Prog Artif Intell (2013) 2:1–11
DOI 10.1007/s13748-012-0035-5

REVIEW

A survey of methods for distributed machine learning

Diego Peteiro-Barral · Bertha Guijarro-Berdiñas

Received: 15 September 2012 / Accepted: 28 October 2012 / Published online: 15 November 2012

Abstract Traditionally, a bottleneck preventing the development of more intelligent systems was the limited amount of data available. Nowadays, the total amount of information is almost incalculable and automatic data analyzers are even more needed. However, the limiting factor is the inability of learning algorithms to use all the data to learn within a reasonable time. In order to handle this problem, a new field in machine learning has emerged: large-scale learning. In this context, distributed learning seems to be a promising line of research since allocating the learning process among several workstations is a natural way of scaling up learning algorithms. Moreover, it allows to deal with data sets that are naturally distributed, a frequent situation in many real applications. This study provides some background regarding the advantages of distributed environments as well as an overview of distributed learning for dealing with "very large" data sets.

Keywords Machine learning · Large-scale learning · Data fragmentation · Distributed learning · Scalability

1 Introduction

Automatic learning has become increasingly important due to the rapid growth of the amount of data available. In the year 2000, the total amount of information on the Web varied somewhere between 25 and 50 terabytes [1]. By 2005, the total size was approximately 600 terabytes [2]. Nowadays, the total amount of information is almost incalculable. This unrestrainable growth of data opens the way for new applications of machine learning. Automatic data analyzers are needed since a human, even an expert, cannot look at a "very large" data set and plausibly find a good solution for a given problem based on those data. In this situation, new challenges are raised regarding the scalability and efficiency of learning algorithms with respect to computational and memory resources. Practically, all existing implementations of algorithms operate with the training set entirely in main memory. If the computational complexity of the algorithm exceeds the main memory then the algorithm will not scale well, will not be able to process the whole training data set or will be unfeasible to run due to time or memory restrictions. However, increasing the size of the training set of learning algorithms often increases the accuracy achieved by classification models [3], and thus, in order to handle "very large" data sets, a new and active research field emerges, large-scale learning [4,5]. It intends to develop efficient and scalable algorithms with regard to accuracy and to requirements of computation (memory, time and communication needs). Large-scale learning has received considerable attention in the recent years and many successful techniques have been proposed and implemented [6–8]. The different techniques proposed in the literature can be categorized into three main approaches where, in most cases, techniques from separate categories are independent and can be applied simultaneously. The three main approaches are to:

- design a fast algorithm,
- use a relational representation, and
- partition the data [9].

D. Peteiro-Barral (✉) · B. Guijarro-Berdiñas
Department of Computer Science, University of A Coruña,
Campus de Elviña s/n, 15071, A Coruña, Spain
e-mail: dpeteiro@udc.es

B. Guijarro-Berdiñas
e-mail: cibertha@udc.es

Following the last approach, one of the most promising research lines for large-scale learning is distributed computing since allocating the learning process among several workstations is a natural way of scaling up learning algorithms. In this study, we will explore this approach by presenting some of the most popular distributed learning algorithms.

Regarding distributed algorithms, we can distinguish between two different contexts. On the one hand, there are models based on distributing data artificially between different computational systems, and normally, they move data between them during the execution of the distributed algorithm. In this context, we can find classical techniques such as [10,11] and nowadays improvements like [12,13]. This context is connected to a new computing paradigm namely "big data" processing [14–16]. For example, the general approach is offered by the Hadoop philosophy [17].

In this survey, we focus our attention on a second context in which data are naturally distributed but moving them between systems is not permitted. This constraint restricts some techniques available for developing distributed algorithms. Note, however, that these classes of algorithms will be valid in both naturally and artificially distributed data.

2 Background

In the form of multi-core processors and cloud computing platforms, powerful parallel and distributed computing systems have recently become widely accessible. This development makes various distributed and parallel computing schemes applicable to a variety of problems that have been traditionally addressed by centralized and sequential approaches. In this context, machine learning can take advantage of distributed computing to manage big volumes of data or to learn over data that are inherently distributed as can be, for example, wireless sensors in a smart city.

2.1 Data fragmentation into distributed databases

In general, the data set used for training consists of a set of instances where each instance stores the values of several attributes. The distributed nature of such a data set can lead to at least two common types of data fragmentation [18]:

- Horizontal fragmentation wherein subsets of instances are stored at different sites.
- Vertical fragmentation wherein subsets of attributes of instances are stored at different sites.

The great majority of distributed data sets are horizontally fragmented since it constitutes the most suitable and natural approach for most applications. Vertical fragmentation is solely useful where the representation of data could vary along time by adding new attributes.

Finally, a third type of data fragmentation is mixed fragmentation wherein subsets of instances, or subsets of attributes of instances, are stored at different sites. It is defined as a process of simultaneously applying horizontal and vertical fragmentation on a data set.

2.2 Why distributed databases? Some initial motivations

In a company, the database organization might reflect the organizational structure, which is distributed into units. Each unit maintains its own data set. In this distributed approach for storing data, both efficiency and security are improved by storing only the data required by local applications and so making data unavailable to unauthorized users.

