Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Learning distributed discrete Bayesian Network Classifiers under MapReduce with Apache Spark

Jacinto Arias*, Jose A. Gamez, Jose M. Puerta

Department of Computing Systems, University of Castilla-La Mancha, Albacete, Spain

A R T I C L E I N F O

Article history:
Received 17 March 2016
Revised 9 June 2016
Accepted 12 June 2016
Available online 22 June 2016

Keywords:
Bayesian Network Classifiers
MapReduce
Big Data
High dimensionality
Apache Hadoop
Apache Spark

A B S T R A C T

The challenge of scalability has always been a focus on Machine Learning research, where improved algorithms and new techniques are proposed in a constant basis to deal with more complex problems. With the advent of Big Data, this challenge has been intensified, in which new large scale datasets overwhelm the majority of available techniques. The community has resorted to Cloud Computing and distributed programming paradigms as the most immediate solution where Apache Spark has proven to be the most promising framework. In this paper we focus on the problem of supervised classification, exploring the family of the so called Bayesian Network Classifiers by studying their adaptability to the MapReduce and Apache Spark frameworks. We will analyse a range of algorithms and propose distributed versions of them. Our approach is based on a general framework for learning this probabilistic models from large scale and high dimensional data, the latter being a problem with less support in the literature. We also present an extensive experimental evaluation of our proposal over a wide set of problems and different elastic configurations of a computing cluster to show the full extent of the scalability properties of our framework. Additional material and the software to reproduce our experiments can be found on the supplementary website http://simd.albacete.org/supplements/distributed_bncs.html.

1. Introduction

Scalability has always been a prolific field of study in the areas of Data Mining and Machine Learning. Algorithms are constantly being improved to scale up in order to tackle bigger and more complex problems. The continuous evolution of the hardware as well as the efforts in the scientific community to provide new and improved techniques has maintained a steady progress in solving the challenges needed to deal with new interesting domains.

During the recent years these challenges have stepped up towards new levels of complexity, following a new progression that has outreached the capabilities of traditional techniques and computing architectures. This is the resulting impact of new emergent technological achievements in Computer Science such as Social Networks, the Internet of Things or the impact of globalization and telecommunications [1–3].

As both the companies and the scientific community invest in these new fields, the underlying technology evolves and cheapens propitiating an environment in which massive amounts of data can be generated and stored. This new data phenomena, usually referred to as Big Data opens a new window for Machine Learning in which large amounts of data can be analysed to extract knowledge from which obtain benefits.

The scale of these new problems requires novel solutions, for which the community has adapted traditional High Performance Computing and Cloud Computing into highly distributed and data-oriented new computing frameworks. Among the existing proposals, MapReduce (MR) [4] obtained great acknowledgement becoming a standard for processing massive datasets.

There has been an active development of Machine Learning solutions under MR since its beginning [5], where particular techniques have obtained more popularity on their adaptation to the framework. Apache Hadoop [6] was established as the standard open source implementation for MR, however it has been quickly succeeded by Apache Spark [7], which proposed an improved data abstraction, a faster execution environment based on main memory, and a friendlier programming interface. The Spark project has an specific focus on data analysis and Machine Learning and is packed with a standard library containing some of the most popular algorithms in the field [8]. This leaves an open space for new proposals from other researchers, with the purpose of discovering new techniques or adapting existing ones that could fit under this paradigm.

* Corresponding author.
E-mail addresses: jacinto.arias@uclm.es (J. Arias), jose.gamez@uclm.es (J.A. Gamez), jose.puerta@uclm.es (J.M. Puerta).

http://dx.doi.org/10.1016/j.knosys.2016.06.013
0950-7051/© 2016 Elsevier B.V. All rights reserved.

In this paper we propose a study on a particular family of supervised classification models called Bayesian Network Classifiers (BNCs) [9]. These techniques have demonstrated competitive results in the past for several domains and are present in many of the most popular Machine Learning libraries [10]. Our proposal focuses on the learning stage of such models, for which we introduce a general framework that characterizes the full family of BNC classifiers under the MapReduce paradigm. We also include a discussion among the best properties and main pitfalls of these models, both theoretic and practically, with the purpose of finding future research lines to approach the problem with specifically designed algorithms.

Although Big Data usually refers to problems in which we have a large volume of examples or observations, there is also a increasing number of problems in which a massive amount of variables are obtained for a given domain, sometimes ranging from thousands to millions of them [11]. This second component of the first V (volume) of Big Data [12] is sometimes ignored by many solutions; however it is a key component for many Machine Learning algorithms, specially regarding their scalability where some algorithms can become intractable if the domain grows excessively. The proposed framework tackles the problem from both perspectives simultaneously, being able to deal with either large scale and high dimensional datasets, for which we identify a common computation pattern that would allow the definition of both horizontal and vertical parallel approaches.

An exhaustive evaluation of our proposal is also presented, for which we have conducted a series of experiments over a large range of both synthetic and real problems on a competitive cluster of computers. We also explore the behaviour of the algorithms when the underlying architecture of the cluster is modified and a different amount of resources is involved in the execution. Thus, we aim to cover the widest spectrum of situations in which the framework can be applied, in order to provide a comprehensive description of the scalability properties of these methods.

The rest of the paper is organized as follows, in Section 2 we describe the background on Bayesian Network Classifiers by providing detailed descriptions of the sequential learning algorithms as well as their complexity; in Section 3 addition the MapReduce paradigm is described along with the particularities of both Apache Hadoop and Spark. In Section 4 we analyse the described algorithms in order to identify the stages that are suitable for parallelisation, then the MapReduce implementation is described and discussed. In Section 5 we conduct and describe a series of experiments for a series of datasets and different cluster configurations. Finally in Section 6 we offer a brief summary of the obtained results and propose future extensions of this work.

2. Background

2.1. Notation

Our context will be the supervised classification problem, in which given a vector $\mathbf{A} = \{X_1 \dots, X_n\}$ of discrete[1] predictor random variables and a class variable C with $c \in \Omega_C = \{c_1, \dots, c_l\}$, we wish to induce a model from a training dataset $\mathcal{D}$ consisting of m labelled examples $\{(\mathbf{x}^{(1)}, \mathbf{c}^{(1)}), \dots, (\mathbf{x}^{(m)}, \mathbf{c}^{(m)})\}$ where $\mathbf{x}^{(j)} = (x_{j1}, \dots, x_{jn})$. A comprehensive enumeration of the different notation symbols used in this study can be found on Table 1.

[1] Probabilistic classifiers are naturally defined for categorical variables, from this point we will assume that all variables in $\mathbf{A}$ are discrete, and therefore each $X \in \mathbf{A}$ has a set of states Ω_X.

Table 1
Summary of the different notation elements used in the paper.

