Efficient Parallel Associative Classification Based on Rules Memoization

Michel Pires[1,3](✉), Nicollas Silva[3], Leonardo Rocha[2], Wagner Meira[3], and Renato Ferreira[3]

[1] Centro Federal de Educação Tecnológica de Minas Gerais, Divinópolis, MG, Brazil
michel@cefetmg.br

[2] Universidade Federal de São João del-Rei, São João del-Rei, MG, Brazil
lcrocha@ufsj.edu.br

[3] Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil
{michelpires,ncsilvaa,meira,renato}@dcc.ufmg.br

Abstract. Associative classification refers to a class of algorithms that is very efficient in classification problems. Data in such domain are multidimensional, with data instances represented as points of a fixed-length attribute space, and are exploited from two large sets: training and testing datasets. Models, known as classifiers, are mined in the training set by class association rules and are used in eager and lazy strategies for labeling test data instances. Because test data instances are independent and evaluated by sophisticated and high costly computations, an expressive overlap among similar data instances may be introduced. To overcome such drawback, we propose a parallel and high-performance associative classification based on a lazy strategy, which partial computations of similar data instances are cached and shared efficiently. In this sense, a PageRank-driven similarity metric is introduced to reorder computations by affinity, improving frequent-demanded association rules memoization in typical cache strategies. The experiments results show that our similarity-based metric maximizes the reuse of rules cached and, consequently, improve application performance, with gains up to 60% in execution time and 40% higher cache hit rate, mainly in limited cache space conditions.

Keywords: Parallel associative classification · Memoization · Class association rules

1 Introduction

Classification is an important task in data mining, and the associative classification (ACs) its branch that describes a class of algorithms based on two well-known mining paradigms, pattern classification and association rules mining [4]. Pattern classification assigns a class label to a given data input instance by a specific model named classifier, and association rules mining is the task that discovers correlations or other relationships in large datasets to turned classification process accurate [5].

Several efforts have shown that ACs are capable of building efficient and accurate classification systems, by three typical steps. Firstly, class association rules (CARs) are mined from attribute space through a training dataset. Such CARs are weighted according to a given function, and support and confidence thresholds pruning weak rules to make up a frequent item set. Posteriorly, frequent items are assumed as part of a classifier into eager or lazy approaches to associate classes for unlabeled input data instances of a test data set [15].

In order to achieve accurate classifiers, eager strategies look at frequent items by a global searching into attribute space. An expressive number of useless rules may be introduced due to weak similarity among part of generated rules and unlabeled data attribute space. To overcome such problem, lazy approaches, as the lazy associative classification (LAC) [21], investigates the attribute space by local searches during the classification time, whenever a novel unlabeled data input is provided. Thus, suitable models are afforded for each data instance while a higher similarity between classifiers and data attribute space is ensured [22].

The local models look at into attribute space by a large variability of descriptions or subproblems, which it can afford a better global approximation of target function [21,22], which leads to a specific advantage in relation to traditional classification models, such as decision trees and rule induction [5,16]. Because local models are designed independently for each unlabeled data instance, an expressive computation overlap between similar classifiers may be introduced. This is because different classifiers overlap on the attribute space, which leads to a costly rework under a significant amount of similar CARs. Indeed, an opportunity to employ typical cache strategies to overcome such drawback, since CARs can be efficiently cached. Further, because classifiers are independent, the classification process can be exploited within parallel execution models.

In this paper, we propose a parallel and high-performance LAC execution model based on MapReduce concepts in that a list of unlabeled data instances is orchestrated toward an effective CARs memoization. We propose modeling unlabeled data instances attribute space as a weighted graph in which vertices are drawn as attributes and edges as correlations among them. The weight in each edge exposes the number of times that each relationship between two vertices occurs. Because the local models are designed based on the attribute space, we evaluate the relationship of different data instances by a similarity metric based on PageRank. Basically, we use PageRank to discover relevant attributes and, the eigenvectors and eigenvalues yielded by it for clustering data instances in an attempt to ensure high cohesion and low coupling among CARs demanded by classifiers. PageRank is used to discover correlated Web pages with a high-accuracy, we use such accuracy to identify data instances with high-similar attribute subsets keeping them close to each other, which translates in local models with the high relationship of CARs. In our experiments, the results show that our similarity-based metric maximizes the number of rules reused in the cache and, consequently, improve application performance, with gains up to 60% in execution time and 40% in the cache hit rate, mainly in limited cache space conditions.