On the other hand, classical machine learning approaches are designed to learn from a unique data set and, thus, to be applied to distributed data, they would require the collection of that data in a database for central processing. However, this is usually either ineffective or infeasible for the following reasons [19]:

- The cost of storing a central data set is much larger than the sum of the cost of storing smaller parts of the data set. It is obvious that the requirements of a central storage system are enormous. A classical example concerns data from the astronomy science and especially images from earth and space telescopes. The size of such databases is reaching the scale of exabytes (10^{18} bytes) and is increasing at a high pace. The central storage of the data of all telescopes of the planet would require a huge data warehouse of enormous cost. Another example of storage cost considers a multinational corporation, with thousands of establishments throughout the world, who wants to store data regarding all purchases of all its customers.
- The computational cost of mining a central database is much bigger than the sum of the cost of analyzing smaller parts of the data. Furthermore, with fragments as units of distribution, the learning task can be divided into several sub-tasks that operate in parallel. A distributed mining approach would make a better exploitation of the available resources. For example, the best way to quickly develop a successful business strategy is to analyze the data in a distributed manner, since the centralized approach takes too long due to the "very large" number of instances.
- The transfer of huge data volumes over network might take extremely much time and also require an unviable financial cost. Even a small volume of data might create problems in wireless network environments with limited bandwidth. Note also that it is common to have frequently updated databases, and communication could

be a continuous overhead that can even impede the usefulness of a learnt machine if its development and/or application take too long.
– Data can be private or sensitive, such as people's medical and financial records. The central collection of such data is not desirable as it puts their privacy into risk when communicating, for example, financial corporations who want to cooperate in preventing fraudulent intrusion into their computing systems [20]. The data stored by financial corporations are sensitive and cannot be exchanged with outsiders. On the other hand, in certain cases, the data might belong to different, perhaps competing, organizations that want to exchange knowledge without the exchange of raw private data.

Distributed algorithms deal with the above issues in order to learn from distributed data in an effective and efficient way.

2.3 Advantages of distributed data learning

Most distributed learning algorithms have their foundations in ensemble learning [21]. Ensemble learning builds a set of classifiers in order to enhance the accuracy of a single classifier. Although there are other methods, the most common one builds the set of classifiers by training each one on different subsets of data. Afterwards, the classifiers are combined in a concrete way defined by the ensemble algorithm. Thus, the ensemble approach is almost directly applicable to a distributed environment since a classifier can be trained at each site, using the subset of data stored in it, and then the classifiers can be eventually aggregated using ensemble strategies. In this sense, the following advantages of distributed learning come from the advantages of ensemble learning [22]:

– Using different learning processes to train several classifiers from distributed data sets increases the possibility of achieving higher accuracy especially on a large-size domain. This is because the integration of such classifiers can represent an integration of different learning biases which possibly compensate one another for their inefficient characteristics. Hansen and Salamon [23] have shown that, for an ensemble of artificial neural networks, if all classifiers have the same probability of making error of less than 0.5 and if all of them make errors independently, then the overall error must decrease as a function of the number of classifiers.
– Learning in a distributed manner provides a natural solution for large-scale learning where algorithm complexity and memory limitation are always the main obstacles. If several computers or a multi-core processor are available, then they can work on a different partition of data in order to independently derive a classifier. Therefore, the memory requirements as well as the execution time, assuming some minor communication overhead, become smaller since the computational cost of training several classifiers on subsets of data is lower than training one classifier on the whole data set.
– Distributed learning is inherently scalable since the growing amount of data may be offset by increasing the number of computers or processors.
– Finally, distributed learning overcomes the problems of centralized storage, already mentioned in Sect. 2.2.

Thus, many researches on distributed data learning have been concentrated on ensemble learning where the emphasis is put on making accurate predictions based on multiple models. As a consequence, one of the most promising research lines in distributed learning is "local learning and global integration".

2.4 Information to be combined

In distributed learning, as well as in ensemble learning, there are several learnt models and therefore several potential answers for a given problem. As the goal is to obtain an unique answer they have to be combined somehow. There are, in general, two types of information that can be combined [24]. On the one hand, the classifiers by themselves and, on the other hand, the predictions of the classifiers.

The first approach presents several limitations. Learning algorithms are concerned with learning concept descriptions expressed in terms of the given attributes. These descriptions can be represented in different ways as, for example, in the form of a decision tree, a set of rules or a neural network. Moreover, in distributed learning, the type of learning technique employed at one learning site might be different from the one employed at another, since there is no restriction on this aspect. Consequently, the learning algorithms to be combined could have different representations and, in order to combine the generated classifiers, we need to define a uniform representation into which the different classifiers are translated. It is difficult to define such a representation to encapsulate all other representations without losing a significant amount of information during the translation. Furthermore, a probable and undesirable consequence of this translation would be the restriction, to a large degree, of the information supported by the classifier. For example, it is difficult to define a uniform representation to merge a distance-based learning algorithm with a rule-based learning algorithm and, even if it were possible, the amount of information lost during translation might be unacceptable.

An alternative strategy to combine classifiers is to merge their outputs instead of the classifiers themselves. In this way, the representation of the classifiers and their internal organization is completely transparent since the type of information is based on the predictions which are the outputs of the classifiers for a particular data set, for example, a

hypothesized class for each input instance. As in the previous case, there is a need to define an unique representation, in this case with regards to predictions as they can be categorical or numerical (associated with some measure like probabilities or distances). However, the difficulty to define a uniform framework to combine the outputs is much less severe than the one to combine classifiers, as numerical predictions can be treated as categorical by simply choosing, as the corresponding categorial label, the class where the outputs reach the highest value. The opposite is not considered, since converting categorical predictions into predictions with numeric measures is undesirable or impossible.

3 Algorithms for distributed machine learning

The great majority of learning algorithms published in the literature focus their development on combining the predictions of a set of classifiers, since any classifier can be employed in this case, avoiding potential problems with concept descriptions and knowledge representation. In this section, several of the most popular distributed machine learning algorithms will be described, that follow this approach.

3.1 Decision rules

One of the simplest way of combining distributed classifiers consists on using non-trainable or adaptable methods of integration [25,26]. Instead, fixed rules are defined as functions that receive as inputs the outputs of the set of learnt classifiers and combine them to produce a unique output.