$\mathcal{D}$	Training dataset
$\mathcal{D}_k$	A subset of the training data
n	Number of predictor attributes
m	Number of examples in the training data
$\mathbf{A}$	Set of predictive attributes or features
C	Class variable
$\bar{v}$	Average number of values per attribute
$\Omega_C = \{c_1, \dots, c_l\}$	Domain of the class variable
l	Number of Classes
x_i	The value for the ith attribute in $\mathbf{x}$
$\mathbf{pa}(X_i)$	The set of parents of X_i on a given BN

2.2. Bayesian Network Classifiers

Among other popular supervised classification models, those based on Bayesian Networks [13,14] have become increasingly popular over the years. From a probabilistic approach, we wish to estimate $p(c|\mathbf{x})$ for $c \in \Omega_C$ given an example $\mathbf{x}$. A Bayes classifier will select the value, c^k, which maximizes the posterior probability (MAP):

$$c^k = \arg\max_c p(c \mid \mathbf{x}) = \arg\max_c p(\mathbf{x}, c) \tag{1}$$

A Bayesian Network (BN) is a probabilistic graphical model that allow us to efficiently represent and manipulate probability distributions. It is defined by two components: A graphical structure $\mathcal{G}$ represented by a directed acyclic graph $(\mathbf{V}, \mathbf{E})$ where $\mathbf{V}$ if a set of nodes representing the variables in $\mathbf{A}$ and $\mathbf{E}$ is a set of edges encoding (in)dependence relations between them; and a set of numerical parameters Θ encoding quantitative information about the variables and the dependences encoded in the graph, concretely, for each variable $X_i \in \mathbf{A}$ and its set of parents in the graph $\mathbf{pa}(X_i)$, a conditional probability distribution table (CPT) $p(X_i|\mathbf{pa}(X_i))$ is stored. This representation encoded by a BN can be used to recover the joint probability distribution thanks to the Markov Rule:

$$p(X_1, \dots, X_n) = \prod_{i=1}^{n} p(X_i \mid \mathbf{pa}(X_i)) \tag{2}$$

Owing to its sound mathematical foundations and its capabilities as both descriptive and predictive model, the Bayesian Network formalism has been widely adopted for many tasks in the field of Artificial Intelligence. However, a general BN model has proven not to be very competitive for the problem of supervised classification [15]. For that reason, specific models have been proposed where the structure of a BN is adapted to deal with this particular scenario, usually by considering the class variable as a node with a more important dependency role. Such models are commonly known as Bayesian Network Classifiers (BNC) [9] and are considered as start-of-the art methods in many domains.

Perhaps the most common BNC model is the popular Naive Bayes (NB) classifier [16], in which all the predictive features in $\mathbf{A}$ are considered to be independent given the class C. Although this is a strong assumption that rarely holds in real data, the performance of the NB classifier has proven to be competitive for many problems [17], and thanks to its efficient training is a suitable candidate for many practitioners. This independence assumption defines a fixed structure for the model, and thus it is not necessary to induce the dependences from the training data, translating into high computational savings. The graphical representation of NB classifier is illustrated in Fig. 1a.

Training a NB classifier is reduced to estimate its parameters from the data, where using a maximum likelihood estimation (MLE) algorithm along with a smoothing strategy such as Laplace, has a computational complexity of $\mathcal{O}(nm)$ and a spatial

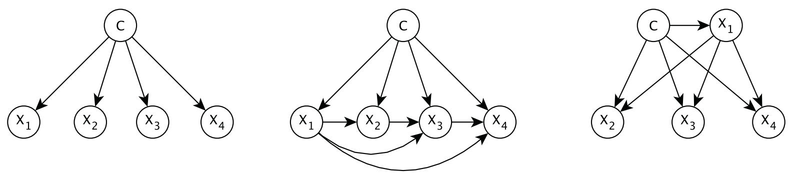

Fig. 1. Augmented Naive Bayes classifiers structures.

complexity of $\mathcal{O}(nl\bar{v})$ which is also the spatial complexity of the resulting model.

There has been a continuous effort to improve the performance of BNCs by proposing improvements over the NB framework. The most usual approaches try to relax its independence assumption by allowing additional dependences between the features to be included in the model. The resulting classifiers are commonly called Augmented Naive Bayes classifiers. Learning these models imply an additional computation effort in the learning process of the algorithm but usually paired with a substantial improvement on the classifier performance. We will describe in depth some of the most known techniques.

2.2.1. Tree-Augmented Naive Bayes

The Tree Augmented NB (TAN) model [15] relaxes the conditional independence assumption by allowing each predictive attribute to depend on an extra variable in addition to the class. To select this particular dependences, a structural learning algorithm is proposed in which the well known Chow and Liu method is used to build a maximum spanning tree by means of the Conditional Mutual Information (CMI):

$$MI(X_i, X_j \mid C) = \sum_{c\in\Omega_C} \sum_{x_i\in\Omega_{X_i}} \sum_{x_j\in\Omega_{X_j}} p(x_i, x_j, c) log\left(\frac{p(x_i, x_j \mid c)}{p(x_i \mid c)p(x_j \mid c)}\right) \tag{3}$$

Once the tree is obtained a DAG is built by arbitrarily selecting a root node and orienting the edges in topological ordering; finally the class variable is added as a common parent to all nodes. This algorithm requires a computational complexity of $\mathcal{O}(mn^2)$ to compute the CMI for each pair of attributes, for which it builds a three-dimensional table with spatial complexity of $\mathcal{O}(ln^2\bar{v}^2)$. The computational complexity for building the maximum spanning tree is $\mathcal{O}(n^2 \log n)$. Parametric learning of the model can reuse the counts computed for the CMI and calculate the required probability tables with $\mathcal{O}(n\bar{v}l)$ complexity; the resulting TAN classifier requires an storage with spatial complexity $\mathcal{O}(ln\bar{v}^2)$.

2.2.2. k-Dependence Estimators

The k-Dependent Estimator [18] can be seen as a generalization of the previous idea, in which a number of k additional parents are allowed in the model for each predictive variable, see Fig. 1b. This algorithm can explore a wider spectrum of classifiers, starting with NB if $k = 0$ towards a full BN model as k increases. The structure of the classifier is learned by a three step algorithm which also relies on the Mutual Information and Conditional Mutual Information:

- A ranking σ is established between the predictive attributes by means of their mutual information with the class variable, $MI(X, C)$ (Eq. 4).

$$MI(X_i \mid C) = \sum_{c\in\Omega_C} \sum_{x_i\in\Omega_{X_i}} p(x_i, c) log\left(\frac{p(x_i, c)}{p(x_i)p(c)}\right) \tag{4}$$

- For each attribute X_i being $X_{\sigma(i)}$ its position on the previous ranking σ, we compute the conditional mutual information $MI(\cdot, X_i|C)$ (Eq. 3) given the class for the subset of attributes preceding X_i in σ: $\{X_1, \ldots, X_i - 1\}$. Then the best k attributes with the highest dependence are set as the parents of X_i.
- Finally the class variable C is added as a parent for all the predictive attributes.

Learning the network structure of a kBD classifier has a computational complexity of $\mathcal{O}(n^2m)$ and a spatial complexity of $\mathcal{O}(l\bar{v}n^2)$ as it also builds a three-dimensional table. Calculating the CPTs for that particular network takes $\mathcal{O}(n(m + \bar{v}^k))$ and needs an spatial capacity of $\mathcal{O}(kn\bar{v}^kl)$.

2.2.3. Averaged k-Dependence Estimators

The idea behind Averaged k-Dependence Estimators (AkDE) [19,20] is slightly different from the previous ones; instead of learning an augmented structure from a NB classifier avoids this costly operation by defining an ensemble of fixed structure simpler models. More concretely, an ensemble of Super Parent One-Dependence Estimators (SPODE, Fig. 1c) is learnt, in which, for each model, all attributes depend on the class and on an additional attribute which is named as the *super-parent*. For a value of $k = 1$, an A1DE classifier is composed of n SPODEs each one of them having a different $X \in \mathbf{A}$ attribute acting as the super-parent. The classification is obtained by averaging the individual predictions of the models in the ensemble.

For higher values of k the super parent attributes are constructed as the Cartesian product of k attributes for a total of $\binom{n}{k}$ SPODE models in the ensemble, increasing the predictive power of the classifier at the cost of a higher training time and space complexity. The A1DE classifier has proven to be a good trade-of between efficiency and accuracy, outperforming the other BNCs approaches in many domains.

Learning an AkDE classifier requires the computation of a $k + 2$ dimensional table to estimate the required parameters, with a computational complexity of $\mathcal{O}(mn^k)$ and a spatial complexity of $\mathcal{O}(\binom{n}{k}l\bar{v}^{(k+1)})$.