To summarize, the main contributions of this paper are: *(i) the proposal of a parallel approach to optimize the data analysis process on LAC for recurrent and high-cost computations; (ii) a powerful cache implementation in which a PageRank-driven similarity metric is employed to deal with computation affinities and CARs demand-relationship; (iii) a thorough evaluation based on different datasets to explain the potential of CAR memoization, as well as execution time improvements and cache hit rate.*

Roadmap. This paper is organized as follows. Section 2 introduces a background of ACs, LAC, MapReduce, and related works. Section 3 describes our approach in details, and as LAC job instance is executed in one of the most important open-source distributed general-purpose engine based on MapReduce, the Spark. Section 4 presents experiments setup and discuss results around the number of generated rules, execution time and cache hit rates. Finally, in Sect. 5, we introduce the conclusions.

2 Background

In this section, we introduce basic definitions that are necessary to understand the associative classification problem, the lazy associative classification approach and MapReduce programming model over the Spark engine. We also show related works to the proposed theme and its benefits for the classification process.

2.1 Associative Classification Problem

AC is a data mining branch in which useful patterns are discovered in large data sets by exploiting class association rules. The first concepts correlated with such domain were introduced in the paper "*Mining association rules between sets of items in large databases*" by [1]. After that, other techniques have been proposed, including emerging patterns methods (CAEP) [8], multiple class association rules (CMAR) [13] and (MCAR) [17], predictive association rules (CPAR) [25], instance centric rule generation (HARMONY) [24]. There are also recent efforts in which parallel and distributed approaches are investigated [5,12,18].

Formally, we explain such approaches denoting training set as Λ and test set as Γ. We address ACs as association rules mining cases in which data instances (also referred to as examples - Λ) are looks at as pairs in the form $\lambda_i = \langle x_i, c_i \rangle$. Each x_i is drawn as a point in a fixed-length attribute space, that is, outlined by an item set $\langle a_1, a_2, \ldots, a_k \rangle$ with a_k as the k^{th} attribute-value mapped in such space. Each c_i is expressed as a value in a discrete and finite set of possibilities $\langle v_1, v_2, \ldots, v_p \rangle$ with $1 \leq i \leq p$ and designates the class to which λ_i belongs. The classification process consists, for the instances of Γ on which c_i is unknown, find a conditional probability distribution $P(c|x)$, mapped of the relationships between points and classes in Λ by a function in a form $F : \chi \rightarrow C$. The performance of a given $f \in F$ is expressed by some accuracy criterion using Γ.

Definition 1: An item in Λ and Γ can be described as a combination of attribute name A_i and value a_i, denoted $\langle (A_i, a_i) \rangle$.

Definition 2: A CAR r, mined in Λ, is denoted as $\chi \rightarrow c_i$ where χ is a itemset (i.e., a set of pairs $\langle (A_1, a_1), \ldots (A_k, a_k) \rangle$) and c_i the i^{th} class in a discrete and finite set of possibilities in C.

Definition 3: The incidence of a r (denoted $I(r)$) is given by the number of data instances in Λ that have as antecedent r, that is:

$$I(r) = \sum_{i=1}^{|\Lambda|} \lambda_i;\ \forall \lambda_i \subseteq \Lambda \mid \lambda_i \in \chi \tag{1}$$

Definition 4: The support threshold of r or $S(r) = S(\chi \rightarrow c_i) = P(c_i | I(r))$ is referenced as:

$$S(r) = \frac{|P|}{|\Lambda|} = \frac{I(r) \rightarrow c_i}{|\Lambda|}. \tag{2}$$

Definition 5: The confidence threshold of r or $E(r) = E(\chi \rightarrow c_i) = P(c_i|I(r))$, is represented as:

$$E(r) = \frac{|P|}{|I(r)|} = \frac{S(\chi \rightarrow c_i)}{S(\chi)} \tag{3}$$

The main task in a CA is to find rules set able of associating classes with unlabeled data instances, that is, for pairs drawn as $\tau_i = \langle x_i, ? \rangle$. In other words, discovering a target function $f \in F$ that express the conditional probability distribution $P(c|x)$ into a higher accuracy function $f(x, c) = \tau$, for each unlabeled data instance $\tau \in \Gamma$.

2.2 Lazy Associative Classification

Introduced by [21], LAC is a demand-driven associative classification that uses local searches in Λ to compose classifiers, whenever an unlabeled data instance $\tau \subseteq \Gamma$ is provided. Therefore, it is assumed that pairs in Λ are in some sense related to pairs in Γ, sampled independently and identically by the same distribution $P(c|x)$. Facts that make LAC capable to afford a better global approximation of the target function F by local models, which leads to greater accuracy than noticed in traditional classification models [19–21]. The classification process performed by LAC is described in the Algorithm 1.

In Algorithm 1, Λ is accommodated into hash tables η_X and η_C. η_X describe the relationship between items and transaction indexes in the dataset. Each item, defined as a key of the hash, is linked to transactions indexes in which the item's pair is addressed, that is, $\langle (A_i, a_i) \rangle \rightarrow \lambda_i\ \forall \lambda_i \subseteq \Lambda \mid \langle (A_i, a_i) \rangle \subseteq \lambda_i$. Similarly, η_C is modeled to express the classes associations.

