POLITECNICO DI TORINO
Repository ISTITUZIONALE

BAC: A bagged associative classifier for big data frameworks

Original
BAC: A bagged associative classifier for big data frameworks / Venturini, Luca; Garza, Paolo; Apiletti, Daniele. - STAMPA. - 637:(2016), pp. 137-146. (3rd International Workshop on Big Data Applications and Principles, BigDap 2016, co-located with the 20th East-European Conference on Advances in Databases and Information Systems, ADBIS 2016 Prague, Czech Republic 28-8-2016) [10.1007/978-3-319-44066-8_15].

Availability:
This version is available at: 11583/2651083 since: 2016-09-28T13:18:28Z

Publisher:
Springer Verlag

Published
DOI:10.1007/978-3-319-44066-8_15

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Springer postprint/Author's Accepted Manuscript

This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/978-3-319-44066-8_15

(Article begins on next page)

13 June 2026

BAC: a Bagged Associative Classifier for Big Data Frameworks

Luca Venturini, Paolo Garza, and Daniele Apiletti

Politecnico di Torino, Dipartimento di Automatica e Informatica, Corso Duca degli Abruzzi 24, Torino, Italy
name.surname@polito.it

Abstract. Big Data frameworks allow powerful distributed computations extending the results achievable on a single machine. In this work, we present a novel distributed associative classifier, named BAC, based on ensemble techniques. Ensembles are a popular approach that builds several models on different subsets of the original dataset, eventually voting to provide a unique classification outcome. Experiments on Apache Spark and preliminary results showed the capability of the proposed ensemble classifier to obtain a quality comparable with the single-machine version on popular real-world datasets, and overcome their scalability limits on large synthetic datasets.

Keywords: associative classifiers, bagging, machine learning, MapReduce, Apache Spark

1 Introduction

Big Data frameworks are becoming increasingly popular thanks to their technological maturity, the wide-spread availability, and the value they allow to extract from very large data sets. Even if *volume*, *velocity*, and *variety* are the typical characteristics of Big Data [1], others are often considered critical for businesses: among the top Vs of Big Data, *value* is probably the most valuable. Hence, Big Data frameworks are technological enablers to not only process larger and more complex data sets, but also to extract higher value from any dataset.

In this work, we aim at distributing the training phase of a special kind of classifiers, namely associative classifiers, whose training is highly sequential in its very own nature. The distribution of the work on a cluster of machines could have several advantages, among which a performance boost, or an increase of the dataset size which can be analyzed. Preliminary experimental results exploiting a distributed approach on Apache Spark applied to real-world and synthetic datasets showed promising results: ensembles of multiple models trained on small subsets of the original dataset (as small as 10%) lead to accuracies comparable with the single model trained on the whole dataset. Our method thus proves to be a viable solution to distribute the workload of this task, without compromising on the quality of the results.

This paper is organized as follows. Section 2 describes the data mining background, Section 3 presents the proposed approach, Section 4 discusses the experimental results, and Section 5 compares our approach with previous works. Finally, Section 6 draws conclusions and presents future developments.

2 Problem statement

In this section, we state the classification problem and describe how association rules are usually exploited to solve it. Finally, we describe the centralized L^3 associative classifier [2], since the distributed BAC algorithm proposed in this paper is based on L^3.

2.1 Classification problem

The input dataset $\mathcal{D}$ is represented as a relation R, whose schema is given by k distinct attributes $A_1 \dots A_k$ and a class attribute C. Each record in R can be described as a collection of pairs *(attribute, integer value)*, plus a class label (a value belonging to the domain of class attribute C). Each pair *(attribute, integer value)* will be called *item* in the following. A training record is a tuple in R where the class label is known, while an unlabelled record is a tuple in R where the class label is unknown. The attributes can be either categorical or continuous attributes. For categorical attributes, for each attribute, all values in its domain are mapped to consecutive positive integers. In the case of continuous attributes, the values of each attribute are first discretized into intervals, and then intervals are mapped into consecutive positive integers.