Consider a classification problem where instance x is to be assigned to one of the C possible classes $c_1, c_2, \ldots, c_C$. Let us assume that we have N classifiers and thus N outputs $y_i, i = 1, \ldots, N$ to take the decision.

According to the Bayesian theory, given measurements $y_i, i = 1, \ldots, N$, the instance x should be assigned to class c_j provided the a posteriori probability of that interpretation is maximum, i.e., assign

$x \rightarrow c_j$ if $P(c_j|y_1, \ldots, y_N) = \max_k P(c_k|y_1, \ldots, y_N)$.

This is a correct statement of the classification problem but the computation of the a posteriori probability functions would depend on the knowledge of high-order statistics described in terms of joint probability density functions which would be difficult to infer. To avoid this problem and make the rules more manageable, they must be expressed in terms of decisions produced by individual classifiers. When a measure of belief, confidence, or certainty is available, posterior probability can be estimated as $P(c_j|x) = y_i$, where y_i is computed as the response of a classifier i. This scenario provides a scope for the development of a range of efficient classifier combination rules[27]. Denote $y_{i_j}(x)$ as the output of the classifier i in the class j for the instance x and provided that the outputs y_i are normalized, i.e., $P(c_j|x) = \frac{y_i}{\sum_k y_k}$ some of the most popular rules are defined as:

– Product rule, $x \rightarrow c_j$ if

$$\prod_{i=1}^{N} y_{i_j}(x) = \max_{k=1}^{C} \prod_{i=1}^{N} y_{i_k}(x)$$

– Sum rule, $x \rightarrow c_j$ if

$$\sum_{i=1}^{N} y_{i_j}(x) = \max_{k=1}^{C} \sum_{i=1}^{N} y_{i_k}(x)$$

– Max rule, $x \rightarrow c_j$ if

$$\max_{i=1}^{N} y_{i_j}(x) = \max_{k=1}^{C} \max_{i=1}^{N} y_{i_k}(x)$$

This rule approximates the sum rule under the assumption that output classes are a priori equiprobable. The sum will be dominated by the output which lends the maximum support for a particular hypothesis.

– Min rule, $x \rightarrow c_j$ if

$$\min_{i=1}^{N} y_{i_j}(x) = \max_{k=1}^{C} \min_{i=1}^{N} y_{i_k}(x)$$

This rule approximates the product rule under the assumption that output classes are a priori equiprobable. The product will be dominated by the output which lends the minimum support for a particular hypothesis.

– Median rule, $x \rightarrow c_j$ if

$$\frac{1}{N} \sum_{i=1}^{N} y_{i_j}(x) = \max_{k=1}^{C} \frac{1}{N} \sum_{i=1}^{N} y_{i_k}(x)$$

– Majority voting, $x \rightarrow c_j$ if

$$\sum_{i=1}^{N} \Delta_{i_j}(x) = \max_{k=1}^{C} \sum_{i=1}^{N} \Delta_{i_k}(x)$$

This rule is obtained from the sum rule under the assumption that classes are a priori equiprobable and soft outputs $y_{i_k}(x)$ are transformed into hard outputs [0,1] where $\Delta_{i_k}(x) = 1$ if $y_{i_k}(x) = \max_{k=1}^{C} y_{i_k}(x)$ and $\Delta_{i_k}(x) = 0$ otherwise.

3.2 Stacked generalization

An alternative type of classifier combination approaches involves *learning* a global classifier that combines the outputs of a number of classifiers instead of using fixed rules. At the outset, stacked generalization [28], also known as staking in the literature, was developed with the aim of increasing the accuracy of a mixture of classifiers using a high-level model for combining the low-level classifiers, but it can be easily adapted to distributed learning.

Stacked generalization is a general method for combining multiple classifiers by learning the way that their output correlates with the true class. It works by deducing the biases of the classifiers with respect to an independent evaluation set. This deduction proceeds by generalizing the already trained classifiers over a second space. In this second space, the inputs x are the predictions of the classifiers for the instances of the independent evaluation set, and the outputs are the true class for those instances. The training procedure, adapted for a distributed environment, is summarized as follows. It is important to remark that, from now on and unless otherwise stated, every step is executed in every learning node,

1. Divide the data into training and validation sets.
2. Train a classifier on the training set.
3. Broadcast the classifier to all other nodes. Note that at the end of this step every node will contain every classifier.
4. Form the meta-level training set. Let $y_i(x)$ be the output of the classifier i for the instance x of the validation set and $\text{class}(x)$ the true class, then the meta-level instances will be of the form:

   $$[y_1(x), y_2(x) \ldots y_N(x), \text{class}(x)]$$

5. Send the meta-level subsets of data to a single node.
6. In this *single node*, a global classifier is trained on the meta-level training set using all meta-level subsets.

When a new instance appears for classification, the output of every local classifier is first computed to form a meta-level instance that will be an input to the global classifier which will be determined the final classification.

3.3 Meta-learning

Meta-learning [24] applies the concept of stacked generalization on combining classifiers but investigates several schemes for structuring the meta-level training set. Thus, both approaches follow the same procedure but they differentiate on how they form the meta-level training set and on how they determine the final output of new instances, as it is explained below.