For each unlabeled data instance τ, a target function $f \in F$ is modeled through a projection of τ in η_X. Only keys in η_X directly related to τ are considered for classification. CARs are mined by the aforementioned Definitions 3, 4 and 5 to extract the relevant rules, named frequent itemset. Thus, each τ is labeled sorting the frequent itemset and finding the higher class threshold in η_C.

Algorithm 1. Lazy Associative Classification [21]

```
input  : $\eta_X$ hash of items
input  : $\eta_C$ hash of classes
input  : $\mathit{\Gamma}$ set of unlabeled data instances
output: $\tau \rightarrow c_i$ for each $\tau \subseteq \mathit{\Gamma}$
for $\tau \in \mathit{\Gamma}$ do
    Given $\eta_\tau$ the projection of $\tau$ in $\eta_X$
    Given $\chi_{\eta_\tau}$ CARs mining in $\eta_\tau$ and submitted to equations 2 and 3
    Sort $\chi_{\eta_\tau}$ by thresholds of each $c_i \subseteq \eta_C \wedge \chi_{\eta_\tau}$
    Get better $c_i \in \eta_C$ and use it to classify $\tau$
end
```

2.3 MapReduce over Spark

MapReduce is a concept for processing a large amount of data in parallel and distributed environments based on a functional programming [7]. Proposed by Google in 2004, its computational flow is based on two core constructions, *map* and *reduce*. The idea behind such construction is providing means to partition data instances of a dataset into independent tasks, while distributing its computations on a cluster, avoiding communications and possible failures, at the same time that ensures an efficient disk usage and partial results arrangements [23].

In a MapReduce execution, a dataset is automatically partitioned into independent subsets and each one processed by an independent machine with a copy of user program. One copy is denoted as a *master* and is used to schedules and handles such subsets for other ones that perform them, named *workers*.

In last years, several efforts have been employed to deal with large datasets over MapReduce concepts. One of such efforts has led to a fast and general engine for large-scale data processing, named Spark [26]. In Spark, the programming model, based on MapReduce, is extended to introduce a data-sharing abstraction denoted as resilient distributed datasets (RDDs), which are fault-tolerant objects distributed across the cluster that can be handled in parallel. In addition, Spark introduces a large range of novel actions and transformation functions, as well as, explicit support for data sharing among computations, such as *accumulator* and *broadcast* variables. Advantages that allows introducing high-performance mining in a wide range of workloads whose composition and size until then required separate engines.

2.4 Related Works

ACs are gaining more and more attention as an ever-increasing amount of data are becoming available about many interesting events of everyday life and, classify such data is an important task of science and engineering. In this sense, the MapReduce is a well-known paradigm addressed for more than one decade that deals with data parallelization in a large number of machines by a series of popular and open-source engines. In recent years, efforts have been directed attention to improving performance for different AC algorithms, many of these based on MapReduce [2,5,9,12,18]. It is also possible to observe a significant trend to improve performance for MapReduce applications by the sophisticated aware-resource usage strategies [11,27] and data schedulers [6,14].

Several researches, e.g., [5,10], show MapReduce applications performance depends on cluster configurations and jobs, as well as, input data. According to [3], an effectively data-scheduling is the major challenges in MapReduce frameworks. Thus, important observations have been comparing First-in-First-Out (FIFO) data scheduler performance with more sophisticated strategies, some of these based on data locality [11,27].

In [11], an extensive investigation of data locality is reported and, data-scheduling issues addressed by a mathematical model and theoretical analysis in which impacts from cluster configurations and tasks are observed. In the same way, [27] investigate a cache strategy and looks at results reuse under a perspective of the Map transformations. Effective solutions for many applications, though is not adequate for situations where results produced by each data instance overlaps in random times during execution, which often happens when a naïve execution order is adopted.

As above mentioned, advances task-scheduler and efficient use of resources in MapReduce have been drawn the attention of different research. In AC, solutions employ such advantages to improve the classification step. In [18], a variation of MCAR (denoted MRMCAR) is presented to search frequent items in the rule discover step under large training data by proposing a new learning method that repeatedly transforms the data spaces. In the work presented by [12], a similar idea is used to develop an online algorithm for performing frequent items discover in a data streaming. In another hand, [2,5,9] use parallelization benefits of MapReduce to introduce classifications strategies that treat with typical methods toward to the big data solutions. In this sense, a fuzzy associative classifier [9], a FP-Growth [5] parallel approach and a new storage format are presented. Solutions that involve several perspectives of performance under different research fields with great results in computational resource and execution time maximization. However, to our best knowledge, such efforts do not consider a pre-analysis of data instances to predict computations and results before executions to achieve high-performance. In addition, many such research fields do not evaluate the overlap among partial computations of different data instances to maximize result reuse as part of their performance goals.

