exhaustive search algorithms [2]. CBA [4] and its improved version CBA2 [17] are two examples of AC algorithms that first mine association rules by means of exhaustive search algorithms. However, this is a major drawback when large datasets are required to be analysed since for a dataset comprising k single items, a total of $3^k - 2^{k+1} + 1$ rules can be computed and saved in memory. To solve this issue, additional approaches, e.g. CMAR [12], have been proposed which are mainly based on novel data structures that avoid the generation of any candidate. Another example is CPAR [13], which combines the advantages of AC with traditional rule induction algorithms. While these algorithms work well for not so big datasets, they cannot be used in Big Data due to the associated complexity [21]. Under these circumstances, new forms of building this type of classifiers from a Big Data perspective is an interesting and emerging topic [22], which has not received yet the needed attention.

Big Data architectures: Apache Spark, Apache Flink and its origins

MapReduce [16] is a recent paradigm of distributed computing in which programs are composed of two main stages, 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. To clarify this idea, let us analyze a social network in which it is required to know the number of friends in common that each pair of friends have. Assume that friends are stored as $Person \rightarrow [List\ of\ friends]$. Thus, the list of friends would be something like: $A \rightarrow BCD, B \rightarrow ACDE, C \rightarrow ABDE, D \rightarrow ABCE$, and $E \rightarrow BCD$. For every friend in the list of friends, the mapper will output a $\langle k, v \rangle$ pairs, where the key k is a friend along with the person (e.g. AB). The value v will be the list of friends (e.g. BCD). The key will be sorted so that the friends are in order, causing all pairs of friends to go to the same reducer. Before we send these $\langle k, v \rangle$ pairs to the reducers, we group them by their keys and get $(AB) \rightarrow (ACDE)(BCD)$. The reduce function will simply intersect the lists of values and output the same key with the result of the intersection. For example, the previous reduce $((AB) \rightarrow (ACDE)(BCD))$ will output $(AB) : (CD)$ which means that friends A and B have C and D as common friends.

Hadoop [23] is the *de facto* standard for MapReduce applications. Even when Hadoop implements these paradigms efficiently, its major drawback is it imposes an acyclic data flow graph, and there are applications that cannot be modeled efficiently using this kind of graph such as iterative or interactive analysis [18]. Besides, MapReduce is not aware of the total pipeline of map plus reduce steps so it cannot cache 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 Resilient Distributed Datasets (RDD) to store data in main memory and a new approach using micro-batch technology. Unfortunately, Spark does not support native iterations, which means that its engine does not directly handle iterative algorithms [24]. 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 where delay of the order of milliseconds can cause significant impact. Furthermore, Flink includes its own memory manager reducing the time required by garbage collector, whereas Spark addresses this issue later with Tungsten optimization project [25]. In this sense, Apache Flink has been proposed to face the problems of Spark and to address the problem of streaming applications differently and in a more native way (vs the micro-batch methodology 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 appear superfluous, but in the recent years the attention on this new platform has risen up considerably.

Methods

In this section the two proposals are fully described. Even though the algorithms have been run on both Spark and Flink, the explanation is in common since the philosophy is the same and the unique difference is the platform. Then, In the experimental stage, these two proposals have been compared to existing AC approaches to demonstrate that both are the most interesting ones. The experimental set-up is also fully described in this section including which comparisons have been performed.

Aim of this work and our proposals

The goal of this work is to propose two different methods based on traditional algorithms for AC, that is, CBA and CPAR, through emerging paradigms of distributed computing (Spark and Flink). In this regard, CBA and CPAR were selected since, according to some authors [4, 12, 13, 17], these are the most interesting ones based on accuracy and interpretability. CBA is considered as the algorithm that obtain the most interpretable classifiers, whereas CPAR is able to obtain very accurate classifiers with accuracy values, in average, greater than those obtained by CBA [13]. In this work, the two proposed methods have been designed on both Apache Spark and Apache Flink, and these two platforms were selected since they are the most representative within the Big Data field [18].

CBA-Spark/Flink

This proposal is based on the well-known CBA [4] algorithm¹, which makes use of two different stages. Firstly, the Apriori [2] algorithm is used to find association rules. Secondly, an accurate classifier is built using the previously mined rules.

Sequential algorithm. Aiming at easing the comprehension of the parallel approach, the original algorithm is briefly described through an example. First, association rules are extracted by means of the Apriori algorithm, only considering those rules having a support and a confidence values higher than a threshold. Let us considered the

well-known weather dataset (the class attribute determines whether someone will play tennis or not) and the rules: $outlook = rainy \rightarrow play = no$, $outlook = sunny \rightarrow play = yes$, $windy = no \rightarrow play = yes$, $windy = no \text{ AND } outlook = sunny \rightarrow play = yes$, $windy = no \text{ AND } outlook = rainy \rightarrow play = no$, $windy = no \text{ AND } outlook = rainy \rightarrow play = yes$. Second, an accurate classifier is built by considering the previously mined rules. In this regard, rules are sorted according to their support, confidence and size [4]. However, sorting is not enough since the final classifier might be formed of many and repetitive rules: $windy = no \rightarrow play = yes$ and $windy = no \text{ AND } outlook = sunny \rightarrow play = yes$ may be fired by any example that satisfies $windy = no$. In this regard, the final step in the algorithm is to remove those rules with a low level of precedence and not covering at least one example on the training dataset.

Generation of association rules. The goal of this phase is to obtain those rules whose frequency of occurrence is greater than a threshold value predefined by the user. In this sense, an iterative algorithm based on the well-known Apriori [2] is considered. An important problem of this kind of methodologies is the extremely high number of rules that may be produced at the same time. To deal with this issue, an option is not to create the whole lattice for each transaction but the l -sized sub-lattice each time, requiring a predefined number of iterations. In this regard, both the memory requirements and the computational time can be reduced. Furthermore, when the l -sized rules are generated, only the supersets from $l-1$ -sized frequent rules are used as a seed, enabling to speed up the runtime. The explanation behind this fact is that any rule can be only frequent if all its sub-rules are also frequent [2].

This phase has been developed with a classical MapReduce application including three different types of processes: 1) driver: it is the main program when running the algorithm; 2) mappers: they aim at processing an input and producing a set of $\langle k, v \rangle$ pairs; 3) reducers: they receive the previously set of pairs in order to aggregate and filter the final results. Each of these steps are fully described as follows:

- Step 1. The driver reads the database from disk and save it in the main memory of the cluster. Either using Spark or Flink, the driver splits the dataset in data subsets in order to ease both the data access and data storage.
- Step 2. Each time this phase (see Listing 1) is performed a MapReduce procedure is run. The proposed model works by running a different mapper for each specific sub-database and, then, the results are collected and filtered in a reducer phase.

These two sub-steps are described as follows.

- Step 2.1. Mappers phase. Each mapper is responsible for mining the complete set of rules of size l for its sub-database. A set of $\langle k, v \rangle$ pairs are produced where k represents a set of items and class values, whereas v denotes an array of values (support of antecedent, support for each class value, and support of the rule considering each possible class value). All the rules produced in this step do not have any sub-set of infrequent itemsets, since the $l-1$ -itemsets are used to avoid infrequent patterns to be generated (see Listing 1, line 3 in *mapper* function).
- Step 2.2 Reducers phase. The rules for each data subsets are collected, aggregated in order to obtain the support value for the whole dataset (see

Listing 1, lines 2 to 4 in *reducer* function), and filtered (see Listing 1, line 5 in *reducer* function) according to a minimum threshold for their support.

- Step 3. After the MapReduce phase is carried out, the best rules of size l are reported to the driver.

Listing 1 CBA-Spark/Flink - Generation of association rules - Step 2

function mapper(*instance*, l , *l-1-sizedRules*)

```

1: candidates  $\leftarrow$  generateRulesSizeL( $l$ , instance)
2: for all candidate in candidates do
3:   if candidate is not a subset of any l-1-sizedRules then
4:     supports  $\leftarrow$  calculateSupports(candidate, instance)
5:     emit(candidate, supports) // Emit  $\langle k, v \rangle$  pair
6:   end if
7: end for

```

end function

function reducer(*candidate*, *supports*)

```

1: finalSupports  $\leftarrow$  {supportAntecedent: 0, supportConsequent: 0, supportRule: 0}
2: for all support in supports do
3:   finalSupports  $\leftarrow$  finalSupports + support
4: end for
5: if finalSupports.supportRule  $\geq$  threshold then
6:   emit(candidate, supports) // Emit  $\langle k, v \rangle$  pair
7: end if

```

end function

It should be noted that all these steps are repeated until l is equal to the number of the items in data. These l -sized rules are kept in a pool of rules, known as R , which will be used in the next phase, that is, the generation of the final classifier.

Building the final classifier. The goal of this phase is to build the final classifier by means of the previously obtained rules. Let R be the set of generated rules and D the training dataset. The basic idea is to choose a set of high accurate rules from R to cover D . The final classifier is therefore formed as a list $[r_1, r_2, \dots, r_n, \text{default_class}]$ where $r_i \in R$ and *default_class* is only used when none of the rules is fired. This phase is tough since a huge number of combinations are possible so a heuristic is usually used to alleviate this problem.

Four different steps are considered to build the final classifier:

- Step 1. Rules are sorted according to a precedence criterion. Given two rules, r_i and r_j , r_i has a higher precedence than r_j :
 - The confidence of r_i is greater than that of r_j .
 - The confidence values are the same for both rules but the support of r_i is greater than that of r_j .
 - The confidence and support values for both rules are the same, but r_i was first generated, that is, $\text{size}(r_i) < \text{size}(r_j)$ where *size* returns the number of attributes in each rule.
- Step 2. After sorting, a MapReduce is required to select candidate rules.

- Step 2.1. Mappers phase. The input of the mappers is a chunk of the dataset and R . The goal is to iterate on each instance of the dataset to find the rule with the highest precedence that it will be fired given that example (see Listing 2). In order to do so, for each instance two rules are selected: 1) $cRule$: the rule with the highest precedence that correctly classify the instance; 2) $wRule$: the rule with also the highest precedence that wrongly classify the instance. When $cRule$ has a higher precedence than $wRule$ is trivial that $cRule$ will be fired, thus it will classify this instance. However, when the contrary happens the problem is more complex and more analysis is required (these rules will be re-studied in the next step of the algorithm). When all the instances for the chunk are processed two kind of $\langle k, v \rangle$ pairs are generated:
 - 1) $cRule$ type. For those rules which were selected at least one time as $cRule$ a $\langle k, v \rangle$ pair is emitted, where k is the rule and v is an array containing two values: $QFlag$ if it had at least one time more precedence than its $wRule$; $classCasesCovered$ an array containing the number of cases per classes which were covered by this rule.
 - 2) $wRule$ type. For those rules which were selected at least one time as $wRule$ and they had a higher precedence than its respective $cRule$ a $\langle k, v \rangle$ pair is emitted, where k is the rule and v is an array containing three values: Instance, $cRule$ and $wRule$.
- Step 2.2. Reducers phase. They receives the two types of $\langle k, [v_0, \dots, v_n] \rangle$ pairs generated in the previous step. In function of the type an action is done:
 - 1) $cRule$ type: they are sent directly to driver, without performing any action. They are saved in the driver as Q .
 - 2) $wRule$ type. The overall count is calculated aggregating the results for each mapper. In case of properties as $classCasesCovered$ the values are added and for binary values, the OR operator is applied. Then, they are sent to the driver saving it in A .
- Step 3. Next, those complex cases which could not be studied in the previous step, are now examined. In this regard, for each case where $wRule$ has a higher precedence than $cRule$ is checked if this very $wRule$ has been used as $cRule$ for other instances (that is, $QFlag$ is enabled) in that case is clear that $wRule$ will cover this instance. For the another case, it is required to find all the rules with higher precedence than $cRule$ and they also have to wrongly classify this instance. These returned rules are those that may replace $cRule$ to cover this instance because they have higher precedences, thus in this sense the counter of covered classes are updated and they are saved. As these rules are candidate of being in the final classifier.
- Step 4. All the previously generated rules are sorted in function of the precedence of the rules. Then, all the rules which do not cover at least one instance are removed. Finally, add one by one those rules to the final classifier until the point where adding a new rule does not improve the overall performance but they worsen.

Finally, the computational complexity of this algorithm is the same as the original approach [4]. As it could be appreciated, this algorithm is computationally expensive since it is based on Apriori [2]. Let us consider N as the number of input transactions, M is the threshold and k the number of unique elements. To generate rules of size i it

Listing 2 CBA-Spark/Flink - Building the final classifier - Step 2

```

function mapper(chunk, R)
1: finalResult  $\leftarrow$  generateCustomStructure(R)
2: for instance in chunk do
3:   cRule  $\leftarrow$  correctClassifyRule(R, instance)
4:   wRule  $\leftarrow$  wronglyClassifyRule(R, instance)
5:   finalResult[cRule.id].classCasesCovered[instance.class]++
6:   finalResult[cRule.id].UFlag  $\leftarrow$  true
7:   if cRule > wRule then
8:     finalResult[cRule.id].QFlag  $\leftarrow$  true
9:   else
10:    emit((cRule, [instance, cRule, wRule])) // Emit (k, v) pair
11:   end if
12: end for
13: emit(finalResult) // Emit each result as one (k, v) pair
end function

```

requires $\mathcal{O}(k^i)$, and for calculating support it requires $\mathcal{O}(N)$. Therefore, time complexity for algorithm would be $\mathcal{O}[(k + N) + (k^2 + N) + \dots] = \mathcal{O}\left(MN + \frac{1-k^M}{1-k}\right)$.

CPAR-Spark/Flink

This proposal is based on the well-known CPAR [13] algorithm, which also works in two different stages. Firstly, a greedy approach is considered in the rule generation phase, which is much more efficient than generating all candidate rules. Secondly, CPAR repeatedly searches for the current best rule and removes all the data records covered by the rule until there is no uncovered data record.

Sequential algorithm. CPAR² extracts rules by means of a greedy algorithm inspired in the well-known FOIL algorithm [13]. FOIL repeatedly searches for the current best rule and removes all the positive examples covered by such rule until all the positive examples in data are covered. To facilitate understanding let suppose the well-known weather dataset, where the rule with a highest gain is *outlook* = *sunny* \rightarrow *play* = *yes*. Then, two items *windy* = *no* and *temperature* = *hot* are found to have similar gain. The rules *outlook* = *sunny* AND *windy* = *no* \rightarrow *play* = *yes* and *outlook* = *sunny* AND *temperature* = *hot* \rightarrow *play* = *yes* are therefore generated. This process will be repeated until all the instances are covered. Finally, to predict an unseen example the best *k* rules for each class is used, with the following procedure: 1) select any rule that satisfies the example; 2) from the rules selected in step 1, take the best *k* rules for each class; and 3) compare the average expected accuracy of the best *k* rules for each class and choose the class with the highest expected accuracy as the predicted class.

Generation of association rules. This phase is responsible for obtaining class association rules. It makes use of an adaptation of FOIL which has proved to obtain good results [13]. This algorithm makes use of a unique iteration on the dataset to build a special data structure which enables to reduce the number of times that a dataset has to be read. Then, the rules could be extracted directly using this data structure. At this point, and similarly to the other proposal (CBA-Spark/Flink), this first phase has been developed with a classical MapReduce framework by including four different types of processes: 1) driver: it is the main program when running the algorithm; 2) mappers: they aim at processing an input and producing a set of $\langle k, v \rangle$ pairs; 3) reducers: they receive

the previously set of pairs in order to aggregate and filter the final results; 4) a set of association rules are produced in a greedy fashion. Each of these steps are fully described as follows:

- Step 1. The driver reads the database from disk and save it in the main memory of the cluster. Either using Spark or Flink, the driver splits the dataset in data subsets in order to ease both the data access and data storage.
- Step 2. Building a data structure to synthesize the dataset. Unlike the previous approach where an iterative approach was used, in this case a unique MapReduce phase is used. The goal is to calculate the frequency of occurrence or support value for each *attribute=value* combined with one value of the class, that is, itemsets of size 2 where one of the items is the class and the other is an item of the form *attribute=value*. Therefore, for binary classification two parallel MapReduce would be required (one for positive and another for negative). In multi-class problems the process is repeated for each class value, considering the current value as positive and the rest of values as negative. This step is split in two different sub-parts as follows.
 - Step 2.1. Mapper phase (see Listing 3 *mapper* function). Each mapper analyzes a data subset and produces a set of $\langle k, v \rangle$ pairs where k is an expression of the form *attribute=value*; and v is an array of the form $(instance.id, instance.weight, instance.class)$, where *instance.id* is a unique identifier for this instance; *instance.weight* is the weight associated to this instance (by default is 1); *instance.class* is the class for this instance. It should be noted that the number of mappers for this phase is limited to the size of the data, and its calculation is delegated to the platforms (Spark/Flink).
 - Step 2.2. Reducer phase (see Listing 3 *reducer* function). The set of $\langle k, v \rangle$ pairs is aggregated in the reducers to produce the final count for both the positive and negative classes of the instances for each *attribute=value*. Each reducer returns its result to the driver, which saves all the final results in *simplified_dataset*. As the number of produced $\langle k, v \rangle$ are very large a unique reducer could be act as a bottleneck. To avoid this situation, several reducers are considered to achieve a higher level of parallelism. In concrete, the meta-data (attributes and values included in data) is used to calculate the number of reducers as $numberOfReducers = \sum_{i=0}^n numberOfValuesForAttribute(i)$, where n is the number of attributes; *numberOfValuesForAttribute(i)* returns the number of different values which could take the attribute i .
- Step 3. Next, the dataset is split in subparts, one for each value of the class. It is required as a previous step of generating rules, since it will be used combined with the *simplified_dataset* to avoid to iterate several times in the original dataset. In case of binary classification, two new sub-sets of the original dataset would be created (see lines 3 to 4, Listing 3, *driver* function). Furthermore, the total weight for positive instances is calculated by the function *totalWeightPositiveInstances* (see line 5, Listing 3, *driver* function), having into account that each instance has a value of weight equal to 1 by default. In multi-class problems the problem of calculating the total weight would be repeated for each class value, iterating on the values and considering the current value as positive and the rest of values as negative.