3. Distributed processing of massive datasets

The MapReduce (MR) framework was proposed in 2004 [4] in order to provide a highly scalable and flexible framework for data-oriented parallel computing, as a solution for the increasing need of processing massive amounts of data. Data analysis has become one of the key technological challenges of Information Technology, which is intended to provide substantial benefits and improvements in a wide range of scientific and commercial disciplines [3].

These technologies arose as a response to the increasing amount of digital data available. The data generation and storage capabilities have grown to the petabyte scale [3] over the last few years, being the results of cheaper hardware and novel processing technologies. The original design of MR introduced by Google served as an optimization layer for their search engine and the massive amount of dataset being generated by their systems. It

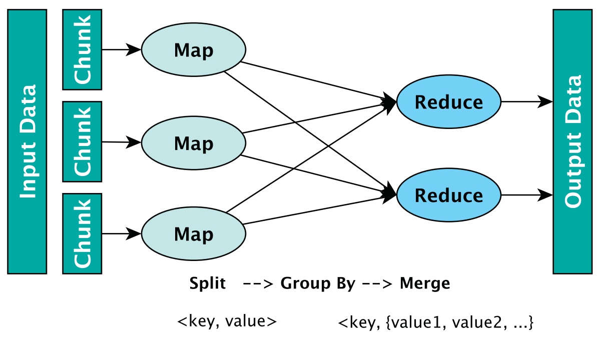

Fig. 2. Graphical representation of a MR flow.

turned out that MR could be also employed as a general computing framework under which many different applications could be defined [21].

The main potential of MR is the computing abstraction that it proposes, in which the whole processing is divided into smaller tasks of types *map* and *reduce* that are distributed and processed evenly along the cluster. The practitioner only has to be in charge of providing these two functions, avoiding to adapt the processing to the underlying architecture of the cluster or to the nature of the data. This framework provides an extremely scalable and fault-tolerant environment for parallel data processing.

A MR procedure follows a two step process in which the architecture of the cluster is organized in a master/slave scheme: One master node configures the job and distributes the computing tasks and input data among a collection of worker nodes which will carry out the processing. First of all, the input data is split into smaller partitions or chunks that are distributed among the worker nodes where they serve as the input of a given number of *map* tasks.

The data managed in a MR workflow is represented by pairs of the form ⟨*key, value*⟩. The initial input for each map task is usually given as an input partition of the original data, associated to an arbitrary key. Each map task process its chunk of data by applying a defined function to it, which generates a number of new intermediate ⟨*key, value*⟩ output pairs. The system collects, shuffles and sorts this intermediate output pairs by their corresponding keys, and sends the matching pairs sharing the same key as the input data of a number *reduce* tasks. This input consists on a pair formed by the common key and a list containing the associated values. Then, the reduce step is performed, in which all the sorted pairs are processed by applying a function to combine the grouped values; as a result, a final output is generated, usually consisting on new formatted ⟨*key, value*⟩ pair. Fig. 2 shows a detailed flow of a MR procedure.

The popularisation of the framework was brought by the itsopen implementation Apache Hadoop [6]; a collective effort of the open software community along with many private companies to provide a widely available software stack for Big Data preprocessing. The Hadoop ecosystem is broad, however two main components are prominent: The generic MapReduce implementation and the Hadoop Distributed FileSystem (HDFS) [22]. They allow to deploy inexpensive large clusters of computers as both a powerful execution engine as well as a reliable and fault-tolerant distributed data storage.

3.1. Apache Spark

The Apache Spark [7] framework can be seen as the next generation of distributed computing frameworks, and as an extension of MapReduce. They key difference between them is that Spark defines a new abstraction for the data being processed called *Resilient Distributed Datasets* (RDD) [23]; which allows the practitioner to define additional operations over the data without strictly sticking to map and reduces functions. However, the MapReduce paradigm is still in the core of the Spark platform, as the majority of the computations follow the same pattern in which several functions are individually applied to every unit of data and them shuffled through the network to be combined.

The main motivation for Spark to become the *de facto* standard for distributed data processing relies on architectural differences: While Hadoop MR relies on hard drives to spill intermediate data between every operation, Spark focuses on much more rapid main memory to maintain its data structures. This allows more complex execution paths to be defined as well as adds support for iterative processes.

Finally, another key factor for the popularity of this new framework is its accessibility to programmers by providing easy to use interfaces for most of the programming languages oriented to Data Science. In this paper we would focus on Spark as the platform in which the algorithms would be implemented and executed; however, our approach to define them would be theoretical and we would stick to the MapReduce framework, thus providing a general definition of the scalability and parallelism capabilities of these methods.

Table 2
Complexity of BNCs.

Classifier	Time	Space	Model	Data passes
NB	$\mathcal{O}(nm)$	$\mathcal{O}(n\bar{v}l)$	$\mathcal{O}(ln\bar{v})$	1
TAN	$\mathcal{O}(n^2m + n^2\log n + n)$	$\mathcal{O}(l\bar{v}^2n^2)$	$\mathcal{O}(ln\bar{v}^2)$	1
*k*DB	$\mathcal{O}(n^2m + n^2\log n + knm)$	$\mathcal{O}(l\bar{v}^2n^2)$	$\mathcal{O}(ln\bar{v}^k)$	2
A*k*DE	$\mathcal{O}(mn^k)$	$\mathcal{O}(l\bar{v}n^k)$	$\mathcal{O}(ln^k\bar{v}^{k+1})$	1

4. A general framework for distributed Bayesian Network Classifiers

In order to design scalable versions of the described algorithms we must identify those stages where parallel processing could have a bigger impact and provide higher computational savings. In MapReduce, as well as in any other data oriented distributed paradigm, the main execution bottlenecks appear when data is transported, specially when reading or writing from disks or when transferring large amounts of data through the cluster's network. Even when using a main memory based framework such as *Apache Spark*, we must design our algorithms to minimize this data movement in order to maximize the parallelism.

A first look into the family of BNC classifiers shows that a taxonomy can be established between those algorithms which require structural learning and those whose structure is fixed. A direct implication of this is the requirement of additional passes through the data, as the algorithms may require to read the dataset more than once for the different learning stages of the network, which is likely to increase their execution time. Although this should be a minor problem for traditional datasets, the overhead can be excessively demanding when the dataset is massive, specially if it does not fit into main memory.

A comprehensive summary on the complexity of the different classifiers can be found on Table 2. We may assume that the algorithms that require more than one data read will involve a significant overhead for larger datasets; however performing a single read through the data comes at the cost of lower accuracy results in the case of NB and a higher model complexity for AODE.

According to the previous analysis, a common bottleneck can be found for the described algorithms: The main computational burden of the learning procedures, either structural or parametric, is driven by the computation of multidimensional contingency tables from the training data. This can be easily seen, as both described metrics MI and CMI (Eqs 4, 3) require the estimation of probability distributions in the same way as the parametric learning for any BN model, if we use a frequentist approach such as MLE. Computing these contingency tables is the most demanding part of the algorithms with complexity $O(mn^k)$ where m corresponds with reading the complete training data and $k+1$ with the dimensionality of such table: i.e, estimating the parameters for a NB classifier requires learning a 2-dimensional table involving each attribute ($k=1$) and the class, whereas for the A1DE classifier a 3-dimensional table is required, involving each pair of attributes ($k=2$) and the class.

In general, these contingency tables can be represented by histograms or frequency counts distributions among the different configurations of a given set of attributes $\mathbf{S} \subseteq A \cup \{C\}$. We will compute the frequency of each occurrence for the joint set of states $\bigotimes \Omega_{X_i} \setminus X_i \in \mathbf{S}$ for the attributes in the subset $\mathbf{S} = \{X_1, \ldots, X_r, C\}$, where the contingency table, $\#_{\mathcal{D}}(\mathbf{S})$, stores the number of occurrences of each possible configuration defined in $\bigotimes \Omega_{X_i}$ in a given dataset $\mathcal{D}$.

