International Journal of Advanced Science and Technology
Vol. 38, January, 2012

Improving Classification Accuracy Using Code Migration

Moez Ben Haj Hmida and Yahya Slimani

Department of Computer Science, Faculty of Sciences of Tunis, Tunisia

Abstract

Classification is a data mining technique widely used in critical domains like financial risk analysis, biology, communication network management, etc. Classification accuracy and learning from distributed datasets are the most challenging topics in the field of supervised learning. In this paper, we first briefly review the background of parallel and distributed classification algorithms and then propose a novel approach for classification in distributed large datasets. This approach is based on code migration instead of data migration. Extensive experimental results using a popular benchmark test suite show the effectiveness of this approach in term of accuracy. These results show also that the proposed method improved slightly classification accuracy over standard methods.

Keywords: *Classification; Classification accuracy; Distributed datasets; Meta-learning; Neural networks*

1. Introduction

Classification is a data mining technique widely used in diverse domains to extract hidden knowledge in distributed large datasets [1]. Actual classification algorithms are based on a set of theories, models, techniques and tools among which we can mention statistics, induction, artificial neural networks [2], or genetic algorithms [3]. Despite this diversity and plurality of methods and tools, existing algorithms suffer both from classification accuracy and computation time [4].

To improve the performances of classification algorithms it is possible to employ parallelisation techniques [4, 5] or ensemble methods [6]. The exploitation of parallelism in classification algorithms follows two main strategies [7]: data parallelism and task parallelism. In data parallelism, the dataset is divided across all processors and each processor performs the same task locally on its own partition. In task parallelism, different tasks are executed in parallel on different partitions but processors need to have access to all partitions. In this case, the dataset is explicitly replicated (distributed memory) or shared among multiple processors (shared memory). The parallel hybrid is the combination of two above strategies [7].

In task parallelism, however, if the size of this dataset is very large or if it is widely distributed, its duplication may cause a communication overhead that can degrade the performance of a classification system. It is therefore preferable to use the data parallelism. Thus, the combination of techniques of parallelization and distribution can help to improve significantly the performance of classifiers. In this paper, we propose a novel classification method based on data distribution.

Ensemble methods are learning algorithms that construct a set of classifiers (called base classifiers) and then classify new data points by taking a (weighted or unweighted) vote of their predictions [6]. Typically, an ensemble learning method runs the learning algorithm several times, each time with a different subset of the training examples.

Boosting [8] is the most popular implementation of such methods. It manipulates the training examples to generate multiple hypotheses. Boosting is a family of methods for accelerating a learning algorithm: AdaBoost (two-class problem), AdaBoost.M1 and AdaBoost.M2 (multiple-class problems) and AdaBoostR (regression) are among the most known [8]. Boosting maintains a set of weights over the training examples. For each iteration, the learning algorithm is invoked to minimize the error on the training set. Weights of misclassified examples are increased and weights of correctly classified examples are decreased. The global classifier is constructed through a weighted vote of the individual classifiers.

The ensemble learning approach [4], in which the same program runs several times on different dataset, can be applied to a distributed environment, where dataset is stored on different sites, thus producing different local models to be aggregated into a global model. Based on this idea, several methods have been proposed such as meta-learning.

Parallelizing a classification algorithm implies considering the method characteristics that permit to find the best parallelism [9]. Most of parallel classification algorithms in literature are based on data parallelism [4, 5]. In this type of algorithms, the same classifier is applied to each partition of the dataset to generate a local model. Subsequently, all these local models will be aggregated by a combining strategy to produce a global model [4].

The rest of this paper is organized as follows. Section 2 makes a review on techniques of parallel classification. Our method is described in Section 3 and its theoretical evaluation is presented in Section 4. Section 5 presents and discusses some experimental results for validating the proposed method. Finally, Section 6 concludes the paper and lists some prospects for future works.