Listing 3 CPAR-Spark/Flink - Generation of association rules

```

function mapper(instance)
1: for all attributeAndValue in instance do
2:   emit(attributeAndValue, (instance.id, instance.weight, instance.class)) // Emit  $\langle k, v \rangle$  pair
3: end for
end function
function reducer(attributeAndValue, values)
1: result  $\leftarrow \{positiveInstances: \emptyset, negativeInstances: \emptyset\}$ 
2: for all value in values do
3:   // value has the form (instance.id, instance.weight, instance.class)
4:   if value.class is positive then
5:     result.positiveInstances  $\leftarrow$  result.positiveInstances  $\cup$  (value.id, value.weight)
6:   else
7:     result.negativeInstances  $\leftarrow$  result.negativeInstances  $\cup$  (value.id, value.weight)
8:   end if
9: end for
10: emit(attributeAndValue, result) // Emit  $\langle k, v \rangle$  pair
end function
function driver(dataset, minGain)
1: R  $\leftarrow \emptyset$ 
2: A  $\leftarrow$  MapReduce to synthesize dataset
3: P  $\leftarrow$  Filter dataset to select only positive instance from dataset
4: N  $\leftarrow$  Filter dataset to select only negative instance from dataset
5: originalTotalWeight  $\leftarrow$  totalWeightPositiveInstances(simplified_dataset)
6: while totalWeightPositiveInstances(simplified_dataset)  $< \delta \cdot$  originalTotalWeight do
7:   N'  $\leftarrow$  N, P'  $\leftarrow$  P, simplified_dataset'  $\leftarrow$  simplified_dataset
8:   r  $\leftarrow$  emptyRule
9:   while (attributeValue = bestAttributeValue(simplified_dataset)).gain  $<$  minGain do
10:    r  $\leftarrow$  r  $\cup$  attributeValue
11:    for all t in P'  $\cup$  N' not satisfying r's body do
12:      remove t from P' or N'
13:      recalculate simplified_dataset according to the removal of t
14:    end for
15:  end while
16:  R  $\leftarrow$  R  $\cup$  r
17:  for all t in P satisfying r's body do
18:    t.weight  $\leftarrow \alpha \cdot$  t.weight
19:    change simplified_dataset according to the weight decreased
20:  end for
21: end while
end function

```

- Step 4. Find a set of rules in a greedy fashion by means of the previously obtained data. Each rule is initialized with an empty set (see line 8, Listing 3, *driver* function), and the best *attribute=value* is selected from *simplified_dataset* using the gain measure. This measure is calculated as $gain(rule) = |P| \left(\log \frac{|P*|}{|P*| + |N*|} - \log \frac{|P|}{|P| + |N|} \right)$, where *N* and *P* is the number of negative instances and positive instances respectively; *N** and *P** are the number of both the number negative and positive instances satisfying the rule body. When the best *attribute=value* is selected, it is added to the current rule *r* (see line 10, Listing 3, *driver* function), then the temporal *P'*, *N'* and *simplified_dataset'* are updated considering that this new rule covers some instances (see lines 11 to 14, Listing 3, *driver* function) and the process of adding a new *attribute=value* is repeated until the gain of the new *attribute=value* is smaller than a threshold

minGain. Once the rule is completely formed, it is added to the final set of rules (see line 16, Listing 3, *driver* function), and P and *simplified_dataset* are updated considering this new rule (see lines 17 to 20, Listing 3, *driver* function). The process is repeated until a sufficient number of positive instances have been covered by means of the threshold δ . All the rules generated are saved in R . It generates rules while the number of positive instances in the remaining dataset are larger than a threshold (δ).

Building the final classifier. The aim of this step is to build the final classifier by means of the previously obtained rules. Before anything, it is required to calculate the power of predictability of the rules. In this sense, the Laplace error estimate is used to calculate the accuracy of rules, which is defined as $LaplaceAccuracy = \frac{n_c + 1}{n_{tot} + k}$, where k is the number of classes, n_{tot} is the total number of examples satisfying the antecedent of the rule, among which n_c examples belong to the predicted class value c .

When an unseen example has to be predicted, the best k rules of each class are used for prediction following the next procedure. First, it selects all the rules whose antecedents are satisfied by this unseen example. Second, from the previously selected rules, it selects the best k rules for each class value. Finally, it compares the average expected accuracy of the best k rules of each class value and the class value with the highest expected accuracy is selected. Finally, it should be considered that multiple rules are used in prediction because two reasons: 1) the accuracy of rules cannot be precisely estimated; 2) it cannot be expected that any single rule can perfectly predict the class value of every example. Moreover, only the best k rules are used instead of all the rules since there are different number of rules for different class values.

Finally, the computational complexity of this algorithm is the same as the original approach [13]. During the process of building a rule, it removes in *simplified_dataset* each example at most once. Moreover, it takes $\mathcal{O}(k)$ time to remove an example from it. So it takes $\mathcal{O}(nk)$ time to build a rule, thus the final time complexity of calculating the ruleset is measured as $\mathcal{O}(nk|R|)$.

Experimental set-up

This section describes both the experimental set-up (algorithms and datasets) and the achieved results. The goal of this experimental analysis is four-fold:

- 1 To compare the quality of the predictions with other well-known algorithms taken from the AC field.
- 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 in terms of both quality (accuracy and kappa) and interpretability.
- 4 To analyze the scalability in Big Data environments when different parallel implementations are considered.

All the results obtained in the experimental analysis are available at <http://www.uco.es/kdis/cba-cpar/>.

Design of the experimental study and criteria for selecting the best algorithms

In this analysis, 40 real-world datasets widely used by researchers in AC and Big Data datasets have been considered. Table 1 shows the number of both attributes and instances, they have been categorized into two different groups: classical datasets and Big Data

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

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

datasets. All of them are publicly available at the KEEL [26] repository. For these datasets, the number of attributes ranges from 2 to 60, the number of class values varies between 2 to 23, and the number of instances ranges from 87 to 11,000,000. A 10-fold stratified cross-validation has been used, and each algorithm has been executed 5 times. Thus, the results shown for each dataset are the average results obtained from 50 different runs.

Additionally, 12 different algorithms have been considered to be analyzed, which are based on different methodologies such as exhaustive search from ARM, bio-inspired methodologies, Big Data approaches as well as classic methodologies. All these algorithms, which are mainly used by researchers in the AC field, have been selected according to their efficiency and significance within the predictive tasks. It is important to note that the configurations for these algorithms are those provided by the authors in their original works. Each of these algorithms have been categorized as follows:

Classical algorithms

- CBA [4]. It is the first algorithm that was proposed in the AC field. It is based on a well-known algorithm from ARM known as Apriori [2].
- CBA2 [17]. It is an improvement of CBA that considers multiple class minimum support in rule generation.
- CMAR [12]. It uses a recognized algorithm (FP-Growth [27]) from ARM to obtain rules without candidate generation.
- CPAR [13]. It adopts a greedy algorithm to generate interval association rules directly.
- C4.5 [1]. One of the most well-known algorithms to generate a decision tree in the same way as ID3 algorithm [5].
- RIPPER [28]. 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 [29]. It is a coevolutionary algorithm for rules induction. It coevolves rules and rule sets concurrently in two cooperative populations.
- OneR [30]. 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 [31]. Distributed association rule-based classification scheme shaped according to the MapReduce programming model.
- MRAC+ [31]. Improved version of MRAC where some time-consuming operations were removed.
- DAC [32]. Ensemble learning which distributes the training of an associative classifier among parallel workers.
- DFAC-FFP [33]. An efficient distributed fuzzy associative classification approach based on the MapReduce paradigm.

In order to analyze each of the aforementioned algorithms, an experimental study has been performed. In this study, the main and most important criteria to choose an algorithm is described as:

- Predictive power. In this regard, accuracy rate [5] and Cohen's kappa rate [34] have been 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 in classification. On the contrary, an due to accuracy may achieve unfair results with imbalanced data, Cohen's kappa rate [34] has been considered, evaluating

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.

- Interpretability has been selected as one of the main reasons of using AC, that is, to obtain interpretable classifiers that facilitate the understanding from an expert in the domain. Rules with a less quantity of attributes and classifiers formed by a small number of rules are denoted as more interpretable from a point of view of a human expert. In this sense, it is straightforward to state that the ideal classifier would be composed of a small number of rules with very few variables. Let C be a classifier including a set of rules, i.e. $C = \{R_0, \dots, R_n\}$, the complexity or interpretability of C is calculated as $complexity(C) = n \sum_{i=0}^n attributes(R_i)$, 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.
- Efficiency is of great interest since, nowadays, more and more data is daily generated and it is vital to propose approaches that are able to be scaled out and to obtain results, in a reasonable quantum of time.

Then, the analysis is exclusively focused on Big Data algorithms. Three well-known metrics have been used to show how our proposals behave [19]. Next, each metric is briefly described.

- Speed-up [19]: given a fixed job run on a small system, and then run on a larger system, the speed-up is measured as $speed - up(p) = \frac{T_1}{T_p}$, where p is the number of nodes, T_1 is the execution time on one node and T_p is the execution time on p nodes. It holds the problem size constant, and grows the system.
- Scale-up [19]: is defined as the ability of a N -times larger system to perform an N -times larger job in the same elapsed time as the original time. Thus, it measures the ability to growth both the system and the problem. It is defined as $scale - up(D, p) = \frac{T_{D1}}{T_{Dp}}$, where D is the dataset, T_{D1} is the execution time for D on one node, T_{Dp} is the execution time for $p \times D$ on p nodes.
- Size-up [19]: it measures how much longer it takes on a given system, when the dataset size is p larger than the original dataset. It is defined as $size - up(D, p) = \frac{T_p}{T_1}$, where T_p is the execution time for processing $p \times D$ and T_1 is the execution time for processing data.

Finally, 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 is 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.

Results

All the algorithms studied in this work have been analyzed in function of the different criteria and considering non-parametric tests. The result for each analysis is as follows:

- Analysis of the predictive power: all the aforementioned algorithms have been analyzed according to both the accuracy and kappa quality measures. CPAR obtained the best result.
- Interpretability: from the best algorithms obtained in the previous study, an analysis of interpretability have been performed considering both number of rules and attributes. CBA obtained the best results.
- Efficiency: after the two previous analysis, the best algorithms have been compared with regard to quantity of time. CBA was the best algorithm.

Additionally, two additional analyses were performed. First, traditional approaches on traditional datasets were considered to prove the importance of parallelizing such algorithms. Second, Big Data algorithms and datasets were considered. The proposals for Big Data (CBA-Spark/Flink and CPAR-Spark/Flink) are deeply analyzed and compared to the state-of-the-art in Big Data proving that they scale very well in terms of metrics such as speed-up, scale-up and size-up.

Discussion

This section discuss the implications of the findings in context of existing research.

Comparative study on predictive power, interpretability and efficiency

The aim of this section is to analyze all the aforementioned algorithms according to three main criteria: predictive power, interpretability and efficiency. It is important to remark that the best algorithms for each criterion are the only ones used in the experimental analysis of the following criterion. 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-art even when a small quantity of data is considered); Big Data algorithms and datasets (aiming at comparing to current state-of-art in Big Data). Finally, it is important to remark that classical approaches cannot be run on Big Data datasets.

Analysis of the predictive power

The goal of this study is to analyze the quality of the solutions, in terms of Accuracy and Kappa measures, obtained by different algorithms.

Classical state-of-art

Analyzing the accuracy (see Table 2), it is obtained that CMAR and OneR obtained the worst results. This behavior is caused by the fact that OneR only uses a unique rule to predict on the whole datasets. It should be noted that in those datasets comprising a huge number of instances, a higher number of rules are required to cover all the instances. Besides, CMAR did not obtain good results since it optimizes the confidence measure in isolation, that is, it generates very specific classifiers that are not able to correctly predict unseen examples. As for CBA and its improved version CBA2, they obtained good results in terms of accuracy, being even close to the results obtained by C4.5. It should be highlighted that, among all the algorithms under study, CPAR obtained the best results for the Accuracy metric. Finally, focusing on the Kappa metric (see Table 2), very similar results were obtained, although in this case the difference between CPAR and C4.5 is not so high.

Table 2 Classical algorithms

Algorithm	Ranking
Ranking for the accuracy measure	
CMAR	6.200
OneR	5.566
CORE	5.516
CBA	4.466
Ripper	4.500
CBA2	3.383
C4.5	3.816
CPAR	2.550
Ranking for the kappa measure	
OneR	6.083
CORE	5.750
CMAR	5.683
CBA	4.633
Ripper	3.816
CBA2	3.516
C4.5	3.300
CPAR	3.216

Average ranking for each algorithm (sorted in descending order) according to the Friedman test. Bold typeface denotes the algorithm whose ranking was the best

In order to analyze whether there are any statistical difference in the previous results, several non-parametric tests have been performed. First, a Friedman test has been run on the Accuracy measure, obtaining a $X_F^2 = 52.894$ with a critical value of 18.475 and a p -value = 3.889^{-9} . Therefore, it is possible to assert that there exist some kind of statistical difference among the algorithms for this measure with $\alpha = 0.01$. In the same way, a $X_F^2 = 50.042$ with a critical value of 18.475 and a p -value = 1.418^{-8} has been obtained for the Kappa metric, meaning that there exist some statistical differences among the algorithms for $\alpha = 0.01$. Next, a post-hoc test has been performed to state among which algorithms there exist any difference. In this regard, Table 3 shows the p -values for the Holland test with $\alpha = 0.01$. CPAR has been selected as control since it achieved the best ranking in the previous analysis. Focusing on the Accuracy measure, results of this post-hoc test (see Table 4) denoted some statistical differences with regard to CMAR, CORE and OneR. The rest of algorithm equally behaves in terms of Accuracy measure. Finally, focusing on the Kappa measure, results of this post-hoc test (see Table 4) revealed some statistical differences with regard to CMAR, CORE and OneR. To sum up, among the ten selected algorithms,

Table 3 Classical algorithms

	CBA2	CPAR	C4.5	Ripper
CBA	0.983	0.000	0.000	0.000

p -value for the Holland test with $\alpha = 0.01$ for the complexity measure. Bold typeface denotes the algorithm whose ranking was the best

Table 4 Classical algorithms

	CBA	CBA2	CMAR	C4.5	Ripper	CORE	OneR
Results for the accuracy measure							
CPAR	0.048	0.810	0.000	0.500	0.042	0.000	0.000
Results for the kappa measure							
CPAR	0.317	0.996	0.002	0.996	0.977	0.002	0.000

p-values for the Holland test with $\alpha = 0.01$. Bold typeface denotes the algorithm whose ranking was the best

five of them equally behave according to the predictive power (measured with Accuracy and Kappa metrics) and, therefore, additional criteria need to be used to select the best approach.

Big Data state-of-art

Table 5 shows the ranking for Big Data algorithms for accuracy measure. DAC obtained close results to those obtained by MRAC. MRAC+ was the next best algorithm, improving its original non-improved version (MRAC). DFAC-FFP obtained also good results but not as good as those obtained by CBA Spark/Flink. CPAR Spark/Flink has been the algorithm which achieved the best performance on accuracy measure. Similarly, Table 5 shows the ranking for kappa measure. The results are very similar to those previously obtained. The unique difference was between MRAC and MRAC+ where in this case this last algorithm obtained worse performance than MRAC. Again, CPAR Spark/Flink has been the best algorithm.

In order to study whether there are any statistical significant difference among the results, several non-parametric tests have been considered. First, a Friedman test has been performed on accuracy measure obtaining a $X_F^2 = 36.5$ with a critical value of 15.086 and a *p*-value = 7.543^{-7} . Hence, it is possible to state some kind of statistical significant differences among the algorithms. Likewise, a Friedman test has also been performed on kappa measure obtaining a $X_F^2 = 45.371$ with a critical value of 15.086 and a *p*-value = 1.219^{-8} .

Table 5 Big Data algorithms

Algorithm	Ranking
Ranking for accuracy measure	
DAC	5.100
MRAC	5.000
MRAC+	4.150
DFAC-FFP	3.550
CBA Spark/Flink	2.000
CPAR Spark/Flink	1.200
Ranking for kappa measure	
DAC	6.000
MRAC+	4.700
MRAC	4.150
DFAC-FFP	2.950
CBA Spark/Flink	1.950
CPAR Spark/Flink	1.250

Average ranking for each algorithm (sorted in descending order) according to the Friedman test when 10 Big Data datasets are considered. Bold typeface denotes the algorithm that achieves the best ranking

Next, a post-hoc test has also been performed to find among which algorithms there are any kind of differences. In this sense, Table 6 shows p -values for the Holland test with $\alpha = 0.01$. Thus, it could be stated that there are some statistical differences with regard to MRAC, MRAC+ and DAC. Then, it has not been possible to find differences in terms of accuracy for CPAR Spark/Flink, CBA Spark/Flink and DFAC-FFP. Finally, focusing on kappa measure (see Table 6), shows statistical differences with regard to MRAC, MRAC+ and DAC.

Considering these results and comparing with those obtained in classical algorithms, it is found the same behavior. CPAR obtained the best performance so much in small data as in big data. CBA did not obtain very different results than those obtained by CPAR, however in average they have been a little worse.

Analysis of the interpretability

Continuing with the analysis of the interpretability, the number of attributes per rule as well as the number of rules have been analyzed.

Classical state-of-art

The average ranking for the complexity measure is shown in Table 7. CBA obtained the best results closely followed by its improved version CBA2. CPAR, C4.5 and Ripper obtained the worst results and, among them, there are not many big differences. In order to analyze these results in a statistical way, a Friedman test has been performed, obtaining a $X_F^2 = 66.647$ with a critical value of 13.277 and a p -value = 2.698^{-14} , meaning that it exists some kind of statistical differences among the complexity of the solutions of these algorithms. Next, a post-hoc test has been performed to state among which algorithms there exist any type of differences. In this sense, Table 3 shows the p -values for the Holland test with $\alpha = 0.01$. Taking into account only the interpretability, the algorithm with the best ranking has been selected as control, that is, CBA. Results of this post-hoc test proved that CPAR, C4.5 and Ripper obtained statistical significant differences with regard to CBA. However, when comparing CBA and CBA2, an additional criterion is required since no statistical difference was obtained among them for the interpretability measure.

Big Data state-of-art

Table 8 shows the average ranking for the complexity measure. CBA Spark/Flink has outperformed the rest of algorithms achieving the most interpretable classifiers. DFAC-FFP obtained better results than CPAR Spark/Flink. This results are very similar to those obtained in classical algorithms. CPAR almost always obtains a very large number of rules hampering interpretability of classifiers. Unlikely, CBA obtained almost always the best

Table 6 Big Data algorithms

	CBA Spark/Flink	MRAC	MRAC+	DFAC-FFP	DAC
Results for the accuracy measure					
CPAR Spark/Flink	0.773	0.000	0.005	0.049	0.000
Results for the kappa measure					
CPAR Spark/Flink	0.643	0.000	0.006	0.229	0.000

p -values for the Holland test with $\alpha = 0.01$. Bold typeface denotes the algorithm whose ranking was the best

Table 7 Classical algorithms

Algorithm	Ranking
Ripper	4.000
C4.5	3.850
CPAR	3.783
CBA2	1.700
CBA	1.667

Average ranking for complexity measure of each algorithm (sorted in descending order) according to the Friedman test. Bold typeface denotes the algorithm whose ranking has been the best

possible results since it obtained small rules (few attributes) and classifiers with a small number of rules.

In order to find some statistical significant differences among the results several non-parametric test have been performed. Firstly, a Friedman test has been performed obtaining a $X_F^2 = 11.450$ with a critical value of 9.210 and a p -value = 0.003 proving that there are some kind of differences. Then, a post-hoc test has been performed to state among which algorithms there are differences. In this way, Table 9 shows the p -values for the Holland test with $\alpha = 0.01$. It proves that there are differences with regard to CPAR Spark/Flink.

Analysis of the efficiency

Once the predictive power and the interpretability of the solutions have been analyzed, the final criteria to be considered is efficiency. In this sense, only the best algorithms until the moment have been taken into account.

Classical state-of-art

The runtime for the CBA and CBA2 algorithms were measured, and a Wilcoxon signed rank test was performed obtaining a Z -value = -2.519 with p -value = 0.005. Results denoted that some statistical differences were found when comparing CBA and CBA2 with $\alpha = 0.01$, CBA2 obtaining the worst results. The explanation behind this fact is simple, CBA2 needs to build a classifier by means of a close adaptation of CBA. Then, it builds a decision tree method as in C4.5 and, at the same time, a Naive-Bayes method is also performed. It means that, in the best case, CBA2 requires the same time as CBA. However, in practice, this best case was rarely found since the building of the tree also consumes a quantity of time that increases the overall runtime.

