JAISCR, 2012, Vol. 2, No. 1, pp. 5 – 20

LEARNING FROM HETEROGENEOUSLY DISTRIBUTED DATA SETS USING ARTIFICIAL NEURAL NETWORKS AND GENETIC ALGORITHMS

Diego Peteiro-Barral[1], Bertha Guijarro-Berdiñas[1] and Beatriz Pérez-Sánchez[2]

[1]*Department of Computer Science, University of A Coruña, Campus de Elviña s/n, 15071, A Coruña, Spain*

[2] *Department of Computer Science, University of A Coruña, Campus de Elviña s/n, 15071, A Coruña, Spain*

Abstract

It is a fact that traditional algorithms cannot look at a very large data set and plausibly find a good solution with reasonable requirements of computation (memory, time and communications). In this situation, distributed learning seems to be a promising line of research. It represents a natural manner for scaling up algorithms inasmuch as an increase of the amount of data can be compensated by an increase of the number of distributed locations in which the data is processed. Our contribution in this field is the algorithm Devonet, based on neural networks and genetic algorithms. It achieves fairly good performance but several limitations were reported in connection with its degradation in accuracy when working with heterogeneous data, i.e. the distribution of data is different among the locations. In this paper, we take into account this heterogeneity in order to propose several improvements of the algorithm, based on distributing the computation of the genetic algorithm. Results show a significative improvement of the performance of Devonet in terms of accuracy.

1 Introduction

The unrestrainable growth of data in fields such as bioinformatics, intrusion detection in computer networks, text classification or engineering, opens the way for new applications of machine learning. Traditional algorithms cannot look at a large data set and plausibly find a good solution on a reasonable time. They can deal with medium-size data sets but they usually have problems with more than $1,000,000$ data [1], data being features times samples. Even so most of them are restricted to much less data due to time of memory constraints. In this case, preprocessing techniques as subsampling are usually applied.

However, this is not a constraint for learning from large data sets but a limitation of learning algorithms. What is more, the larger the number of samples for training the better the performance of classifiers [2].

In order to deal with large data sets, a new field of research has emerged, called *large-scale learning* [3, 4]. The aim is to develop efficient and scalable algorithms with regard to their requirements of computation, memory, time and communications. Large-scale learning has received considerable attention in recent years and some methods have been proposed so far in the literature [5, 6, 7, 8, 9, 10, 11].

In particular, distributed learning seems to be one of the most promising lines of research. It involves learning from subsets of data allocated at physically separated locations and then combining all knowledge. It represents a natural manner for scaling up algorithms inasmuch as an increase of the amount of data can be compensated by an in-

crease of the number of locations. In this manner, there is no need of learning on a large data sets but learning on smaller data sets and then combining all models.

The fact that machine learning applications are most often distributed means that traditional algorithms are becoming obsolete. They need a monolithic data set so the only solution is to gather all data at a single location. However, this is usually inefficient. On the one hand, the cost of storing a single data set is much higher than the sum of the costs of storing smaller parts of it. On the other hand, the necessary bandwidth to transmit the data might not be available [12]. Furthermore, some constraints may prevent exchanging private data at all. Additionally, the current trend of reducing the speed of processors in favor of multi-core processors and computer clusters leads to an appropriate situation in which distributed learning may play a central role [13, 14, 15, 16, 17].

Our contribution in the field of distributed learning is the algorithm *Devonet* [18], an abbreviation of *Distributed evolved networks*. Devonet is a fast and accurate distributed learning algorithm based on one-layer artificial neural networks (ANNs) and genetic algorithms (GA). It makes possible privacy-preserving classification by not exchanging raw data across distributed locations but only the ANNs. Furthermore, it minimizes communications by using this approach since the size of the ANNs is negligible in comparison with the size of the raw data. Devonet performs better than other algorithms in several data sets [18] but some limitations were reported when the distributions of data are heterogeneous, i.e. the prevalence of classes is different among the locations. In this regard, real-world distributed data sets are usually heterogeneous, e.g. data related to diseases from hospitals around the world or attacks from an intrusion detection perspective. With the aim of overcoming this issue, in this paper we take into account this heterogeneity in order to propose several improvements of the algorithm, based on distributing the computation of the genetic algorithm. Moreover, another contribution of this paper is the assessment of the algorithms in terms of different degrees of heterogeneity.

The remainder of the paper is structured as follows. Section 2 presents the distributed learning algorithm Devonet, Section 3 describes its improvements proposed in this paper, Section 4 describes the experimental study, Sections 5 and 6 discuss the experimental results, and Section 7 presents the conclusions of this research.

2 Devonet

In a formal way, we can define a distributed computing environment as a set of locations physically separated but mutually connected by a communication network, including in this definition multi-core processors. In order to learn from distributed data, one of the most promising strategies is local learning and model integration, i.e. in the first place, the classifiers are trained on their corresponding, local, subset of data and then they are integrated using some combination method. Devonet takes advantage of this learning paradigm avoiding moving raw data across the locations and therefore allows privacy-preserving classification as well as minimizing communications, as described below.

2.1 Local Learning

As local classifiers, Devonet uses one-layer ANNs trained with an efficient learning algorithm [19]. It is well known that, when working with large data sets, it is essential to employ low complexity algorithms due to time and memory restrictions. This training algorithm presents some advantages which make it suitable for our purposes on distributed learning. On the one hand, it is very efficient since the weights of the ANNs are computed analytically and, on the other hand, it is able to learn in an incremental manner.

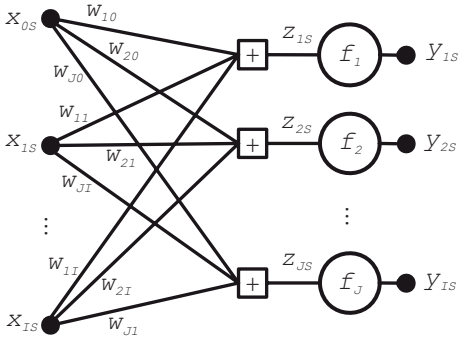

Figure 1. Architecture of a single-layer ANN

Assume that the one-layer ANN shown in Figure 1 in which the set of equations relating inputs and outputs is given by

$$y_{js} = f_j\left(\sum_{i=0}^{I} w_{ji}\cdot x_{is}\right); j = 1\ldots J; s = 1\ldots S \quad (1)$$

where I, J, S are the number of inputs, outputs and training samples, respectively; $x_{0s} = 1$; w_{ji} is the weight of the connection between the ith input and the jth output neuron; and f_j is the nonlinear activation function of jth output neuron. The system described by Eq. 1 has $J \times S$ equations in $J \times (I+1)$ unknowns. However, due to the situation that the number of samples of data is often larger than the number of inputs $(S >> I+1)$, in practice this set of equations in w_{ji} is not compatible and has no solution. Consequently, some errors ε_{js} between the real output, y_{js}, and the desired output, d_{js}, of the ANN have to be considered,

$$\varepsilon_{js} = d_{js} - y_{js} = d_{js} - f_j\left(\sum_{i=0}^{I} w_{ji}\cdot x_{is}\right) \quad (2)$$