2. Parallel and Distributed Classification Algorithms

Classification aims at assigning data items to one of *n* predefined categorical classes [1]. Since the category being predicted is pre-labelled, classification is also known as supervised induction [1]. The classification task requires two phases: learning and classification. The first phase generates a classification model from a specific set of training data with a predefined class for each datum. Then, the accuracy of the generated model is evaluated on the test data to measure the learning quality. The classification (or prediction) phase is the use of the model, found in the first phase, to classify new data.

In the following we present a brief review of parallel classification algorithms and methods. First, we introduce parallel algorithms based on decision trees and artificial neural networks, then we describe ensemble based methods such as Boosting [10] and meta-learning [11].

2.1. Parallel Decision Trees

The initial approaches to parallelizing decision trees were not scalable on large datasets and massively parallel machines [4, 5]. These limitations were resolved by the SPRINT algorithm [12]. SPRINT reduces the multiple sorting passes on the dataset to only one pass and it partitions horizontally the data among all processors. The decision tree is replicated on all processors. Each processor computes locally the best split point then exchanges its frequency statistics to determine the global best split point. SPRINT scalability is limited because the hash table used to split the attributes generates a communication overhead of *O(N)* messages per processor, where *N* is the number of examples. The ScalParc algorithm [13] is an extension of SPRINT that uses a distributed hash table to reduce the communication overhead to *O(log(N))*. Srivastava [14] proposed a dynamic hybrid approach that migrates gradually from data parallelism to task parallelism in order to reduce the load imbalance and the communication overhead incurred by the decision tree when it becomes bushy. This approach distributes the dataset among *P* processors. All processors are assigned to a single group. Each processor computes synchronously with the other processors the same decision tree node. The local information is exchanged by global reduction. This exchange overhead grows with the number of leaf nodes. When the communication becomes costly, the hybrid algorithm splits these nodes equally among two processor groups. On each group of processors, the tree nodes are synchronously computed then split among two processor groups.

2.2. Parallel Artificial Neural Networks

There are three main approaches in parallel neural networks formulation [7]. The first approach partitions the network. Nodes and weights are partitioned among processors and the computation is parallelized [15]. The neural network can also be represented by a weight matrix that can be parallelized.

In the second approach, namely pattern partition, nodes and weights are replicated on each processor and the patterns are partitioned among the processors. In this case, weights are communicated to update the network [16]. This approach is preferred when the number of patterns is large. It is considered as a coarse-grained parallelism but the first ANN approach is considered as fine-grained parallelism. The two approaches can be combined on a hybrid one [4].

Neural networks performances are sensitive to initial conditions and network topology. Thus, neural networks learning can be seen as an optimization problem that can be resolved by Genetic algorithms to find an optimal topology or to find an initial set of good weights [17].

2.3. Parallel Boosting

The first distributed version of the boosting method was proposed in [10], where each classifier is trained using a fraction of the training set. This version had worse performance compared to sequential algorithm [10]. Another distributed version of the boosting algorithm is proposed in [9]. At each boosting round, the classifiers are first learned on disjoint datasets. The produced models are exchanged among the sites to build an ensemble. Afterwards, the exchanged models are combined and their weighted voting-ensemble is constructed on each disjoint dataset.

P-AdaBoost [18] is a parallelization of the AdaBoost algorithm based on early estimates of the asymptotic frequency distribution of weights. In the first phase, the AdaBoost algorithm is run in sequential to estimate the asymptotic frequency distribution for the weights. In the second phase, instance models are trained in parallel along with their respective coefficients in the aggregated predictor. The global model is constructed through a linear combination of the trained instance models.