A classifier is a function from $A_1, \dots, A_n$ to $\mathcal{C}$, that allows the assignment of a class label to an unlabelled records. Given a collection of training data, the classification task is the generation of a classifier able to predict the class label for the unlabeled data with high accuracy.

2.2 Associative classification

Associative classifiers (e.g., [3, 2]) are well-known classifiers based on association rules [4]. Association rules [4] are rules in the form $X \rightarrow Y$, where both X and Y are set of items. When using them for classification purposes, X is a set of items (i.e., (attribute, value) pairs), while Y is a class label. A record d is said to match a collection of items X when $X \subseteq d$. The quality of an association rule is measured by two parameters, its support, given by the number of records matching $X \cup Y$ over the number of records in the training dataset, and its confidence given by the the number of records matching $X \cup Y$ over the number of records matching X. Hence, the classification task can be reduced to the generation of the most appropriate set of association rules for the classifier.

The model generation phase of the associative classifiers (e.g., [3, 2]) is based on two steps: (i) Extraction of all the classification rules with a support higher than a minimum support threshold *minsup* and a minimum confidence threshold

minconf and (ii) Rule selection by means of the database coverage technique. Step (i) extracts all the rules that are potentially significant, in terms of frequency and confidence, while the Step (ii) selects the best rules by evaluating their accuracy.

Once the model is built, the prediction of the class label of a generic unlabelled record d is performed by considering the the first rule r matching d, based on a quality ranking, and the class label of r is used to label d.

The L^3 classifier The proposed distributed algorithm (Section 3) is based on the L^3 associative classifier. Hence, we describes the centralized version of L^3. The model generation phase of L^3 algorithm is based on two main steps: (i) frequent classification rule mining and (ii) rule selection by means of a lazy database coverage technique. To address the mining step efficiently, L^3 exploits an FP-growth like association rule mining algorithm. The exploited mining algorithm is optimized to directly extract classification rules (i.e., association rules with a class level as consequent).

Once the potentially large of set of frequent classification rules is available, a lazy pruning step, based on the database coverage approach is applied. Before performing the pruning phase, a global order is imposed on the extracted frequent classification rules. Let r_1 and r_2 be two classification rules. Then r_1 precedes r_2, denoted as $r_1 > r_2$ if

1. $\text{conf}(r_1) > \text{conf}(r_2)$, or
2. $\text{conf}(r_1) = \text{conf}(r_2)$ and $\text{sup}(r_1) > \text{sup}(r_2)$, or
3. $\text{conf}(r_1) = \text{conf}(r_2)$ and $\text{sup}(r_1) = \text{sup}(r_2)$ and $\text{len}(r_1) > \text{len}(r_2)$, or
4. $\text{conf}(r_1) = \text{conf}(r_2)$ and $\text{sup}(r_1) = \text{sup}(r_2)$ and $\text{len}(r_1) = \text{len}(r_2)$ and $\text{lex}(r_1) > \text{lex}(r_2)$

where $\text{len}(r)$ denotes the number of items in the body of r, and $\text{lex}(r)$ denotes the position of r in the lexicographic order on items.

After the rule sorting operation, only the rules satisfying the chi-square test, at a significance level of 95% are selected (i.e., the rules with a correlation between the antecedent and the consequent of the rule).

Finally, the lazy pruning approach is applied. The idea behind the lazy pruning is to discard only the rules that do not correctly classify any training record, i.e., the rules that only negatively contribute to the classification of training records and organize the other rules in two levels (first level and second level). To achieve this goal, the lazy pruning approach splits the frequent association rule set in three subsets:

- **First level rules**. This rule set contains the mining set of rules that is needed to properly "cover" the training data. This set of rules represents the main characteristics of the majority of the training records.
- **Second level rules**. These rules are not included in the first level, but are potentially useful to represent the characteristics of "special" records that are slightly different with respect to the most common ones appearing in

the training set. These rules are also called "spare" rules and are useful to properly label new unlabelled data not represented by the rules of the first level.
- **Harmful rules**. Some rules, even when applied on the training set, always perform wrong predictions. These rules must be removed since they contribute only negatively to the quality of the generated model.