Learning a particular type of BNC requires computing a given number of contingency tables, defined for a collection of sets of variables $\mathbf{\Psi} = \{\mathbf{S_1}, \ldots, \mathbf{S_v}\}$; e.g. learning a Naive Bayes classifier requires estimating the counts for each attribute and the class, and thus the sets of variables from which estimating the counts will be $\mathbf{\Psi} = \{\mathbf{S_i} = \{X, C\} \forall X \in \mathbf{A}\}$. In previous work, MapReduce algorithms for learning particular models were proposed [24], however, a common pattern can be established in order to define a general framework to cover all the proposed BNC algorithms. From now on, we will pose our problem in such general view, with the purpose of defining a common procedure to learn any of the described models; later on, we would be able to instantiate this framework to match a particular algorithm. This general procedure strategy is based on computing the contingency tables for a given set of attribute combinations $\mathbf{\Psi}$ that would be set according to the particular model selected.

Computing such contingency tables for a large volume of data is a problem that falls naturally on the MapReduce paradigm. In general, we can identify two different strategies to define a parallel scheme for obtaining the counts given a set of attribute/class combinations, each one of them intended to parallelize a different component of the scalability problem:

- **Horizontal parallelism** is the most straightforward approach which targets the m component of the complexity. It starts by splitting the training dataset into different chunks and distributing them across the available computing nodes. From there, the contingency tables are computed partially for each chunk of data and combination of attributes $\mathbf{S} \in \mathbf{\Psi}$. Finally the partial distributions can be collected and aggregated to recover the full distribution over the whole dataset.
- **Vertical parallelism** aims to distribute the computation for the different subsets in $\mathbf{\Psi}$. This strategy is meant for high dimensional domains where there is a large number of attributes, making the size of the model to grow with polynomial complexity. In these cases, a simple horizontal distribution of the dataset would imply a large overhead when collecting the different partial counts, so vertical parallelism can be used even combined with the previous approach. This is done by computing each contingency table as a separate parallel task. This strategy involves a vertical distribution of the dataset according to the the different subsets $\mathbf{S} \in \mathbf{\Psi}$; and, as the subsets may overlap, this involves a complex combinatorial problem in order to minimize data replication and to assure computational balance across the different nodes.

4.1. Horizontal parallelism in MapReduce

According to the previous definition, our problem is reduced to estimate the contingency tables for a particular number of attribute subsets $\mathbf{\Psi} = \{\mathbf{S_1}, \ldots, \mathbf{S_v}\}$ over a training dataset $\mathcal{D}$. A horizontal strategy starts by partitioning the data into chunks $\{\mathcal{D}_1, \ldots, \mathcal{D}_h\} \mid h \leq m$ that will be distributed into, ideally, h map tasks. Each of these tasks computes a partial contingency table $\#_i(\mathbf{S_j})$ for the available local chunk $\mathcal{D}_i$, and for each subset $\mathbf{S_j} \in \mathbf{\Psi}$. Then, each map task emits a collection of key-value pairs $\langle \mathbf{S_j}, \#_i(\mathbf{S_j}) \rangle$ for each subset, associating them with their corresponding distribution. A graphical representation of this process is illustrated in Fig. 3.

In the the reduce stage, these pairs will be grouped by their keys (representing a subset of attributes) and sent to the corresponding reduce tasks, where the ideal number of tasks is v, one for each subset. The reduce stage aggregates the partial distributions for the different attribute subsets in parallel and emits a new key-value pair containing the corresponding subset and its full contingency table computed for the complete dataset.

The previous framework can be instantiated into any of the described BNSs by providing a particular range of attribute subsets for $\mathbf{\Psi}$.

4.1.1. Distributed Naive Bayes

Being the most simple of the classifiers in our study, we will start by characterizing the previous MapReduce algorithm for the NB learning algorithm. As this model does not require a previous structural learning stage we will just define the estimation via MLE of the parameters. Concretely, we need to set the topology of the multidimensional contingency tables to be computed by correctly identifying $\mathbf{\Psi}$; in this case, the NB classifier is encoding the conditional probabilities for each attribute and class, thus defining a 2-dimensional table with complexity represented by the subsets $\mathbf{\Psi^1} = \{\mathbf{S_i} = \{X, C\} \forall X \in \mathbf{A}\}$.

4.1.2. Distributed A*k*DE

Owing to the lack of structural learning in the A*k*DE classifier we can proceed exactly as with the NB model. However, the number of parameters to compute would depend on the value of k, which affects the dimension of the contingency tables to estimate. Concretely, the dimension of such tables would be 3 for A1DE with $\mathbf{\Psi^2} = \{\mathbf{S} = \{X_i, X_j, C\} \forall X_i, X_j \in \mathbf{A}\}$ and a 4-dimensional table with $\mathbf{\Psi^3} = \{\mathbf{S} = \{X_i, X_j, X_k, C\} \forall X_i, X_j, X_k \in \mathbf{A}\}$ in the case of A2DE.

4.1.3. Distributed TAN

The case of the TAN classifier is particular as it requires a previous structural learning stage before estimating the parameters. However, this second stage can reuse the statistics computed for the structural learning part, obtaining the parameters of the final model without performing an additional pass through the data. In order to induce the structure of the model, this algorithm builds a 3-dimensional table identically to A1DE with $\mathbf{\Psi^2}$ to compute the CMI statistics. Then, the statistics are used to build the tree model by using the maximum spanning tree (MST) algorithm.

At this stage we have already compressed the data and removed m component of the problem, thus the complexity of the MST algorithm is n^2 which is independent on the number of instances in the data. In practice, the cost of this operation should not have a real impact on the overall performance of the algorithm when the size of the dataset m is large; however, given that this stage is sequential, the combination of a large number of attributes in a small dataset can introduce an overhead in the execution time of the learning algorithm.

After the structure of the classifier is induced, the parameters can be estimated from the same contingency tables computed to

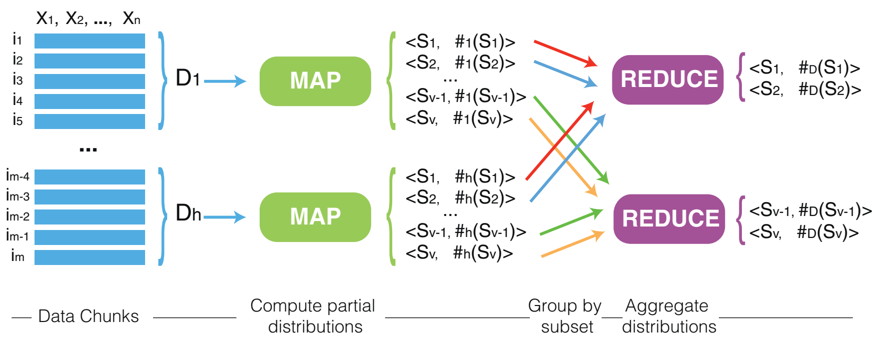

Fig. 3. Graphical scheme of the proposed MapReduce algorithm. Horizontal distribution of data and computations.

obtain the CMI. This stage would be efficient as well as it is also independent from the size of the data.

4.1.4. Distributed kDB

Learning a kDB model involves two different passes through the data, one during the structural learning stage and a second pass to estimate the parameters. Unlike the TAN algorithm, the statistics computed during the first step can not be reused to estimate the CPTs, as the cardinality of the latter would be higher for values of $k > 1$. The structural learning complexity is identical to the described for the TAN classifier in which 3-dimensional contingency tables are computed for $\mathbf{\Psi^2}$ in order to obtain the CMI for each pair of attributes and the class. Once the metric has been computed, the graph structure is induced by the algorithm described in 2.2.2, which again turns out to be sequential but can be considered efficient when compared to the computation of the previous statistics for being independent from the size of the data.