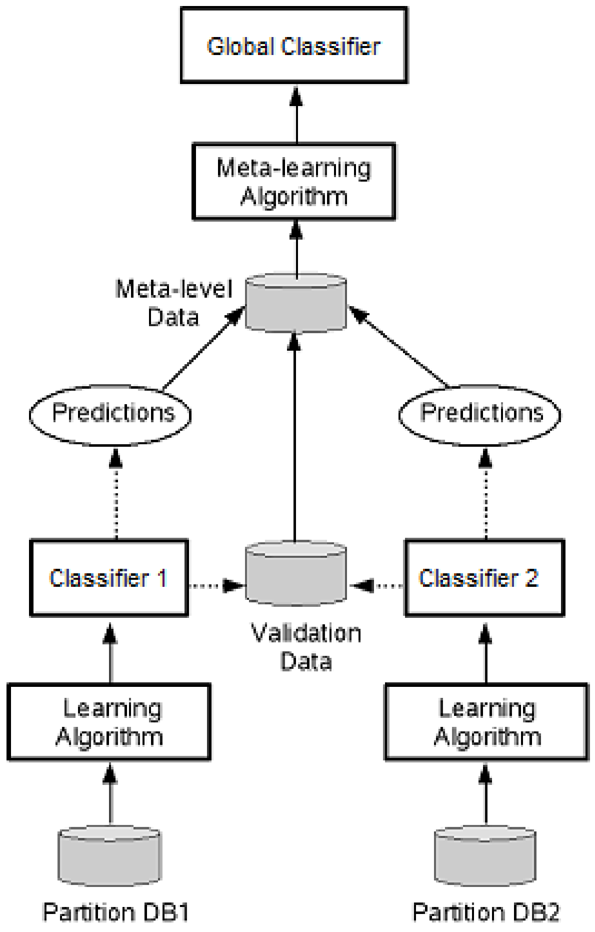

Figure 1. Meta-learning from Two Data Sources

2.4. Meta-learning

The meta-learning method was proposed for homogeneous distributed datasets [4]. First, a base classifier is trained from the local training-set at each site. The produced classifiers are collected to a master site where their predictions are generated on a separate validation-set to produce meta-level data. Finally, the global classifier (meta-classifier) is trained from the meta-level data. The meta-classifier algorithm can be an arbiter or a combiner of the different predictions composing the meta-level data. Figure 1 illustrates the various stages of meta-learning applied to a dataset distributed over two sites.

An example of meta-learning based system is JAM (Java Agents for Meta-Learning) [11]. By combining different learning systems, meta-learning improves the predictive performance and exploit task parallelism. It also reduces the communication cost when it exchanges the learned models instead of training examples. Meta-learning is scalable w.r.t the data size when it learns from small subsets that fit in memory. For the scalability w.r.t the number of sites, meta-learning has been enriched with a hierarchical approach, yielding however admittedly poor results: reduced predictive performance and increased communication overhead with large number of subsets [4].

3. Our Proposed Method

In this section, we propose a classification method inspired by the parallel genetic algorithm called Island Model [3]. This model creates, randomly, a set of starting solutions and divides it into islands. Each solution is considered as an individual who migrates from one island to another. The purpose of migration is the improvement of individuals (solutions) by undergoing mutations and crosses.

By analogy with the island model, and in the case of a geographically distributed dataset, each partition of the dataset will be regarded as an island. An individual (solution) will be a classification model. Initially, a classification model (neural network, decision tree, Bayesian network, etc.) is created on each partition. By migrating from one site to another, the model must adapt to the data of the hosting island. The models are constructed using artificial neural networks with backpropagation learning algorithm [19]. In what follows we describe the problem to be addressed and then we detail our proposed method.

3.1. Position of the Problem

Advances in networking technology and computational infrastructure induced (automatic) generation of large volumes of distributed data, which contain hidden source of knowledge, very rich and very useful for policy makers. The discovery of this knowledge requires the implementation of data mining techniques. In the case of a distributed data mining system, the first phase of the discovery process is to define the data distribution model. Data distribution considers two main models: horizontal partitioning (tuples) and vertical partitioning (attributes). In the first model, various distributed data sites store the same set of attributes while in the second model sites contain different sets of attributes across distributed datasets. In this paper, we investigate the case of horizontally distributed datasets like most of existing algorithms.

Typically a distributed data mining algorithm applies the same algorithm on the different sites to produce a local model that will be aggregated to build a global model. This method is appropriate for distributed architectures but can lead considerable accuracy lost [11]. Thus applying a single algorithm to whole data should produce more accurate classification models. This can be done by transferring data to a single site or migrating the code through the various data sites. Because a produced model is more compact than the data on which it was applied, we choose to use code migration instead of data migration.