In order to learn the weights w_{ji}, the sum of squared errors is usually employed as the objective function to be minimized,

$$P = \sum_{s=1}^{S}\sum_{j=1}^{J}\varepsilon_{js}^2 = \sum_{s=1}^{S}\sum_{j=1}^{J}\left(d_{js} - f_j\left(\sum_{i=0}^{I} w_{ji}\cdot x_{is}\right)\right)^2 \quad (3)$$

Assuming that the nonlinear activation functions f_j are invertible (as it is the case for the most commonly used functions), alternatively, the system of equations in Eq. 2 can be rewritten in the following way [19]

$$\bar{\varepsilon}_{js} = \bar{d}_{js} - z_{js} = f_j^{-1}(d_{js}) - \sum_{i=0}^{I} w_{ji}\cdot x_{is} \quad (4)$$

Notice that in Eq. 4, as opposed to Eq. 3, the unknowns, representing the weights of the ANN, are not affected by the nonlinear activation functions f_j, Thus, we can define a new error objective function as [20]

$$Q = \sum_{s=1}^{S}\sum_{j=1}^{J}(\bar{\varepsilon}_{js})^2 = \\ = \sum_{s=1}^{S}\sum_{j=1}^{J}\left(f_j^{-1}(d_{js}) - \sum_{i=0}^{I} w_{ji}\cdot x_{is}\right)^2 \quad (5)$$

whose global minimum can be computed by deriving it with respect to the weights,

$$\frac{\partial Q}{\partial w_{jp}} = -2\cdot Q\cdot x_{ps}; p = 1\ldots I \quad (6)$$

and solving the system of equations obtained by equalizing its derivative function to zero,

$$\sum_{i=0}^{I} A_{pi}\cdot w_{ji} = b_{pj} \quad (7)$$

where

$$A_{pi} = \sum_{s=1}^{S} x_{is}\cdot x_{ps} \\ b_{pj} = \sum_{s=1}^{S} \bar{d}_{js}\cdot x_{ps} \quad (8)$$

For every output j, Eq. 8 has $I+1$ linear equations and unknowns so there exists only one real solution which corresponds to the global optimum of the objective function in Eq. 5. Several computationally efficient methods can be used to solve these kinds of systems with a complexity of $O(J\times(I+1)^2)$ [21, 22]. Moreover, this training algorithm is able to learn incrementally since the coefficients A_{pi} and b_{pj} are calculated as a sum of terms in Eq. 8 on the number of samples S. Due to the commutative and associative properties of the sum, the same solution is obtained independently of the order of occurrence of the samples.

2.2 Model Integration

The weights of a single, global, ANN representing the union of all, local, ANNs may be computed by summing their corresponding matrices of coefficients A and b (see Fig. 2), solving the new system of linear equations. Notice that even when the knowledge of an ANN is contained in its weight matrix W it can be computed from the coefficients A and b (see Eq. 7). However, this method may be very sensitive to data skewness since the training algorithm achieves the global optimum for each subset of data. For this reason, Devonet computes a global ANN using a GA. Basically, the GA is defined by the next three elements in which the initial population corresponds to the set of local classifiers, ANNs, represented by the matrices of coefficients A and b.

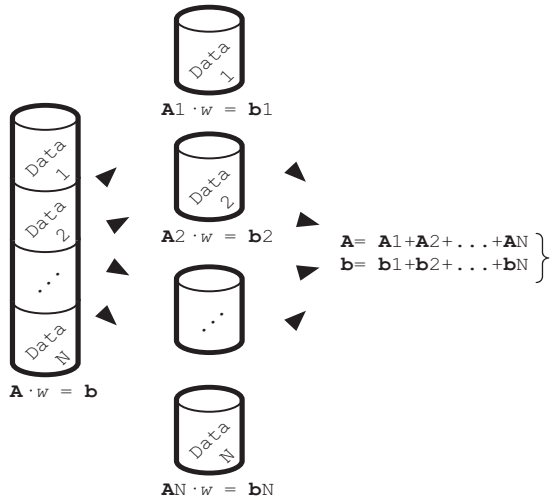

Figure 2. Combination of ANNs using the matrices of coefficients A and b.

– The *fitness function* defines the optimality of a solution and it is minimized during the execution of the algorithm. A solution represents an individual in the population, i.e. a classifier. In this case, the standard class accuracy [23] was used.

– The *crossover operator* defines how two individuals are combined. Due to the training algorithm allows incremental learning, it is simply defined as the sum of the matrices of coefficients A and b of the parents.

– The *mutation operator* defines the impact of small random disturbances in an individual. In this case, few coefficients of the matrices A and b are altered with small normally-distributed probability.

The hypothesis here is that the crossover operator will find the best combination of local ANNs and the mutation operator will reduce the effect of data skewness. In order to do this, the GA is run in a central location wherein independent, validation, data is contained.

Fig. 3 shows the architecture of Devonet. The training procedure is summarized in Algorithm 0.

Algorithm 0 Devonet.

1 For each location, train a local classifier on the data.
2 Broadcast each local model to the central location. Remember that each model is represented by matrices A and b.
3 Execute the GA at the central location, using the data stored in it.
4 For the central location, select the individual with the best fitness of the population as the global classifier and broadcast it to all locations. In this manner, we will be able to classify new samples at every location.

Notice that during this process only the classifiers, which have a negligible size in comparison with the data, are sent across a communication network. In this manner, Devonet is able to learn from distributed data sets without exchanging any raw, private, data.

3 The Proposed Method

Devonet shows good performance on many data sets [18] but some limitations were pointed out regarding the two following issues.

– Distributed real-world data sets are usually not symmetric, i.e. the distributions of data for different locations may not be the same. So, in order to train a proper global classifier the GA has to be executed on unbiased independent data allocated in a *central location*. But the concept of a central location with unbiased data does not exists in most real-world applications. We could select at random any location to play the role of the central location but, in this case, the quality of the data is not assured. It could be biased. Notice that the question is not the computation itself of the GA but the data used.