In summary, methods and strategies discussed have shown relevance and great benefits to different research fields. However, such efforts deal with performance through solutions that promote more responsible execution time for applications by an individual analysis of each data instance on execution time, making it the singular point of attention. According to our investigations, different approaches present performance variations if the execution order of data instances changes, in particular, there are significant impacts in the relevance of partial computations under data locality principle when cache strategies are considered. In this sense, we perceived a lack of alternatives that addressed these issues without requiring a costly redefinition of concepts and application under novel paradigms, which led us to discuss our parallel LAC approach, as presented in next section.

3 Parallel Lazy Associative Classification

In the aforementioned section, we discussed the importance of associative classification in current days and some of their problems, concentrating our attention in the lazy associative classification demand driven-basis approach – LAC. We demonstrated the advantages of such an approach when we described the ability of the local models generating well-approximations of the target function F in classification time. We explained MapReduce concepts and their benefits in a recent open-source solution named Spark. Now, we describe an approach that combines LAC and Spark advantages in a parallel AC in which computations of similar data instances can be cached and shared efficiently.

Our insight to propose such a solution is the drawback imposed by sequential LAC execution process, in which similar classifiers introduce an expressive computation overlap in the classification time. A condition caused by similar CARs used to label classes for distinct but related unknown data instances. Because CARs of independent local models are directly related to the attribute space of Γ, we propose a two-step based strategy to improve execution performance during classifications. For explain it, let us consider τ_1 to τ_4 as four data instances of Γ with some similarity and a 5-dimensional attribute space as shown in Fig. 1.

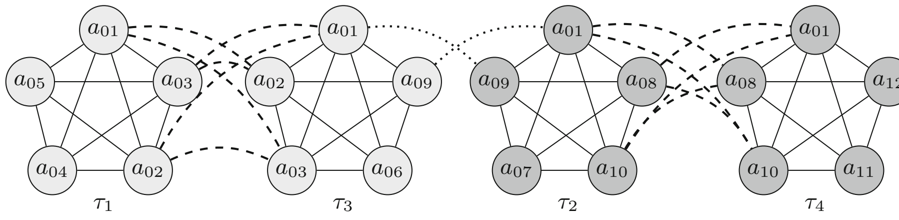

Fig. 1. A toy example of the modeling stage for attribute relationships in Γ using four data instances with some overlap on a 5-dimensional attribute space.

In Fig. 1, attributes are described as vertices and edges are drawn to show the relationship between each attribute pair, in a model designed as undirected and weighted graph $G = \{V, E\}$. Each data instance is reported as a complete graph with a high-correlation among their attributes. In another hand, edges are mapped in an adjacency matrix M, with $M_{n,m} \in \mathbb{R}$ corresponding to the relationship weight between n^{th} and m^{th} attributes, that is, the number of incidences of each attribute pair in the attribute space of Γ.

Considering a naive execution order as τ_1, τ_2, τ_3 and τ_4, CARs yielded by τ_1 will prematurely leave the cache since τ_2 not overlapping them and cache space is naturally limited. As a consequence, an expressive re-work is employed on classification time. However, in G there are some subsets of higher relationship attributes (i.e, linked by dotted edges), that is, τ_1 and τ_3, as well as τ_2 and τ_4 drawn greater overlap than τ_1 and τ_2 or τ_2 and τ_3. Thus, if Γ data instances are reordered by such high-correlation, we believe that it is possible to reduce the re-work on CARs mining process. Consequently, an outperform execution time is achieved. In this sense, we introduce a PageRank-driven similarity metric to compose a pre-analysis that investigates the relevance for each attribute pair in G and, from such relevance, decides the data execution order. As G is a weighted graph with similar representativeness than Web pages relationship description, such a strategy can be used toward an execution order effective.

As incidence values of each attribute pair in G are mapped in M with $M_{i,j}$ showing occurrence value of some pair, we introduce the PageRank to getting an eigenvalue for each attribute in G, which is used as score for mapping data instances in Γ in a pre-analysis stage that uses the following equation:

$$\tau_k = \sum_{a_i \wedge a_j \in \tau_k} G_{v_i, v_j} \tag{4}$$

such that, $\forall i \wedge \forall j \mid M_{i,j} > 0$ with a_i and a_j as i^{th} and j^{th} items of the data instance k.

Data instances are sorted in descending order by representativeness look at attribute-eigenvalues that designed it on the attribute space. Considering the toy example of Fig. 1, data instances are performed as $\tau_1, \tau_3, \tau_2, \tau_4$, ensuring a effective CAR memoization. In addition, a process performed in a second stage through a parallel execution model with a typical cache strategy is employed to reducing computations among similar classifiers and improve classification time outperform. A high-level scheme of such parallel execution model is described in the Fig. 2.