3.2. Methodology

Consider a distributed dataset D partitioned on n sites $S_1 S_2 \dots S_n$, geographically distributed. The partitions of the dataset at these sites are respectively $P_1, P_2, \dots, P_n$. Instead of exchanging the different partitions between computing sites, we propose to build a local classification model that travels all the sites to produce a global model.

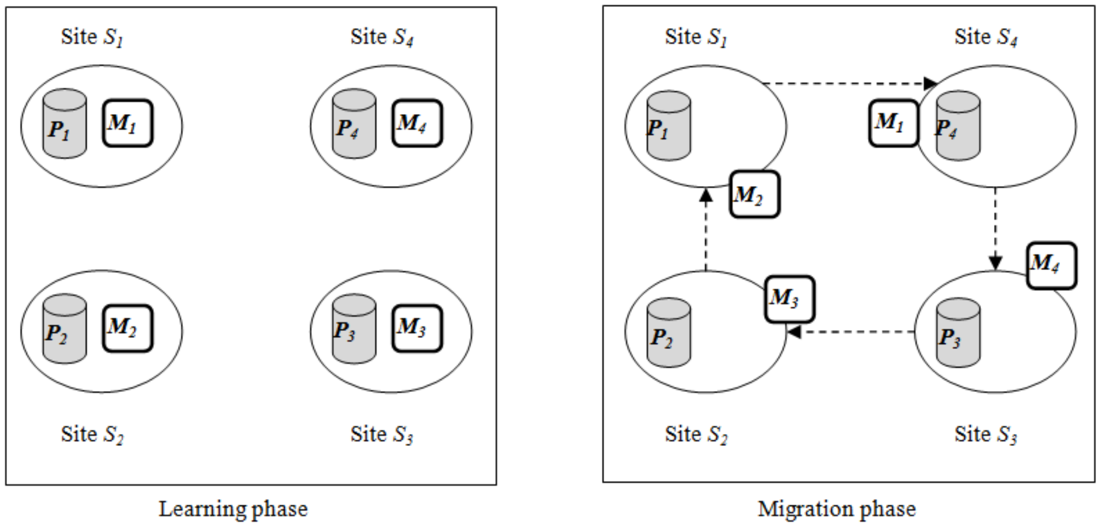

Figure 2. Different Phases of Distributed Learning Algorithm

Figure 2 shows the various phases of the distributed learning algorithm that we propose. It is indeed a repetition of two essential phases: learning and migration. The learning phase is the application of a classification algorithm to the local partition of each site to produce a local model per site. The migration phase consists on the migration of each local classification model to the successor site based on a virtual ring topology. At each new learning phase, each site performs the same classification algorithm to train the migrant model on local partition.

Generally, a classifier is built from an empty initial model (decision tree) or a randomly generated model (neural network). This means that the learning phase is done without prior knowledge. Contrariwise, in our approach, each model is built from a previously constructed model (coming from another site). We call this form of incremental learning, relearning.

Neural networks are ideally suited to this type of treatment through the iterative aspect of the learning algorithms they use. Unfortunately, the neural networks are known for their sensitivity to initial conditions. Network architecture, number of layers and numbers of internal nodes, can affect the performance of the produced model. They are also sensitive to the order of instances processing. To overcome these problems, we propose to use an approach based on data distribution. Thus, in each site the same algorithm is applied to the local partition. The models constructed in parallel will migrate from one site to another to produce N global models. Each model is characterized by its architecture and an order of partitions traversal. The choice the best model will be based on the quality of its performances.

3.3. Distributed Learning Algorithm

Our algorithm takes as input a dataset geographically distributed over *n* nodes, each containing a partition P_i and a test dataset *TD* built from these partitions. Initially, a neural network with random weights (initial model) is created on each node. Each model performs the back-propagation algorithm [20] on the local partition. Then the model is evaluated on the test data. According to the migration scheme, each model is sent to the next node where it resumes learning on the hosted partition. The process is repeated for each model by moving it from one node to another until it reaches the start node (Algorithm 1).