Table 8 Big Data algorithms

Algorithm	Ranking
CPAR Spark/Flink	2.600
DFAC-FFP	2.250
CBA Spark/Flink	1.150

Average ranking for complexity measure of each algorithm (sorted in descending order) according to the Friedman test. Bold typeface denotes the algorithm whose ranking has been the best

Table 9 Big Data algorithms

	DFAC-FFP	CPAR Spark/Flink
CBA Spark/Flink	0.028	0.004

p-value for the Holland test with $\alpha = 0.01$ for the complexity measure. Bold typeface denotes the algorithm whose ranking was the best

Big Data state-of-art

Finally, the runtime of the two best algorithms have been studied. Time for CBA Spark/Flink has been the average obtained in the two platforms, being both very similar. In the next section, a different study is performed to prove whether exists differences between these two platforms. A Wilcoxon signed rank test has been performed obtaining a Z -value = -2.310 obtaining that classical CBA implemented on current distributed computing obtained statistically significant differences with regard to DFAC-FFP.

Conclusions achieved with this analysis on three different criteria

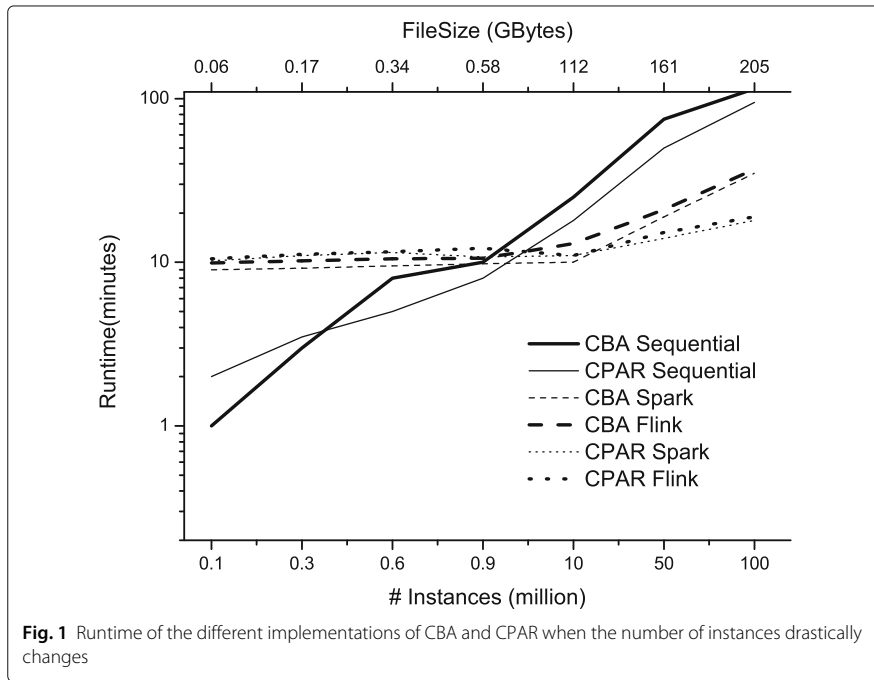
After performing a complete analysis based on three different criteria, next the following conclusions could be stated. Two algorithms for AC have been selected to be adapted to Big Data platforms. On the one hand, CBA has been considered since it obtained a good trade-off among predictive power, interpretability and efficiency. On the other hand, CPAR has proved to obtain very accurate classifiers in a reduced quantum of time but the interpretability of the results is not so good compared to other such as CBA and CBA2. In this regard, if the time required to produce results can be improved, it is obvious that CPAR should be used due to its good results in predictive power. On the contrary, when a high interpretability is required, CPAR is not recommended but CBA.

Very similar results have been obtained in Big Data. CPAR Spark/Flink obtained the best results in terms of performance. When interpretability is considered, CBA Spark/Flink is the winner outperforming both DFAC-FFP and CBA Spark/Flink. Finally, CBA Spark/Flink has also obtained the most efficient results.

Scalability of the different proposals in Big Data

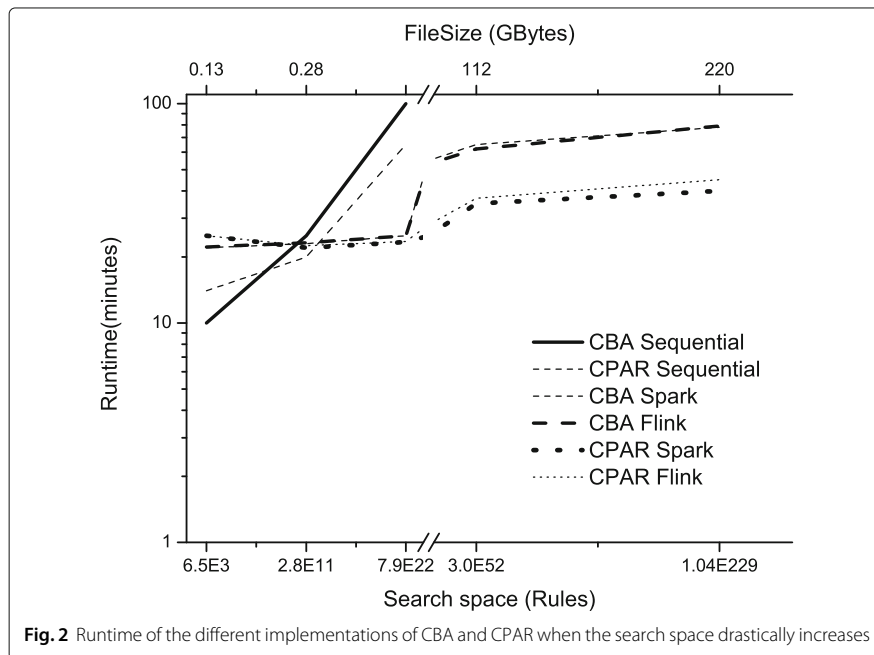
The goal of this analysis is to study the scalability of the different proposals in Big Data. In this regard both the original and the adaptations have been run on a series of synthetic datasets. This kind of datasets has been selected due to the fact that only the runtime is analyzed because both algorithms have proved to obtain accurate and interpretable classifiers on real-world datasets. Furthermore, synthetic datasets enables to change both the number of instances and the search space easily to study the behavior when different data sizes are considered. On this matter, the datasets have been generated following a Gaussian distribution where the number of instances ranges from $1 \cdot 10^4$ to $1 \cdot 10^8$, with a search space ranging from 6500 to 1.04^{229} , and file sizes up to 200 GBytes have been included.

Figure 1 shows the behavior of the selected algorithms when the number of instances changes. As it is illustrated, when the number of instances is low CBA-Sequential is more efficient than CPAR-Sequential, that is because CBA is more direct in building the classifier than the greedy algorithm used in CPAR. Neither the implementations based on

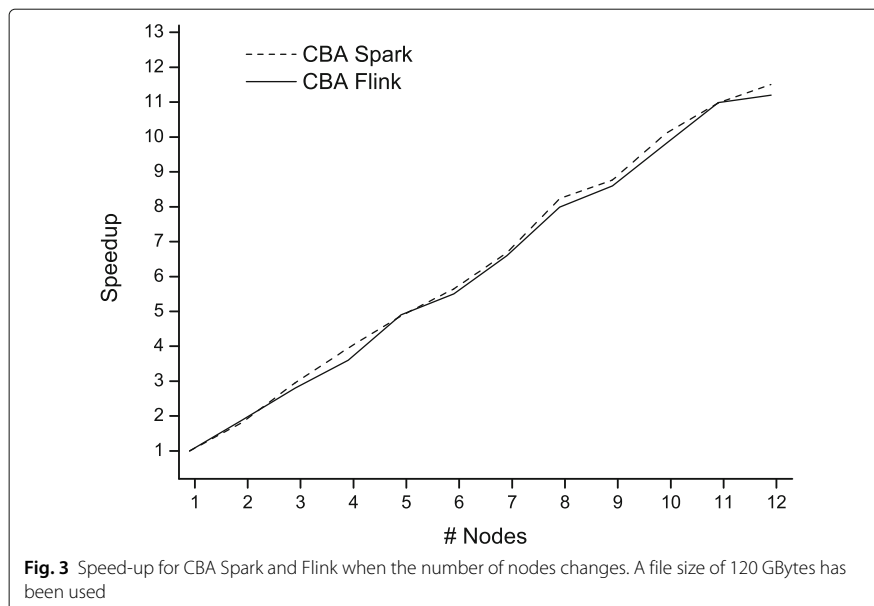


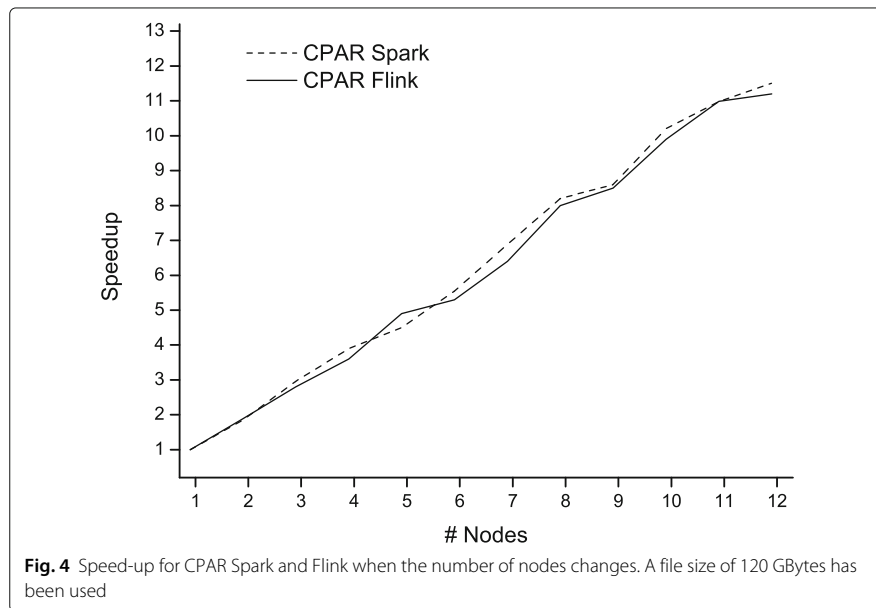
Spark or Flink obtained good runtime with few instances. When the number of instances continues growing, the approaches based on Spark And Flink obtained much better performance than sequential methodologies. Finally, it should be remarked that between the implementations of Spark and Flink of each algorithm there are not many differences, Spark obtained a small better performance than Flink although it is not game-changing. Between CBA-Spark/Flink and CPAR-Spark/Flink, this last method obtained a better performance than CBA-Spark/Flink thanks to its greedy approach that in this case is more efficient than considering all the possible cases as an exhaustive search like the used in CBA-Spark/Flink does.

Continuing with this study, it has also been considered of high interest to analyze how the behavior of the proposals varies when the search space changes. The search space was calculated as the number of feasible rules that can be mined from data ($3^k - 2^{k+1} - 1$ where k is the number of items). To perform this analysis several synthetic datasets were used where the number of attributes were changed to analyze the performance on different search spaces. In this regard, Fig. 2 shows the performance, proving that the behavior is more different than in the previous analysis. It is due to the fact that CPAR-Sequential is not as affected as CBA-Sequential thanks to its greedy methodology. With a small search space, the proposals based on Spark and Flink do not obtain a good performance however when the search space increases, they begin to obtain a very good performance. Finally when the number of search spaces highly grows, CPAR-Spark/Flink obtained a good performance followed by CBA-Spark/Flink. With this large search space sequential approaches are not able to be run, requiring several hours to end.

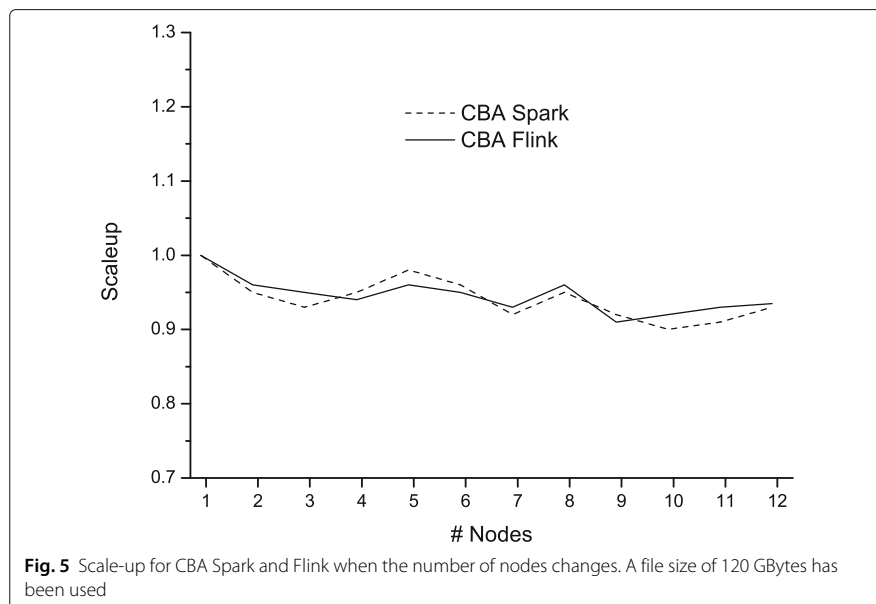


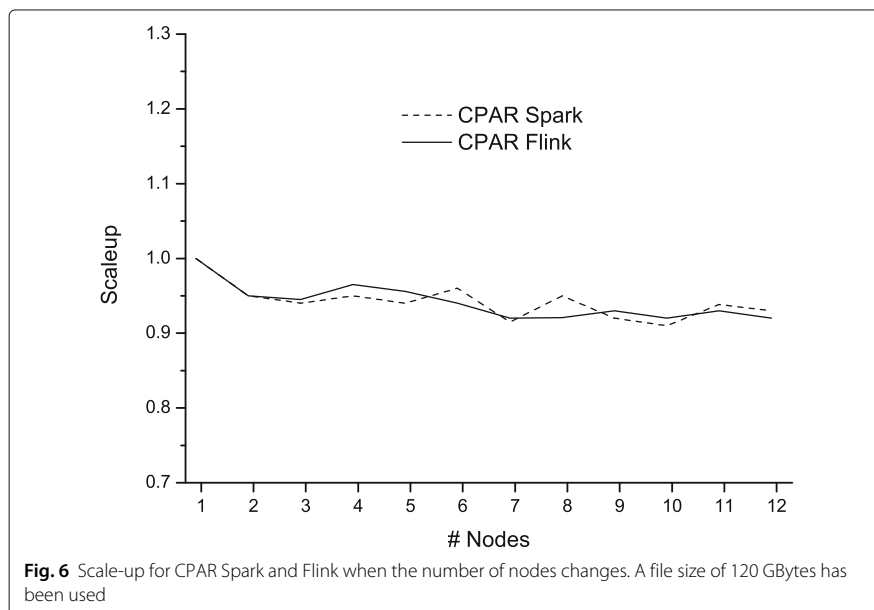
The analysis now continues considering only Big Data approaches. In this regard, three well-known metrics have been studied [19]. Firstly, speed-up is analyzed aiming at measuring how algorithms behave when parallelism increases (without altering data size). Figures 3 and 4 show the results when the number of nodes increases from 1 to 12 with different data sizes, as it could be seen speed-up holds linear proving that performance





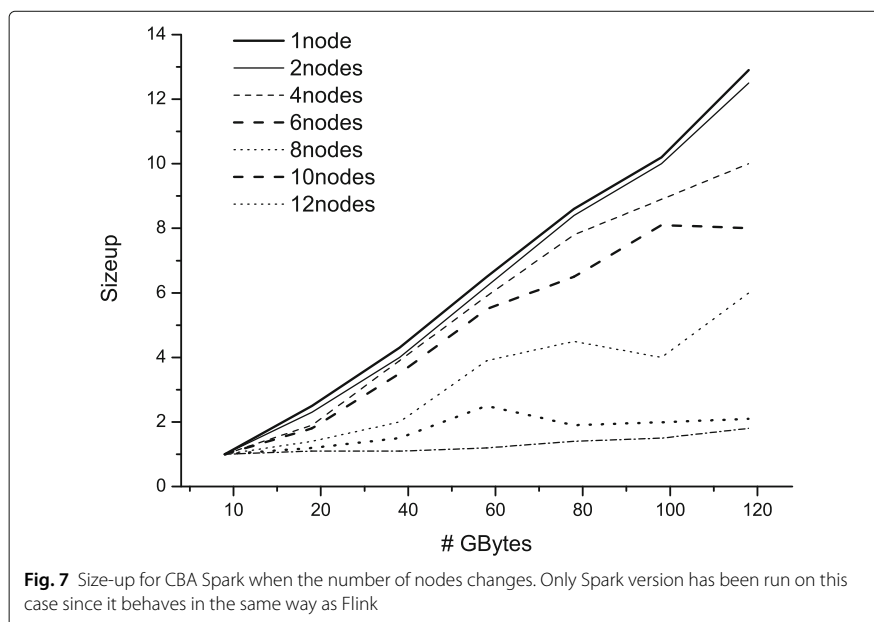
increases linearly with the number of nodes. Then, scale-up is analyzed to see how well the proposed algorithms handle larger datasets when more nodes are available. In this case the size of the dataset is increased in direct proportion with the number of nodes in the system. Figures 5 and 6 shows that it is practically evaluated to 1 in almost the cases being linear and proving good scalability [19]. Lastly, size-up is analyzed where the number of nodes grows from 1 to 12 and the sizes of datasets from 10 GBytes to 120 GBytes (see Figs. 7 and 8). As the result shows, the size-up performance of our proposals is also very good.

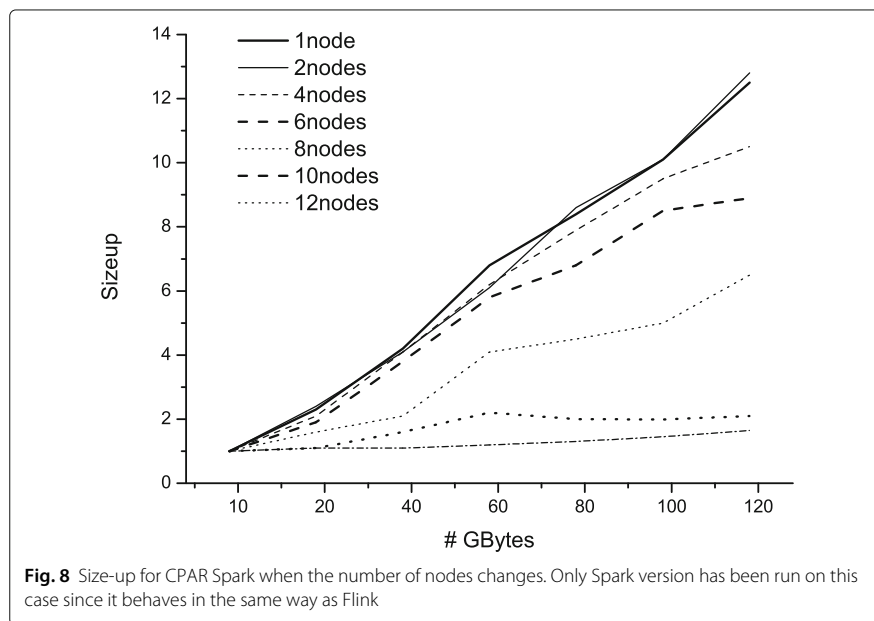




Conclusion

In this work an experimental study including 40 datasets and 12 different algorithms have been performed. This analysis has been arranged having into account three different criteria. First, predictive power of the algorithms have been measured by means of both kappa and accuracy. Second, the interpretability of the classifiers have been studied. Finally, the efficiency has been also measured. After performing this experimental study, two different algorithms of the state-of-art have been selected. On the one hand, CBA has been selected since it obtained a very good predictive power, and the best results in terms





of interpretability. On the other hand, CPAR was selected after obtaining the very best results for predictive power in a reduced quantum of time.