– As is common in algorithms that involve learning in the model integration step, Devonet requires to *retain independent data* in order to train the global classifier. But this approach has two major drawbacks. On the one hand, it de-

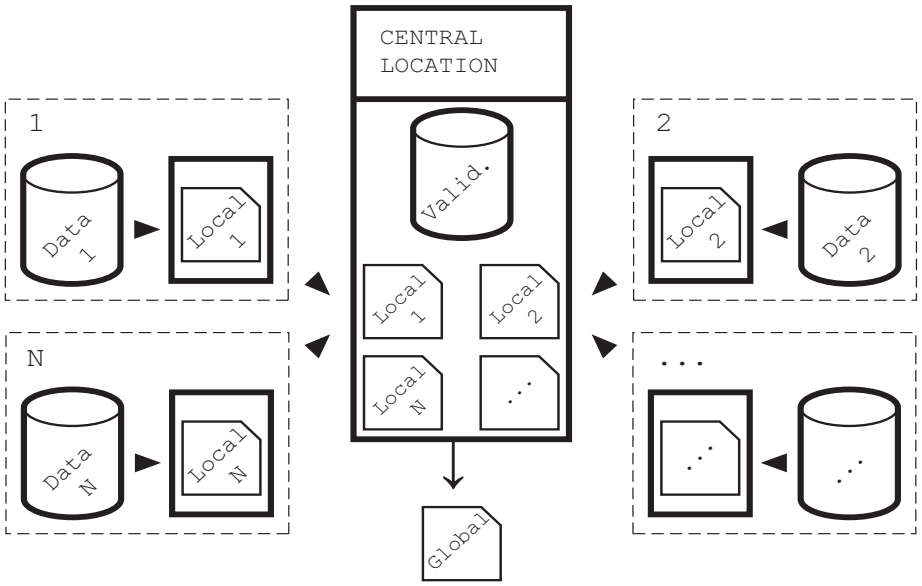

Figure 3. Architecture of Devonet.

prives the local classifiers of some training data and, on the other hand, the global classifier is trained only on a small subsample of all data.

While the size of the data sets is large the necessity of retaining a subset of data as the validation set is not a severe issue. However, the presence of a central location with unbiased data could be the most important thing for the good performance of Devonet.

With the aim of overcoming these limitations three different improvements of Devonet are proposed in this paper. Notice that any improvement of the algorithm must keep the advantages of the original, that is allowing privacy-preserving classification and minimizing communication costs.

Improvement 1

It is illustrated in Fig. 4. For purposes of simplicity, this figure as well as figures 5 and 6 show a scenario with three distributed locations. We believe that other scenarios with different number of locations can be easily inferred from them.

This approach overcomes the limitation of the need of a central location by computing the fitness function of the GA in a distributed manner. Thus, each location has its own validation set to compute *part* of the fitness function, which will be coalesce with all other parts at a designated location. We simply referred to the location chosen to gather all data and build the global model as the designated location. Notice that, in our design, any distributed location can play this role. The hypothesis here is that subsampling each location to form the validation data set we will be able to make an unbiased sample of all data. Algorithm 1 summarizes the procedure of the algorithm and describes how the global classifier is obtained.

Algorithm 1 Improvement 1 of Devonet.

1 For each location, divide the data into training and validation sets. Train a local classifier on the training data.
2 Broadcast each local classifier to all other locations, i.e. at the end of this step each location will contain every local model or, what is the same, the whole population.
3 Execute the GA at the designated location. Repeat until convergence (epochs, error. . .),
3.1 For each location, compute the values of the fitness function on the validation set for every individual of the population and send them to the designated location.
3.2 For the designated location, evolve the population and broadcast to all other locations.
4 For the designated location, select the best individual of the population as the global classifier and broadcast it to all other locations.

Improvement 2

This approach overcomes the limitation of retaining independent data for training the global

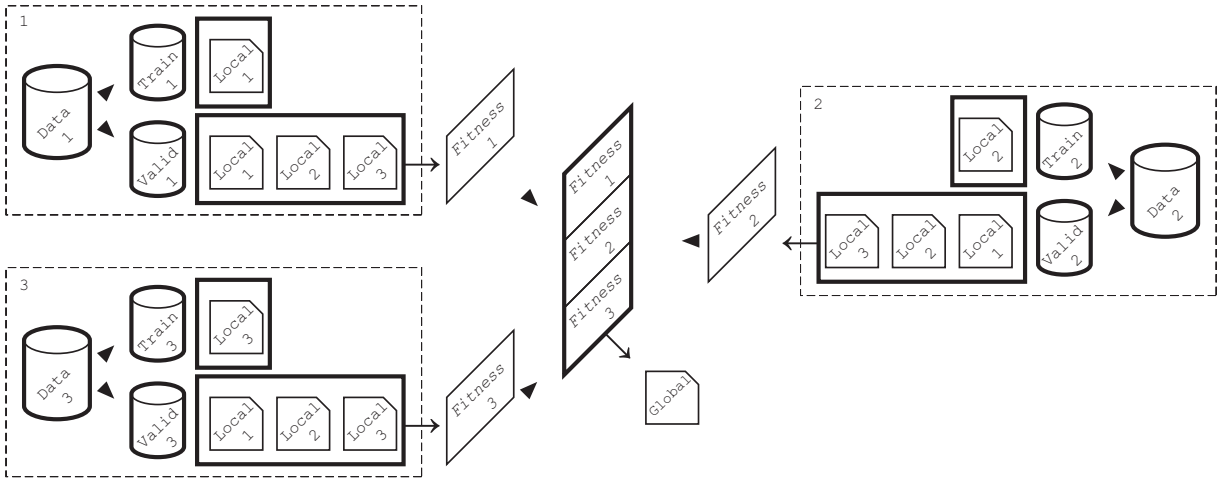

Figure 4. Architecture of improvement 1 of Devonet.