In our parallel execution model, each partition defined by Spark scheduler is executed in the workers from threefold fundamental stages. Firstly, local searches identify well-associate items between both Λ and Γ, in a filter step. A map transformation is then used to generate a temporary item set for each $\tau \in \Gamma$. Then, the generated item sets are performed in a parallel flow in which CARs are mining and shared them in a cache structure. Finally, a filter gets the better c_i for each τ and a reduce action consolidates results on the master, as well as, updates the cache with the frequent CARs. To ensure adequate CARs sharing, the cache structure is designed from accumulators and broadcast variables. Accumulators are employed to storing novel CARs while broadcast variables spread them to the workers. Thus, when each novel task partition is performed, the similar CARs are sharing and reused among classifiers avoiding costly computations on classification time by effective memoization. The performance of our proposal is reported based on observations of different naïve execution performed by the same parallel classification model through a FIFO execution order.

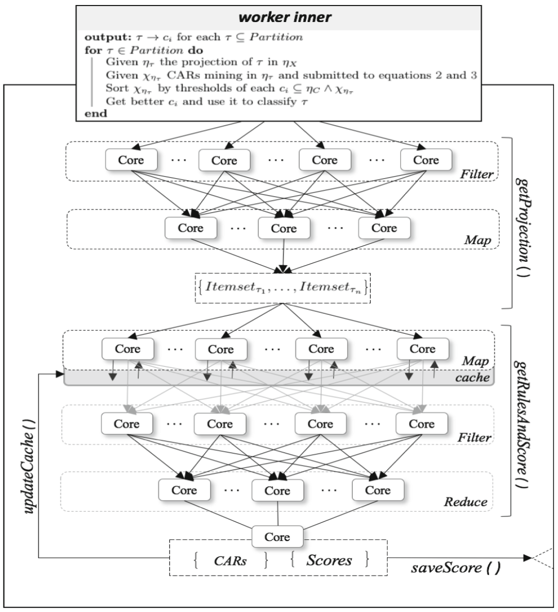
```
worker inner
output: $\tau \rightarrow c_i$ for each $\tau \subseteq Partition$
for $\tau \in Partition$ do
    Given $\eta_\tau$ the projection of $\tau$ in $\eta_X$
    Given $\chi_{\eta_\tau}$ CARs mining in $\eta_\tau$ and submitted to equations 2 and 3
    Sort $\chi_{\eta_\tau}$ by thresholds of each $c_i \subseteq \eta_C \wedge \chi_{\eta_\tau}$
    Get better $c_i$ and use it to classify $\tau$
end
```

Fig. 2. Execution flow of LAC in Apache Spark programming model in which each unlabeled data partition is performed in threefold core stages: (1) projection, (2) CARs mining and class score, and (3) cache update with frequent CARs.

4 Experimental Results

In order to evaluate the effectiveness of the proposed parallel LAC and measurement the impacts of our similarity metric in memoization of frequent CARs, we investigated the Round-Robin (RR) and Least Recently Used (LRU) cache policies to different CARs storage spaces. For each list of unlabeled data instance evaluated, we performed initial computations, without data collection, to generate training models and cache buffers. Training models, containing data produced in the pre-processing step, were employed in all experiments. The cache buffers, in turn, were used to create a baseline to demonstrate, from 100% advance knowledge of the CARs, how far the executions with caches of smaller size, i.e., 20%, 40%, 60%, and 80%, and without previous initialization are optimal. Initial computations were executed with a thread as master and worker while the parallel evaluation addressed by one master and twelve worker threads.

To baseline for our evaluations, we addressed a First-Come-First-Served (FIFO) with the same cache sizes but without reordering the data instances. We evaluate our proposal (i.e., PageRank), denoted in the graphics as T3, comparing the results achieved by it with the results of FIFO execution with the same data input but randomly organized in three different manners, expressed as T0, T1, and T2. The architecture used to generating the experiments was constituted from four quad-core machines with a 2.7 GHz Intel i5 processor, 16 GB of RAM, an HD of 1 TB, and Linux OS. As our goal is to provide an inherent maximization of frequent CARs memoized in the cache, we considering such architecture more than enough.

We evaluated LAC behavior in the Spark and configurations above proposed using typical datasets treated by [19–22], availables in UCI machine learning repository[1] with 50% for training and the remainder as test data instances. For each dataset, the support and confidence were instantiated as 0, and the maximum size of CARs in 3. We measurement the number of rules produced on classification time as well as the runtime and hits on the cache. The number of rules is shown in Fig. 3.