To identify the three described subsets, L^3 applies a database coverage technique. Specifically, L^3 considers one rule r at a time in the sort order and selects the training records matched by r. For each matched record d, L^3 checks also if r classifies properly d. If r classifies properly at least one training record, then r is stored in the first level of the classifier. Differently, if all the training records matched by r have a class label different from the one of r, then r is discarded (i.e., it is an harmful rule). If r does not match any training data then r is store in the second level of the final classifier. Once r has been analysed, all the training records matched by r are removed from the training set and the next classification rule is analysed by considered only the remaining training records.

To predict the class label of an unlabelled record d, the rules of the first level are initially considered. If no rule in the first level matches d, the second level is used.

2.3 Bagging

The bagging technique is frequently used to build accurate models by combining a set of "weak" classifiers. The basic idea is that a set of classifiers can provide better predictions than a single model.

The bagging technique works as follows.

1. Generate N datasets by applying random sampling with replacement on the training dataset.
2. Build N classification models (one for each dataset generated during Step 1).
3. Predict the class label of the new unlabelled records by using a majority voting approach combining the predictions of the N classifiers that have been built during Step 2.

3 BAC: Bagged Associative Classifier

This Section describes the proposed approach to scale on different machines the training of an associative classifier, namely L^3, which is presented in Section 2. The Big Data framework we exploited is the well-known Apache Spark, which poses some specific technological issues to the design of an associative classifier.

We recall, as mentioned in Section 2, that an important phase of the generation of such a classifier is represented by the database coverage (lazy pruning in L^3). Unfortunately, the database coverage algorithm does not fit a map-reduce approach, as it is sequential in its nature: the database must be covered in the

strict order of the rules for the algorithm to be effective. Such requirement prevents most of the work of this phase to be executed in parallel.

To cope with the scalability of the process, in order to fully exploit the distributed framework, we adopt bagging. The solution consists in generating several models, each one from a portion of the original full dataset. Each model can then be trained independently, thus also in parallel on multiple machines. The model trained locally on each portion of the dataset is a variant of L^3, where we discharge the second level rules after the database coverage phase. The FP-growth algorithm used is the one implemented in Apache Spark, slightly modified to run locally. The ensemble of the single models eventually generate a single prediction by majority voting.

In the following we provide details about the split of the dataset among the different machines (i.e., Apache Spark workers). The ensemble of models is eventually collected and can be used on any number of machines to classify new records.

3.1 Dataset distribution

Each model, as said, is trained on a different portion of the original dataset. Each portion is drawn by sampling the original records with replacement, as this method is well-proven in literature, like mentioned in Section 2.3. Moreover, whereas this method allows for obtaining any sort of combination for number of models and partitions size, sampling without replacement would limit the total number of records to the original dataset size. In other words, sampling with replacement can produce, for example, multiple models trained on a dataset as large as the original dataset (and still different), or even larger; or we might have a dozen models, each trained on half of the original dataset, while without replacement we would have just two halves available.

The core APIs of Apache Spark provide a method for performing sampling with replacement. The method is characterized by two parameters: the input dataset D and a real number λ that is used to specify the size of the sample as a fraction of the input dataset. Specifically, a sample of size $|D| \times \lambda$ is generated. The provided method performs a sampling with replacement, generating k copies for each input record, where k is drawn from a Poisson distribution.