These two algorithms have been adapted to be run on Big Data platforms considering both Apache Spark and Apache Flink. These adaptations obtained the same results as the sequential approaches but in a reduced quantum of time. Finally, an analysis of the scalability has also been performed considering files sizes up to 200 GBytes proving that our methods are able to work in an efficient way in Big Data, where sequential approaches would never be able to work in a efficient way.

Endnotes

¹ The original pseudocode can be found at <http://www.uco.es/kdis/cba-cpar/>

² The original pseudocode can be found on at <http://www.uco.es/kdis/cba-cpar/>.

Abbreviations

AC: Associative classification ARM: Association Rule Mining

Acknowledgments

Not rule mining and association rule mining (ARM) are two important and different fields in data mining. Whereas the first one has a predictive purpose by discovering a small applicable.

Funding

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

Availability of data and materials

All the results which have been used to obtain the tables and figures shown in this work are available at <http://www.uco.es/kdis/cba-cpar/>.

Authors' contributions

All the authors contributed in equal parts. All authors read and approved the final manuscript.

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 15 May 2018 Accepted: 11 December 2018

Published online: 14 January 2019

References

1. Quinlan R. C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufmann Publishers; 1993.
2. Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. *SIGMOD Rec.* 1993;22(2):207–16.
3. Ventura S, Luna JM. Supervised Descriptive Pattern Mining; 2018.
4. 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. 80–6.
5. Han J. Data Mining: Concepts and Techniques. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2011.
6. Cortes C, Vapnik V. Support vector networks. *Mach Learn.* 1995;20:273–97.
7. Valdes G, Luna J, Eaton E, B Simone C, H Ungar L, D Solberg T. Mediboost: A patient stratification tool for interpretable decision making in the era of precision medicine. In: scientific reports. 2016;6:37854.
8. Kim SG, Theera-Ampornpunt N, Fang C-H, Harwani M, Grama A, Chaterji S. Opening up the blackbox: an interpretable deep neural network-based classifier for cell-type specific enhancer predictions. *BMC Syst Biol.* 2016;10(2):54. <https://doi.org/10.1186/s12918-016-0302-3>.
9. Clark P, Niblett T. The cn2 induction algorithm. *Mach Learn J.* 1989;3(4):261–83.
10. Thabtah FA. A review of associative classification mining. *Knowl Eng Rev.* 2007;22(1):37–65.
11. Fong RC, Vedaldi A. Interpretable explanations of black boxes by meaningful perturbation. In: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22–29, 2017; 2017. p. 3449–57. <https://doi.org/10.1109/ICCV.2017.371>.
12. 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.
13. Yin X, Han J. Cpar: Classification based on predictive association rules. In: 3rd SIAM International Conference on Data Mining (SDM03); 2003. p. 331–5.
14. Gumbus A, Grodzinsky F. Era of big data: Danger of descrimination. *SIGCAS Comput Soc.* 2016;45(3):118–25. <https://doi.org/10.1145/2874239.2874256>.
15. Wu X, Zhu X, Wu GQ, Ding W. Data mining with big data. *IEEE Trans Knowl Data Eng.* 2014;26(1):97–107. <https://doi.org/10.1109/TKDE.2013.109>.
16. Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. *Commun ACM - 50th Anniversary Issue: 1958 - 2008.* 2008;51(1):107–13.
17. Liu B, Ma Y, Wong C-K. In: Grossman RL, Kamath C, Kegelmeyer P, Kumar V, Namburu RR, editors. Classification Using Association Rules: Weaknesses and Enhancements. Boston, MA: Springer; 2001, pp. 591–605.
18. 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: USENIX Association; 2010.
19. DeWitt D, Gray J. Parallel database systems: The future of high performance database systems. *Commun ACM.* 1992;35(6):85–98. <https://doi.org/10.1145/129888.129894>.
20. Ventura S, Luna JM. Pattern Mining with Evolutionary Algorithms; 2016.
21. Oneto L, Bisio F, Cambria E, Anguita D. Slt-based elm for big social data analysis. *Cogn Comput.* 2017;9(2):259–74.
22. Siddique N, Adeli H. Nature inspired computing: An overview and some future directions. *Cogn Comput.* 2015;7(6):706–14.
23. Lam C. Hadoop in Action, 1st edn. Greenwich, CT, USA: Manning Publications Co.; 2010.
24. Padillo F, Luna JM, Ventura S. Exhaustive search algorithms to mine subgroups on big data using apache spark. *Prog Artif Intell.* 2017;6(2):145–58.
25. Xin R, Rose J. Project Tungsten: Bringing Apache Spark Closer to Bare Metal; 2015. <https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html>.
26. Triguero I, González S, Moyano JM, García S, Alcalá-Fdez J, Luengo J, Fernández A, del Jesús MJ, Sánchez L, Herrera F. Keel 3.0: an open source software for multi-stage analysis in data mining. *Int J Comput Intell Syst.* 2017;10(1):1238–49.
27. Han J, Pei J, Yin Y, Mao R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min Knowl Discov.* 2004;8(1):53–87.
28. Cohen WW. Fast effective rule induction. In: Machine Learning: Proceedings of the Twelfth International Conference; 1995. p. 1–10.
29. Tan KC, Yu Q, Ang JH. A coevolutionary algorithm for rules discovery in data mining. *Int J Syst Sci.* 2006;37(12):835–64.
30. Holte RC. Very simple classification rules perform well on most commonly used datasets. *Mach Learn.* 1993;11:63–91.
31. Bechini A, Marcelloni F, Segatori A. A mapreduce solution for associative classification of big data. *Inf Sci.* 2016;332:33–55.
32. Venturini L, Baralis E, Garza P. Scaling associative classification for very large datasets. *J Big Data.* 2017;4(1):44. <https://doi.org/10.1186/s40537-017-0107-2>.
33. Segatori A, Bechini A, Ducange P, Marcelloni F. A distributed fuzzy associative classifier for big data. *IEEE Trans Cybern.* 2018;48(9):2656–69.
34. Ben-David A. Comparison of classification accuracy using cohen's weighted kappa. *Expert Syst Appl.* 2008;34(2):825–32.

TITLE:

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

AUTHORS:

F. Padillo, J.M. Luna and S. Ventura



Cognitive Computation, *Volume 11, Issue 3, pp. 331-446, 2019*

RANKING:

Impact factor (2018 JCR): 4.287

Knowledge area: Computer Science; Neurosciences & Neurology

DOI: 10.1007/s12559-018-9617-2



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

F. Padillo¹ · J. M. Luna^{1,3} · S. Ventura^{1,2,3}

Received: 18 April 2018 / Accepted: 12 November 2018 / Published online: 16 January 2019
© Springer Science+Business Media, LLC, part of Springer Nature 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 $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$ be the set of items, features, or attributes in a dataset comprising a set of transactions $\mathcal{T} = \{t_1, t_2, \dots, t_m\}$. Here, each transaction t_j comprises a subset of items $\{i_k, \dots, i_l\}$, $1 \leq k, l \leq n$. 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., $|\{t_j \in \mathcal{T} : X \subseteq t_j; t_j \subseteq \mathcal{I}\}|$. 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., $|\{t_j \in \mathcal{T} : X \subset t_j, Y \subset t_j; t_j \subseteq \mathcal{I}\}|$. 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 Σ_T represents the alphabet of terminal symbols, Σ_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>  $\Leftarrow$  <Antecedent>, <Consequent>;
          <Antecedent>  $\Leftarrow$  <Condition> (AND <Condition>)*;
          <Consequent>  $\Leftarrow$  class = value;
          <Condition>  $\Leftarrow$  <Numerical> | <Nominal>;
          <Numerical>  $\Leftarrow$  name IN Min_value, Max_value;
          <Nominal>  $\Leftarrow$  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

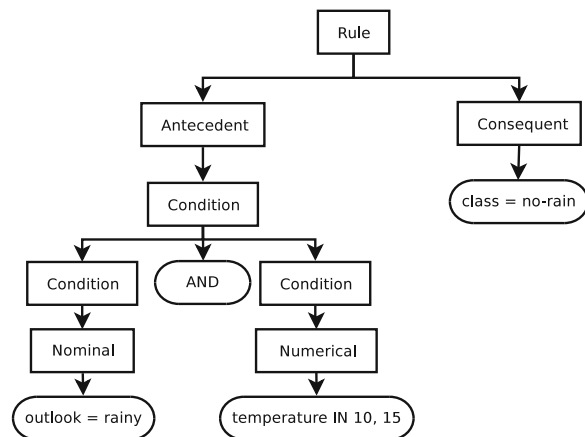


Fig. 2 Sample rule encoded through the grammar defined in Fig. 1

following rule: **If** *outlook = rainy* **AND** *temperature IN 10, 15* **THEN** *no-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 $F(R) = \text{confidence}(R) \times \text{support}(R)/\text{support}(Y)$, 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:    $\text{auxiliary\_population}_c \leftarrow \emptyset$ 
4:   for  $i = 0$  to NumberOfGenerations do
5:      $\text{evaluate\_rules}(P_{i,c})$ 
6:      $\text{auxiliary\_population}_c \leftarrow$  Maintain elitism using
      $P_{i,c} \cup \text{auxiliary\_population}_c$ 
7:     Stop if  $\text{mean}(\text{auxiliary\_population}_c)$  has not
     changed in a number of generations specified by the
     user
8:      $\text{selected\_individuals} \leftarrow$  Apply tournament selector
     to  $P_{i,c}$ 
9:     for all  $\text{pair}$  in  $\text{selected\_individuals}$  do
10:        $\text{offspring} \leftarrow \text{pair}$ 
11:       if  $\text{Rand\_number}(0, 1) > \text{Prob}_{\text{cro}}$  then
12:          $\text{offspring} \leftarrow$  Apply crossover operator
         ( $\text{offspring}$ )
13:       end if
14:       for all  $\text{individual}$  of the  $\text{offspring}$  do
15:         if  $\text{Rand\_number}(0, 1) > \text{Prob}_{\text{mut}}$  then
16:            $\text{offspring} \leftarrow$  Apply mutation operator
           ( $\text{individual}$ )
17:         end if
18:       end for
19:     end for
20:      $P_{i+1,c} \leftarrow \text{offspring} \cup$  best  $n$  individuals from
      $\text{auxiliary\_population}_c$ 
21:   end for
22:    $\text{pool\_rules} \leftarrow \text{pool\_rules} \cup \text{auxiliary\_population}_c$ 
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
 Σ_N = {<Classifier>, <Rules>, <DefaultClass>}
 Σ_T = {rule, class, =, value}
 P = {<Classifier> \Leftarrow <Rules> <DefaultClass>;
 <Rules> \Leftarrow rule (rule)*;
 <DefaultClass> \Leftarrow 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.

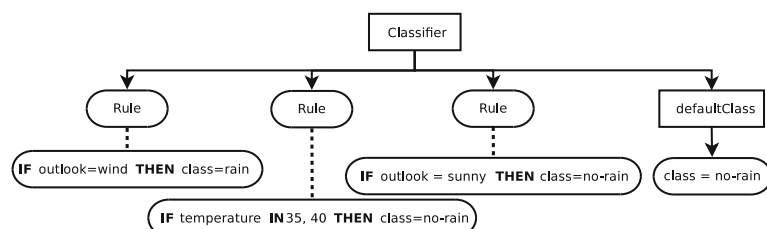
Step 2: Rule Selection

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 classifier. 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 $F(ind) = \frac{1}{l} * \sum_{c=1}^m \frac{accuracy_c}{m}$. 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:    $selected\_individuals \leftarrow$  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:    $P_{i+1} \leftarrow offspring \cup best\_individual$ 
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.availables() > 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  $\leftarrow$  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 $\langle \text{key}, \text{value} \rangle$ 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 $\langle \text{key}, \text{value} \rangle$ pairs as input. Here, the global support values for antecedent ($\text{support}(X)$), consequent ($\text{support}(Y)$), and rule ($\text{support}(R)$) are obtained for each individual. Once, these three measures have being calculated, the fitness function is obtained as $F(R) = \text{confidence}(R) \times \text{support}(R) / \text{support}(Y)$ 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. $\sum_{i=0}^m \text{numberGenerations}(c_i)$ represents the number of MapReduce jobs, where m is the number of classes, and $\text{numberGenerations}(c_i)$ 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:      $P_{0,c} \leftarrow$  Generate a random population of  $n$  rules following
       the grammar with class c
4:     auxiliary_populationc  $\leftarrow \emptyset$ 
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  $\leftarrow$  pool_rules  $\cup$  auxiliary_populationc
10:    end
11: end for
end procedure

procedure mapper(instance,  $P_{i,c}$ )
1: for all rule in  $P_{i,c}$  do
2:   measures  $\leftarrow$  rule.evaluate(instance)
3:   emit(rule, measures)
4: end for
end procedure

procedure reducer(rule, measures[])
1: finalMeasures  $\leftarrow (0, 0, 0)$  // Support antecedent, conse-
   quent and rule
2: for all measure in measures[] do
3:   for  $i = 0$  to 2 do
4:     finalMeasures[i]  $\leftarrow$  finalMeasures[i] +
       measure[i]
5:   end for
6: end for
7: fitness  $\leftarrow$  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 $F(rule - set) = \frac{1}{l} * \sum_{m=1}^m \frac{accuracy_c}{m}$ 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  $\leftarrow (0, \dots, 0)$  // 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  $\leftarrow (0, \dots, 0)$  // 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 $Kappa = \frac{N \sum_{i=1}^k x_{ii} - \sum_{i=1}^k x_{i \cdot} x_{\cdot i}}{N^2 - \sum_{i=1}^k x_{i \cdot} x_{\cdot i}}$, where x_{ii} is the count of cases in the main diagonal of the confusion matrix, N is the number of instances and, finally, $x_{i \cdot}$ and $x_{\cdot i}$ 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

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
<i>G3P-ACBD</i>	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
<i>G3P-ACBD</i>	3.716

Italic typeface denotes the algorithm that achieves the best ranking

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.

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
<i>G3P-ACBD</i>	<i>1.650</i>
(b) Ranking for kappa measure	
DAC	4.600
MRAC	4.100
MRAC+	2.600
DFAC-FFP	1.900
<i>G3P-ACBD</i>	<i>1.800</i>

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-FFP 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
<i>FARCHD</i>	<i>2.066</i>

Italic typeface denotes the algorithm that achieves the best ranking

expert's point of view). Given a classifier C including a set of rules $C = \{R_0, \dots, R_n\}$, then $complexity(C) = n \sum_{i=0}^n attributes(R_i)$ represents the interpretability 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 -value = $2.2 \cdot 10^{-16}$, 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

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	<i>0.000</i>	0.687	<i>0.000</i>	<i>0.000</i>

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
<i>G3P-ACBD</i>	<i>1.000</i>

Italic typeface denotes the algorithm that achieves the best ranking

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.

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 interpretability. 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	<i>0.000</i>	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
<i>G3P-ACBD (Multi-thread)</i>	<i>2.167</i>

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 = 1.399×10^{-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	<i>0.000</i>	0.156	<i>0.001</i>	<i>0.000</i>

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
<i>G3P-ACBD (Spark)</i>	<i>1.650</i>

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)	<i>0.001</i>	0.193	0.862

Italicized values represent the best ranking value

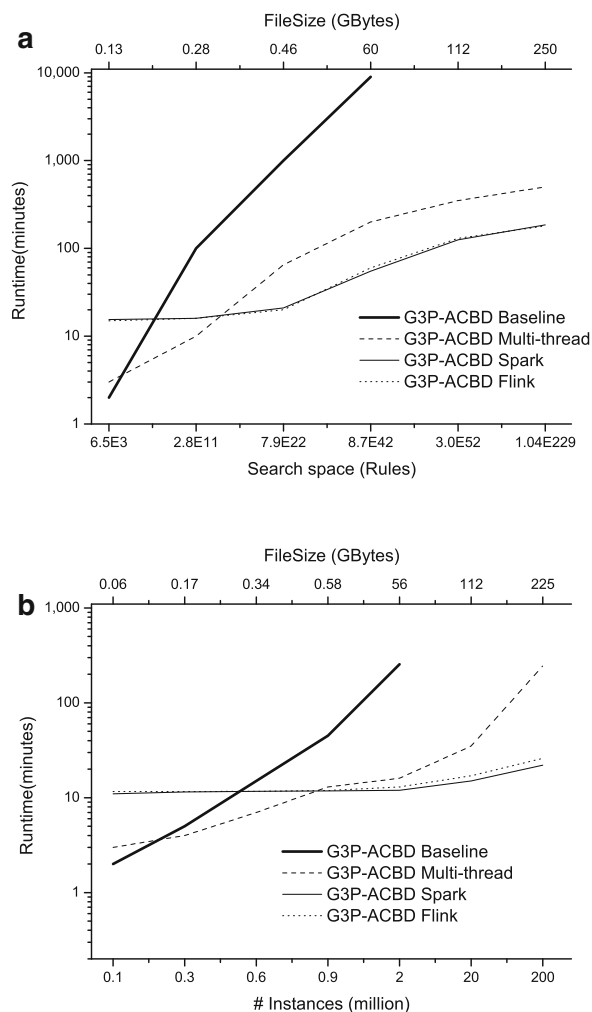


Fig. 5 Runtime of the different implementations considering truly Big Data

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.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

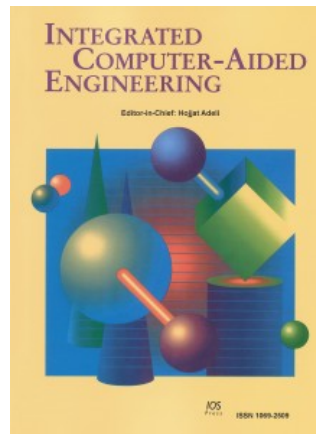
- Fernández A, del Río S, Chawla NV, Herrera F. An insight into imbalanced big data classification: outcomes and challenges. *Complex & Intelligent Systems*. 2017;3(2):105–20.
- Chen H, Chiang R, Storey V. Business intelligence and analytics: from big data to big impact. *MIS Quarterly: Management Information Systems*. 2012;36(4):1165–88.
- Cambria E, Chattopadhyay A, Linn E, Mandal B, White B. Storages are not forever. *Cogn Comput*. 2017;9(5):646–58.
- Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. *SIGMOD Rec*. 1993;22(2):207–16.
- Han J, Kamber M. Data mining: concepts and techniques. Morgan Kaufmann. 2011.
- Quinlan R. C4.5: Programs for machine learning. San Mateo: Morgan Kaufmann Publishers; 1993.
- Cortes C, Vapnik V. Support vector networks. *Mach Learn*. 1995;20:273–97.
- Thabtah FA. A review of associative classification mining. *Knowl Eng Rev*. 2007;22(1):37–65.
- 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.
- 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.
- Bechini A, Marcelloni F, Segatori A. A MapReduce solution for associative classification of big data. *Inf Sci*. 2016;332:33–55.
- 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.
- 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.
- Venturini L, Baralis E, Garza P. Scaling associative classification for very large datasets. *Journal of Big Data*. 2017;4(1):44.
- 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.
- Ventura S, Luna JM. Pattern mining with evolutionary algorithms. New York: Springer International Publishing; 2016.
- Oneto L, Bisio F, Cambria E, Anguita D. SLT-based ELM for big social data analysis. *Cogn Comput*. 2017;9(2):259–74.
- 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.
- 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.
- 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.
- Siddique N, Adeli H. Nature inspired computing: an overview and some future directions. *Cogn Comput*. 2015;7(6):706–14.
- Lam C. Hadoop in action, 1st ed. Greenwich: Manning Publications Co.; 2010.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- Ben-David A. Comparison of classification accuracy using Cohen's Weighted Kappa. *Expert Syst Appl*. 2008;34(2):825–32.
- Triguero I, González S, Moyano JM, García 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.
- Yin X, Han J. CPAR: classification based on predictive association rules. In: 3rd SIAM International Conference on Data Mining(SDM03); 2003. p. 331–5.
- 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.
- Liu B, Ma Y, Wong CK. In: Classification Using Association Rules: Weaknesses and Enhancements. Kluwer Academic Publishers; 2001. p. 591–601.
- 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.
- Cohen WW. Fast effective rule induction. In: Machine Learning: Proceedings of the 12th International Conference; 1995. p. 1–10.
- Tan KC, Yu Q, Ang JH. A coevolutionary algorithm for rules discovery in data mining. *Int J Syst Sci*. 2006;37(12):835–64.
- Holte RC. Very simple classification rules perform well on most commonly used datasets. *Mach Learn*. 1993;11:63–91.
- Segatori A, Bechini A, Ducange P, Marcelloni F. A distributed fuzzy associative classifier for big data. *IEEE Trans Cybern*. 2018;48(9):2656–69.
- 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.