Algorithm 1 Distributed learning algorithm

```
For site i out of n
Input:
P_i, TD
Output:
Classification model M
Main:
    1. Create a random classification model M
    2. For k out of n
        a. Train M on P_i
        b. Test M on TD
        c. Send M to Successor(P_i)
        d. Receive M' from Predecessor(P_i)
        e. M = M'
    3. Broadcast accuracy of the model local M
    4. Choose the best model
```

The Distributed learning algorithm is executed in parallel by each participating site according a predefined virtual ring of communication. It should be noted here that we can build N! possible communication rings. But in our algorithm the same communication ring is used by all sites.

Figure 3 highlights the various steps executing the proposed algorithm on a dataset distributed across four partitions:

1. Step 1: a model (M_1) is generated by learning from the first partition (P_1). After testing (M_1) on test data, the model (M_1) is moved to the next partition (P_2).
2. Step 2: the learning algorithm re-adapts the model (M_1) to new data while keeping the knowledge previously learned to produce a new model named (M_2). This is what we called relearning. Accuracy of (M_2) is measured then it is moved to the next site (P_3) of the virtual ring of communication.
3. Step 3: model (M_3) is built from the previous knowledge of model (M_2) and the data of partition (P_3). Accuracy of (M_3) is measured then it is moved to the next site (P_4).
4. Step 4: model (M_4) is built from the previous knowledge of model (M_3) and the local partition (P_4). Accuracy of (M_3) is measured then it is moved to the next site (P_4). Finally, we obtain a global model built by learning from the whole dataset.

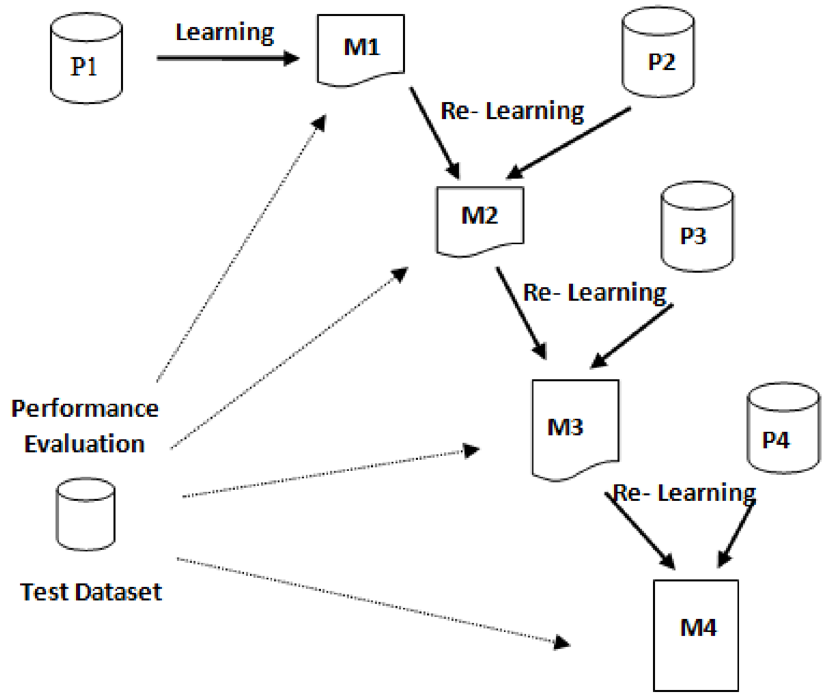

Figure 3. Learning from Distributed Dataset (Four Partitions)

In this algorithm, to build a classification model we have to choose a classification technique suitable for incremental learning like artificial neural networks. Thus to run and evaluate the performance of our algorithm we use a learning algorithm based on neural networks. Initially, the algorithm builds a neural network with a predefined architecture and random weights. Learning occurs by adjusting the weights based on instances of the partition (P_1). The second step is to apply the learning algorithm to the next partition (P_2). In this step, the initial network is the model generated by the first step instead of a random network. Thereafter this treatment is repeated for the remaining partitions. Thus, the neural network runs through all partitions by accumulating new knowledge.