After the structural learning stage, it is necessary to estimate the parameters of the model. However, this step would require a second pass through the data, as the size of the new statistics would require obtaining new contingency tables for a different set of attribute combinations. We require to estimate a CPT of each node in the model so, in this case, the complexity of the process is linear regarding the number of attributes n. For each distribution we must obtain the frequency counts of each attribute given the different combinations of its parent in the induced structure; by using our proposed framework, this is described given the set of attributes $\mathbf{\Psi} = \{\mathbf{S_i} = \{X_i\} \cup pa(X_i) \forall X_i \in \mathbf{A}\}$.

4.2. Vertical parallelism in MapReduce

The previous horizontal strategy implies that each parallel task would compute partial contingency tables for all the combinations in $\mathbf{\Psi}$, which, in terms of spatial complexity is equivalent to holding the full set of parameters in memory for each task. Adding vertical parallelism would alleviate this constraint by allowing each task to target only a partial number of sets from $\mathbf{\Psi}$. The strategy can be combined with the previous one being identical: Partial frequency counts are estimated during the map stage for a given chunk, and then aggregated in the reduce stage; however, in a vertical approach, a number of map tasks would target the same chunk D_i to compute the partial contingency table for a subset of the combinations in $\mathbf{\Psi}$. Fig. 4 shows an example in which vertical parallelism is added to the horizontal basic scheme proposed in Fig. 3, notice that the shuffle and reduce processes are identical ones from the previous definition. In an extreme case with no horizontal partitions, the reduce stage can be suppressed, as the contingency tables estimated during the map stage would not be partial.

This scheme would change the way in which the parallelism is distributed, by requiring less partitioning of the data and a different configuration of map tasks. In Section 5.3.1, a practical example of how this approach could be effectively configured on a real cluster is provided.

As opposed to horizontal parallelism where data partitioning is trivial, the main drawback of vertical parallelism is that finding optimal data partitions for different subsets of the elements in $\mathbf{\Psi}$ turns out to be a very complex combinatorial problem, in which the data must be divided by optimally reducing replication. Different strategies have been proposed to solve this problem [25] in which smart statistical techniques are used in order to guarantee a balance according to amount of data replicated for each vertical map task.

In this paper we will assume a trivial strategy in which the data is not partitioned but replicated entirely, so the different tasks can access the variables they need for their computations. This indeed adds a notable overhead to the execution as a large amount of redundant data would be read from disk and transported through the network.

5. Empirical evaluation

We have conducted a series of experiments to evaluate the performance of the different algorithms for different levels of parallelism, while combining the horizontal and vertical strategies proposed. We present two series of experiments, the first one based on synthetic generated data, which will allow us to evaluate the improvement in a wide range of scalability problems. Finally we present a series of experiments using real word data obtained from challenges to show the performance of the algorithms in particular domains.

In both cases we report several metrics regarding the execution time of the algorithms when different computational resources are present. The accuracy of the classifiers is not evaluated on this paper as the algorithms are identical to their corresponding sequential version, and thus the resulting models will maintain its discriminative power.

5.1. Execution environment

We will be using a cluster of computers consisting of one master and six slave nodes equipped with dual Intel Xeon E5-2609v3 1.90 GHz hexacore processors and 64GB of RAM each one. Each worker node is running the HDFS file system on 4x1TB disks and managed by Cloudera CDH 5.5 distribution.

The MapReduce environment selected in Spark 1.6.0 on a standalone deploy. We will launch different configurations of the cluster by providing a different amount of resources, in order to test the behaviour of the algorithms for different architecture layouts.

5.2. Synthetic data experiments

The aim of conducting experiments on synthetic data is to provide an intensive study on the scalability of the proposed framework. These experiments are based on an incremental synthetic

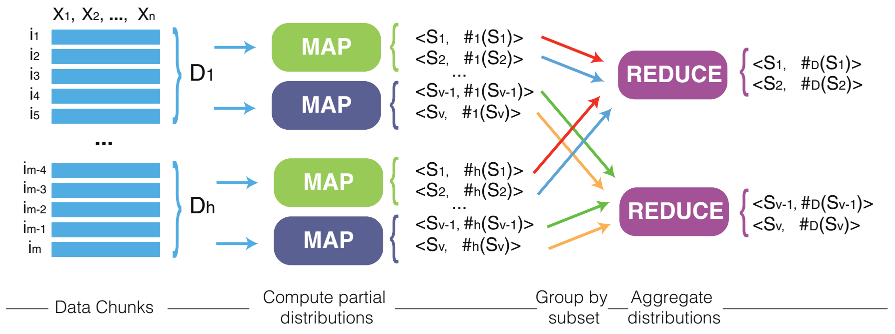

Fig. 4. Graphical scheme of the proposed MapReduce algorithm. Horizontal distribution of data and vertical combination of computations distribution. Notice that this time there are two different types of map tasks being applied simultaneously to the data on two different levels: On the horizontal level, a collection of map tasks would target the same data chunk to compute the Ψ contingency tables; on the vertical level, each one of these particular map task would be configured to compute only a subset of the Ψ contingency tables. This strategy reproduces the previous layout from Fig. 3, however the data chunks would be smaller and the parallelism would be obtained by the vertical distribution of the computations. Notice that the reduce phase is identical.

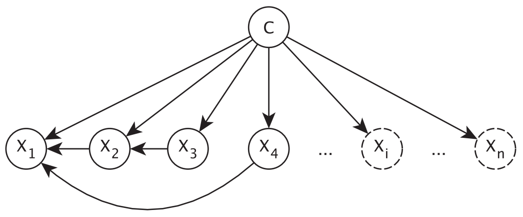

Fig. 5. Structure of a random synthetic network generated where the number of attributes is indefinite.

Table 3
Properties of the synthetic datasets generated for the study, with the number of variables and instances increasing with different proportions. The random model has been created by assuring an average of 3 parents for each model, with discrete variables with 3 states.

m\n	200	400	600	800
4M	1.6GB	3.2GB	4.8GB	6.4GB
8M	3.2GB	6.4GB	9.6GB	12.8GB
16M	6.4GB	12.8GB	19.2GB	25.6GB
32M	12.8GB	26.6GB	38.4GB	51.2GB

model which allow us to obtain different problems covering a wide spectrum of the scalability challenge, regarding both the number of attributes and the number of instances. For that we define a random synthetic Bayesian network model from which we will sample a series of datasets with different properties.

More concretely, we have constructed a random model by using a modified version of the procedure described in [26]. In our case, we will focus on creating a model to match a supervised classification problem; more concretely, we will construct an augmented Naive Bayes model over an indefinite set of variables $\{C\} \cup \{X_1, \ldots, X_n\}$, in which all attributes are dependent on the class and on a random set of parents. To set such dependencies, a topological ordering is imposed over the variables and an average of p parents are randomly selected for each node. The parameters are then obtained accordingly to the given structure by following uniform distributions. A graphical representation of a possible model is shown on Fig. 5. We have generated a range of models with different number of attributes and several samples with incremental number of instances for each one of them, details are shown in Table 3.

This approach fits specifically in this study as we are not evaluating the classification performance of the models, we are assuming that it would be equivalent to the performance in traditional domains that has been tested many times in the literature. Such performance in these specific domains should be compared with relevant data and additional indicators, this has been left for a future work intentionally.

Fig. 6 shows the summarized performance for each learning algorithm over the synthetic samples when using the proposed MapReduce implementation with 64 parallel tasks. The most noticeable result is perhaps the huge difference between Naive Bayes and the other classifiers, showing that an algorithm with linear complexity is not comparable with the others having quadratic complexity. The second important observation is that neither the sequential structural learning stages in TAN and KDB nor the second pass through the data of the latter have a significant impact on the performance of these classifiers; as we can compare their results are equivalent to A1DE.