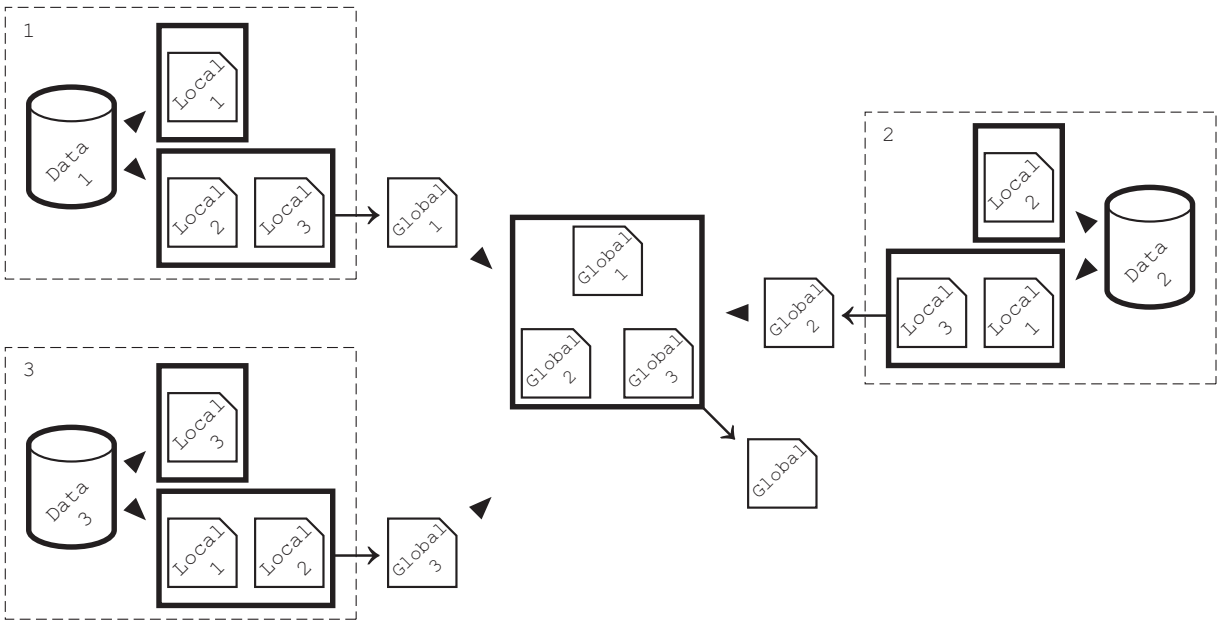

Figure 5. Architecture of improvement 2 of Devonet.

model by following the approach proposed in [16] (see Fig. 5). The key in this approach is to bring the concept of *partial*-global classifiers in the algorithm (see Algorithm 2 for further details). The hypothesis here is that training both local and global models on more data will improve the performance of the classifiers.

Algorithm 2 Improvement 2 of Devonet.

1 For each location, train a local classifier on the data.
2 Broadcast each local model to all other locations, i.e. at the end of this step each location will contain every local classifier or, what is the same, the whole population.
3 For each location, execute the GA on the data using all classifiers except the local one. Do this in order to avoid biased classifiers since the local classifier was trained on that data. Select the best individual of the population as the *partial*-global classifier and broadcast it to the designated location.
4 For the designated location, build the global classifier by combining the *partial*-global classifiers. Due to the incremental features of the training algorithms, we are able to combine them by simply summing their matrices of coefficients. Broadcast the global classifier to all other locations.

Improvement 3

This approach merges, up to a point, the two previous improvements (see Fig. 6). The hypothesis here is that, even when the complexity of the model is higher than the previous approaches, we will be able to exploit the strengths of them. Algorithm 3 summarizes the procedure of the algorithm.

Algorithm 3 Improvement 3 of Devonet.

1 For each location, divide the data into training and validation. Train a local classifier on the training data.
2 Broadcast each local classifier to all other locations, i.e. at the end of this step each location will contain every local model or, what is the same, the whole population.
3 For each location, execute the GA on the training data using all classifiers except the local one. Do this in order to avoid biased classifiers since the local classifier was trained on that data. Select the best individual of the population as the *partial*-global classifier and broadcast it to the designated location.
4 Execute the GA at the designated location using the *partial*-global classifiers as initial population. Repeat until convergence (epochs, error...),
4.1 For each location, compute the values of the fitness function on the validation set for every individual of the population and send to the designated location.
4.2 For the designated location, evolve the population and broadcast to all locations.
5 For the designated location, select the best individual of the population as the global classifier and broadcast it to all other locations.

4 Experimental Study

The experimentation presented below is focused on the assessment of the improvements proposed in this paper. With this aim, the best improvement among them, or even the original algorithm if there is no improvement at all, will be selected in the first place. After that, the method selected in the previous step will be compared against other distributed algorithms. We will also aim attention at the impact of different distributions of data on the performance of the algorithms. This is important due to the inherent data skewness features shown in most real-world distributed data sets. Indeed, this fact may cause that the locations perform quite differently from one another in terms of accuracy.

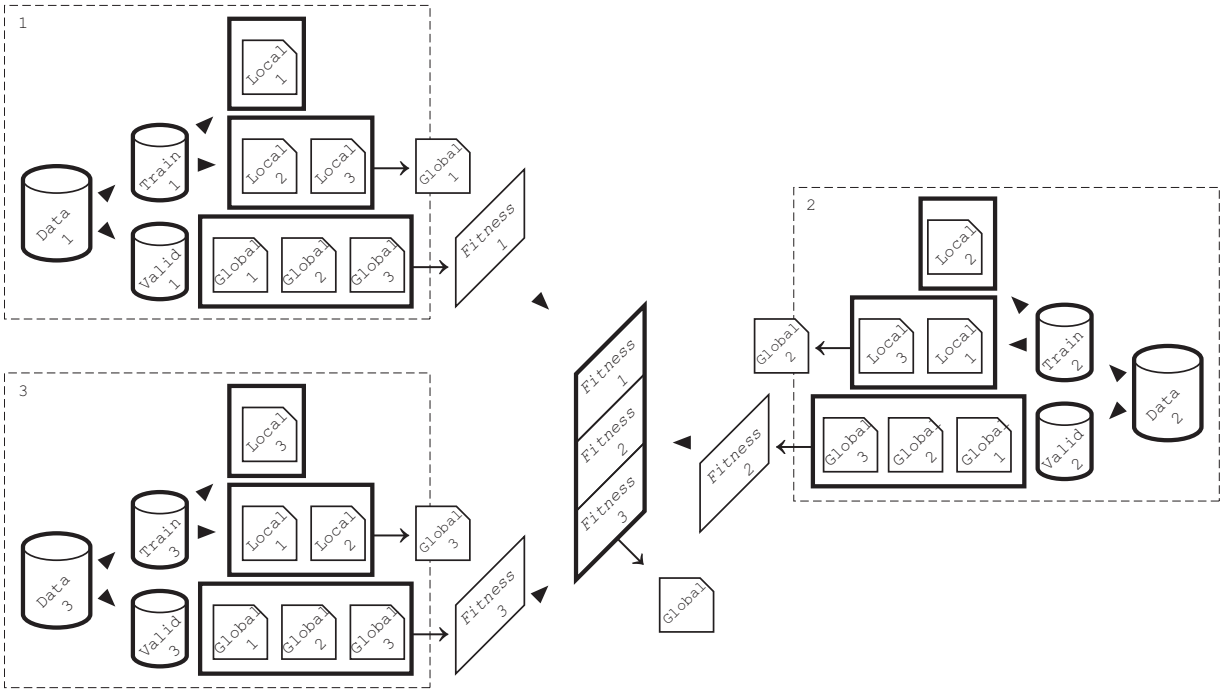

Figure 6. Architecture of improvement 3 of Devonet.

4.1 Data Sets

Four well-known data sets selected from the *UCI Machine Learning Repository* [24] were used. For each data set, number of inputs, classes and samples are shown in Table 1. In order to be as representative as possible of different data sets, we included binary and multi-class medium-large size data sets.

Data set	Inputs	Classes	Samples
Adult	14	2	30,162
Connect4	42	3	67,557
Covertype	54	7	581,012
Shuttle	9	7	43,500

Table 1. Characteristics of each data set used during experimentation.