3.4. Reduction of Communication Cost

Consider a dataset distributed among N sites, each partition consisting of I instances, C classes and A attributes. Thus, the size of each partition is $(I*A)$ and its transfer over network will produce the exchange of $(I*A)$ messages. However, the size of a neural network learned on a partition is on order of $O(A+C)$. Since the number of classes is often negligible compared to the number of attributes, the size of a neural network is estimated to $O(A)$. According proposed algorithm, each model traverses N sites, so it produces $(A*N)$ messages. The communication gain is the ratio of $(I*A)$ on $(A*N)$ which equals (I/N). Since the datasets sizes are on the order of thousands of instances when the number of attributes does not exceed tens, our approach brings us to reduce very substantially the communication cost.

4. Experimental Evaluation

In this section, we discuss some experimentation results to compare the performance of our approach versus a centralized approach. These experiments were performed on various datasets of the UCI site (http://kdd.ics.uci.edu/), which are Waveform, Colic, Letter, German, Breast cancer, Heart, Diabetes, Thyroid and Titanic datasets. Main characteristics of these datasets are summarized in Table 1. The learning algorithm used is back propagation algorithm of the gradient of the platform Weka 3.5.3 [21] that we extended to implement our approach.

Table 1. Datasets Employed in the Experiments.

Dataset	**# Instances**	**# Instances Training/Test**	**# Attributes**
German	1000	700/300	20
Breast c.	277	200/77	9
Heart	270	170/100	13
Diabetes	768	468/300	8
Thyroid	215	140/75	5
Titanic	2201	150/2051	3

To test our approach, each training set will be divided into four sites. Performance metric to consider is the accuracy (classification error of unseen data).

We used neural networks with a single hidden layer. The number of neural network nodes is equal to the sum of the number of attributes and classes. The training epochs is set to 400 for the centralized approach and 100 for each phase of the distributed learning approach. Now we have to fix the topology of the communication ring.

The learning datasets are partitioned into four partitions A, B, C and D. In order to estimate the influence of the partitions traversal order we considered six possible communication rings: (ABCD), (BCDA), (CDAB), (DABC), (ADBC) and (ACDB). For simplicity these six virtual rings are noted respectively VR_1, VR_2, ... and VR_6.

Figure 4 compares the values of the squared error and classification error obtained by our approach with those of the centralized one, compared on the Waveform dataset according to the six schemes defined above. We note that the performance of our approach depends on the migration scheme, but remain better than the centralized one.

In Figure 5, we see that our approach performs well than the centralized approach on the Colic dataset. This improved performance is due to the rehabilitation of the classifier during partitions traversal, which produces a more general learning and avoids overfitting. Figures 4 and 5 illustrate the sensitivity of classification accuracy between the different migration schemes. Indeed, whenever a model is applied to a new partition the knowledge is adapted to the information it contains. This adaptation can improve or degrade the quality of the classifier depending on the topology of the communication ring but not on data analyzed on the first or last phase.

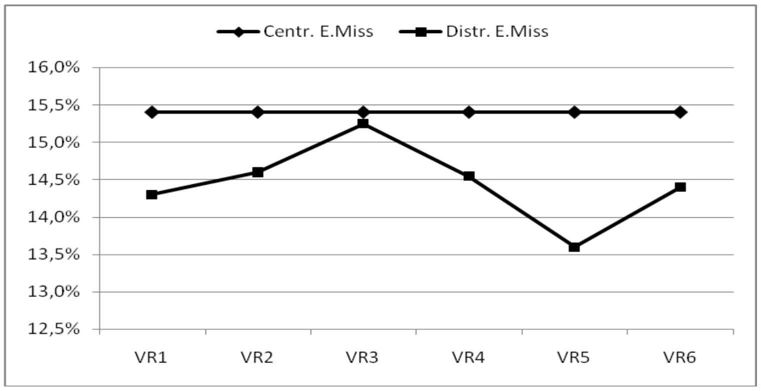

Figure 4. Influence of Order of Partitions Traversal on the Classification Accuracy of Distributed Learning Algorithm for the Waveform Dataset