1. Divide the data into training and validation sets.
2. Train a classifier on the trainingdata.
3. Broadcast the classifier to all other nodes.
4. Form the meta-level training set. There exist three types of meta-learning strategies for combining classifiers.
   - In the *combiner* strategy, the outputs of the classifiers for the validation set form the meta-level set. A *composition rule* determines the content of the instances of the meta-level based on different schemes,
     - *Meta-class*. Similar to stacked generalization, it uses the outputs of the classifiers together with the true class,

       $$[y_1(x), y_2(x) \ldots y_N(x), \text{class}(x)]$$

     - *Meta-class-attribute*. Similar to *Meta-class* with the addition of the attributes of the instance in the validation set,

       $$[y_1(x), y_2(x) \ldots y_N(x), x, \text{class}(x)]$$

     - *Meta-class-binary*. Similar to *Meta-class*, again the outputs of the classifiers for the validation set are included, but in this case, every output contains C binary predictions, as a strategy *one-versus-rest* is followed for every classifier,

       $$[y_{1_{\ldots C}}(x), y_{2_{\ldots C}}(x) \ldots y_{N_{\ldots C}}(x), \text{class}(x)]$$

   - In the *arbiter* strategy, the meta-level set M is a subset of the validation set, i.e., the meta-level is a particular distribution of the validation set. A *selection rule* determines the subset of instances of the validation set that will contain the meta-level set based on different schemes,
     - *Meta-different*. Select the instances with outputs that disagree to form the meta-level set M_d,

       $$M_d = \{x |\ y_1(x) \neq y_2(x) \vee y_1(x) \neq y_3(x) \vee \cdots \vee y_{n-1}(x) \neq y_n\}$$

     - *Meta-different-incorrect*. In this case, also the instances with outputs that agree but do not predict the true class are added to M,

       $$M = M_d \cup \{x |\ y_1(x) = y_2(x) = \cdots = y_n(x) \wedge \text{class}(x) \neq y_1(x)\}$$

   - The *hybrid* strategy integrates the combiner and arbiter strategies. Here, a *composition rule* form the meta-level set on the subset of instances returned when using a *selection rule*.
5. Send the meta-level subsets of data to a single node.
6. In this *single node*, build the meta-level training set containing all meta-level subsets of data and train a global classifier on the meta-level training set.

When a new instance appears for classification, the output of the global classifier will depend on the scheme.

– In the combiner and hybrid strategies, the output of every classifier is first computed to form a meta-level instance to the global classifier (*combiner*) which outputs the final classification.
– In the arbiter strategy, the output is the class obtained with majority vote among the local classifiers and the global one (*arbiter*), breaking ties in favor of the arbiter.

In light of the above, we can say that the combiner strategy tries to find a relationship among the outputs of the classifiers and the desired output. On the contrary, the arbiter strategy attempts to mediate among conflictive outputs wherein the global classifier was trained on a biased distribution of the validation set. Lastly, the hybrid strategy attempts to improve the arbiter strategy by correcting the predictions of "controversial" instances.

3.4 Knowledge probing

Knowledge probing [22] is also based on the idea of meta-learning, but it uses an independent set of instances to build a comprehensible classifier. Although meta-learning provides useful solutions to distributed learning, the authors of knowledge probing pointed out some fundamental limitations. The principal one is the problem of knowledge representation in the descriptive function, to which the final classifier does not provide any understanding of the data as it is not the integration of the knowledge from the base classifiers but the statistical combination of their predictions.

The key idea underlying knowledge probing is to derive a descriptive model by learning from un-seen data and the corresponding set of predictions made by the black box. The basic steps of knowledge probing can be presented as follows:

1. Divide the data into training and validation sets.
2. Train a classifier on the training set.
3. Broadcast the classifier to all other nodes.
4. Form the "probing" set using as inputs the inputs x of the validation set and as desired class $d(x)$ the one obtained by applying a simple decision rule, like majority vote, to the output of the classifiers. The probing instances will be of the form:

   $$[x, d(x)]$$

5. Send the probing subsets of data to a single node.
6. In this [*single node*], build the probing set containing all probing subsets of data and train a global classifier on the probing set.

When a new instance appears for classification, the global classifier will simply compute the final classification.

3.5 Distributed pasting votes

Pasting votes [29] was proposed to build ensembles of classifiers from small pieces or "bites" of data. Two strategies were implemented, Ivote (I = importance) and Rvote(R = random). *Ivote* sequentially generates data sets, and thus classifiers, by sampling with replacement in a way that each new training set has more instances that are more likely to be misclassified by the ensemble of classifiers generated up to that point. Thus, subsequent classifiers rely on the combined hypothesis of the previous classifiers. The sampling probabilities are based on the out-of-bag error, that is, a classifier is only tested on the instances not belonging to its training set. The out-of-bag error gives good estimates of the generalization error [30]. *Rvote* requires the creation of many bites of data by random and is a fast and simple approach. Distributed pasting votes [31] proposes a distributed approach following pasting votes. The authors call the distributed approaches of pasting Ivotes and Rvotes as DIvote and DRvote, respectively. The procedure for DIvote is as follows. Note that every step is executed in every node,

1. Build the first bite of data by sampling with replacement z instances from the available subset of data.
2. Train a classifier on the first bite.
3. Compute the out-of-bag error as follows:

   $$e(k) = p \times e(k-1) + (1-p) \times r(k)$$

   where p is the p value (the use of $p = 0.75$ is recommended [29]), k the number of classifiers in the ensemble so far, and $r(k)$ is the error rate of the k-th classifier on the subset of data. The probability $\Pr(k)$ of selecting a correctly classified instance is defined as follows:

   $$\Pr(k) = \frac{e(k)}{1-e(k)}$$

4. Build the subsequent bite of data. An instance is drawn at random from the subset. If this instance is misclassified by the majority vote of the out-of-bag classifiers, i.e., those classifiers for which the instance was not in their training set, then put it in the subsequent bite. Otherwise, put this instance in the bite with a probability of $\Pr(k)$. Repeat until z instances have been selected for the bite.
5. Train a new classifier on the $(k+1)$-th bite.
6. Repeat steps 3 and 4 for a given number of iterations to produce a desired number of classifiers.