TITLE:

Mining association rules on Big Data through MapReduce genetic programming

AUTHORS:

F. Padillo, J.M. Luna, F. Herrera and S. Ventura



Integrated Computer-Aided Engineering, *Volume 25, Issue 1, pp. 31–48, 2018*

RANKING:

Impact factor (JCR): 4.904

Knowledge area: Computer Science; Engineering.

DOI: 10.3233/ica-170555

Mining Association Rules on Big Data through MapReduce Genetic Programming

F. Padillo^a, J.M. Luna^{bf}, F. Herrera^{cd} and S. Ventura^{aef*}

^aDepartment of Computer Science and Numerical Analysis, University of Cordoba, 14071 Cordoba, Spain.

^bDepartment of Computer Science, University of Jaén, 23071, Jaén, Spain.

^cFaculty of Computing and Information Technology, North Jeddah, Saudi Arabia Kingdom.

^dDepartment of Computer Science and Artificial Intelligence, University of Granada, 18071, Granada, Spain.

^eFaculty of Computing and Information Technology, King Abdulaziz University, Saudi Arabia Kingdom.

^fKnowledge Discovery and Intelligent Systems in Biomedicine Laboratory, Maimonides Biomedical Research Institute of Cordoba, Spain.

Abstract. Association rule mining is one of the most important tasks to describe raw data. Although many efficient algorithms have been developed to this aim, existing algorithms do not work well on huge volumes of data. The aim of this paper is to propose a new genetic programming algorithm for mining association rules in Big Data. The genetic operators of our proposal have been specifically designed to avoid a growing in the complexity of the solutions without an improvement in their fitness function values. Furthermore, it introduces a repairing operator to improve the convergence. Additionally, to facilitate its application on real world problems a grammar has been included, allowing it to introduce subjective knowledge into the mining process and to reduce the search space. Due to the growing interest in data gathering, a unique implementation of the proposed algorithm is not useful so different implementations (considering different architectures such as RMI, Hadoop and Spark) are required depending on the data size. All these adaptations obtain exactly the same solutions as those of the original algorithm since they only differ on the software architectures. The experimental study considers more than 75 datasets and 14 algorithms and the results reveal that the proposed algorithm obtains excellent results for more than 12 quality measures. The scalability of the proposal is also analyzed by considering the three parallel implementations on high dimensional datasets (3,000 millions of instances) and file sizes up to 800 GB.

Keywords: Association Rules, Big Data, MapReduce, Hadoop, Spark

1. Introduction

As technology advances, high volumes of valuable data are generated in modern organizations. Nowadays, the extraction of knowledge from such raw massive data is a priority to support decision making. It has driven and motivated the research in improving techniques for data analysis in such massive datasets, giving rise to the new buzzword Big Data [30]. This term encompasses a set of techniques to face up the problems derived from the management and analysis of these huge quantities of data [10].

The extraction of patterns of interest that represent intrinsic and important properties of data plays an important role in data analysis. Nevertheless, the knowledge extracted by a single pattern might be meaningless, and a more descriptive analysis can be

required. In this sense, the concept of association rules was proposed by *Agrawal et al.* [2] as a way of describing correlations within patterns of potential interest. Association rule mining was firstly described in the context of market basket analysis [40], where it was used to determine which products were bought together. Nowadays, though, the applications are not limited to the market basket analysis and more and more domains [16, 18, 26] are interested in using this kind of relationships.

First approaches in the association rule mining [1] field were based on the Apriori algorithm. It is an exhaustive search algorithm that extracts associations of interest by dividing the problem into two sub-tasks: (1) finding patterns whose frequency of occurrence is greater than a minimum threshold; and (2) extracting association rules from the previously ob-

tained patterns. Despite the fact that this first algorithm [2] worked well in many different fields, the number of applications has grown exponentially and this rapid increment in size and number of datasets has given rise to some limitations. For instance, obtaining all the rules could be unfeasible if the dataset has a high number of k different single items, producing $2^k - 1$ patterns and $3^k - 2^{k+1} + 1$ rules to be analyzed and saved in main memory. In addition, real-world datasets include continuous features so the high number of distinct values produces extremely large search spaces to be considered by traditional approaches.

In order to overcome existing drawbacks in the mining of association rules, many research studies [36] have been focused on extracting these relationships by means of Evolutionary Algorithms (EAs). The use of EAs enables to extract association rules in a single step, not requiring a previous subtask for mining frequent patterns. Multi-objective optimization [31, 32] has been also considered by different researchers to extract association rules by means of EAs [12, 20, 21]. Additionally, some researchers

have applied EAs for optimizing patterns in continuous domains, not requiring any previous discretization step. But even more important than all of this is the reduction in both the computational time and the memory requirements by considering the pattern mining problem as a combinatorial optimization issue. Even when really efficient algorithms have been proposed for mining association rules, truly Big Datasets hamper the process of mining association rules. In the 1990s, 20 attributes were called a large-scale problem [27]. Nowadays, the number of attributes in many areas, e.g. gene analysis, can easily reach thousands or even millions [33]. Under these circumstances, new forms of processing data are needed to enhance the process of decision making and knowledge discovery when massive data are considered [14, 17]. Additionally, parallel computing is being applied in this field, by considering both multi-core processors and multiple computers through a Remoted Method Invocation (RMI). In this sense, *Cano et al.* [5] proposed the use of Graphics Processor Units (GPUs) to speed up the process of mining association rules. GPU computing allows thousands of cores to be used at the same time, however, it could not be enough when truly Big Data are considered. In this regard, MapReduce [6] has emerged as a paradigm to tackle Big Data. It uses multiple machines in a distributed way, enabling a higher level of parallelism. Neverthe-

less, not only are new huge quantities of data available but also a massive number of decision variables, different mathematical properties of the data or even various type of constraints. Thus, this problem cannot be solved by only increasing the computational power or by using paradigms such as MapReduce. Hence, novel methods and algorithms based new advances in distributed computing [27] have become a necessity [25].

The goal of this paper is therefore to propose a new efficient EA to extract association rules in Big Data. The baseline of this work is a new Grammar-Guided Genetic Programming algorithm to optimize Leverage, Support and Confidence, known as G3P-LSC. The proposed model makes use of a context-free grammar to encode the solutions and it allows to restrict the search space by adding some syntax constraints, i.e. it enables expert's knowledge to be introduced into the mining process. Furthermore, our proposal is eminently designed to be as parallel as possible so Big Data can be tackled, and its operators have been specifically designed to avoid the loss in large search spaces as well as to maintain diversity in the solutions. In this regard, its genetic operators provide a reduced set of rules with high values for many different quality measures and few attributes, making it easier to understand from a user's perspective. Taking the proposed sequential algorithm G3P-LSC as a starting point, different approaches have been finally implemented considering both RMI and MapReduce (Hadoop and Spark). In order to analyze the scalability of the proposal and its parallel versions, the experimental study includes different data sizes, considering datasets with more than 3,000 millions of instances. G3P-LSC has been compared with other 14 algorithms and using more than 75 datasets. Results state that the proposed G3P-LSC algorithm mines rules by optimizing the desired qualities, providing the user with rules of high interest. Finally, the proposed approach presents a good computational cost and a promising scalability when the size of the problem increases.

The rest of the paper is organized as follows. Section 2 presents the most relevant definitions and related work; Section 3 describes the proposed algorithm; Section 4 presents the datasets used in the experiments and the results; finally, some concluding remarks are outlined in Section 5.

2. Preliminaries

In this section, the association rule mining task is formally defined, and the MapReduce paradigm is analyzed.

2.1. Association Rule Mining

Association Rule Mining (ARM) [40] is considered as one of the most relevant tasks in unsupervised learning. It aims to discover accurate associations between item-sets of interest for the application domain. These associations have a descriptive nature, describing useful behaviors for the end user.

In a formal way, it is possible to define an association rule as follows [36]. Let $I = \{i_1, i_2, i_3, \dots, i_n\}$ be the set of items or features, and let define a set of all transactions $T = \{t_1, t_2, t_3, \dots, t_m\}$ in a dataset, where each transaction t_j comprises a subset of items $\{i_k, \dots, i_l\}$, $1 \leq k, l \leq n$. An association rule is formally defined [1] as an implication of the form $X \rightarrow Y$ where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The meaning of an association rule [13] 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. $Y \subset t_j$. Nevertheless, in some scenarios the extraction of rules of the form $X \rightarrow Y$ could not be enough, and it may be interesting to extract rules such as $X \rightarrow \neg Y$. This kind of rules relates the presence of X to the absence of Y [13].

ARM obtains relationships from data where no information is known, so the extracted knowledge might be hardly quantifiable sometimes. In general, association rules represent data behavior and their interest is quantified by means of metrics that determine how representative a specific rule is within a dataset. Tons of quality measures have been defined for this aim [3], being support and confidence the two most widespread metrics in literature [1]. The support of the item-set X (see Equation 1) is defined [1, 40] as the number of transactions that satisfies $X \subset t_z \in T$.

$$\text{Support}(X) = |\{t_j \in T, X \subset t_j : t_j \subseteq I\}| \quad (1)$$

In the same way, the support of an association rule $X \rightarrow Y$ (see Equation 2) is defined as the number of transactions from T that satisfies both X and Y [1, 13, 40].

$$\text{Support}(X \rightarrow Y) = |\{t_j \in T, X \subset t_j \wedge Y \subset t_j : t_j \subseteq I\}| \quad (2)$$

As for the confidence quality measure [1, 13, 40], it determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure (see Equation 3) is defined as the proportion of transactions that satisfies both the antecedent X and the consequent Y among those transactions that contain only the antecedent X [1].

$$\text{Confidence}(X \rightarrow Y) = \text{Support}(X \rightarrow Y) / \text{Support}(X) \quad (3)$$

Even though these quality measures are extensively used in the field, they have some downsides [13, 36]. First, the confidence measure does not detect statistical independence or negative dependence between items. Second, item-sets with very high support are a source of misleading rules. To overcome these drawbacks, many researchers have proposed several measures for the selection of interesting rules, and leverage is one of the most alluring since it satisfies the three properties proposed by *Piatetsky-Shapiro* [28]. Leverage (see Equation 4) calculates how different is the co-occurrence of the antecedent X and consequent Y from expected [1], i.e. from independence. This quality measure [36] takes values in the range $[-0.25, 0.25]$, and a zero value states for statistical independence between X and Y .

$$\text{Leverage}(X \rightarrow Y) = \text{Support}(X \rightarrow Y) - (\text{Support}(X) \times \text{Support}(Y)) \quad (4)$$

2.2. MapReduce

MapReduce [6] is a recent paradigm of distributed computing in which programs are composed of two main phases defined by the programmer: map and reduce. MapReduce considers that the input and output are based on (*key, value*) pairs, which are also denoted as tuples $\langle k, v \rangle$. In the map phase, each mapper processes a sub-set of input data and produces $\langle k, v \rangle$ pairs. Then, an intermediate step is carried out, known as shuffle phase, which merges all the values associated with the same key. 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. The flowchart of a generic MapReduce framework is depicted in Figure 1.

Hadoop [15] is the de facto standard for MapReduce applications. Hadoop implements the MapRe-

duce paradigm and provides a distributed filesystem known as Hadoop Distributed File System (HDFS), which replicates file data in multiple storage nodes that can concurrently access to the data. The main drawback of Hadoop is that it imposes an acyclic data flow graph, and there are applications that cannot be modeled efficiently using this kind of graph such as iterative or interactive analysis [38]. Moreover, the communication among mappers and reducers are performed using disk. This operation could cause problems of I/O, when the number of (*key*, *value*) pairs are extremely large. The disk being the main bottleneck due to the slow speed of read/write. All of this has hampered the modeling of efficient iterative algorithms in this platform. To solve these downsides, a novel solution has been proposed known as Spark [38]. This new proposal is eminently designed to be used in iterative and interactive algorithms. To speed up the process of tackling huge amounts of data, it introduces an abstraction called Resilient Distributed Datasets (RDDs). RDD represents a read-only collection of objects partitioned across a set of machines stored in main memory. It allows us to load a dataset in memory one time, and read multiple times without having to load it from disk in each iteration as Hadoop does. Furthermore, the communication among mappers and reducers are performed in memory, being much faster than the approach followed by Hadoop. One of the main strengths of Spark is its rich application program interface, which provides a set of in-memory primitives facilitating the modeling of algorithms.

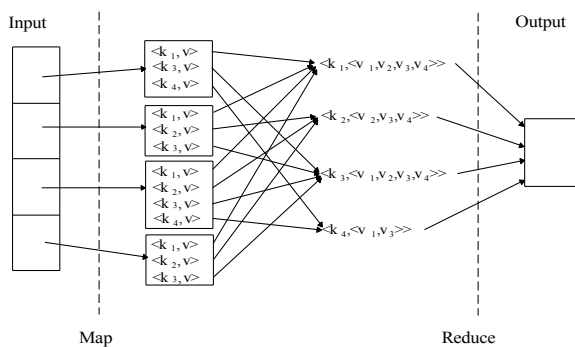


Figure 1. Diagram of a generic MapReduce framework

3. Evolutionary algorithm based on grammars for mining association rules in Big Data

The main motivation of this work is to propose an EA based on grammars for mining association rules.

This work has been eminently designed to tackle massive amounts of data, where its genetic operators enable to scale on huge search spaces without losing accuracy and maintaining diversity among solutions. Its repairing operator allows to improve the convergence in complex spaces. Additionally, the fitness function has been explicitly designed to find frequent and reliable rules whose antecedent and consequent are not independent. To accomplish this, two different populations and three genetic operators as well as a grammar have been used. Also, due to the growing interest in data gathering, a unique and universal implementation of the proposed algorithm is not useful, so different adaptations are carried out depending on the data size. Hence, different adaptations have been performed in function of the used architecture, and all of them are fully described in further sections. Finally, it is worth noting that all of these adaptations obtain exactly the same solutions, the unique difference among them is the used software architecture.

3.1. Baseline

The baseline of this work is a new EA, known as G3P-LSC, that makes use of a context-free grammar to constrain the search space. Many authors have explored the use of grammars in pattern mining, achieving excellent results in both introducing subjective external knowledge into the mining process and restricting search space by introducing some syntax constraints [19]. A major strength of using grammars is the adaptability to represent solutions with different forms and features in such a way that a simple change in the grammar is able to produce completely different solutions. However, the user should be cautious when using grammars in pattern mining since the fact of reducing the search space may produce the loss of high interesting solutions, e.g. those that do not satisfy the constraints provided by the grammar. Besides, another major limitation of using grammars is the possibility of bloating by which the trees expand without control and the complexity increases. All these downsides are overcome in the proposed G3P-LSC as it is described below.

Encoding. G3P-LSC represents each solution as a derivation syntax tree encoded by means of a set of production rules from the context-free grammar shown in Figure 2. It is defined as a four-tuple $(\Sigma_N, \Sigma_T, P, S)$ where Σ_N and Σ_T represent the alphabet of non-terminal and terminal symbols, respectively; and they have no common elements, i.e. $\Sigma_N \cap \Sigma_T = \emptyset$.

Terminal symbols are literals of the grammar and cannot be changed using the rules of the grammar. For example, the value of an attribute does not change even when the rules of the grammar are modified. Additionally, terminal symbols do not appear in the left-hand side of any production rule. On the contrary, non-terminal symbols are lexical elements used to form a grammar, and they can be replaced to produce different solutions. Non-terminal symbols may appear in both left and right-hand side of the production rules. In order to encode a solution, a number of production rules from the set P are applied beginning from the start symbol denoted by S . A production rule is defined as $\alpha \rightarrow \beta$ where $\alpha \in \Sigma N$, and $\beta \in \{\Sigma T \cup \Sigma N\}^*$. After applying the production rules, a derivation syntax tree is obtained for each solution, where internal nodes contain only non-terminal symbols, and leaves contain only terminal symbols.

```
G = ( $\Sigma N$ ,  $\Sigma T$ ,  $P$ ,  $S$ ) with:
S = Rule
 $\Sigma N$  = {Rule, Antecedent, Consequent, Condition, Nominal,
        Numerical}
 $\Sigma T$  = {'name', '=', 'value', 'IN', 'Min_value', 'Max_value'}
P = {
    Rule := Antecedent, Consequent;
    Antecedent := Condition | Condition, Antecedent;
    Consequent := Condition | Condition, Consequent;
    Condition := Numerical | Nominal;
    Numerical := 'name' 'IN' 'Min_value', 'Max_value';
    Nominal := 'name' '=' 'value';
}
```

Figure 2. Context-free grammar defined by G3P-LSC

To generate each solution, the derivation syntax tree is obtained by applying a series of derivation steps from the start symbol of the grammar. From this symbol, the algorithm searches solutions belonging to the set P , until a valid derivation chain is reached. Additionally, in order to avoid bloating that is one of the main problems of using grammars, a maximum number of derivations is previously determined as an input parameter. In this regard, there is a maximum length that no rule can exceed. Genetic operators are aware of this maximum length so they are specifically designed to avoid an uncontrolled growth of the solutions.