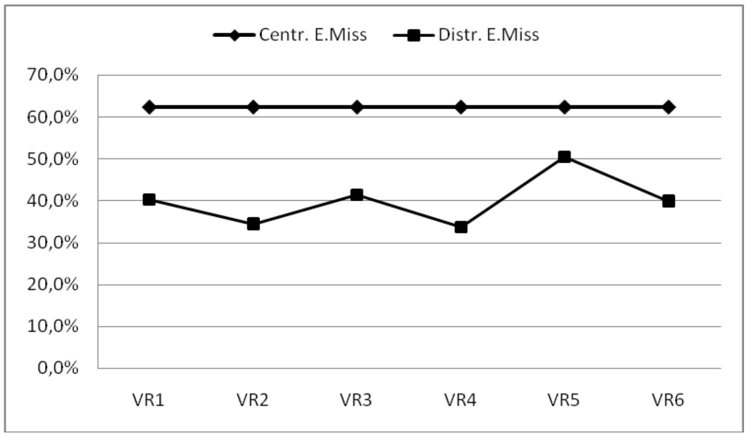

Figure 5. Influence of Order of Partitions Traversal on the Classification Accuracy of Distributed Learning Algorithm for the Colic Dataset

Table 2 compares misclassification errors on the test set and relative standard deviations are reported for AdaBoost and Bagging methods (A and B, respectively) built by aggregating a 1000 model instances, for P-AdaBoost algorithm (P) built by aggregating 1000 models; the length of the P-AdaBoost sequential stage is S = 100 (as reported in [18]) and for the Distributed learning algorithm (D) by partitioning training dataset on four equal partitions. As Table 2 shows, our algorithm achieves classification accuracy comparable or slightly better than that of other methods. Compared to the parallel method P-AdaBoost (P), the Distributed learning algorithm (D) achieves better accuracy except for Heart and Thyroid datasets.

Table 2. Accuracy on Benchmark Datasets

Method	Diabetes	Breast c.	German	Heart	Titanic	Thyroid
A	26.4 ± 1.4	24.5 ± 4.8	**23.2 ± 1.8**	**22.2 ± 3.1**	23.5 ± 1.6	**4.7 ± 1.8**
B	**23.3 ± 1.8**	25.1 ± 4.4	24.3 ± 2.3	24.0 ± 2.3	27.0 ± 3.7	6.5 ± 2.3
P	26.0 ± 1.7	24.7 ± 3.4	23.5 ± 2.3	22.2 ± 4.3	23.1 ± 1.3	4.9 ± 1.8
D	24.3 ± 1.3	**23.4± 4.7**	23.3 ± 1.6	23.0 ± 3.8	**22.1 ± 4.3**	5.3 ± 1.3

Table 3 compares performances of the Distributed learning algorithm to the meta-learning method (based on the C4.5 algorithm [22]) provided by the platform WekaMetal [23] for the dataset Letter. The values in the table represent the percentage of misclassified examples in the test phase by varying the number of partitions. These values show that the performance of our approach degrades when the number of partitions increases. Except that these degradations are less important than the meta-learning method.

Table 3. Accuracy on Letter Dataset Varying the Number of Partitions

# Partitions	**Meta-learning**	**Distributed Learning Algorithm**
1	14	11
4	15	10
8	15	12
16	17	12

5. Conclusion

In this paper, we proposed a distributed approach to supervised learning in which we use code migration instead of data migration. This approach, applied to distributed datasets, has allowed distributed learning avoiding communication overhead. The effectiveness of our approach was compared against a centralized approach, where learning is performed from a dataset stored on a single site. The first experiments we conducted have shown fairly significant gain in performance obtained by our proposal. These encouraging results suggest new directions for improving our approach. Thus, we believe it would be interesting to use genetic algorithms to find an optimal setting of initial parameters. This setting is to find a number of internal layers and the number of nodes in layers that improve the performance of the neural network.