Pasting DRvotes follows a procedure similar to DIvotes. The only difference is that each bite is a bootstrap replicate of size z, where each instance has the same probability of being selected. DRvote is very fast, as no intermediate steps

of DIvote are required. However, DRvote does not provide the accuracies achieved by DIvote [31].

DIvotes and DRvotes can essentially build numerous classifiers fast, as on each node numerous classifiers are built independently. It is important to remark that the global classifier consists of multiple classifiers from multiple nodes. When a new instance appears for classification, the global classifier will simply compute the final classification by combining the predictions of the ensembles of classifiers using a voting mechanism.

3.6 Effective stacking

Effective stacking [32] is motivated by several problems that arise in approaches based on stacked generalization (see Sect. 3.2) when dealing with large scale, high dimensional data. In stacking, the number of inputs in the meta-level depends on the number of classes and nodes (it becomes the number of classes times the number of nodes), a condition that goes against scalability. Another problem is that it is necessary to retain independent, validation, instances to train a global classifier during the combination step in order to avoid overfitting of the classifier. This can be a major drawback since the global classifier is trained using only a sub-sample of all available data.

Effective stacking attempts to counter the problems of stacking in large scale by averaging the outputs of the classifiers and applying a sort of cross validation: each subset of data serves as the training set for a local model and the evaluation set for a global model. The basic steps of effective stacking can be presented as follows:

1. Train a classifier on the subset of data.
2. Broadcast the classifier to all other nodes.
3. Form the meta-level training set using the outputs of the classifiers along with the true class in the subset of data, as in stacked generalization. Combine all classifiers apart from the local one. This is done to insure unbiased results, as the combination of classifiers requires training on the local data upon which it was induced. The combination performs stacked generalization of the averages of the predictions of the classifiers. Thus, the meta-level instances will be of the form:

$$\left[\frac{1}{N}\sum_{i=1}^{N} y_{i_1}(x), \frac{1}{N}\sum_{i=1}^{N} y_{i_2}(x), \ldots, \frac{1}{N}\sum_{i=1}^{N} y_{i_C}(x), \text{class}(x)\right]$$

   Note that the size of the meta-level training examples stays equal to the number of classes in the domain regardless of the number of local classifiers.

4. Train N global classifiers, one in each site, on the corresponding meta-level training set. These global classifiers describe the knowledge of all classifiers apart from the local one with respect to the local subset of data.

When a new instance appears for classification, the final classifier will simply compute the final classification by combining the predictions of the N global classifiers using the sum rule (see Sect. 3.1).

3.7 Distributed boosting

Distributed boosting [33] modifies the AdaBoost [34] for its application to a distributed environment. This algorithm proceeds in a series of T rounds. In every round t, a classifier is trained using a different distribution D_t for its training data that is altered by emphasizing particular training examples. Specifically, the distribution is updated to give wrong classifications higher weights than correct classifications. The entire weighted training set is given to the classifier to compute the hypothesis h_t. At the end, all hypotheses are combined into a final hypothesis h_{fn}.

During the boosting rounds, the node j maintains a local distribution $\Delta_{j,t}$ and the local weights $w_{j,t}$ that directly reflect the prediction accuracy on that node. The goal is to emulate the global distribution D_t obtained through iterations when standard boosting is applied to a single data set obtained by merging all subsets from the distributed nodes. The procedure is as follows. Note that every step is executed in every node unless otherwise stated. On node j, $j = 1, \ldots, N$ we are given set $S_j = (x_{j,1}, y_{j,1}), \ldots, (x_{j,m_j}, y_{j,m_j}), x_{j,i} \in X_j$, with labels $y_{j,i} \in Y_j = 1, \ldots, C$.

Let $B_j = (i, y_j) : i = 1, \ldots, m_j, y_j \neq y_{j,i}$

1. Initialize the distribution $\Delta_{j,1}$ over the instances, such that $\Delta_{j,1} = \frac{1}{n}$.
2. Make a version of the global distribution $D_{j,1}$, by initializing the j-th interval $[\sum_{p=1}^{j-1} m_p + 1, \sum_{p=1}^{j} m_p]$ in the distribution $D_{j,1}$ with values $\frac{1}{m_j}$.
3. Normalize $D_{j,1}$ with a normalization factor such that $D_{j,1}$ is a distribution.
4. For $t = 1, 2, 3, 4, \ldots, T$
   (a) Draw the indices of the instances according to the distribution $D_{j,t}$ and train a classifier $L_{j,t}$ on the sample.
   (b) Broadcast the classifier $L_{j,t}$ to all other nodes.
   (c) Create an ensemble $E_{j,t}$ by combining the classifiers $L_{j,t}$, and compute the hypothesis $h_{j,t} : X \times Y \rightarrow [0, 1]$ using the ensemble $E_{j,t}$.
   (d) Compute the pseudo-loss of hypothesis $h_{j,t}$ as

$$\epsilon_{j,t} = \frac{1}{2}\sum_{(i,y)\in B_j} \Delta_{j,t}(i, y)(1 - h_{j,t}(x_{j,i}, y_{j,i}) + h_{j,t}(x_{j,i}, y_j))$$

(e) Compute $V_{j,t} = \sum_{(i,y_j)\in B_j} w_{j,t}(i, y_i)$ where

$$w_{j,t}(i, y_j) = \frac{1}{2}\frac{(1 - h_{j,t}(x_{j,i}, y_j) + h_{j,t}(x_{j,i}, y_j))}{\text{acc}_j^p}$$

and $p \in 0, 1, 2$.

(f) Broadcast $V_{j,t}$ to all other nodes. Note that merging all the weight vectors $w_{j,t}$ requires a huge amount of time for broadcasting, since they directly depend on the size of the distributed data sets. In order to reduce this transfer time, only the sums $V_{j,t}$ of all their elements are broadcasted.