We can observe that the complexity of the algorithm is driven by the dimensionality of the problem, as the execution time experiments a larger increase when the number of attributes is incremented. It is also relevant to evaluate the extent to which this MapReduce approach increases the efficiency of the algorithms for a given configuration of parallel resources in the cluster. For that, we have conducted an exhaustive comparison among all the problems using a different amount of parallelism, that is shown in Fig. 7.

These experiments confirm that using a higher number of parallel tasks is providing a consistent scalable behaviour for the algorithms, which is in fact larger in the case of more complex models like A1DE. These new results are also reflecting that scalability in terms of the dimensionality of the problem is more costly that escalating the algorithms for larger datasets in terms of examples.

Using a MapReduce approach introduces a certain overhead in the execution as the number of parallel tasks in increased, Fig. 8 compares the scalability of the different executions of the algorithm for a given problem by measuring the the speed-up and efficiency of the execution. For the speed-up, the results are compared with those using the minimum amount of parallelism (4 tasks), the efficiency is obtained as the ratio between speed-up and the actual number of parallel tasks; this measure can be interpreted as the proportional performance of a parallel task (a physical core) compared to a single sequential task executing the algorithm.[2]

[2] An optimal theoretical value for efficiency would be 1, where each parallel task would perform equally as compared to the single sequential task, thus obtaining a perfect complete parallelism. In practice this is not possible, as parallelism involves an amount of overhead which is proportional to the number of active tasks, we can observe that in fact, efficiency is reduced proportionally to the number of cores used in the execution.

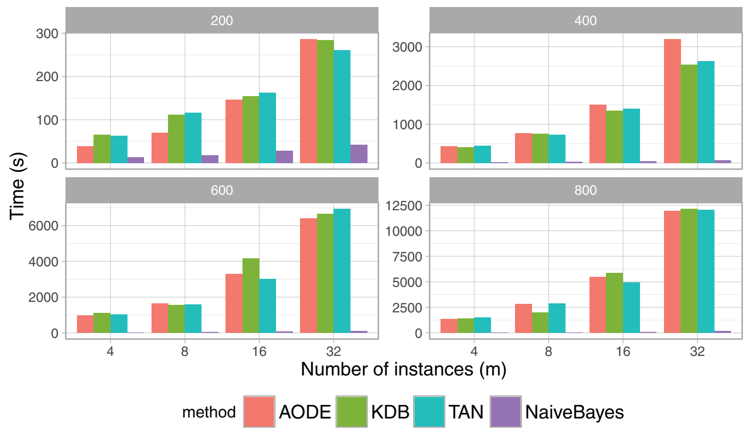

Fig. 6. Results for each 4 learning algorithms and for each synthetic problem generated. The y-axis shows the execution time of the algorithms, notice that each scale is unique for each plot. The x-axis shows the different number of samples (m) for the given problem while the number of attributes (n) is separated into the different charts.

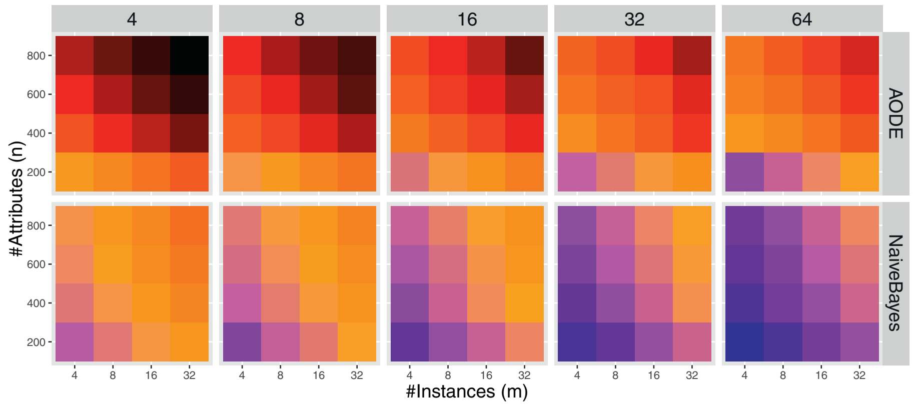

Fig. 7. Execution time (logarithmic scale) for all combination of number of attributes (y-axis), number of instances (y-axis) and number o parallel tasks (different plots). The color represents the execution time where blue is the lowest towards orange, red and black being the greatest.

Table 4
Properties of the real datasets included in the experimentation.

Name	#Attributes (n)	#Instances(m)	Size	Ref
Splice	141	50M	14.20GB	[27]
ECBLD14[a]	631	4M[a]	5.26GB	[28]
Epislon	2000	500K	1.90GB	[29]

[a] This is a sample from the original dataset.

These results reflect that the overhead is higher in the case of Naive Bayes and in the presence of a lower number of attributes in the problem. Our conclusion is that the proposed implementation is favourable for larger datasets and high dimensional problems when the size of the cluster grows.

5.3. Real data experiments

As an additional evaluation of our proposal, we have conducted a range of experiments over a series of real world problems obtained from public challenges. These real datasets present different properties regarding scalability, a summary is shown in Table 4.

Results for execution time are shown in Fig. 9, where we can observe the same behaviour of the algorithms in the case of *splice* and *EBLD14** but an anomaly in the case of *epsilon* and the TAN classifier. This dataset is unique in our study given the small amount of instances that it contains, specially if we take into account that is has a very large dimensionality. The most direct explanation for this overhead is that the sequential stage of the structural part of the TAN algorithm is introducing a noticeable overhead as the complexity of computing the contingency tables is relatively small given the size of m. This shows that in real problems where the proportion of n and m is not balanced, the performance of the algorithms that require two steps (TAN and kDB) can be compromised.

A final experiment compares our proposal of a distributed classifier against a sequential version. It is interesting to contrast the performance of each Spark task against a single core running optimized Java code. For that we have implemented the A1DE algorithm in Java using the *MOA* library [30] to read big datasets as a data stream, thus eliminating any memory restrictions. The code has been carefully optimised in order to provide a fair comparison between the two approaches when focusing on a real context. Fig. 10 shows the runtime, speed-up and efficiency progression for executions of the A1DE algorithm using different amounts of resources, including the sequential approach. We can confirm that the initial impact on scalability, when moving from sequential (1 task) to MapReduce (4 tasks), is significant, however the scalability is more evident when additional Spark nodes are included. A certain amount of overhead can be noticed when the number of parallel tasks grows larger, however, the speed-up plots shows that in the case of *splice* this overhead is almost non-existent, which means that using MapReduce and Spark favours big problems with a massive amount of examples.

5.3.1. Vertical parallelism

As stated in the previous section, in this paper we have just explored a naive approach on vertical parallelism. Distributing the

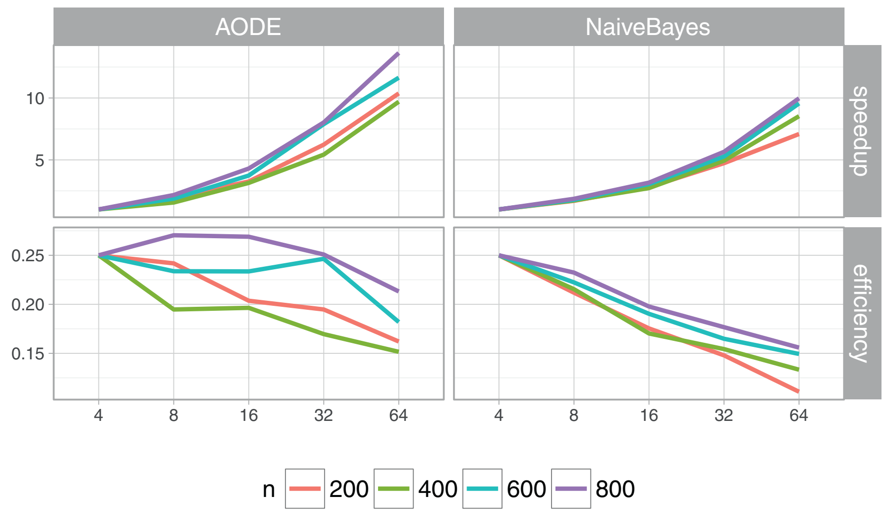

Fig. 8. Speed-up and efficiency for the experiments described in Fig. 7, the values are computed as the execution time ratio (y-axis) between a given experiment for a particular problem and given number of parallel tasks (x-axis) and the same configuration using only 4 parallel tasks. The different lines represent the average speed-up and efficiency for the different samples sharing the same dimensionality with different number of instances. Efficiency is computed as the ratio between the speed-up and the number of parallel tasks.