Evaluation procedure. In any evolutionary approach, the evaluation process is cornerstone since it is responsible for assigning a fitness value to determine how promising each rule is for a specific aim. The proposed fitness function F of a solution (rule) $R \equiv X \rightarrow Y$ is the product of support, confidence and

leverage, i.e. $F(R) = \text{support}(R) \times \text{confidence}(R) \times \text{leverage}(R)$. This fitness function takes values in the range $[-0.25, 0.25]$. Support and confidence are the most widespread measures in association rule mining and they are related in such a way that the confidence value of a rule cannot be lower than its support [36]. It means that high support values imply high confidence values but in those datasets where the maximum feasible support value is not too high, then confidence is required to be maximized and, therefore both metrics should be considered at time. Even though these two quality measures are extensively used in the field they should be considered together with additional quality measures [35]. In this regard, Leverage appears as a good metric to determine co-occurrence of the antecedent and consequent of a rule from independence. A major feature of this quality measure with regard to similar metrics [36] is that its values have predefined lower and upper bounds, i.e. $[-0.25, 0.25]$, zero value denoting a statistical independence between antecedent and consequent.

The fitness function has been precisely designed to avoid frequent (support) and reliable (confidence) rules where the antecedent and consequent are not independent. Using the product of these metrics it is obtained that when leverage is zero, then the overall fitness value is also zero.

Finally, it is important to highlight that the evaluation procedure is carried out in a sequential way, where the whole dataset has to be read from one processor, evaluating each rule of the main population in each instance.

Algorithm. The G3P-LSC algorithm proposed as baseline for mining association rules in Big Data environments is depicted in Listing 1. It starts by encoding rules (line 1, Listing 1) by using the context-free grammar defined in Figure 2 and the extracted metadata. Additionally, the maximum feasible length of this grammar is set to the number of attributes in data, so it is possible to obtain rules comprising the whole set of features.

After selecting a set of individuals to work as parents (line 7, Listing 1), the next step is to apply the crossover operator with a certain probability. If the crossover operator is applied, two offspring will be generated, which could be independently mutated with a certain probability. On the contrary, if the crossover operator is not applied, then the two parents could be separately mutated with a certain probability (see lines 7 to 16, Listing 1). A major feature of G3P-LSC is the elitism, in which best solutions are guaranteed a place in the next generation. It is especially important in mining association rules since

both crossover and mutation are too disruptive operators and may cause the loss of really promising solutions. This evolutionary process is repeated a number of generations specified by the user.

Listing 1 G3P-LSC sequential algorithm

```

1: Initialize a random population of N rules as  $P_0$ 
2: auxiliary_population  $\leftarrow \emptyset$ 
3: for  $i = 0$  to NumberOfGenerations do
4:   offspring  $\leftarrow \emptyset$ 
5:   evaluate_rules( $P_i$ )
6:   Maintain elitism using auxiliary_population
7:   Apply BetterSelector to  $P_i$ 
8:   for each pair in  $P_i$  do
9:     if Rand_number(0, 1) <  $P_{cro}$  then
10:      pair  $\leftarrow$  Apply crossover and repairing operator (pair)
11:     end if
12:     for each individual of the pair do
13:       if Rand_number(0, 1) <  $P_{mut}$  then
14:         individual  $\leftarrow$  Apply mutation and repairing operator (individual)
15:       end if
16:     end for
17:     offspring  $\leftarrow$  offspring + pair
18:   end for
19:    $P_{i+1} \leftarrow$  offspring + auxiliary_population
20: end for

```

Genetic operators. The crossover genetic operator works by interchanging a random sub-tree between two parents, whereas the mutation operator applies changes to attributes from the antecedent and the consequent. These are high disruptive operators, giving rise to solutions whose fitness values highly vary from the original solution (parents). This issue is caused by the ARM problem itself since the simple fact of changing a single attribute in a rule may produce wrong solutions or even completely different leverage, support and confidence values. In this regard, G3P-LSC also proposes a repairing operator used to improve the algorithm's performance by modifying invalid rules. The main idea is really simple since this operator checks whether the antecedent and consequent of the rule include similar items, and it is checked on each of the resulting solutions. It is important to highlight that, according to the formal definition of association rules provided in Section 2.1, both the antecedent and consequent cannot include the same items, i.e. $X \cap Y \neq \emptyset$. Thus, a rule is considered as invalid if both the antecedent and the consequent comprise common items and the repairing

operator therefore works by removing those repeated items and providing rules satisfying $X \cap Y = \emptyset$. In general, invalid elements are randomly removed either from antecedent or consequent with the only constraint that the resulting solution satisfies that $X = \emptyset$ and $Y = \emptyset$.

3.2. Scaling G3P-LSC using parallel and distributed computing

Focusing on the same idea of G3P-LSC, different parallel and distributed computing architectures have been used to speed up the mining of association rules in Big Data environments. In these implementations, the evaluation procedure has been the unique parallel phase since it has been proved to be the most time consuming [5]. Each version is developed following a different kind of implementation, although all the versions share the same idea and the same results are therefore obtained.

RMI Version. The first parallel version of the G3P-LSC algorithm is based on a master/slave architecture that uses RMI to communicate the master process with each slave. This version, known as G3P-LSC RMI, uses multiple threads and different processes that are distributed among a cluster of machines (See Listing 2).

Listing 2 G3P-LSC RMI-rules are distributed among slaves

```

function evaluate_rules( $P_i$ )
1: Split  $P_i$  in subPopulations // As much as slaves
2: for each subPopulation in subPopulations do
3:    $Slave_j$ .evaluate(subPopulation)
4: end for
end function

```

Focusing on the master process, only a single master procedure is used since it is the coordination point. The master process is almost the same as the baseline approach shown in Listing 1, being the difference the function *evaluate_rules*. In this case, this function splits the main population in as many subpopulations as number of slaves exist (see Listing 2). The aim of each slave is to evaluate the subpopulation of rules in the whole dataset. Furthermore, each slave is located in a different computer node, enabling parallel and distributed computing, and achieving a better performance.

Although RMI provides a high-level programming interface, the load balancing, fault tolerance and the coordination among slaves are very troublesome to manage. In this approach, the dataset has to be repli-

cated in each slave, provoking both high network and I/O activity. Even when this implementation works well for large datasets, the use of truly Big Data hampers the process of replicating data. Some researchers have proposed distributed file systems to solve this issue. However, it is not enough when truly Big Data is considered since each slave process has to read the whole dataset and this operation becomes impossible if the file size is big enough.

MapReduce Versions. These versions have been implemented in two different MapReduce architectures (Apache Hadoop and Apache Spark). Both implementations share the same approaches, even though slight adaptations have been required to adapt to the used architecture. They require three different processes: (1) the driver performs the main code of G3P-LSC baseline, however, the *evaluate_rules* function has been implemented to use MapReduce. Thus, in each generation of the evolutionary process a MapReduce phase is required to evaluate the main population; (2) Mappers in which the main population of rules is evaluated; (3) Reducers which collect the data produced by mappers, and return the evaluated main population for the whole dataset.

Listing 3 G3P-LSC MapReduce

```

function evaluate_rules(population)
  1: MapReduce to evaluate rules in population
end function

procedure evaluatorMapper(instance)
  1: for each rule in population do
  2:   measures  $\leftarrow$  rule.evaluate(instance)
  3:   emit(rule, measures)
  4: end for
end procedure

procedure reducer(rule, measures)
  1: finalMeasure 0
  2: for each measure in measures do
  3:   finalMeasure finalMeasure + measure
  4: end for
  5: emit(rule, finalMeasure)
end procedure

```

In the mapper phase (see *evaluatorMapper* procedure, Listing 3) each mapper receives as input a subset of the dataset and the main population. A group of pairs (*key*, *value*) are generated by each mapper, where the key is the individual (rule or solution within the population set), and the value is a tuple consisting of values for the support of the antecedent, consequent and rule. Hence, each *evaluatorMapper* procedure produces the same number of (*key*, *value*)

pairs as number of individuals exist in the main population. Hence, the number of (*key*, *value*) pairs in each generation is calculated as the number of mappers multiplied by the number of individuals in the main population. The reducer phase, on the contrary, receives these (*key*, *value*) pairs, calculating the total sums of support antecedent, consequent and rule for each individual. Hence, the reducer produces the same quantity of (*key*, *value*) pairs as individuals exist in the main population (see reducer procedure, Listing 3). At the end of this phase, the output is the evaluated main population. Finally, the driver continues the evolutionary procedure similar to the baseline of G3P-LSC. The procedure followed by MapReduce versions can be summarized as illustrated in Figure 3, where the input of the MapReduce phase is the dataset and the main population (shaded rectangle). The output is the evaluated main population which is returned to the driver.

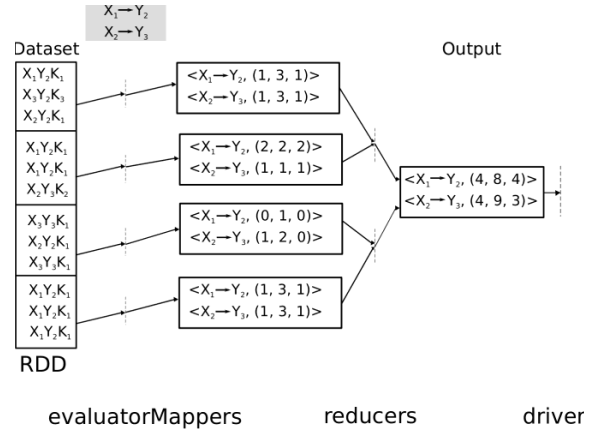


Figure 3. Flowchart for each generation in G3P-LSC versions based on MapReduce. It receives the main population and the dataset as inputs, and it produces as output the evaluated main population. The main population is returned to the driver

Although the two implementations (Hadoop and Spark) share the same approaches some slight differences exist. These differences are produced by the structure of each platform. Hadoop version uses disk as the way of communicating. Thus, communication among driver-mappers, mappers-reducers and reducers-driver are performed using disk. Furthermore, Apache Hadoop does not provide any way of saving the dataset in main memory so each generation has to read the dataset from disk. It should be noted that the process of this algorithm in each generation is exactly the same, so the dataset is loaded in each generation and the outputs are written on disk. In cases where

the dataset is big enough, the reading process could be very time consuming so reading once, saving in main memory and using multiple times could be more efficient. On the other hand, Apache Spark allows to communicate among processes using main memory and enabling a faster communication. The main difference between Hadoop and Spark versions is that in the first generation of Spark, the whole dataset will be loaded in main memory using a RDD. It is split into the cluster, and each mapper could access to one different partition (sub-dataset). Unlike the previous Hadoop version, the dataset is loaded in memory only once, so it does not require a loading for each generation and the global performance can be substantially improved. Moreover, the communication among mappers and reducers are not performed using disk but memory, being this much faster. Both versions return exactly the same set of rules as the previous versions. In order to clarify these points, all the source code is available at <http://www.uco.es/grupos/kdis/wiki/G3PLSC>.

4. Experiments

The aim of this section is to study the performance of the G3P-LSC algorithm and its versions on different parallel architectures when different data dimensionalities are considered. The goal of this study is three-fold:

1. To prove the quality of the obtained solutions. In this sense, a comparative study has been carried out by using exhaustive search algorithms to prove that our proposal is able to discover global optimum solutions in a reduced quantum of time. Additionally, a set of EAs for mining association rules has also been considered in this experimental study, and a comparative analysis of different quality measures has been carried out. A set of statistical tests [8, 9, 10, 11] has been applied to compare the differences.
2. To analyze the scalability of the proposed algorithm when different parallel implementations are considered truly Big Datasets are used in this section to prove scalability considering both synthetic and real-world datasets.
3. To show the interesting of using grammars is to restrict the search space and to include external subjective knowledge that helps in the process of mining association rules.

All the experiments have been run on an HPC cluster comprising 16 computing nodes, with two Intel E5-2620 microprocessors at 2 GHz and 64 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 Hadoop 2.6.0 and Spark 1.6.0.

4.1. Study of the grammar

As previously described, the grammar considered in this approach is defined in Figure 2. Considering this grammar G , the following language is obtained $L(G) = \{condition (condition)^* condition (condition)^*\}$ where the first part of the language, i.e. $\{condition (condition)^*\}$, corresponds to the antecedent of the rule, whereas the second part represents the consequent of the rule.

According to the grammar G (see Figure 2), the minimum length of a rule includes a single item in the antecedent, and a single item in the consequent. Hence, the minimum tree will have a depth of 4, a total of 7 internal nodes (one of them is the starting symbol *Rule*), and requiring 7 derivations to create the final tree. Additionally, the maximum derivation size is fixed to the number of attributes in data, so the maximum value depends on the dataset to be used. Taking a dataset comprising 3 features, the number of internal nodes in the tree is 9, requiring 10 derivations; 11 internal nodes if the dataset comprises 4 features, requiring 13 derivations; and so on. As a result, the maximum number of internal nodes is equal to $3 + 2n$, n being the number of features in data satisfying that $n > 1$; whereas the number of derivations required to create the final tree is $3n + 1$. As for the depth of the tree, it should be highlight that it remains the same for whatever number of features. It implies that the size of the resulting trees increases in width, not in depth.

4.2. Computational complexity

An analysis of the computational complexity is essential to determine the efficiency of the proposed approach. In this sense, we analyze each of its main procedures: encoding criterion, evaluator procedure and genetic operators. According to all these procedures, it is possible to determine the computational complexity of the whole algorithm.

As for the computational complexity of the encoding criterion, it depends on the number of derivations

required to form the tree, which depends on the number of attributes in data as described in Section 4.1, i.e. $3n + 1$, n being the number of features in data. Hence, the final complexity of the encoding criterion will be determined by the number of times (individuals) the derivation process is carried out, resulting as $O(3n \times m + m)$, n denoting the number of features in data, and m stating for the number of individuals to be created. Concerning the evaluator procedure, its complexity depends on the number m of individuals, t instances in data and n attributes. Mathematically, the final complexity is defined as $O(m \times t \times n)$. Finally, the computational complexity of the three genetic operators depends on both the derivation tree size (number of derivations, i.e. $3n + 1$, for a dataset with n attributes or features) and the number of individuals m . In consequence, the final complexity order is defined as $O(3n \times m + m)$.

Analyzing the computing requirements for each procedure, it is stated that the number m of individuals is previously fixed, so it is considered as a constant with a complexity $O(1)$. Additionally, all the procedures are repeated as many times as the predefined number of generations, which is also a constant value predefined. Therefore, bearing in mind all these issues, the resultant computational complexity of the complete algorithm is defined as $O(n \times t)$. Thus, the complexity of the proposed approach is linear with regard to the number of instances and the number of attributes. It is important to highlight that any of the evolutionary approaches used in the experimental stage presents the same computational complexity, being linear with regard to the number of instances and attributes in data. On the contrary, exhaustive search approaches require the whole search space to be generated, existing $2^n - 1$ different patterns in data when n attributes are considered. Thus, the computational complexity is exponential with regard to the number of attributes in data, and it becomes prohibitively expensive for really large datasets.

4.3. Analysis of the genetic operators

In this section, an interesting analysis of the usefulness of the proposed genetic operators is carried out. The aim of this section is to demonstrate that the fitness function improves when the three proposed genetic operators are considered at time. To this aim, it is analyzed how the average fitness function improves along the generations when considering the same algorithm with, and without, the proposed genetic operators.

According to the results illustrated in Figure 4, the worst results are obtained when the algorithm includes only a single genetic operator (either crossover or mutation). In fact, the behavior of the algorithm is almost the same for these two genetic operators, and only a slight difference is obtained close to the generation number 1,000 (the crossover operator converges faster than the mutation operator). However, when the repairing operator comes into play the results obtained by each of the previously analyzed genetic operators (mutation and crossover) are improved. Now, both genetic operators achieve a higher convergence. Finally, both mutation and crossover are considered at time including (or not) the repairing operator. As a result, it is obtained that the fact of considering the two main genetic operators (mutation and crossover) produces a huge improvement in the convergence of the algorithm. In fact, the resulting average fitness values are one order of magnitude better than those obtained when the genetic operators were considered in isolation. These results are even better when the three genetic operators (crossover, mutation and repairing) are considered at time.

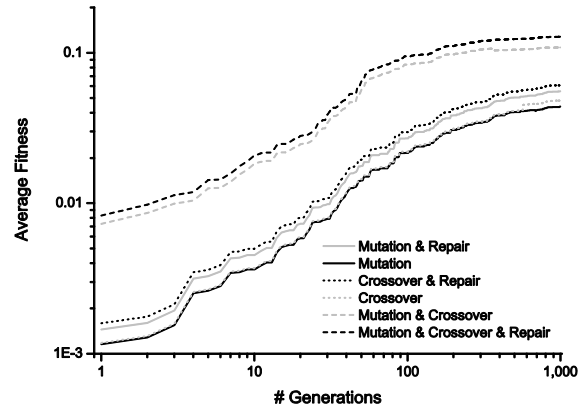


Figure 4. Analysis of the convergence of the three proposed genetic operators

To sum up, both crossover and mutation genetic operators play an important role in the convergence of the algorithm and they should be considered at time. The fact of including the repairing operator in the proposed algorithm slightly improves the obtained results, and this improvement is higher when the number of generations increases.

4.4. Datasets included in the experimental study

This experimental section considers a large number of datasets (see Table 1) comprising both synthet-

ic and real-world datasets. The goal of these studies is to analyze the performance of different algorithms for mining association rules on a high number of datasets and to prove the scalability of the proposed approach. Thus, the used datasets depend on the type of experiment as described below.

Table 1: Datasets considered for the experimental study

Dataset	Attributes (R/I/N)	File size (MB)	Instances
Bolts	8 (2/6/0)	0.20	40
Breast Cancer Wisc. ¹	10 (0/10/0)	0.10	699
Stock Price	10 (10/0/0)	0.06	950
Tic-Tac-Toe ¹	9 (0/0/9)	0.02	958
Statlog ¹	20 (0/7/13)	1.50	1,000
Flare ²	11 (0/0/11)	0.02	1,066
Car ²	6 (0/0/6)	0.04	1,728
Chess ¹	36 (0/0/36)	0.40	3,196
Texture ²	40 (40/0/0)	1.50	5,500
Optdigits ²	64 (0/64/0)	0.80	5,620
Satimage ²	36 (0/36/0)	0.70	6,435
Marketing ²	13 (0/13/0)	0.10	6,876
Thyroid ²	21 (6/15/0)	0.40	7,200
Ring ²	20 (20/0/0)	0.70	7,400
Twonorm ²	20 (20/0/0)	1.20	7,400
Mushroom ¹	22 (0/0/22)	0.20	8,124
Coil2000 ¹	85 (0/85/0)	1.80	9,822
PenBased ²	16 (0/16/0)	0.60	10,992
Nursery ²	8 (0/0/8)	1.20	12,690
Magic ¹	10 (10/0/0)	1.50	19,020
Letter ²	16 (0/16/0)	0.70	20,000
UJIIndoorLoc ¹	529 (2/527/0)	45.00	21,048
House16H ²	17 (10/7/0)	3.80	22,784
Grammatical ¹	100 (100/0/0)	56.00	27,965
ChessKrkP ¹	6 (0/0/6)	0.70	28,056
Adult ¹	14 (6/0/8)	3.00	48,842
Statlog (Shuttle) ¹	10 (0/10/0)	1.60	58,000
Connect4 ¹	42 (0/0/42)	11.00	67,557
ColorTexture ²	17 (16/1/0)	11.00	68,040
ColorHistogram ²	33 (32/1/0)	20.00	68,040
Fars ²	29 (5/0/24)	11.00	100,968
Census ²	41 (1/12/28)	60.00	299,284
Epsilon	2000 (2000/0/0)	11,000.00	500,000
Covtype ¹	54 (0/10/44)	72.00	581,012
Transactions90k ²	3 (0/3/0)	11.00	855,367
Poker ²	10 (0/10/0)	25.00	1,025,010
US Census Data 1990 ¹	68 (0/0/68)	328.00	2,458,285
SUSY ¹	18 (18/0/0)	2,000.00	5,000,000
HEPMASS ¹	28 (28/0/0)	7,000.00	10,500,000
HIGGS ¹	28 (28/0/0)	7,300.00	11,000,000
Protein Structure ³⁴	631 (77/462/92)	62,000.00	34,890,838

¹ UCI repository: <https://archive.ics.uci.edu/ml/datasets.html>

² KEEL repository: <http://keel.es>

³ LIBSVM repository: <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/>

In the first analysis only real-world datasets have been considered since the quality of the solutions have been studied. In this sense, a total of 41 real-world datasets have been analyzed, their main characteristics are shown in Table 1. Here, the label *Attributes(R/I/N)* states for the number of Real, Integer, and Nominal features included in each dataset; *File size* denotes the size in terms of memory for each dataset; whereas the *Instances* variable indicates the number of transactions within each dataset. It should be pointed out that all these datasets are freely available and the source to be downloaded is specified for each one. Finally, the second study considers 35 synthetic datasets so the scalability of the proposal can be analyzed. Synthetic datasets have been required since the number of both instances and attributes can be changed to illustrate the performance of different implementations. In these datasets, the number of instances ranged from $1 \cdot 10^5$ to $3 \cdot 10^9$, the number of continuous and discrete attributes ranged from 8 to 48 distributed following a Gaussian distribution throughout the whole set of instances, the file size varies from 46 MB to 804 GB.