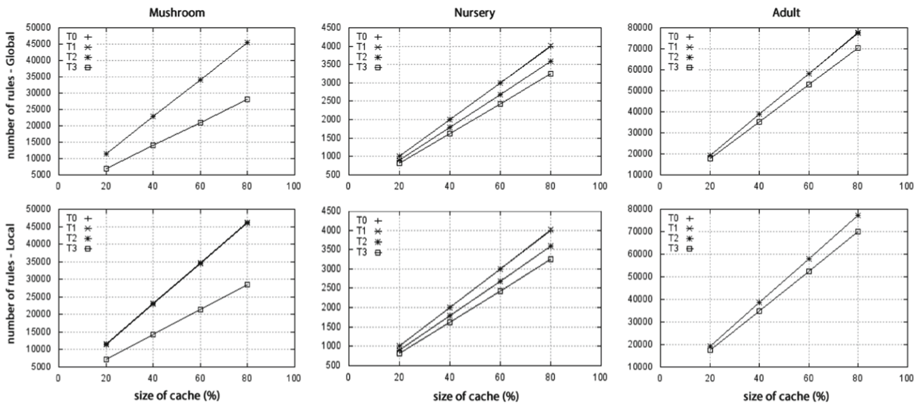

Fig. 3. Experiments overview for the number of rules produced using a typical cache Least Recently Used (LRU) and different data distributions based on FIFO (T0 to T2) and PageRank (T3) execution order.

The results expressed in Fig. 3 make it clear that an expressive reduction in CAR generation is achieved and, as a consequence, relevant gains are acquired to memoization of frequent itemset. Because PageRank produces greater coherence of frequent CARs on the cache, an outperform in the execution time and cache hit rate can be achieved. The Figs. 4 and 5 present such gains.

As can be seen in above results (i.e., Figs. 4 and 5), our PageRank-driven similarity metric can offer significant gains in CAR memoization (i.e., high hit rate), which consequently improves outperform to the total execution time and turned low re-work with costly computations (i.e., Fig. 3). An improvement that occurs because our proposal is able to exploit correlations among items in attribute space in each dataset given as input. Furthermore, the most meaningful results are observed when the cache size is more restrictive, a typical condition in many applications and real computational structures.

[1] https://archive.ics.uci.edu.

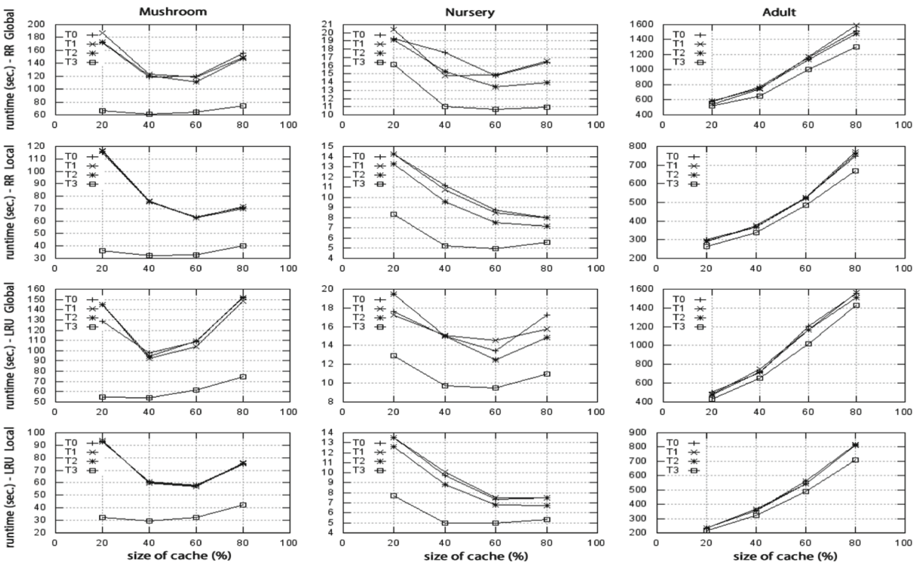

Fig. 4. Experiments overview to LAC execution time on Apache Spark broadcast and accumulators variables, Round-Robin (RR) and Least Recently Used (LRU) typical caches, based on FIFO (T0 to T2) and PageRank (T3) execution orders.