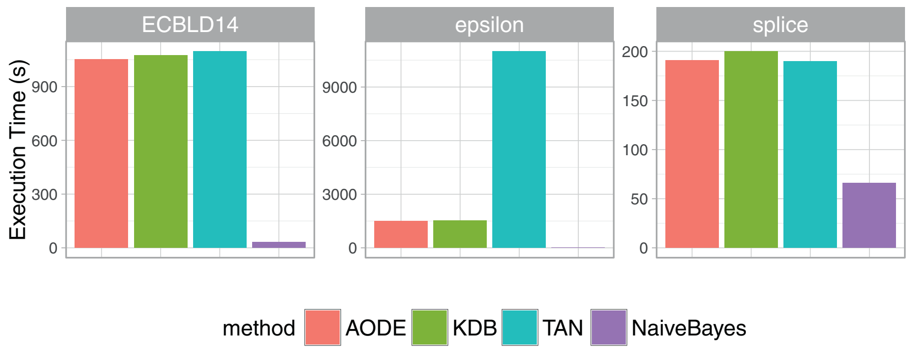

Fig. 9. Results for each 4 learning algorithms and for each real dataset. The y-axis shows the execution time of the algorithms, notice that each scale is unique for each plot. The x-axis shows the different algorithms.

data across the cluster vertically is not a trivial task when the computation carried out in each tasks involves subsets of several variables that may overlap, as it is the case on A*k*DE, TAN or *k*DB but not on Naive Bayes. Additional data and computation distribution schemes may be explored in future work, where complex metrics may used to ensure load balance and data locality in the cluster; however, in this paper we have conducted a series of experiments that follow a very trivial solution to the distribution problem, which is replicating all the data for all the nodes, so every task will have access to the full set of variables. This solutions comes at the cost of a very large overhead introduced by the additional amount of data distribution and replication that is required.

The proposed distributed implementation is compatible with a vertical approach so we have adapted the A1DE algorithm to combine both horizontal and vertical approaches. Fig. 11 shows the results for this new implementation, in which the runtime is indicated as a proportion for each combination of vertical vs horizontal tasks and each problem. More concretely, we run the algorithm using always 64 parallel tasks, but with a different setup regarding horizontal (h) vs vertical (v) tasks configurations ($h \cdot v$): 64 · 1, 32 · 2, 16 · 4, 8 · 8.

We can observe a notable overhead in the bigger dataset *splice*, however it is less significant for the other two datasets and even non-existent. In fact, we can observe that in the high dimensional domains the vertical approach is even more efficient that the pure horizontal one, especially if m is not very large like in the case of *epsilon*. This supports the idea that vertical parallelism is beneficial for high dimensional domains.

5.3.2. Size of the models

Another interesting indicator for our proposal is the resulting size of the models. Although spatial complexity of the resulting model has never been a main concern when dealing with traditional sized problems, it can become a barrier for high dimensional domains, specially when learning models with an order of complexity higher than linear. This is the case for A1DE and A2DE where the size of the resulting model grows with quadratic and cubic progressions along with the dimensionality of the classifier.

Fig. 12 shows a graphical comparison for the different classifiers and problems. It is clear that the difference between A1DE and the other approaches is huge, the former having model sizes that could impact on the performance of the classification step. However, the real problem comes when A2DE and higher order versions of A*k*DE are considered as suitable candidates for solving high dimensional problems. Table 5 shows the model sizes for A1DE and A2DE, where we can observe that learning an A2DE model becomes an intractable problem even for the proposed domains. The only plausible experiment was *splice*, for which we obtained a critical size of the classifier, the remaining experiments exceed the available resources and the size was just estimated. It is clear that new approaches should be proposed in order to learn more powerful classifiers like A2DE in those domains, for which

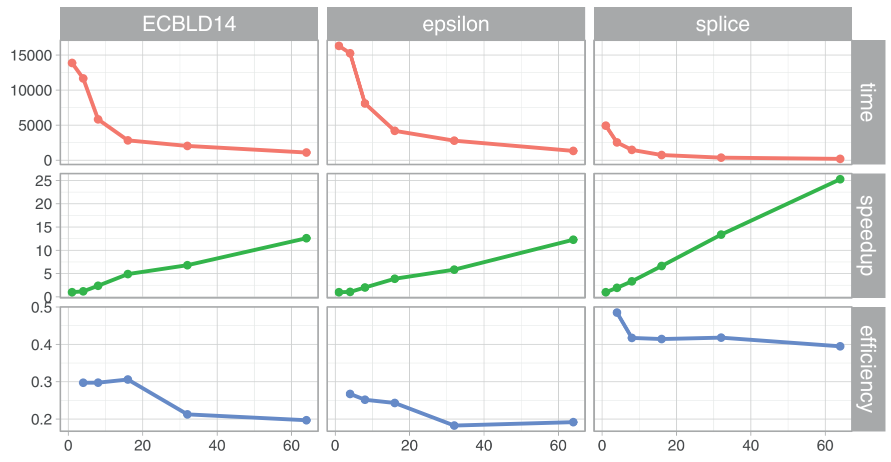

Fig. 10. Progression for (from top to bottom) runtime, speed-up and efficiency, for the A1DE algorithm using different amount of resources for parallelism for each dataset. The left most value in each plot represents $h = 1$ which corresponds with a sequential execution of the algorithm. The following execution correspond with the sequential approach followed by several MapReduce implementation with an increasing number parallel tasks (x2). The speed-up is computed as the ratio of each execution and the corresponding sequential one, and the efficiency is the ratio of the speed-up and the number of parallel tasks.

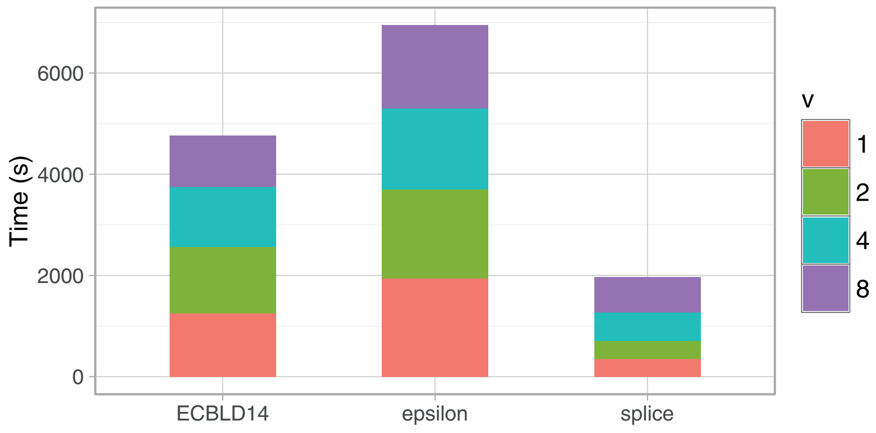

Fig. 11. Runtime for the A1DE classifier and the three real datasets for different configurations of horizontal and vertical parallel tasks. The stacked columns show the proportion for using different configurations when distributing 64 parallel tasks.