4.5. Sequential algorithms and set up

In these experimental studies, 14 different non-parallel algorithms have been considered to be analyzed, comprising both exhaustive search and evolutionary approaches. All these algorithms have been selected according to their efficiency and significance within the association rule mining field. The aim of this first study is therefore to analyze the resulting set of solutions in terms of quality measures. Each of these non-parallel algorithms has been briefly described as follows:

- 1) Apriori [2]: first algorithm for mining association rules, which is based on an exhaustive search methodology. It exploits the search space by means of the downward closure property.
- 2) Eclat [39]: it employs a depth-first strategy by extending prefixes of candidate itemsets.
- 3) GENAR [23]: genetic algorithm for mining quantitative association rules without a previous discretization step. Each solution is encoded by using the minimum and maximum intervals of each numerical attribute.
- 4) GAR [24]: extension of the GENAR [23] algorithm. Each solution is encoded by using all the attributes within data.

- 5) EARMGA [37]: genetic algorithm for mining association rules in continuous domains. It does not require a minimum predefined support threshold.
- 6) Alatasetal [3]: genetic algorithm for mining quantitative association rules. This algorithm is able to extract both positive and negative relationships between item-sets.
- 7) G3PARM [19]: grammar-guided genetic programming algorithm for mining different types of association rules by means of a predefined grammar.
- 8) MOEA_Ghosh [12]: multi-objective genetic algorithm that extracts useful and interesting association rules. It is based on three measures: comprehensibility, interestingness and accuracy.
- 9) MOPNAR [21]: multi-objective evolutionary algorithm that mines positive and negative quantitative association rules. It looks for a good trade-off between comprehensibility, lift and performance (product of support and certainty factor).
- 10) MODENAR [4]: multi-objective differential evolutionary algorithm based on the proposed algorithm in [3]. It weights four quality measures: support, confidence, comprehensibility and amplitude of the attributes.
- 11) ARMMGA [29]: multi-objective evolutionary algorithm based on EARMGA [37]. It looks for a good trade-off between support and confidence.
- 12) NSGA-G3P [20]: multi-objective version of the G3PARM [19] algorithm. It is based on the well-known NSGA-II multi-objective algorithm [7].
- 13) SPEA-G3P [36]: multi-objective version of the G3PARM [19] algorithm. It is based on the well-known SPEA multi-objective algorithm [36].
- 14) QAR-CIP-NSGA-II [22]: multi-objective evolutionary algorithm that extends the well-known NSGA-II algorithm [7]. It performs an evolutionary learning of the intervals of continuous attributes.

All the configurations are those provided by the original authors. A summarizing table could be found in the supplementary material.

4.6. Quality evaluation on sequential algorithms

The aim of this study is to analyze the quality of the solutions obtained by G3P-LSC. First, our proposal is compared to exhaustive search algorithms so only datasets comprising nominal attributes are considered. No minimum quality threshold has been considered so any solution present in the dataset is obtained. In this regard, the best solution found by Apriori or Eclat represents the best possible solution within the dataset since all the existing rules are discovered by these algorithms. Any solution is ranked by the aforementioned fitness function ($F(R) = support(R) \times confidence(R) \times leverage(R)$) and the top twenty rules are analyzed. Similarly, the G3P-LSC algorithm returns the best twenty discovered rules along its evolutionary process, so the aim is to check whether these solutions are good enough.

Table 2 shows the results, where the average fitness function represents the obtained average by the 20 top rules, whereas the maximum fitness function states for the best value $F(R)$ of any association rule R within the set of discovered rules. Finally, *Ratio time* means how many times G3P-LSC is faster than Apriori and Eclat, and it is calculated as the runtime of Apriori (and Eclat) divided by the runtime required by G3P-LSC. Analyzing the results shown in Table 2, it is obtained that G3P-LSC discovers the best rule (maximum fitness function value) in all the datasets, so the proposed algorithm is able to converge to the global optimum. Additionally, if the set of top rules is analyzed, it is discovered that G3P-LSC obtains really promising results since the average value of its resulting set of rules is close to the global optimum (remember that the set of top rules for Apriori and Eclat includes the best rules within each dataset).

Table 2: Comparative of efficiency and effectiveness among Apriori, Eclat and G3P-LSC

Dataset	Algorithm	Fitness function		Ratio time
		Average	Maximum	
Tic-Tac-Toe	G3P-LSC	0.006	0.021	-
	Apriori	0.006	0.021	28.390
	Eclat			32.615
Flare	G3P-LSC	0.066	0.066	-
	Apriori	0.066	0.066	19.910
	Eclat			22.270
Car	G3P-LSC	0.008	0.033	-
	Apriori	0.008	0.033	1.4
	Eclat			1.29
Chess	G3P-LSC	0.095	0.146	-
	Apriori	0.141	0.146	158.22
	Eclat			142.45
	G3P-LSC	0.117	0.121	-

Mushroom	Apriori	0.118	0.121	75.95
	Eclat			61.47
Nursery	G3P-LSC	0.074	0.017	-
	Apriori			18.46
	Eclat			22.41
ChessKrkP	G3P-LSC	0.001	0.004	-
	Apriori			19.708
	Eclat			31.12
Connect4	G3P-LSC	0.067	0.074	-
	Apriori			398.84
	Eclat			230.27
USCensus1990	G3P-LSC	0.123	0.137	-
	Apriori			1,230,045
	Eclat			1,680,098

Finally, the average values for each of the quality measures (support, confidence and leverage) that form the fitness function are also illustrated in Table 3. As it is shown, the results obtained by G3P-LSC are quite similar to those obtained by exhaustive search approaches.

Table 3: Comparative of the single values obtained within the fitness function Apriori, Eclat and G3P-LSC

Dataset	Algorithm	Fitness function		
		Support	Confidence	Leverage
Tic-Tac-Toe	G3P-LSC	0.235	0.600	0.047
	Apriori	0.235	0.600	0.047
	Eclat			
Flare	G3P-LSC	0.310	1.000	0.214
	Apriori	0.309	1.000	0.213
	Eclat			
Car	G3P-LSC	0.173	0.884	0.048
	Apriori	0.173	0.885	0.048
	Eclat			
Chess	G3P-LSC	0.589	0.978	0.165
	Apriori	0.611	1.000	0.107
	Eclat			
Mushroom	G3P-LSC	0.559	0.979	0.213
	Apriori	0.556	1.000	0.213
	Eclat			
Nursery	G3P-LSC	0.167	0.849	0.108
	Apriori	0.018	0.900	0.107
	Eclat			
ChessKrkP	G3P-LSC	0.087	0.512	0.032
	Apriori	0.082	0.546	0.032
	Eclat			
Connect4	G3P-LSC	0.646	0.983	0.108
	Apriori	0.641	1.000	0.116
	Eclat			
USCensus1990	G3P-LSC	0.654	0.982	0.195
	Apriori	0.734	0.997	0.187
	Eclat			

To prove that there is no statistically significant difference in the average fitness function values, a Wilcoxon signed rank test has been used. A p -value of 0.1814 has been obtained, so it possible to assert that, at a significance level of $\alpha = 0.01$, there is no significant difference between exhaustive search al-

gorithms and G3P-LSC on the average set of solutions. It is interesting to note that the global optimum was attained in all datasets (see Table 2) so the convergence to the global optimum is guaranteed. Finally, according to the runtime, our proposal performs better than Apriori and Eclat, and this performance is even better when large datasets are considered (see USCensus1990).

Once it is demonstrated that G3P-LSC converges well to the global optimum, a comparative study among different EAs have been performed by considering continuous and discrete attributes. In this experimental study, all the described datasets in Table 1 have been used, and each EA has been run 10 times for each dataset. Note that EAs are non-deterministic algorithms so the results shown are the obtained average results for these executions. Table 4 shows the obtained average ranking for different quality measures after running all the algorithms on all the datasets. It should be noted that these algorithms were run on their original versions, so each one optimizes its own fitness function. Due to space limitation only the ranking table has been illustrated in this paper, although the whole set of results can be checked in the supplementary material. Different quality measures have been selected to quantify the quality of the rules. In Table 4, each column represents a different EA, whereas each row is used to represent a different quality measure.

As it is illustrated in Table 4, G3P-LSC obtains the best results in Leverage, NetConf, Pearson and Laplace quality measures. If we focus on the Confidence quality measure, the ranking values determine that G3P-LSC does not behave well for this specific quality measure. However, if we analyze the confidence values for each run 5 it is obtained that all the values are above 0.9 and, in some specific datasets, the confidence values obtained are greater than 0.95. Thus, it is possible to assert that the results of G3P-LSC for this quality measure are alluring enough. In the same way, the ranking values for support seem to be worse than for other algorithms. It is worth noting that extremely high support values, as it is obtained by some algorithms, imply misleading rules according to some of the quality measures. In fact, maximum support values, i.e., 1.00, imply misleading rules since they do not provide unknown information about the dataset. As it is demonstrated by analyzing the rankings (see Table 4), the rules obtained by G3P-LSC present values for the interestingness measures that are better than or similar to the obtained by the analyzed algorithms.

In order to prove if the differences obtained by the ranking analysis are statically significant, a Friedman test [10, 11] for each quality measure is carried out. In this regard, we consider the null hypothesis H_0 that all algorithms equally perform. Table 5 shows whether the null hypothesis is rejected, considering a criti-

cal interval $F_{0.01,29,348} = 2.23$. According to the results of the Friedman statistical test, it is possible to assert that the null hypothesis is rejected for each of the analyzed quality measures since F_F obtains the following values: 23.08 for CF; 47.06 for Confidence; 58.03 for Cosine; 35.57 for Gain; 85.59 for Laplace;

Table 4: Obtained ranking after executing 10 times each algorithm in the whole set of real-world datasets. Different well-known quality measures have been calculated. Values in bold typeface represent the best ranking for each quality measure

Measure	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
CF	4.167	9.083	5.283	11.267	9.750	5.750	9.117	3.400	6.633	8.500	7.817	7.400	2.833
Confidence	8.100	8.750	5.950	1.883	9.933	2.900	10.967	9.167	10.367	6.883	2.883	4.850	8.366
Cosine	4.467	5.733	9.733	8.900	11.383	3.600	7.867	12.167	9.233	4.750	1.833	4.300	7.033
Gain	2.933	6.750	6.717	9.850	8.783	9.400	5.383	8.267	3.367	8.417	10.517	9.117	1.500
Laplace	1.267	6.663	9.183	7.433	11.300	3.833	8.300	12.300	9.033	5.483	2.550	4.200	9.433
LeastContradiction	4.933	5.367	8.717	7.967	9.867	3.350	11.167	10.800	11.133	5.150	1.717	3.900	6.933
Leverage	1.067	5.383	8.100	9.067	10.117	8.617	6.800	8.467	5.417	6.800	9.783	8.483	2.900
Lift	3.950	7.133	6.983	8.300	9.867	8.650	3.767	12.167	2.833	7.633	9.600	8.383	1.733
NetConf	2.033	6.200	7.950	10.617	9.617	9.800	7.617	2.633	5.900	7.533	10.150	8.817	2.133
Pearson	1.700	6.333	8.233	11.200	10.500	7.017	6.233	10.017	5.700	6.900	7.000	8.200	1.966
Support	6.250	5.617	9.983	8.000	11.367	3.117	7.933	9.067	9.600	4.817	1.617	3.267	10.366
YulesQ	2.567	6.617	7.533	10.683	9.983	8.233	8.483	2.217	6.167	7.117	9.600	8.433	3.366
Zhang	4.283	8.833	5.250	10.683	9.500	6.050	8.067	10.667	5.633	8.483	7.133	4.283	2.133

- (1) G3P-LSC (4) EARMGA (7) MOEA_Ghosh (10) ARMMGA (12) SPEA-G3P
(2) GAR (5) Alatasetal (8) MOPNAR (11) NSGA-G3P (13) QAR-CIP-NSGA-
(3) GENAR (6) G3PARM (9) MODENAR

Table 5: F_F Friedman's values to test whether the null hypothesis H_0 that all the algorithms equally behave for each quality measure can be rejected or not, considering the critical interval $F_{0.01,29,348} = 2.23$

Measure	F_F	H_0
CF	23.08	Rejected
Confidence	47.06	Rejected
Cosine	58.03	Rejected
Gain	35.57	Rejected
Laplace	85.59	Rejected
LeastContradiction	59.26	Rejected
Leverage	26.62	Rejected
Lift	49.87	Rejected
NetConf	46.21	Rejected
Pearson	33.91	Rejected
Support	53.80	Rejected
YulesQ	31.81	Rejected
Zhang	27.18	Rejected

26.62 for Leverage; 49.87 for Lift; 46.21 for NetConf; 33.91 for Pearson; 53.80 for Support; 31.81 for YulesQ; and 27.18 for Zhang. Thus, none of the F_F values belongs to the critical interval $F_{0.01,29,348} = 2.23$, so it is not possible to statistically assert that all algorithms equally behave. In this regard, a Bonferroni-Dunn test [8] has been considered to determine the statistical differences among the algo-

rithms under study. According to the Bonferroni-Dunn test, the obtained critical difference is 3.36 for $\alpha = 0.01$. In this regard, Table 6 shows the number of quality measures in which G3P-LSC is better than other algorithms (# Wins), worse (# Losses) or there is no significant difference between them (# Draws). Alatasetal is the algorithm most different with respect to G3P-LSC, since our proposal has obtained 12 wins of a total of 13. The highest number of losses is obtained with NSGA-G3P. Although, G3P-LSC loses 2 times with NSGA-G3P, our proposal obtains 7 wins, thus G3P-LSC behaves better than the rest. The most similar algorithm is QAR-CIP-NSGA-II, obtaining 11 draws. However, G3P-LSC obtains 2 wins and no losses.

As a result, G3P-LSC behaves statistically better for more quality measures and in very few cases the quality measures obtained by other algorithms behave statistically better than G3P-LSC.

Table 6: Statistical differences among distinct measures and algorithms according to the Bonferroni-Dunn test. G3P-LSC achieves the highest number of significant differences and it does not almost loss. QAR-CIP-NSGA-II is the algorithm more similar to our proposal

G3P-LSC vs	#Wins	#Losses	#Draws
GAR	8	0	5

GENAR	9	0	4
EARMGA	10	1	2
Alatasetal	12	0	1
G3PARM	7	1	5
MOEA_Ghosh	9	0	4
MOPNAR	8	0	5

MODENAR	7	0	6
ARMMGA	9	0	4
NSGA-G3P	7	2	4
SPEA-G3P	3	1	9
QAR-CIP-NSGA-II	2	0	11

Table 7: Rules obtained by G3P-LSC when different grammars are considered on the *stock* dataset.

Type of grammar	Rule	Support	Confidence	Leverage
Original	IF <i>Company1</i> IN [17.38, 40.73] THEN <i>Company5</i> IN [59.41, 93.54]	0.51	0.98	0.23
Positive and negative rules	IF <i>Company1</i> IN [42.023, 60.05] AND <i>Company3</i> IN [34.83, 56.31] THEN <i>Company4</i> NOT IN [28.29, 57.96]	0.54	0.99	0.23
Only one item into the antecedent and consequent	IF <i>Company4</i> IN [58.67, 93.73] THEN <i>Company1</i> IN [42.51, 60.16]	0.54	0.98	0.22

4.7. Different grammars for G3P-LSC

The aim of this section is to demonstrate how the grammar can be modified to achieve different results (see Table 7). All these alluring quantitative association rules have been obtained on the *Stock* dataset and the proposed G3P-LSC algorithm with different grammars each time. Taking the original grammar (see Figure 2), the following rule is obtained: **IF** *Company1* IN [30.46, 58.83] **THEN** *Company10* IN [42.67, 61.01]. As shown, this rule describes continuous patterns by means of enclosed values (lower and upper bounds). One of the main advantages of using grammars is the ability to introduce syntax constraints and to apply external knowledge to the mining process. In this regard, for some specific domains, it is possible to require specific rules, determining the position of an attribute or even the range in which is defined.

$G = (\Sigma N, \Sigma T, P, S)$ with:
 $S = \text{Rule}$
 $\Sigma N = \{\text{Rule, Antecedent, Consequent, Condition, Nominal, Numerical, NegativeNominal, NegativeNumerical}\}$
 $\Sigma T = \{\text{'name', '=', '!=', 'value', 'IN', 'NOT IN', 'Min_value', 'Max_value'}\}$
 $P = \{$
 $\text{Rule} := \text{Antecedent, Consequent};$
 $\text{Antecedent} := \text{Condition} \mid \text{Condition, Antecedent};$
 $\text{Consequent} := \text{Condition} \mid \text{Condition, Consequent};$
 $\text{Condition} := \text{Numerical} \mid \text{Nominal} \mid$
 $\text{NegativeNumerical} \mid \text{NegativeNominal};$
 $\text{Numerical} := \text{'name' 'IN' 'Min_value', 'Max_value'};$
 $\text{NegativeNumerical} := \text{'name' 'NOT IN' 'Min_value', 'Max_value'};$
 $\text{Nominal} := \text{'name' '=' 'value'};$
 $\text{NegativeNominal} := \text{'name' '!=' 'value'};$
 $\}$

Figure 5. Context-free grammar modified to obtain negative items

As a matter of example, Table 7 shows some obtained rules for the *stocks* dataset when different restrictions are defined. For instance, it is possible to include not only positive but also negative associations, so a simple change in the grammar (see Figure 5) allows to obtain rules as the one depicted in the second row. The consequent of this rule denotes that *Company4* cannot include a value in the range [28.29, 57.96].

Finally, the third example is obtained from a grammar (see Figure 6) that allows to obtain only rules having a single item in both the antecedent and consequent. Considering this grammar, an example of rule obtained from the *stocks* dataset is **IF** *Company4* IN [58.67, 93.73] **THEN** *Company1* IN [42.51, 60.16].