In our case, we are interested in N samples, each one with a size equal to $|D| \cdot f$, since we want to build N models for N different samples. f is real parameter of the algorithm and it is used to specify the size of each sample. We can generate the N samples invoking, sequentially, the Spark sampling API N times setting λ to f. However, this approach will reduce the parallelism of our algorithm. Hence, we decided to use another approach that allows us generating simultaneously N samples, each of size $|D| \cdot f$. Specifically, we proceed in the following way. First, we sample the dataset D with $\lambda = N \cdot f$, as to have an expected single sample with a total size equal to $|D| \cdot N \cdot f$. Now we need to split this sample in N subsamples. We can perform this operation by using the parallel Spark API `repartition` that splits the input data in a user-specified

number of partitions (in our case we set it to N). Hence, each partition contains a sample of size $|D| \cdot f$ of the input dataset.

Once obtained N samples with the desired fraction of data, each on a separate partition, we can simply call `mapPartitions` in Spark to apply the same function on each, in our case the training function of the model.

4 Experimental validation

To validate the proposed approach, we implemented BAC in Scala on top of Apache Spark. Experiments aim at assessing the usefulness of a distributed approach against two alternatives: (i) working on the whole dataset on a single machine, which is not always feasible, and (ii) working on a sample of the original dataset, that is always feasible for small portions, but at the costo of lower quality of the model (e.g., accuracy). Experiments also evaluate the effect of the number of estimators (models) of the ensemble on the final quality of the whole ensemble model.

As evaluation criterion, we focus primarily on the accuracy, computed on a 10-fold cross-validation of the selected datasets. For each mean accuracy, we computed a 95% confidence interval based on the standard error of the mean, using a t-student statistics. We also computed the average time for training.

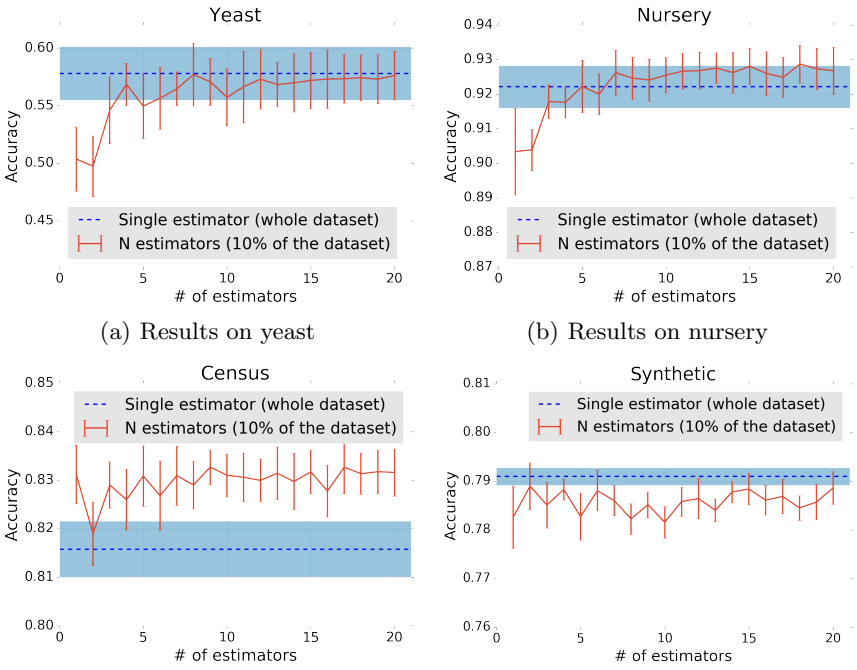
Three very popular datasets have been used for the experiments: `yeast`, `nursery`, and `census`, from the UCI repository [5]. We have chosen these datasets as they are heterogeneous in dimensions, shape and distribution, and state-of-art associative classifiers do not perform well, so that there is still margin for improvements. The continuous attributes have been discretized applying the entropy-based discretization technique [6]. Since the bigger of the three datasets counts for 30162 records only, we generated a fourth synthetic dataset containing 1 million tuples and 9 attributes with different distributions, using the IBM data generator. Continuous attributes have then been discretized to 10 bins each.