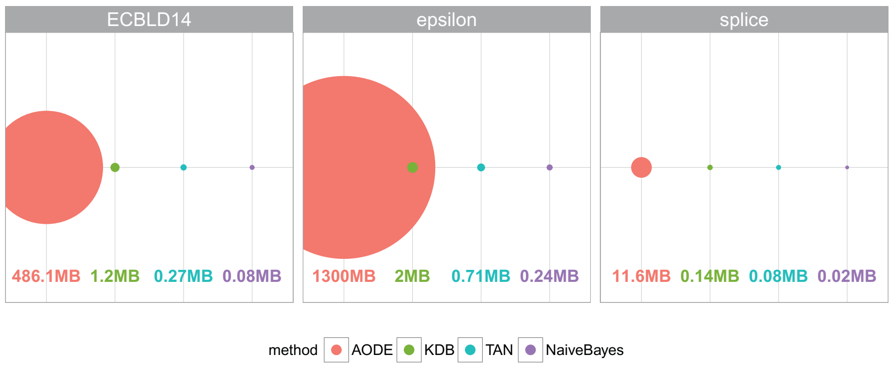

Fig. 12. Sizes of the resulting models for each classifier and problem. The size of the circles represent approximate proportions regarding the size of the model for each dataset. They show to what extent A1DE is generating models whose size is in a different scope when compared to the other algorithms.

Table 5
Sizes of the resulting model for the A1DE and A2DE classifiers and for each one of the real datasets. Boldfaced values represent approximations, due to the impossibility of running those experiments.

	Splice	ECBLD*	Epsilon
A1DE	11.60 MB	486.10 MB	1300.00 MB
A2DE	2059.07 MB	**≈79.16 GB**	**≈3427.53 GB**

some of the constrains of the algorithm must alleviated to reduce the size of the model.

6. Conclusions

In this paper we have analysed the most popular BNCs models and their corresponding learning algorithms, aiming to propose a suitable adaptation for large scale and high dimensional domains. Our main focus has been on Big Data technologies such as

MapReduce, proposing a solution that would fit into the current Cloud Computing and high performance distributed programming paradigms. We have introduced a general computational framework that can be instantiated to learn any of the considered models, adding also elasticity and scalability over a very wide spectrum of problems by targeting massive volumes of data with either a large number of examples or attributes.

We have also extended our proposal by proposing different strategies for distributing the computations and the data along a computing cluster, also introducing new future lines of research to optimize this framework for more complex domains. More concretely, we have defined a basic approach to verticalization of parallel computing, in addition to the usual horizontal strategy that is followed by most of the MapReduce adapted algorithms in Machine Learning. This vertical strategy could be further extending with smart procedures to balance the data replication among the different vertical map tasks to optimize the scalability of the algorithms for high dimensional domains.

A particular implementation of this framework has been proposed, under the state-of-the-art Apache Spark computing framework. This software has been tested on a large and varied benchmark of synthetic and real problems, for which numerous performance indicators have been obtained and discussed. In addition to this benchmark, several configurations of the computing architecture has been tested to provide a general overview of the scalability properties of our proposal.

Finally, some of the main pitfalls of classical BNCs have been identified in order to propose additional research lines. In future works, we will define new classifier models specifically adapted for these complex domains and being native to the MapReduce/Spark programming paradigm. In addition, the software package released along with this paper is aimed to fit the requirements of other researchers and practitioners and it is our purpose to maintain it and spread it among the community.

Acknowledgments

This work has been partially funded by FEDER funds and the Spanish Government (MINECO) through project TIN2013-46638-C3-3-P and by JCCM through project PEII-2014-049-P. Jacinto Arias is also funded by the MECD grant FPU13/00202.

References

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, Big Data: The Next Frontier for Innovation, Competition, and Productivity, Technical Report, 2011. June
[2] A. McAfee, E. Brynjolfsson, T.H. Davenport, D. Patil, D. Barton, Big data, the management revolution, Harvard Bus. Rev. 90 (10) (2012) 61–67.
[3] M. Chen, S. Mao, Y. Zhang, V.C. Leung, Big Data: Related Technologies, Challenges and Future Prospects, Springer, 2014.
[4] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM (2008) 1–13.
[5] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action (2011).
[6] T. White, Hadoop: The Definitive Guide, O'Reilly Media, Inc., 2012.
[7] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets., HotCloud 10 (2010).
[8] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., Mllib: Machine Learning in Apache Spark, arXiv:1505.06807(2015).
[9] C. Bielza, P. Larrañaga, Discrete Bayesian network classifiers, ACM Comput. Surv. 47 (1) (2014) 1–43, doi:10.1145/2576868.
[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, ACM SIGKDD Explor. Newsl. 11 (1) (2009) 10–18.
[11] Y. Zhai, Y.-S. Ong, I.W. Tsang, The emerging "big dimensionality", Comput. Intell. Mag., IEEE 9 (3) (2014) 14–26.
[12] D. Laney, 3D data management: controlling data volume, velocity and variety, META Group Res. Note 6 (2001) 70.
[13] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Elsevier Science, 2014.
[14] T.D. Nielsen, F.V. Jenses, Bayesian Networks and Decision Graphs, Springer Science & Business Media, 2009.
[15] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Mach. Learn. 29 (2-3) (1997) 131–163.
[16] P. Domingos, M. Pazzani, On the optimality of the simple Bayesian classifier under zero-one loss, Mach. Learn. 29 (2-3) (1997) 103–130.
[17] I. Rish, An empirical study of the naive Bayes classifier, in: IJCAI Workshop on Empirical Methods in Artificial Intelligence, 3, 2001, pp. 41–46.
[18] M. Sahami, Learning limited dependence Bayesian classifiers, in: Second International Conference on Knowledge Discovery and Data Mining, 1996, pp. 91–94.
[19] G.I. Webb, J.R. Boughton, Z. Wang, Not so naive Bayes: aggregating one-dependence estimators, Mach. Learn. 58 (1) (2005) 5–24.
[20] G.I. Webb, J.R. Boughton, F. Zheng, K.M. Ting, H. Salem, Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly naive Bayesian classification, Mach. Learn. 86 (2) (2011) 233–272, doi:10.1007/s10994-011-5263-6.
[21] A. Rajaraman, J.D. Ullman, J.D. Ullman, J.D. Ullman, Mining of Massive Datasets, 1, Cambridge University Press Cambridge, 2012.
[22] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, IEEE, 2010, pp. 1–10.
[23] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association, 2012.
[24] J. Arias, J.A. Gamez, J.M. Puerta, Scalable learning of k-dependence Bayesian classifiers under mapreduce, in: Trustcom/BigDataSE/ISPA, 2015 IEEE, 2, IEEE, 2015, pp. 25–32.
[25] A.L. Madsen, F. Jensen, A. Salmerón, M. Karlsen, H. Langseth, T.D. Nielsen, A new method for vertical parallelisation of TAN learning based on balanced incomplete block designs, Lect. Notes in Artif. Intell. 8754 (2014) 302–317.
[26] J.I. Alonso-Barba, L. delaOssa, J.A. Gámez, J.M. Puerta, Scaling up the greedy equivalence search algorithm by constraining the search space of equivalence classes, Int. J. Approx. Reas. 54 (4) (2013) 429–451.
[27] S. Sonnenburg, V. Franc, COFFIN: a computational framework for linear SVMs, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 999–1006.
[28] J. Bacardit, P. Widera, A. MÃ!'rquez-Chamorro, F. Divina, J.S. Aguilar-Ruiz, N. Krasnogor, Contact map prediction using a large-scale ensemble of rule sets and the fusion of multiple predicted structural features, Bioinformatics 28 (19) (2012) 2441–2448, doi:10.1093/bioinformatics/bts472.
[29] G.-X. Yuan, C.-H. Ho, C.-J. Lin, An improved glmnet for l1-regularized logistic regression, J. Mach. Learn. Res. 13 (1) (2012) 1999–2030.
[30] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604.