(g) Create a weight vector $U_{j,t}$ such that the j-th interval $[\sum_{p=1}^{j-1} m_p + 1, \sum_{p=1}^{j} m_p]$ is the weight vector $w_{j,t}$, while the values in the q-th interval, $q \neq j, q = 1, \ldots, N$ are set to $\frac{V_{q,t}}{m_q}$.

(h) Update $D_{j,t} : D_{j,t+1}(i, y_j) = \frac{D_{j,t}(i,y_j)}{Z_{j,t}}\beta_{j,t}^{U_{j,t}(i,y_j)}$, where $Z_{j,t}$ is a normalization constant chosen such that $D_{j,t+1}$ is a distribution. The values in the j-th interval of the $D_{j,t}$ after normalization make the local distribution $\Delta_{j,t}$.

Each node at round t maintains its version $D_{j,t}$ of the global distribution D_t and its local distribution $\Delta_{j,t}$. The samples in boosting rounds are drawn according to the distribution $D_{j,t}$. In order to simulate the boosting on centralized data, our intention was to draw more data instances from the sites that are more difficult for learning. The weights $w_{j,t}$ directly reflect the prediction capability for each data point, thus satisfying our goal to sample more examples from the sites that are more difficult to learn. In order to further emphasize sampling from the sites that are difficult for learning, we consider dividing the weights $w_{j,t}$ by the factor acc_j^p, $p \in 0, 1, 2$, such that the difference between the weights from two sites is further increased.

4 Other strategies for distributed machine learning

In this section, two other strategies for distributed machine learning are described that can be used alone or in combination with the algorithms previously described in Sect. 3.

4.1 Distributed clustering

Most distributed classification approaches view data distribution as a technical issue and combine local models aiming at a single global model. This, however, is unsuitable for inherently distributed databases, which are often described by more than one classification models that might differ conceptually.

Clustering distributed classifiers is based on: (i) a measure of classifier distance, (ii) an efficient algorithm to compute this distance measure for classifiers induced at physically distributed databases, and (iii) a clustering algorithm that will receive as input the calculated distances and will output the clusters.

1. Train a classifier L at each site on the subset of data D.
2. Broadcast the classifier to all other nodes.
3. Compute the distance of all pairs of classifiers apart from the local one,
   (a) At every site l, compute the disagreement measure as follows:
   $$d_{D_l}(L_i, L_j) = \frac{\sum_{r=1}^{M} \delta_{i,j}^{(r)}}{M},\ \forall (i, j) \in S_l^2 : i < j$$
   where M is the size of D_l, $S_l = \{1, \ldots, N\} - \{l\}$, and $\delta_{i,j}^{(r)}$ equals 1 if classifiers L_i and L_j have different output on instance r, and 0 otherwise.
   (b) Broadcast the distance for each pair of classifiers to all other nodes. Note that, at the end of this step, every node will contain $N - 2$ calculated distances for each pair of classifiers. The distance of each pair of local classifiers is evaluated in all N nodes, apart from the two nodes that were used for training these two classifiers.
   (c) Compute the average of these distances as the overall distance for each pair of classifiers,
   $$d(L_i, L_j) = \frac{1}{N-2}\sum_{l \in S_i \cap S_j}^{N} d_{D_l}(L_i, L_j)$$
4. Cluster the classifiers using hierarchical agglomerative clustering [35]. The sequence of merging the clusters can be visualized as a tree-shaped graph, which is called a dendrogram. For the automatic selection of a single clustering result from the sequence, a user-specified cutoff value can be provided, that affects when the agglomeration of clusters will stop.

The descriptive knowledge that the final clustering result conveys about the distributed classifiers, can be used for guiding the combination of the classifiers. Specifically, the classifiers of each cluster can be combined in order to produce a single classifier corresponding to each cluster.

4.2 Effective voting

Effective voting [36] is an effective extension to classifier evaluation and selection [37]. A paired t test with a significance level of 0.05 for each pair of classifiers is applied to evaluate the statistical significance of their relative performance. Effective voting stands between methods that

combine all classifiers, such as decision rules or meta-learning, and methods that just select a single model, such as evaluation and selection. The former uses error correcting through different learning biases at the expense of combining some classifiers with potentially inferior predictive performance. The latter selects one classifier at the expense of not always being the most accurate one. Effective voting attempts to select the most significant classifiers based on statistical tests and then combine them by voting.

The training procedure is summarized as follows:

1. Divide the data into training and validation sets.
2. Train a classifier on the training set.
3. Broadcast the classifier to all other nodes.
4. Compute the error of the classifiers in the validation set.
5. Send the errors to a single node.
6. At this *single node*, compute the overall significance index for every classifier L as:

$$Sig(L_i) = \sum_{j=1}^{N} \text{test}(L_i, L_j)$$

where

$$\text{test}(L_i, L_j) = \begin{cases} 1 & \text{if } L_i \text{ is significantly better than } L_j \\ -1 & \text{if } L_j \text{ is significantly better than } L_i \\ 0 & \text{otherwise} \end{cases}$$

When a new instance appears for classification, the output of the global classifier will depend on the strategy.

- Select the classifiers with the highest significance index and combine their decisions.
- Select the classifier with the lowest error rate along with any others that are not significantly worse than this, and combine their outputs.
- Select the three classifiers with the highest significance index and combine their outputs.

Effective Voting attempts to first select the most significant models with the aid of statistical tests and then combine them through a voting process. It can therefore be considered as a pre-processing method, rather than an actual combination method.