References

[1] Ye N. *The Handbook of Data Mining*. 1st ed. Mahwah, New Jersey: Lawrence Earlbaum Associates, **(2003)**, p.689.

[2] Michie D, Spiegelhalter D and Taylor C. Machine Learning, Neural and Statistical Classification. Artificial Intelligence. Ellis Horwood **(1994)**

[3] Skolicki Z and De Jong K. Improving evolutionary algorithms with multi-representation island models. In: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature PPSN VIII, Birmingham, UK, **(2004)** 420–429.

[4] Kargupta H and Chan P. *Advances in Distributed and Parallel Knowledge Discovery*. Cambridge: MIT Press, **(2000)**

[5] Zaki MJ and Ho CT. *Large-Scale Parallel Data Mining, Lecture Notes in Computer Science*. Germany: Springer, **(2000)** p.261.

[6] Dietterich TG. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning* 2000; 40: 139–157 **(2000)**

[7] Freitas AA. A survey of parallel data mining. In: Proceedings of the 2nd International Conference on the Practical Applications of Knowledge Discovery and Data Mining **(1998)** London, UK, 287–300.

[8] Freund Y and Schapire RE. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computing System Science* **(1997)**; 55(1): 119–139.

[9] Lazarevic A and Obradovic Z. Boosting algorithms for parallel and distributed learning. *Distributed and Parallel Datasets* **(2002)**; 11(2): 203-229.

[10] Fan W, Stolfo S J and Zhang, J. The application of Adaboost for distributed, scalable and on-line learning. In: Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining **(1999)** San Diego, CA, USA, 362–366.

[11] Chan P, Prodromidis A and Stolfo G. Meta-learning in distributed data mining systems: Issues and approaches. *Advances of Distributed Data Mining*. Menlo Park: AAAI Press, 74–81, **(2000)**

[12] Shafer JC, Agrawal R and Mehta M. SPRINT: A scalable parallel classifier for data mining. In: Proceedings of the 22th International Conference on Very Large Datasets, Mumbai **(1996)** Bombay, India, 544–555.

[13] Joshi A, Karypis G and Kumar V. ScalparC: A new scalable and efficient parallel classification algorithm for mining large datasets. In: Proceedings of the 11th International Parallel Processing Symposium. IEEE Computer Society Press **(1998)** Washington DC, USA, 573–579.

[14] Srivastava A, Han E, Kumar V and Singh V. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery* **(1999)**; 3: 237–261.

[15] Nordström T and Svensson B. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing* **(1992)**; 14: 260–285.

[16] Foo SK, Saratchandran P and Sundararajan N. Parallel implementation of backpropagation neural networks on a heterogeneous array of transputers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **(1997)**; 27(1): 118–126.

[17] Abraham A and Nath B. Hybrid heuristics for optimal design of artificial neural networks. In John R & Birkenhead R (Eds.) Advances in Soft Computing Techniques and Applications. Germany: Springer-Verlag. **(2000)**, pp. 15–22.

[18] Merler S, Caprile B and Furlanello C. Parallelizing AdaBoost by weights dynamics. *Computational Statistics and Data Analysis* **(2007)**; 51(5): 2487–2498.

[19] Bishop C. *Neural Networks for Pattern Recognition*. Walton Street, Oxford: Oxford University Press, 1995.

[20] Hinton GE. Connectionist learning procedures. *Artificial Intelligence* **(1989)**; 40: 185–234.

[21] Witten I and Frank E. *Data Mining: Practical machine learning tools and techniques*. 2nd ed. San Francisco: Morgan Kaufmann, **(2005)**, p.560.

[22] Quinlan J R. C4.5: Programs for Machine Learning. *Machine Learning* **(1993)**; 16: 235–240.

[23] Soares C and Brazdil P. Zoomed Ranking: Selection of Classification Algorithms Based on Relevant Performance Information. In: Zighed DA, Komorowski HJ and Zytkow JM (eds.) Lecture Notes In Computer Science London: Springer-Verlag, **(2000)**, pp.126–135.