$G = (\Sigma N, \Sigma T, P, S)$ with:

$S = \text{Rule}$
 $\Sigma N = \{\text{Rule, Antecedent, Consequent, Nominal, Numerical}\}$
 $\Sigma T = \{\text{'name', '=', 'value', 'IN', 'Min_value', 'Max_value'}\}$
 $P = \{$
 $\text{Rule} := \text{Antecedent, Consequent};$
 $\text{Antecedent} := \text{Numerical} \mid \text{Nominal};$
 $\text{Consequent} := \text{Numerical} \mid \text{Nominal};$
 $\text{Numerical} := \text{'name' 'IN' 'Min_value', 'Max_value'};$
 $\text{Nominal} := \text{'name' '=' 'value'};$
 $\}$

Figure 6. Context-free grammar modified to obtain rules having a single item in both the antecedent and consequent

4.8. Scalability on parallel implementations

The aim of this study is to analyze the performance of the different proposed implementations when the number of both attributes and instances increases. In the first part of this study, a set of synthetic datasets has been used that have been properly created to analyze how the number of both instances and attributes

affect to the algorithms' performance. Finally, note that all these algorithms obtain the same results being the followed approach to parallelize the unique difference among them. Analyzing Figure 7, the results illustrate that the baseline implementation of G3P-LSC is the most efficient when a small number of instances is bore in mind. However, when the number of instances continues growing (up to $6 \cdot 10^5$), its performance begins to decrease starting to be needed some kind of parallelization to improve the runtime. Although Big Data architectures provide a way to improve runtime using distributed computing, they are totally unsuitable for small datasets as it is shown. It is due to the expensive cost of communication and scheduling of the platform, in cases where the data are smaller this cost is not justified. Thus, other kind of parallelization is needed. RMI has been considered as a simple way of parallelization achieving excellent results when the number of instances ranges from $6 \cdot 10^5$ to $9 \cdot 10^5$, obtaining better results than sequential approach in this range of examples. Up to $6 \cdot 10^5$ examples, Sparks' performance is almost the same as RMI, while the performance of Hadoop is far from being the best. It demonstrates that the sequential baseline approach and RMI version are appropriate to be used on small datasets unlike Big Data implementations, which are useful for truly large datasets. Hadoop is the worst since it has to read the whole dataset from disk in each generation and write on disk each communication among mappers and reducers, hampering the overall performance. Continuing the experimental study, the number of instances is increased from $9 \cdot 10^5$ to $1 \cdot 10^9$. Figure 8 illustrates that a sequential implementation is meaningless when datasets with millions of instances are considered ant its runtime exponentially increases. This baseline implementation is unfeasible when file sizes of GB are considered since it takes more than 300 days on mining a dataset with $1 \cdot 10^9$ instances.

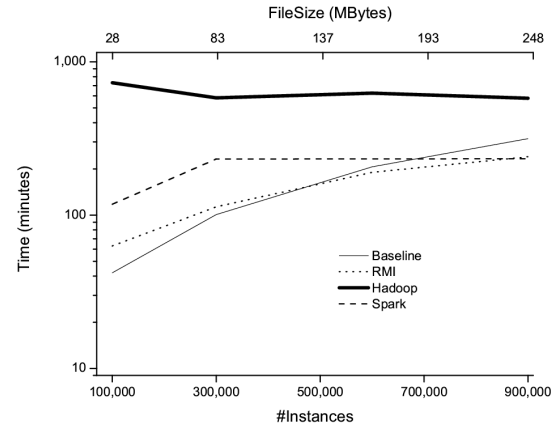


Figure 7. Runtime of different implementations when they are run on datasets comprising 48 attributes and a number of instances that varies from $1 \cdot 10^5$ to $9 \cdot 10^5$. The file size varies from 28 MB to 248 MB.

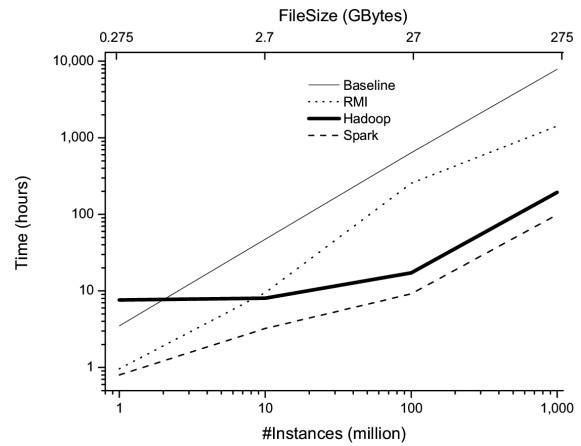


Figure 8. Runtime of different implementations when they are run on datasets comprising 48 attributes and a number of instances that varies from $1 \cdot 10^6$ to $1 \cdot 10^9$. The file size varies from 275 MB to 275 GB.

Although RMI obtains the same results using only a 15% of sequential runtime, it is not enough when huge datasets are considered. Spark's implementation obtains the best results since the datasets could be stored in main memory. On the other hand, the implementation based on Hadoop obtains worse results than Spark. Again, the number of instances is increased from $1 \cdot 10^9$ to $3 \cdot 10^9$, with a range of file size from 275 GB to 804 GB. Figure 9 only shows Hadoop and Spark since both have proved to obtain the best performance in

Big Data. It illustrates that Spark achieves a better performance until the file size of the dataset could not be stored in main memory (≈ 790 GB).

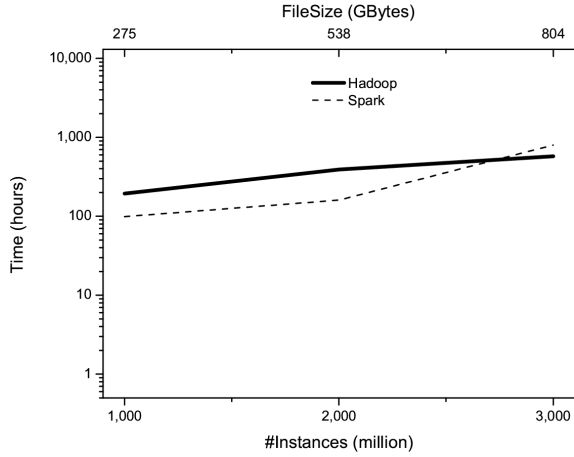


Figure 9. Runtime of different implementations when they are run on datasets comprising $1 \cdot 10^9$ to $3 \cdot 10^9$ instances and a number of attributes of 48. The file size varies from 275 GB to 804 GB.

At this point, Spark begins to use both disk and memory as cache system (hardware limitations), thus its behavior is almost the same as Hadoop having to read almost the whole dataset from disk in each generation. Thus, Spark is appropriate to handle an extremely large number of examples when data can be stored in main memory. On the other hand, Hadoop is quite appropriate to handle huge datasets that cannot be stored in main memory.

Additionally, an analysis has been carried out to study how the number of attributes affects. In this regard, Figure 10 illustrates how the number of attributes linearly affects the global performance of the algorithms. Note that the higher the number of attributes, the higher the search space, so the convergence time increases linearly with the number of attributes.

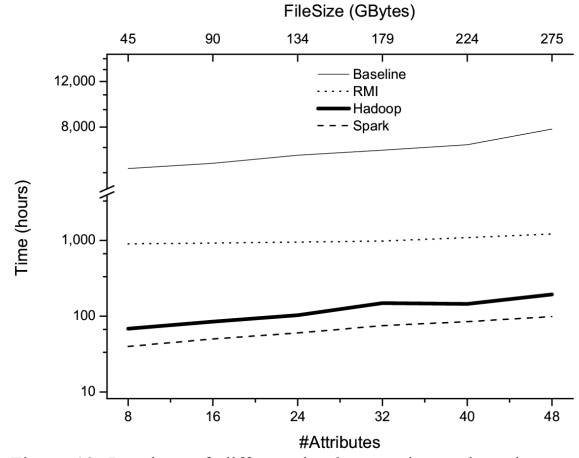


Figure 10. Runtime of different implementations when they are run on datasets comprising $1 \cdot 10^9$ instances and a number of attributes that varies from 8 to 48. The file size varies from 45 GB to 275 GB.

In a third study, it has been considered interesting to note how relevant is the cluster capacity in the decision on which implementation should be used. Figure 11 illustrates how the performance is affected by the cluster capabilities. RMI is the implementation less improved when the number of nodes increases since each node has to evaluate the whole dataset. However, Apache Hadoop and Apache Spark benefit because a greater number of nodes means a greater level of parallelism, since each node evaluates a subset of the dataset.

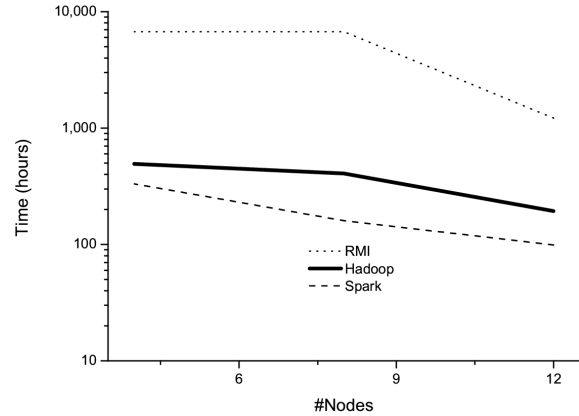


Figure 11. Runtime of parallel approaches when the number of nodes decreases. A dataset comprising $1 \cdot 10^9$ instances with 48 attributes and a file size of 275 GB is used.

Table 8: Runtime in hours required when large real-world datasets are used by running Spark & Hadoop implementation. As all these datasets could be stored in main memory.

Dataset	Hadoop	Spark
---------	--------	-------

Epsilon	8.37	1.42
Poker	7.65	0.01
US Census 1990	7.50	0.83
Susy	14.31	8.20
Heap Mass	33.33	14.25
Higgs	24.46	12.33
Protein Structure Prediction	37.76	18.44

Once it has been demonstrated that implementations based on Spark and Hadoop obtain alluring runtime on synthetic datasets, they have been run on real-world datasets. No comparison with sequential approaches have been considered since it was previously demonstrated that these algorithms cannot be run efficiently. Table 8 shows the obtained results for a set of Big Data real-world datasets. As it could be appreciated, the behavior is the same as was shown in previous studies. Spark achieves a better performance since the whole dataset could be stored in main memory. It achieves the same results as Hadoop but using only a half amount of the time, and in some cases Spark only needs a 13% of the time required by Hadoop. The issue of Hadoop is also shown, needing almost the same time to process several GB (Epsilon datasets) than several MB (Poker dataset). It is due to that much of the time is used to orchestrate the platform and no to our main computations, thus, Hadoop always needs a quantity of time to coordinate the platform independently of the size of the data. However, Spark does not require this large time in orchestration.

5. Concluding remarks

In this work, a genetic programming algorithm based on grammars for mining association rules in Big Data has been proposed, known as G3P-LSC. The novelty of this work is that it proposed a new evolutionary approach which is able to be run in a distributed way enabling the use of parallel paradigms such as MapReduce. Our proposal provides a reduced set of rules, easy to understand, and with a good level in many quality measures. The main aim of G3P-LSC is to optimize a set of well-known quality measures in the association rule mining field.

Currently, due to the increasing interest in data storage, a unique and universal implementation of a single algorithm is unfeasible and different adaptations should be done depending on the data size. In this sense, the proposed algorithm has been imple-

mented on different architectures including a sequential approach, RMI and MapReduce. It should be noted that all the implementations return exactly the same results and the unique change is the architecture.

When comparing the obtained results with other 14 algorithms and using more than 75 datasets, it is obtained that the proposed G3P-LSC algorithm mines rules with better values for interesting measures and few attributes, providing the user with high quality rules. As a grammar is used, the user could even specify which set of attributes must appear in the proposed solutions, allowing to constrain the search space only for rules of his interest. Finally, the proposed approach presents a good computational cost in all datasets and good scalability when the size of the problem increases.

5.1. Future work

Finally, as a future work some parallel searches could be considered to be adapted to the extraction of association rules in Big Data. Its application is not trivial and these paradigms have not been studied yet in the association rule mining field and further research is encouraged. Next, each of the most well-known parallel models is analyzed [34]. Island models or even cellular EAs could be studied to mine association rules in a parallel way. Indeed, hybrid models could improve the convergence of the previous models, where some relaxations of the original methods could further improve the diversity.

Acknowledgment

This work was Supported by the Spanish Ministry of Economy and Competitiveness under the project TIN2014-55252-P, and FEDER funds. This work is also supported by the *Juan de la Cierva Formacion* post-doctoral grant, reference FJCI-2015-23560.

References

- [1] Aggarwal C. C. and Han J. Frequent Pattern Mining. Springer International Publishing, 2014.
- [2] Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. *SIGMOD Rec.* 1993: 22(2):207–216.
- [3] Alatas B. and Akin E. An efficient genetic algorithm for automated mining of both positive and negative quantitative association rules. *Soft Computing*. 2006:10(3):230–237.

- [4] Alatas B., Akin E. and Karci A. MODENAR: Multi-objective Differential Evolution Algorithm for Mining Numeric Association Rules. *Applied Soft Computing*. 2008: 8:646–656.
- [5] Cano A., Luna J. M. and Ventura S. High performance evaluation of evolutionary-mined association rules on GPUs. *The Journal of Supercomputing*. 2013: 66(3): 1438–1461.
- [6] Dean, J. and Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM - 50th anniversary issue: 1958 – 2008*. 2008: 51(1):107–113.
- [7] Deb K., Pratap A., Agarwal S. and Meyarivan T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*. 2002: 6(2): 182–197.
- [8] Demsar J. Statistical comparisons of classifiers over multiple data sets, *Journal of Machine Learning Research*. 2005: 7: 1–30.
- [9] García S., Fernández A., Luengo J. and Herrera F.. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences. Special Issue on Intelligent Distributed Information Systems*. 2010: 180(10): 2044 – 2064.
- [10] García S. and Herrera F. An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. *Journal of Machine Learning Research*. 2008: 9: 2677–2694.
- [11] García S., Molina D., Lozano M. and Herrera F. A study on the use of non-parametric tests for analyzing the evolutionary algorithms’ behaviour: A case study. *Journal of Heuristics*. 2009: 15(6): 617–644.
- [12] Ghosh A. and Nath B. Multi-objective Rule Mining Using Genetic Algorithms. *Information Science*. 2004: 163(1-3): 123–133.
- [13] Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2011.
- [14] Kyriklidis C. and Dounias G. Evolutionary computation for resource leveling optimization in project management. *Integrated Computer-Aided Engineering*. 2016: 23(2): 173–184.
- [15] Lam C. Hadoop in Action. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.
- [16] Li T. and Li X. Novel alarm correlation analysis system based on association rules mining in telecommunication networks. *Information Sciences*. 2010: 180(16): 2960–2978.
- [17] Luna J. M., Cano A., Pechenizkiy M. and Ventura S. Speeding-Up Association Rule Mining with Inverted Index Compression. *IEEE Transactions on Cybernetics*. 2016: 46(12): 3059–3072.
- [18] Luna J. M., Romero C., Romero J. R. and Ventura S. An Evolutionary Algorithm for the Discovery of Rare Class Association Rules in Learning Management Systems. *Applied Intelligence*. 2015: 42(3): 501–513.
- [19] Luna J. M., Romero J. R. and Ventura S. Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules. *Knowledge and Information Systems*. 2012: 32(1): 53–76.
- [20] Luna J. M., Romero J. R. and Ventura, S. Grammar-based multi-objective algorithms for mining association rules. *Data & Knowledge Engineering*. 2013: 86: 19–37.
- [21] Martín D., Rosete A., Alcalá-Fdez J. and Herrera, F. A new multiobjective evolutionary algorithm for mining a reduced set of interesting positive and negative quantitative association rules. *IEEE Transactions on Evolutionary Computation*. 2014: 18(1): 54–69.
- [22] Martín D., Rosete A., Alcalá-Fdez J. and Herrera F. QAR-CIP-NSGA-II: A new multi-objective evolutionary algorithm to mine quantitative association rules. *Information Sciences*. 2014: 258: 1–28.
- [23] Mata J., Alvarez J. L. and Riquelme J. C. Mining numeric association rules with genetic algorithms. In *Proceedings of the 5th International Conference on Artificial Neural Networks and Genetic Algorithms*, ICANNGA 2001. Taipei, Taiwan. 2001: 264–267.
- [24] Mata J., Alvarez J. L. and Riquelme J. C. Discovering numeric association rules via evolutionary algorithm. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD 2002. Taipei, Taiwan. 2002: 40–51.
- [25] Mencá R., Sierra M. R., Mencá C. and Varela R. Genetic algorithms for the scheduling problem with arbitrary precedence relations and skilled operators. *Integrated Computer-Aided Engineering*. 2016: 23(3): 269–285.
- [26] Ordoñez N., Ezquerro C. and Santana C. Constraining and Summarizing Association Rules in Medical Data. *Knowledge and Information Systems*. 2006: 9(3): 259–283.
- [27] Pan L., He C., Tian Y., Su Y. and Zhang X. A region division based diversity maintaining approach for many-objective optimization. *Integrated Computer-Aided Engineering*. 2017: 24(3): 1–18.
- [28] Piatetsky-Shapiro G. Discovery, analysis and presentation of strong rules. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*. AAAI Press. 1991: 229–248.
- [29] Qodmanan H. R., Nasiri M. and Minaei-Bidgoli B. Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence. *Expert Systems with Applications*. 2011: 38: 288–298.
- [30] Ramírez-Gallego S., Fernández A., García S., Chen M. and Herrera F. Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. *Information Fusion*. 2018: 42: 51–61.
- [31] Rostami S. and Neri F. Covariance matrix adaptation pareto archived evolution strategy with hypervolume-sorted adaptive grid algorithm. *Integrated Computer-Aided Engineering*. 2016: 23(4): 313–329.
- [32] Rostami S., Neri F. and Epitropakis M. Progressive preference articulation for decision making in multi-objective optimisation problems. *Integrated Computer-Aided Engineering*. 2017: 24(4): 315–335.
- [33] Sabar N. R., Abawajy J. and Yearwood J. Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems. *IEEE Transactions on Evolutionary Computation*. 2017: 21(2): 315–327.
- [34] Sudholt D. Parallel evolutionary algorithms. In *Springer Handbook of Computational Intelligence*. 2015: 929–959.
- [35] Tan P. and Kumar V. Interestingness Measures for Association Patterns: A Perspective. In *Proceedings of the Workshop on Postprocessing in Machine Learning and Data Mining*, KDD ’00. New York, USA, 2000.
- [36] Ventura S. and Luna J. M. Pattern Mining with Evolutionary Algorithms. Springer International Publishing, 2016.
- [37] Yan X., Zhang C. and Zhang S. Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. *Expert Systems with Applications*. 2009: 36: 3066–3076.
- [38] Zaharia M., Chowdhury M., Franklin M. J., Shenker S. and 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.
- [39] Zaki M. J. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*. 2000: 12(3): 372–390.
- [40] Zhang C. and Zhang S. Association rule mining: models and algorithms. Springer Berlin / Heidelberg, 2002.

Publications in conferences

- J.M. Luna, F. Padillo and S. Ventura, "Associative Classification in Big Data through a G3P Approach", 4th International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS, 2019, po. 94-102. DOI: 10.5220/0007688400940102
- F. Padillo, J. M. Luna and S. Ventura, "An evolutionary algorithm for mining rare association rules: A Big Data approach", 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 2007-2014. DOI: 10.1109/CEC.2017.7969547