5 Evaluating distributed algorithms

In the past years, the theory and practice of machine learning have been focused on monolithic data sets from where learning algorithms generate a single model. In this setting, evaluation metrics and methods are well defined [38]. Nowadays, several sources produce data creating environments with several distributed data sets. Also big data sets collected in a central repository in which processing imposes quite high computing requirements. Then one actually thinks about distributed processing of the data as a way to have a more powerful computing platform.

In this novel situation, classical evaluation methods and metrics are unsuitable as new variables appear, like communication costs, data distribution, etc. On the one hand, simulation runs the algorithm in a simulated execution environment [39]. Such simulations often lead to models and metrics that do not capture important aspects in distributed learning. The availability of distributed data sets for experimenting is limited, an important obstacle to empirical research on distributed learning. This raises the issue of how to simulate the data properties of inherently distributed databases, in order to setup a robust platform for experiments [40], e.g., natural skewness and variability in context, which are found in real-world distributed databases.

On the other hand, there are no standard measures for evaluating distributed algorithms. Many existing measures are inadequate in distributed learning, showing low reliability or poor discriminant validity. Measures might be concerned with the scalability and efficiency of distributed approaches with respect to computational, memory or communication resources. Researchers usually vary the number of subsets of data and measure the prediction accuracy on a disjoint test set. The scalability of the proposed approaches is evaluated by analyzing their computational complexity in terms of training time. But this is a very narrow view of distributed learning and scalability. Many comparisons are presented in the literature but these usually focus on assessing a few algorithms or considering a few data sets. Indeed, they most usually involve different evaluation criteria. As a result, it is difficult to determine how does a method behave and compare with the other ones in terms of test error, training time and memory requirements, which are the practically relevant criteria, from the size or dimensionality of the data set, and from the trade-off between distributed resolution and communication costs.

In the authors' opinion, the PASCAL Challenge [41] provides a good starting point for anyone interested in pursuing a more in-depth study of scalability and distributed systems. To assess the models in the parallel track, the PASCAL Challenge define four quite innovative plots measuring *training time* versus *area over the precision recall curve*, *data set size* versus *area over the precision recall curve*, *data set size* versus *training time*, and *training time* versus *number of CPUs*. Additionally, it may be useful to borrow some ideas from [42] in which the authors are concerned with the scalability and efficiency of existing feature selection methods.

6 Conclusions

An overview of distributed learning was presented in this work. Distributed learning seems essential in order to provide solutions for learning from both "very large" data sets (large-scale learning) and naturally distributed data sets. It provides a learning scalable solution since the growing volume of data may be offset by increasing the number of learning sites. Moreover, distributed learning avoids the necessity of gathering data into a single workstation for central processing, saving time and money. Despite these clear advantages, new problems arise when dealing with distributed learning as, for example, the influence on accuracy of the heterogeneity of data among the partitions or the need to preserve privacy of data among partitions. Therefore, this is already an open line of research that will need to face these new challenges.

Acknowledgments This research was supported in part by the Secretaría de Estado de Investigación of the Spanish Goverment (grant code TIN2009-10748), the Xunta de Galicia (grant code CN2011/007) and the European Union by FEDER funds. Diego Peteiro-Barral acknowledge the support of Xunta de Galicia under Plan I2C Grant Program.