The lack of publicly available distributed data sets is one of the most important issues we have to deal with. With the aim of overcoming this issue, we develop distributed data sets based on these publicly available, monolithic, data sets. However, as mentioned above, notice that distributed data sets may show different distributions of data among locations. In order to take this issue into account, the concept of class-probability distributions of data is introduced below.

4.2 Class-Probability Distribution of Data

For a given data set, we can define the class-probability distribution of that data as the *a priori* probability of classifying a sample in each class. In a less rigorous manner, it is the percentage of samples per class at each location. If we use this concept in a distributed context, we can define a uniform, or homogeneous, distributed data environment as an alike class-probability distribution for *every* location, that is, the prevalence of each class is the same among the locations. On the other hand, a nonuniform, or heterogeneous, distributed data environment occurs when there is an unlike class-probability distribution for at least two locations. The latter opens the way for different degrees of heterogeneity.

The heterogeneity is an important factor that affects learning. Therefore, in order to assess the performance of the distributed algorithms in a proper manner, we simulate several distributions of data among locations; one uniform and three nonuniform. The difference among the nonuniform distributions lies in the degree of heterogeneity, or skewness; slight, quite and severe.

Figure 7 shows the different environments for each data set considered in this research. The darker the shading the more nonuniform the distribution.

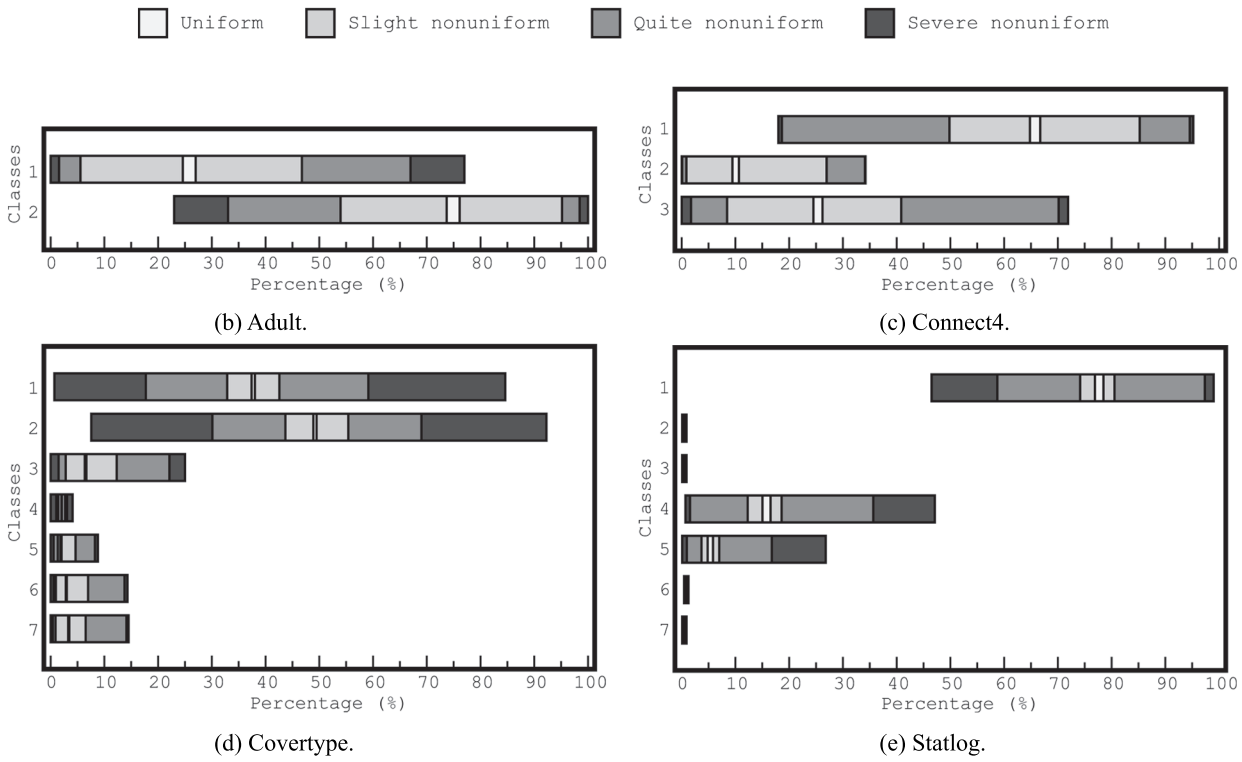

Figure 7. Class-probability distributions of data.

The range of prevalence of each class in a given environment is depicted as a segment bounded by the minimum and the maximum prevalence of that class among the locations. Notice that the segments are continuous and they are shown superimposed. In order to make clear the representation consider *Adult* data set. In the uniform distribution of data, lightest gray, the prevalence of class 1 is around 25% for every location. In the opposite case, darkest gray for severe nonuniform distributions of data, its prevalence lies between 0 and 75%, i.e. at least one location does not contain any sample of class 1, at least one location contains 75% of samples of that class and, finally, all other locations show a prevalence of class 1 somewhere between 0 and 75%. The remaining classes and data sets can be interpreted in the same manner.

4.3 Measure of Disparity among Locations

Once the classifiers are trained any location could receive new samples for classification. Furthermore, those samples should be classified at that location since usually we can not send raw samples of data to other locations due to privacy issues. This is the point for maintaining the global model at every location. All algorithms used in this research use a global model for classifying new samples, and this global model is the same for every location. However, notice that we assume an alike class-probability distribution between the training samples and the new samples received for every location. Thus if the distribution of data is not homogeneous the global classifier could perform differently at different locations. In order to be able to quantify how differently each algorithm performs on each location, we also compute a measure of disparity (disp) by simply averaging all pairwise differences in accuracy (acc) among locations,

$$disp = \frac{N \cdot (N-1)}{2} \sum_{i=0}^{N-1} \sum_{j=i+1}^{N} |acc_i - acc_j| \qquad (9)$$

N being the number of locations. Even when the divergence is not as critical as the class accuracy, we believe that it is a good measure for having a new clue of how well a distributed algorithm performs in different distributions of data. The lower the disparity the better.

4.4 Learning Algorithms

In order to assess the performance of the proposed learning algorithms in comparison with others, two of the most popular distributed learning algorithms were used: Majority Vote and Stacking. Notice that they are independent of the type of classifiers in contrast to Devonet. In order to set up a fair comparative, the same classifiers used in Devonet were used, that are single-layer ANNs trained with the learning algorithm presented in Section 2.1. Remember that this algorithm reaches the global minima of the error function.