All experiments share the same minimum support threshold (1%) and the same minimum confidence value (50%), as in previous works of associative classifiers [2] they proved to generate a good amount of significant rules. Experiments were performed on a cluster with 30 worker nodes running Cloudera Distribution of Apache Hadoop (CDH5.5.1), which comes with Spark 1.5.0. The cluster has 2.5TB of RAM, 324 cores, and 773TB of secondary memory. The size of each container has been set to 2GB.

4.1 Results

Focusing on real datasets, that are `yeast`, `nursery`, and `census`, Figures 1(a), 1(b), and 1(c) show the average accuracy. The dashed line represents the accuracy obtained by the classifier trained over the whole dataset on a single machine. The range of its confidence interval, highlighted in the figures, serves as reference for our evaluation. On the x axes we have the number of models trained, each on a 10%-portion of the original dataset. For $x = 1$, we have the simple sampling.

For `yeast` (Fig. 1(a)), we see how the simple sampling can reduce the total accuracy of more than 8% with respect to the level of the classifier trained on the whole dataset. In `nursery` (Fig. 1(b)) we see the same behaviour. Here the drop is less marked, less than 2%, but still significant if we compare the confidence intervals. Curiously, `census` (Fig. 1(c)) shows a completely different behaviour. In this dataset, representing a US census of the population, sampling obtains an accuracy even higher than our reference. This surprising result can be due to a simpler, even if weaker, model less prone to overfitting the training data. The 10-fold cross-validation indeed penalizes overfitting models.



(a) Results on yeast

(b) Results on nursery

(c) Results on census

(d) Results on the synthetic dataset

Fig. 1. Accuracy results

Let us now inspect the results of BAC, with an increasing number of models, up to 20. For all datasets, the accuracy level stabilizes way before 10 models. We see that `yeast` (Fig. 1(a)), for example, enters the confidence interval of the reference model from 3 models on, becoming statistically indifferent. Large confidence intervals are a peculiarity of the k-fold cross-validation, and highly depend on the dataset. It is as well a measure of the robustness of the model towards different training sets. `Nursery` (Fig. 1(b)) and `census` (Fig. 1(c)) confirm the same trend as `yeast`, with BAC entering the confidence interval of the centralized classifier from 3 models onward. These results are interesting, since

3 models cover less than a third of the original dataset, and they are enough to reach similar accuracies to our reference model. This means also that distributing the dataset only once, i.e., 10 models of 10% each, we can be confident enough of having entered already the steady region, without the need of more replicas/models. Furthermore, having a number of models equal to one corresponds to the simple sampling, whereas two models would affect the contribution of the majority voting among models. Thus 3 is also the minimum sensible setting of this parameter.

Results for the synthetic dataset are shown in Figure 1(d). The dataset counts for a million records, and tries to emulate a possible use case of BAC, to show off its potential on large dataset. Firstly, the accuracy of the reference classifier is characterized by a very narrow confidence interval. This, as mentioned above, comes from the shape and distribution of the dataset: since it is synthetic, the distribution of the attributes and of the labels is uniform among the folds, thus the standard error of the mean accuracy is very little compared to the real datasets, feature that is further sharpened by the greater size of the folds. Sampling the 10% of the dataset results in an almost negligible detriment to the quality, less than a point of accuracy. This outcome is not surprising, as the distribution of the attributes in the sample can not vary too much from the whole, being the attributes generated from a given distribution. The accuracy of BAC increasing the number of models steadily stays in the range of the single sample, often overlapped with the results of the reference model, showing that adding data from the remainder of the dataset does not add information to our model. Though in this case we cannot conclude for absolute better performances on either side (namely, BAC or single-machine), we see an interesting fact in the way these results were obtained. On the cluster described at the beginning of the section, indeed, the generation of the reference classifier, the single model trained on the whole dataset, failed, running out of the memory available to its (single) container. To obtain the value of reference plotted in Figure 1(d) we executed the training on a standalone Spark machine, where we set the amount of memory for the Java VM to the whole RAM available on the machine itself (32 GB). All experiments for BAC, instead, completed successfully, proving that distributing workload can be a way to overcome the limits of the single machine and expand the size of the explorable datasets.