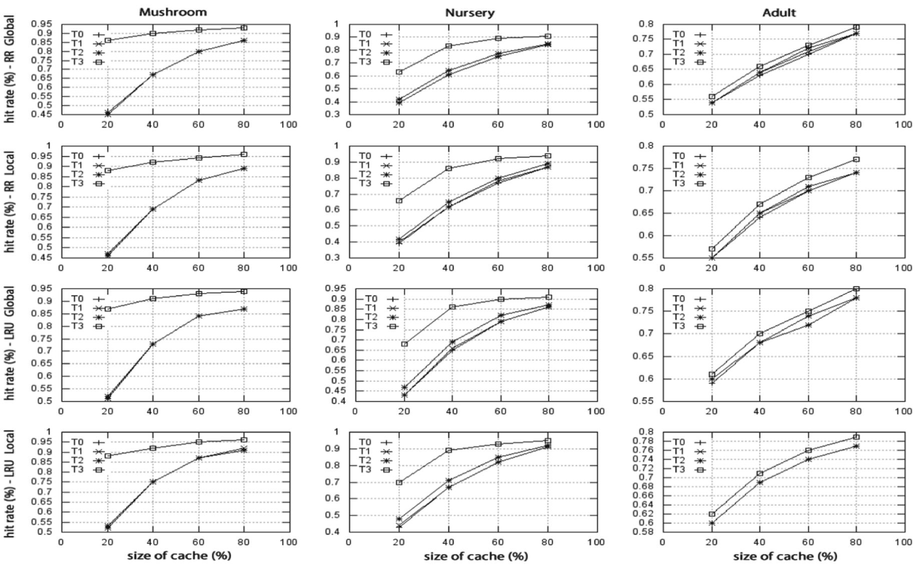

Fig. 5. Experiments overview to LAC cache hit rate on Apache Spark broadcast and accumulators variables, Round-Robin (RR) and Least Recently Used (LRU) typical caches, based on FIFO (T0 to T2) and PageRank (T3) execution orders.

As communication is a significant factor for distributed applications, we drawn twofold scenarios to evaluate the impacts of the cache structure. We evaluated a local cache in each worker (i.e, inside LAC structure) and we compared its performance with a global strategy. As expected, the local cache avoids a significant communication overhead between nodes and our similarity metric ensures high cache hit rates, equivalent those obtained in global accumulators-based structure and broadcast variables available on the Spark engine. Demonstrating that our proposal is adequate and that it induces significant benefits on classification time and memoization of CARs, without leads to a high communication among distinct workers.

5 Conclusions

In this paper, we presented a parallel and suitable approach of associative classification that employs exploration and analysis of unlabeled data instances by a similarity metric based on the PageRank concepts, in particular for the lazy associative classification. We discussed that computational overlap among similar classifiers can be solved ordering input data instances according to attribute space, and demonstrated a parallel implementation of the lazy associative classification can benefits from such order toward to an execution time outperforming. The main contribution of the proposed approach is the embedded similarity metric that can be used to maximize the memoization of computed rules (i.e, CARs). Different similarity metrics can be integrated under our approach to coordinate the distribution and execution of data instances, as well as to treat CARs into typical cache strategies since our metric does not produce any additional cost for the parallel implementation design.

We conducted a series of experimental evaluations and shown that our parallel LAC, idealized on the Spark engine, not only improves response time for different datasets but also leads to considerable improvements in the cache data locality, especially under severe space limitations. Furthermore, we have shown that handling appropriately CARs in associative classification prevents a considerable amount of re-work in consecutive unlabeled data instances. Indeed, for many cases, this tends to be the most significant factor between high-performance behavior and a costly and underutilized execution.

We are currently working on three concepts that are directly derived from this work: (1) we consider the situation in which the application receives data instances interactively at run-time, so there is no way to order the entire set of instances, and we should consider ordering partial views in connection with the current state of the cache, (2) performing a global optimization in scenarios with large datasets, and (3) for different data mining applications. In such scenarios, we have an additional concern about how to distribute the computation between several workers, while we maintaining high cache utilization on each of them. Lastly, we will tackle the problem in a dynamic scenario in which data instances are created at run-time while at the same time workers come and go, with their respective caches. Further, we have also begun conducting evaluations for larger scenarios and massively parallel environments.