Majority Vote

Fixed combining rules are the most straightforward manner of combining classifiers, in which a rule is a function defined on the outputs of the classifiers. Since the rules are defined a priori there is no need for training in the combination step, i.e. no validation data set is required. However, notice that the local classifiers have to be trained in any case. Majority Vote [25] is probably the most often used fixed combining rule in practice, in which a sample is classified as the class with the highest number of votes among the classifiers. Since no training is needed in the combination step, the procedure of this algorithm is very simple (see Algorithm 4).

Algorithm 4 Majority Vote.

1 At each location, train a local classifier on the data.
2 Broadcast each local classifier to all other locations, i.e. at the end of this step each location will contain every local model. Remember that we do this in order to be able to classify new samples at every location.

Stacked Generalization

Stacked Generalization [26], or Stacking, combines classifiers by learning the way that their outputs correlates with their true class. Stacking minimizes the generalization error by inferring the biases of the classifiers on an independent set of samples. Even though Stacking was originally developed for improving accuracy in an ensemble of classifiers, it can be easily adapted to a distributed learning task as described in Algorithm 5 and illustrated in Fig. 8.

Algorithm 5 Stacked Generalization.

1 At each location, divide the data into training and validation. Train a local classifier on the training data.
2 Broadcast each local classifier to all other locations, i.e. at the end of this step each location will contain every local model.
3 At each location, apply all classifiers to its validation set to build a new data set at the *meta*-knowledge level that, later on, will be used to train the global classifier. A sample of this new data set will be formed by the outputs of the local classifiers, that will act as inputs, or features, for the global classifier; along with the true class, that will be the desired output for the global classifier. Notice that the outputs of the local classifiers can be categorical or in terms of probabilities. Notice also that the outputs in terms of probabilities can be translated to categorical by simply selecting the class with the highest confidence. Thus, two different approaches can be followed in this step in order to form the inputs of the new data set at the *meta*-knowledge level,
- Use the label of class, categorical outputs, so the number of features is the number of locations. Lower complexity but lower level of knowledge.
- Use the probabilities for each class so the number of features is the number of locations times the number of classes in the domain. Higher level of knowledge but higher complexity.
4 Broadcast each *meta*-knowledge level data set to the designated location.
5 At the designated location, train a global classifier on the *meta*-knowledge level data set.

4.5 Experimental Procedure

A set of 10 distributed locations was used since a larger number of distributed locations would lead to too small training sets increasing the risk of overfitting [1]. With the aim of assessing the performance of the distributed learning algorithms, the following procedure was performed.

- For each algorithm; repeat $I = 100$ times
  - Divide the data set into $N = 10$ subsets. In

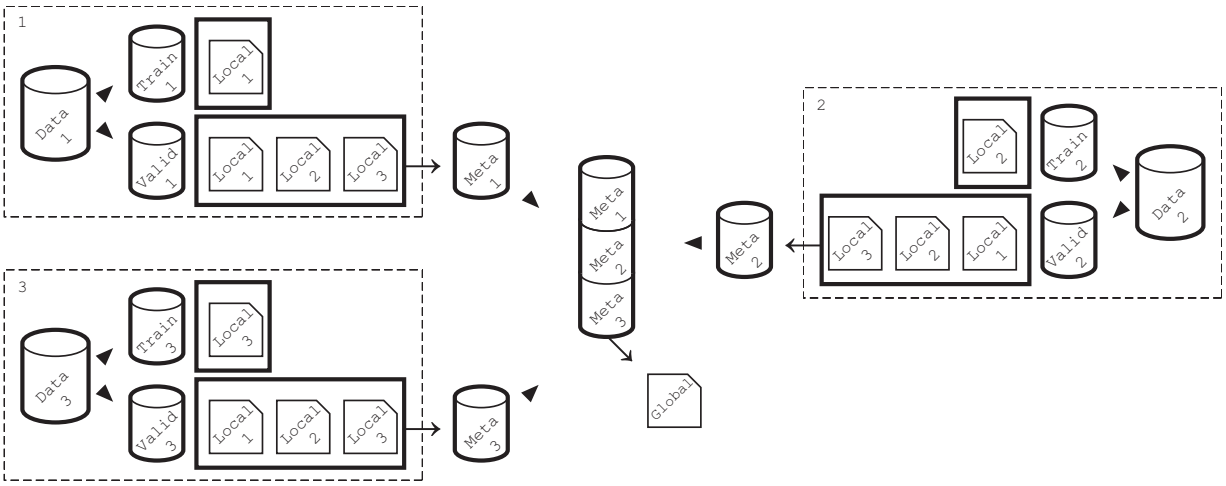

Figure 8. Architecture of Stacked Generalization.

this manner, each subset of data will represent a distributed location.

- For each location, divide the data using *hold-out validation*, i.e. a subset of samples is chosen at random to form the test set and the remaining observations are retained as the training set. 10% of data is used for testing while 90% are for training. This kind of validation is suitable because the size of the data sets is large. Notice that some algorithms need to use a validation data set. In these cases, the following distribution was used; 10% test, 10% validation and 80% training.
- Train the distributed model.
- Test the model at each location by computing the standard class accuracy [23] on the test data set. Also compute the global test accuracy by averaging the test accuracy among the locations.

– Compute the *mean* test accuracy and *standard deviation* of each model over the 100 simulations. Compute also the *disparity* (see Eq. 9) among locations.

– Apply a Kruskal-Wallis test [27] to check if there are significant differences among the medians of the models for a level of significance $\alpha = 0.05$.

– If there are differences among the medians, then apply a multiple comparison procedure [28] to find the simplest model, lowest complexity, whose error is not significantly different from the model with the best mean accuracy rate. In this work, a Tukey's honestly significant criterion [28] was used as multiple comparison test.

Finally, all experiments were carried out in MATLAB® [29].

5 Results and Discussion: Devonet Versus its Improvements

For purposes of simplicity, the experimental results are shown as follows. In the first place, a comparison among the original Devonet and the three different improvements proposed in this paper are presented in this section. In the second place, we select the best algorithm among them and it is compared against Majority Vote and Stacking in Section 6. Notice that the two alternatives of Stacking for forming the *meta*-knowledge level were included.

5.1 Experimental Results

Table 2 shows the mean test accuracies and standard deviations of the original Devonet algorithm (see Section 2) and the three improvements proposed in this research (see Section 3). Note that *Improvement* is abbreviated as *Imprv*.

In order to select the best algorithm in terms of accuracy, the Kruskal-Wallis test was applied.