References

1. School of Information and Management and Systems. How much information? http://www2.sims.berkeley.edu/research/projects/how-much-info/internet.html (2000). Accessed 27 Sept 2010
2. D-Lib Magazine. A research library based on the historical collections of the Internet Archive. http://www.dlib.org/dlib/february06/arms/02arms.html (2006). Accessed 27 Oct 2010
3. Catlett, J.: Megainduction: machine learning on very large databases. PhD thesis, School of Computer Science, University of Technology, Sydney, Australia (1991)
4. Bottou, L., Bousquet, O.: The tradeoffs of large scale learning. Adv. Neural Inf. Process. Syst. **20**, 161–168 (2008)
5. Sonnenburg, S., Ratsch, G., Rieck, K.: Large scale learning with string kernels. In: Bottou, L., Chapelle, O., DeCoste, D., Weston, J. (eds.) Large Scale Kernel Machines, pp. 73–104. MIT Press, Cambridge (2007)
6. Moretti, C., Steinhaeuser, K., Thain, D., Chawla, N.V.: Scaling up classifiers to cloud computers. In: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), pp. 472–481 (2008)
7. Krishnan, S., Bhattacharyya, C., Hariharan, R.: A randomized algorithm for large scale support vector learning. In: Proceedings of Advances in Neural Information Processing Systems (NIPS), pp. 793–800 (2008)
8. Raina, R., Madhavan, A., Ng., A.Y.: Large-scale deep unsupervised learning using graphics processors. In: Proceedings of the 26th Annual International Conference on Machine Learning (ICML), pp. 873–880 (2009)
9. Provost, F., Kolluri, V.: A survey of methods for scaling up inductive algorithms. Data Min. Knowl. Discov. **3**(2), 131–169 (1999)
10. Giordana, A., Neri, F.: Search-intensive concept induction. Evol. Comput. **3**(4), 375–416 (1995)
11. Anglano, C., Botta, M.: Now g-net: learning classification programs on networks of workstations. IEEE Trans. Evol. Comput. **6**(5), 463–480 (2002)
12. Rodríguez, M., Escalante, D.M., Peregrín, A.: Efficient distributed genetic algorithm for rule extraction. Appl. Soft Comput. **11**(1), 733–743 (2011)
13. Lopez, L.I., Bardallo, J.M., De Vega, M.A., Peregrin, A.: Regal-tc: a distributed genetic algorithm for concept learning based on regal and the treatment of counterexamples. Soft Comput. **15**(7), 1389–1403 (2011)
14. Trelles, O., Prins, P., Snir, M., Jansen, R.C.: Big data, but are we ready? Nat. Rev. Genetics **12**(3), 224–224 (2011)
15. Stoica, I.: A berkeley view of big data: algorithms, machines and people. In: UC Berkeley EECS Annual Research, Symposium (2011)
16. LaValle, S., Lesser, E., Shockley, R., Hopkins, M.S., Kruschwitz, N.: Big data, analytics and the path from insights to value. MIT Sloan Manag. Rev. **52**(2), 21–32 (2011)
17. Borthakur, D.: The hadoop distributed file system: architecture and design. Hadoop Project Website **11**, 21 (2007)
18. Caragea, D., Silvescu, A., Honavar, V.: Analysis and synthesis of agents that learn from distributed dynamic data sources. In: Wermter, S., Austin, J., Willshaw D.J. (eds.) Emergent Neural Computational Architectures Based on Neuroscience, pp. 547–559. Springer-Verlag, Berlin (2001)
19. Tsoumakas, G., Vlahavas, I.: Distributed data mining. In: Erickson J. (ed.) Database Technologies: Concepts, Methodologies, Tools, and Applications, pp. 157–171. IGI Global, Hershey (2009)
20. Kargupta, H., Byung-Hoon, D.H., Johnson, E.: Collective Data Mining: A New Perspective Toward Distributed Data Analysis. In: Kargupta, H., Chan, P. (eds.) Advances in Distributed and Parallel Knowledge Discovery, AAAI Press/The MIT Press, Menlo Park (1999)
21. Dietterich, T.: Ensemble methods in machine learning. In: Gayar, N.E., Kittler, J., Roli, F. (eds.) Multiple classifier systems, pp. 1–15. Springer, New York (2000)
22. Guo, Y., Sutiwaraphun, J.: Probing knowledge in distributed data mining. In: Zhong, N., Zhou, L. (eds.) Methodologies for Knowledge Discovery and Data Mining, pp. 443–452. Springer, Berlin (1999)
23. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. **12**(10), 993–1001 (1990)
24. Chan, P.K., Stolfo, S.J.: Toward parallel and distributed learning by meta-learning. In: AAAI Workshop in Knowledge Discovery in Databases, pp. 227–240 (1993)
25. Kittler, J.: Combining classifiers: a theoretical framework. Pattern Anal. Appl. **1**(1), 18–27 (1998)
26. Ho, T.K., Hull, J.J., Srihari, S.N.: Decision combination in multiple classifier systems. IEEE Trans. Pattern Anal. Mach. Intell. **16**(1), 66–75 (1994)
27. Kittler, J., Hatef, M., Duin, R.P.W., Matas, J.: On combining classifiers. IEEE Trans. Pattern Anal. Mach. Intell. **20**(3), 226–239 (1998)
28. Wolpert, D.H.: Stacked generalization. Neural Netw. **5**(2), 241–259 (1992)
29. Breiman, L.: Pasting small votes for classification in large databases and on-line. Mach. Learn. **36**(1), 85–103 (1999)
30. Breiman. L.: Out-of-bag estimation. Technical report. Available at ftp://ftp.stat.berkeley.edu/pub/users/breiman/OOBestimation.ps (1996)
31. Chawla, N., Hall, L., Bowyer, K., Moore, T., Kegelmeyer, W.: Distributed pasting of small votes. In: Gayar, N.E., Kittler, J., Roli, F. (eds.) Multiple Classifier Systems, pp. 52–61. Springer, New York (2002)
32. Tsoumakas G., Vlahavas, I.: Effective stacking of distributed classifiers. In: ECAI, pp. 340–344 (2002)
33. Lazarevic, A., Obradovic, Z.: Boosting algorithms for parallel and distributed learning. Distrib. Parallel Databases **11**(2), 203–229 (2002)

34. Freund,Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: International Conference on Machine Learning, pp. 148–156. Morgan Kaufmann Publishers, Inc., San Francisco (1996)
35. Hand, D.J., Mannila, H., Smyth, P.: Principles of data mining. The MIT press, Cambridge (2001)
36. Tsoumakas, G., Katakis, I., Vlahavas, I.: Effective voting of heterogeneous classifiers. In: Machine Learning: ECML, pp. 465–476 (2004)
37. Woods, K., Kegelmeyer, W.P. Jr., Bowyer, K.: Combination of multiple classifiers using local accuracy estimates. IEEE Trans. Pattern Anal. Mach. Intell. **19**(4), 405–410 (1997)
38. Gama, J., Rodrigues, P.P., Sebastião, R.: Evaluating algorithms that learn from data streams. In: Proceedings of the 2009 ACM symposium on Applied Computing (ACM), pp. 1496–1500 (2009)
39. Urban, P., Défago, X., Schiper, A.: Neko: a single environment to simulate and prototype distributed algorithms. In: 15th International Conference on Information Networking, pp. 503–511. IEEE (2001)
40. Tsoumakas, G., Angelis, L., Vlahavas, I.: Clustering classifiers for knowledge discovery from physically distributed databases. Data Knowl. Eng. **49**(3), 223–242 (2004)
41. Sonnenburg, S., Franc, V., Yom-Tov, E., Sebag, M.: PASCAL large scale Learning challenge. In: 25th International Conference on Machine Learning (ICML2008) Workshop. http://largescale.first.fraunhofer.de. J. Mach. Learn. Res. **10**, 1937–1953 (2008)
42. Peteiro-Barral, D., Bolon-Canedo, V., Alonso-Betanzos, A., Guijarro-Berdinas, B., Sanchez-Marono, N.: Scalability analysis of filter-based methods for feature selection. In: Howlett R. (ed.) Advances in Smart Systems Research, vol. 2, no. 1, pp. 21–26. Future Technology Publications, Shoreham-by-sea, UK (2012)