Acknowledgements. This work was partially funded by INCT Cyber, MASWeb, CAPES, CNPq, Finep, FAPEMIG, and CEFET-MG, as well as EUBRA-BigSea and Atmosphere projects.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. SIGMOD 1993, pp. 207–216. ACM, New York (1993). https://doi.org/10.1145/170035.170072
2. Almasi, M., Abadeh, M.S.: A new MapReduce associative classifier based on a new storage format for large-scale imbalanced data. Cluster Comput. **21**(4), 1821–1847 (2018). https://doi.org/10.1007/s10586-018-2812-9
3. Althebyan, Q., Jararweh, Y., Yaseen, Q., AlQudah, O., Al-Ayyoub, M.: Evaluating map reduce tasks scheduling algorithms over cloud computing infrastructure. Concurrency Comput.: Pract. Exp. **27**(18), 5686–5699 (2015). https://doi.org/10.1002/cpe.3595
4. Antonelli, M., Ducange, P., Marcelloni, F., Segatori, A.: A novel associative classification model based on a fuzzy frequent pattern mining algorithm. Expert Syst. Appl. **42**(4), 2086–2097 (2015). https://doi.org/10.1016/j.eswa.2014.09.021. http://www.sciencedirect.com/science/article/pii/S0957417414005600
5. Bechini, A., Marcelloni, F., Segatori, A.: A MapReduce solution for associative classification of big data. Inf. Sci. **332**, 33–55 (2016). https://doi.org/10.1016/j.ins.2015.10.041
6. Cheng, D., Rao, J., Guo, Y., Zhou, X.: Improving MapReduce performance in heterogeneous environments with adaptive task tuning. In: Proceedings of the 15th International Middleware Conference. Middleware 2014, pp. 97–108. ACM, New York (2014). https://doi.org/10.1145/2663165.2666089
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008). https://doi.org/10.1145/1327452.1327492
8. Dong, G., Zhang, X., Wong, L., Li, J.: CAEP: classification by aggregating emerging patterns. In: Arikawa, S., Furukawa, K. (eds.) DS 1999. LNCS (LNAI), vol. 1721, pp. 30–42. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46846-3_4
9. Ducange, P., Marcelloni, F., Segatori, A.: A MapReduce-based fuzzy associative classifier for big data. In: 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), August, pp. 1–8 (2015). https://doi.org/10.1109/FUZZ-IEEE.2015.7337868
10. Gautam, J.V., Prajapati, H.B., Dabhi, V.K., Chaudhary, S.: Empirical study of job scheduling algorithms in hadoop MapReduce. Cybern. Inf. Technol. **17**(1), 146–163 (2017). https://content.sciendo.com/view/journals/cait/17/1/article-p146.xml
11. Guo, Z., Fox, G., Zhou, M.: Investigation of data locality in MapReduce. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012), May, pp. 419–426 (2012). https://doi.org/10.1109/CCGrid.2012.42
12. Lakshmi, K.P., Reddy, C.R.K.: Fast rule-based prediction of data streams using associative classification mining. In: 2015 5th International Conference on IT Convergence and Security (ICITCS), pp. 1–5. IEEE (2015)

13. Li, W., Han, J., Pei, J.: CMAR: accurate and efficient classification based on multiple class-association rules. In: Proceedings of the 2001 IEEE International Conference on Data Mining. ICDM 2001, pp. 369–376. IEEE Computer Society, Washington, DC (2001). http://dl.acm.org/citation.cfm?id=645496.657866
14. Lin, C., Guo, W., Lin, C.: Self-learning MapReduce scheduler in multi-job environment. In: 2013 International Conference on Cloud Computing and Big Data, December, pp. 610–612 (2013). https://doi.org/10.1109/CLOUDCOM-ASIA.2013.95
15. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: 1998 Knowledge Discovery and Data Mining Conference (KDD), pp. 80–86 (1998)
16. Qureshi, M.N., Aldheleai, H.F.H., Tamandani, Y.K.: An improved documents classification technique using association rules mining. In: 2015 IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), November, pp. 460–465 (2015). https://doi.org/10.1109/ICRCICN.2015.7434283
17. Thabtah, F., Cowling, P., Peng, Y.: MCAR: multi-class classification based on association rule. In: The 3rd ACS/IEEE International Conference on Computer Systems and Applications, January (2005). https://doi.org/10.1109/AICCSA.2005.1387030
18. Thabtah, F., Hammoud, S.: Parallel associative classification data mining frameworks based MapReduce. Parallel Process. **25**(02), 1550002 (2015)
19. Veloso, A., Meira, W., Gonçalves, M., Almeida, H.M., Zaki, M.: Calibrated lazy associative classification. Inf. Sci. **181**(13), 2656–2670 (2011). https://doi.org/10.1016/j.ins.2010.03.007. http://www.sciencedirect.com/science/article/pii/S0020025510001192. Including Special Section on Databases and Software Engineering
20. Veloso, A., Meira Jr., W., Gonçalves, M., Almeida, H.M., Zaki, M.: Calibrated lazy associative classification. Inf. Sci. **181**(13), 2656–2670 (2011)
21. Veloso, A., Meira Jr, W., Zaki, M.J.: Lazy associative classification. In: ICDM 2006: Proceedings of the Sixth International Conference on Data Mining, December, pp. 645–654. IEEE Computer Society (2006)
22. Veloso, A.A.: Classificação Associativa sob Demanda. Ph.D. thesis, Universidade Federal de Minas Gerais, March 2009
23. Wang, J., Li, X.: Task scheduling for MapReduce in heterogeneous networks. Cluster Comput. **19**(1), 197–210 (2016). https://doi.org/10.1007/s10586-015-0503-3
24. Wang, J., Karypis, G.: Harmony: efficiently mining the best rules for classification. In: Proceedings of SDM, pp. 205–216 (2005)
25. Yin, X., Han, J.: CPAR: classification based on predictive association rules. In: Proceedings of the International Conference on Data Mining. SIAM (2003)
26. Zaharia, M., et al.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016). https://doi.org/10.1145/2934664
27. Zhao, Y., Wu, J., Liu, C.: Dache: a data aware caching for big-data applications using the MapReduce framework. Tsinghua Sci. Technol. **19**(1), 39–50 (2014). https://doi.org/10.1109/TST.2014.6733207