We believe that the most interesting results for

2*DATA SET	HETERO-GENEITY	ALGORITHM			
		Devonet	*Imprv1*	*Imprv2*	*Imprv3*
4*Adult	None	83.59 ± 0.72	83.22 ± 0.94	81.61 ± 2.61	83.43 ± 0.65
	Slight	82.35 ± 1.25	83.46 ± 0.61	79.21 ± 5.04	83.30 ± 0.62
	Quite	79.06 ± 3.61	83.59 ± 0.86	77.71 ± 7.71	82.86 ± 1.10
	Severe	75.92 ± 5.71	83.15 ± 0.76	78.09 ± 7.59	83.34 ± 0.62
4*Connect4	None	75.56 ± 0.50	75.17 ± 0.64	73.82 ± 4.92	75.16 ± 0.52
	Slight	74.43 ± 1.83	75.25 ± 0.31	73.89 ± 3.41	75.24 ± 0.64
	Quite	71.56 ± 4.09	75.25 ± 0.53	61.77 ± 9.99	75.14 ± 0.52
	Severe	68.20 ± 6.29	75.06 ± 0.50	67.90 ± 9.74	74.93 ± 0.55
4*Covertype	None	70.24 ± 0.24	70.18 ± 0.20	69.78 ± 1.18	70.33 ± 0.33
	Slight	69.34 ± 1.49	70.48 ± 0.30	70.10 ± 0.54	70.59 ± 0.16
	Quite	66.67 ± 1.99	70.38 ± 0.24	68.97 ± 1.35	70.09 ± 0.98
	Severe	64.75 ± 4.84	69.79 ± 1.02	56.28 ± 9.99	68.68 ± 3.94
4*Shuttle	None	90.89 ± 0.98	90.30 ± 1.20	77.06 ± 9.99	94.13 ± 2.06
	Slight	89.82 ± 1.23	90.63 ± 1.36	86.48 ± 4.27	90.54 ± 1.63
	Quite	89.91 ± 1.94	91.40 ± 1.46	78.33 ± 9.99	92.47 ± 1.64
	Severe	88.87 ± 3.57	91.80 ± 0.73	87.46 ± 3.77	90.98 ± 2.53

Table 2. Mean test accuracy (%) and standard deviation of Devonet and its improvements for each data set and the four class-probability distributions.

the purposes of this research are those computed on severe heterogeneous distributions of data (see Section 4.5) since they show the robustness of the algorithms. Therefore, the statistical assessment of the algorithms is performed on those results.

A p-value of $p < 0.001$ was obtained for each data set so we can reject the null hypothesis, all medians are equal, for a level of significance of 0.05. Consequently, a multiple comparison test to look at pairwise comparisons among the algorithms was applied. Results show that *Imprv1* and *3* are the best algorithms and also they are not significantly different from each other. The original Devonet and *Imprv2* are in any case significantly worse than these two.

Table 3 shows the disparity among locations. Remember that it is a measure of how different the locations perform by averaging all pairwise differences in accuracy among them.

On the other hand, in order to select the best algorithm in terms of disparity, the Kruskal-Wallis test was applied. Remember that the smaller disparity the better. A p-value of $p < 0.001$ was obtained for each data set so we can reject the null hypothesis for a level of significance of 0.05, i.e. significative differences were found among the medians of the models in terms of disparity. A multiple comparison test show that *Imprv1* is the best algorithm in terms of disparity, better or not significantly worse than the others.

5.2 Discussion

As can be inferred from Table 2,

- If we compare the performance of the *original Devonet* between none and severe heterogeneous distributions it falls by 7.67, 7.36, 5.49 and 2.02% on *Adult*, *Connect4*, *Covertype* and *Shuttle* data sets, respectively. On average, its performance is 5.64% worse. Notice that the probability of selecting a biased location to play the role of the validation set increases with the heterogeneity of data. Thus, it is logical that the higher the heterogeneity the lower the accuracy.
- Almost the same performance is reached by *Imprv1* for every distribution of data. Small variabilities can be explained as the effect of randomness. As can be seen, distributing the computation of the GA works much better than selecting a location by random as happened in the original Devonet. This improvement is able to form an unbiased, and *distributed*, validation set.
- Notice that the performance of *Imprv2* is uncorrelated with the distribution of data. In fact,

2*DATA SET	HETERO-GENEITY	ALGORITHM			
		Devonet	*Imprv1*	*Imprv2*	*Imprv3*
4*Adult	None	2.35 ± 0.54	2.29 ± 0.32	2.42 ± 0.49	2.14 ± 0.74
	Slight	4.39 ± 2.19	5.13 ± 1.37	6.55 ± 3.37	5.10 ± 1.44
	Quite	9.11 ± 9.63	8.52 ± 1.78	5.46 ± 2.42	9.79 ± 3.53
	Severe	13.04 ± 7.48	8.48 ± 2.73	8.05 ± 6.57	9.71 ± 3.01
4*Connect	None	1.89 ± 0.40	1.71 ± 0.38	1.77 ± 0.39	1.63 ± 0.32
	Slight	7.05 ± 1.32	8.01 ± 1.55	8.07 ± 2.01	7.55 ± 1.63
	Quite	14.19 ± 6.04	11.14 ± 2.60	13.06 ± 5.89	13.71 ± 4.31
	Severe	18.78 ± 9.31	12.15 ± 3.79	13.60 ± 6.17	12.22 ± 2.65
4*Covtype	None	0.70 ± 0.15	0.56 ± 0.13	0.59 ± 0.12	0.60 ± 0.20
	Slight	2.49 ± 0.64	1.84 ± 0.31	1.67 ± 0.47	1.78 ± 0.60
	Quite	6.69 ± 1.71	4.30 ± 0.98	4.63 ± 1.42	5.08 ± 1.02
	Severe	11.83 ± 6.02	5.40 ± 1.76	11.42 ± 7.99	7.26 ± 4.13
4*Shuttle	None	1.50 ± 0.33	1.27 ± 0.44	1.72 ± 0.63	0.97 ± 0.32
	Slight	2.57 ± 1.19	2.04 ± 0.91	2.60 ± 1.53	1.62 ± 1.25
	Quite	7.28 ± 2.40	4.81 ± 1.31	6.79 ± 1.99	4.63 ± 2.22
	Severe	9.38 ± 2.60	8.78 ± 3.23	10.91 ± 4.22	11.18 ± 6.51

Table 3. Disparity (%) among locations of Devonet and its improvements for each data set and class-probability distribution.

it shows a worse performance than the original Devonet in many cases. Notice also that the standard deviation for *Imprv2* is much greater than the others. Even when the local classifiers were trained on a larger training set, the integration of biased classifiers by simply summing led to a biased global classifier.

– Almost the same performance is reached by *Imprv3* for each distribution of data. The performance of *Imprv1* and *3* is not significantly different, but the former is much simpler than the latter.

Finally, regarding the disparity in accuracy of the algorithms as can be seen Table 3, the higher the heterogeneity the larger the disparity, this result being logical for every algorithm. However, the disparity of *Imprv1* is significantly smaller or equal than that of the others. In view of the above, *Imprv1* is chosen as the best Devonet.