Finally, Table 1 shows the average training time for the 10% sample, for BAC with 10 models on a tenth of the data each, and the reference model, that is the classifier trained on the whole data. As said, the last-mentioned classifier failed its execution for the synthetic dataset, so its timing is not fairly comparable with the others. On the three real datasets, we notice that, unsurprisingly, simple sampling outperforms the more accurate models. BAC is the slowest, as the overhead of the distributed framework and the communication costs are very high for such small datasets. On the larger synthetic dataset, these overheads are absorbed by the real computational costs, resulting in little difference between one or 10 machines working on models of the same size/complexity. We need

further investigations to show the real impact of parallelization on a real dataset of this size or larger.

To sum up, from these results we can conclude that:

1. the mere sampling is not always sufficient to reach a good accuracy,
2. training an associative classifier over the whole dataset is not always feasible,
3. bagging is a viable solution to reach the quality of a single classifier trained on the whole dataset, and offers an easy way to distribute the work among multiple workers.

Table 1. Average training time of the different approaches.

Dataset	Records	Avg training time for a fold [ms]		
		1 model, 10%	10 model, 10%	1 model, 100%
Census	30162	57692	340020	302147
Nursery	12960	983	3457	2181
Yeast	1484	642	1799	850
Synthetic	1000000	14235	15150	n.a.

5 Related work

The classification problem is a well-known problem and many approaches have been proposed to solve this problem [7, 8, 3] Among the others, associative classification is a well-known technique for structured data classification [3, 2]. Associative classifiers are accurate. However, their are based on two complex and time consuming steps: (i) association rule mining and (ii) database coverage [2]. Out paper is the first to propose an associative classifier based on Spark to addresss the complexity of the two steps by means of a distributed approach.

The increase of the overall accuracy of the predictions is also addressed by means of ensemble techniques [6]. The paper [9] analysed the impact of the boosting ensemble technique when a set of associative classifiers are used as building block. However, the impact of the bagging ensemble approach based on associative classifiers has never been analysed. Differently from [9], in this paper we perform this analysis by proposing a bagging version of an associative classifier.

6 Conclusions and future work

In this paper, a novel distributed associative classifier based on bagging has been proposed. Preliminary experiments showed that bagging can achieve the quality of the single-machine version, with an accuracy always better or as good as the sampling-only approach. Therefore, BAC proved to be a simple and effective

way to distribute the workload among the machines of a cluster. The achieved results are promising and hence we are performing further analysis, on other real-datasets, to analyse in more detail both the quality, in terms of accuracy, and the execution time of BAC. We also aim at improving BAC by evaluating a unified rule-generation phase, shared among the machines, to further reduce the total memory used by the cluster. Additional investigations should also identify the limits of this approach, such as dataset cardinality, dimensionality, and density.

Acknowledgment

The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 619633 ("ONTIC" Project).

References

1. Tsai, C.W., Lai, C.F., Chao, H.C., Vasilakos, A.V.: Big data analytics: a survey. Journal of Big Data **2**(1) (2015) 1–32
2. Baralis, E., Garza, P.: A lazy approach to pruning classification rules. In ICDM'02, Maebashi, Japan (December 2002)
3. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In KDD'98, New York, NY (August 1998)
4. Agrawal, R., Imilienski, T., Swami, A.: Mining association rules between sets of items in large databases. In SIGMOD'93 , Washington DC (May 1993)
5. Blake, C., Merz, C.: UCI repository of machine learning databases (1998)
6. Han, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
7. Quinlan, J.: C4.5: program for classification learning. Morgan Kaufmann (1992)
8. Rokach, L., Maimon, O.: Data Mining with Decision Trees: Theory and Applications. World Scientific Publishing Co., Inc., River Edge, NJ, USA (2008)
9. Sun, Y., Wang, Y., Wong, A.K.C.: Boosting an associative classifier. IEEE Transactions on Knowledge and Data Engineering **18**(7) (July 2006) 988–992