6 Results and Discussion: Devonet Versus Others

Once the best Devonet was chosen in Section 5, it is compared against *Majority Vote* and *Stacking* in this section. Remember that *Imprv1* was chosen as the best version of Devonet, and it is referenced as Devonet-*Imprv1*.

6.1 Experimental Results

Table 4 shows the mean test accuracies and standard deviations of Devonet-*Imprv1*, Majority Vote and Stacking.

As in the previous section, the statistical assessment of the algorithms is performed on the results related to severe heterogeneous distributions of data. A p-value of $p < 0.001$ was obtained in the Kruskal-Wallis test so we can reject the null hypothesis for a level of significance of 0.05. The results of the multiple comparison test showed that,

– Majority Vote performs significantly worse than the others for every data set

– Stacking-*Class*, Stacking-*Prob* and Devonet-*imprv1* are the best algorithms with no significant differences on *Adult* and *Connect4* data sets

– Stacking-*Prob* is the best algorithm on *Covertype* and *Shuttle* data sets, followed by Devonet-*imprv1*

Table 5 shows the disparity among locations of Devonet-*Imprv1*, Majority Vote and Stacking. In

2*DATA SET	HETERO-GENEITY	ALGORITHM			
		Devonet-*Imprv1*	Majority Vote	Stacking-*Class*	Stacking-*Prob*
4*Adult	None	83.22 ± 0.94	82.88 ± 0.87	83.06 ± 0.67	83.09 ± 0.73
	Slight	83.46 ± 0.61	82.11 ± 3.01	83.28 ± 1.56	83.40 ± 1.91
	Quite	83.59 ± 0.86	80.87 ± 3.97	82.91 ± 1.75	83.57 ± 3.92
	Severe	83.15 ± 0.76	80.56 ± 3.35	82.72 ± 2.93	83.47 ± 3.16
4*Connect4	None	75.17 ± 0.64	75.11 ± 0.58	75.03 ± 0.61	75.13 ± 3.46
	Slight	75.25 ± 0.31	75.23 ± 2.36	75.34 ± 2.27	75.28 ± 0.54
	Quite	75.25 ± 0.53	74.33 ± 6.33	75.22 ± 4.10	75.23 ± 3.18
	Severe	75.06 ± 0.50	73.44 ± 4.95	74.83 ± 5.75	75.28 ± 1.78
4*Covertype	None	70.18 ± 0.20	70.26 ± 0.47	67.29 ± 0.39	71.42 ± 0.42
	Slight	70.48 ± 0.30	69.85 ± 0.83	67.87 ± 1.22	71.81 ± 0.46
	Quite	70.38 ± 0.24	69.90 ± 1.46	68.45 ± 2.10	72.14 ± 1.47
	Severe	69.79 ± 1.02	67.52 ± 3.90	68.21 ± 1.87	71.99 ± 1.29
4*Shuttle	None	90.30 ± 1.20	87.12 ± 0.28	88.87 ± 0.63	97.64 ± 0.15
	Slight	90.63 ± 1.36	87.54 ± 0.93	88.76 ± 1.39	97.33 ± 0.34
	Quite	91.40 ± 1.46	86.33 ± 2.02	88.39 ± 2.08	97.43 ± 0.33
	Severe	91.80 ± 0.73	84.75 ± 4.81	88.56 ± 3.62	96.52 ± 1.14

Table 4. Mean test accuracy (%) and standard deviation of Devonet-Imprv1, Majority Vote and Stacking for each data set and class-probability distribution.

2*DATA SET	HETERO-GENEITY	ALGORITHM			
		Devonet-*Imprv1*	Majority Vote	Stacking-*Class*	Stacking-*Prob*
4*Adult	None	2.29 ± 0.32	2.13 ± 0.40	2.05 ± 0.27	2.04 ± 0.61
	Slight	5.13 ± 1.37	8.25 ± 2.55	6.02 ± 1.31	7.62 ± 1.04
	Quite	8.52 ± 1.78	17.24 ± 6.48	8.04 ± 1.96	10.38 ± 2.07
	Severe	8.48 ± 2.73	17.72 ± 5.90	12.02 ± 4.90	10.77 ± 1.80
4*Connect	None	1.71 ± 0.38	1.90 ± 0.44	1.61 ± 0.36	1.63 ± 0.45
	Slight	8.01 ± 1.55	8.09 ± 2.13	7.44 ± 0.93	8.58 ± 1.81
	Quite	11.14 ± 2.60	15.33 ± 2.60	11.59 ± 1.30	11.83 ± 1.99
	Severe	12.15 ± 3.79	16.09 ± 5.90	15.31 ± 4.57	12.25 ± 1.27
4*Covtype	None	0.56 ± 0.13	1.85 ± 0.49	1.80 ± 0.35	1.90 ± 0.49
	Slight	1.84 ± 0.31	2.70 ± 0.69	3.43 ± 1.16	2.44 ± 0.37
	Quite	4.30 ± 0.98	5.76 ± 1.66	6.47 ± 1.76	4.38 ± 1.01
	Severe	5.40 ± 1.76	10.02 ± 2.03	9.06 ± 2.30	5.50 ± 1.88
4*Shuttle	None	1.27 ± 0.44	1.69 ± 0.31	1.94 ± 0.90	0.77 ± 0.30
	Slight	2.04 ± 0.91	2.41 ± 1.00	2.70 ± 1.41	0.82 ± 0.22
	Quite	4.81 ± 1.31	8.14 ± 2.34	5.38 ± 2.43	1.12 ± 0.58
	Severe	8.78 ± 3.23	14.15 ± 3.19	8.62 ± 5.14	2.34 ± 1.56

Table 5. Disparity (%) among locations of Devonet-Imprv1, Majority Vote and Stacking for each data set and class-probability distribution.

order to select the best algorithm in terms of disparity, the Kruskal-Wallis test was applied. A p-value of $p < 0.001$ was obtained for each data set so we can reject the null hypothesis for a level of significance of 0.05. A multiple comparison test shows that only Stacking-*Prob* outperforms Devonet-*Imprv1* when *Shuttle* data set are considered.

6.2 Discussion

As can be inferred from Table 4,

– The impact of the heterogeneity of the distribution of data on *Majority Vote* is not negligible. On average, its performance falls by 2.28% from homogeneous to severe heterogeneous distributions. Moreover, statistical tests shows that this degradation in performance can not be explained as the effect of randomness.

– In general terms, Stacking-*Prob* performs better than Stacking-*Class* but at the expense of a higher complexity in the *meta*-knowledge level.

– In terms of accuracy, we can say that Devonet-*Imprv1* lies somewhere in between Stacking-*Class* and Stacking-*Prob*.

In summary, we can say that Stacking-*Prob* performs slightly better than Devonet-*Imprv*. However, notice that the complexity of the *meta*-knowledge level of Stacking in real-world large-scale data sets could be an issue. For example, considering a distributed environment with 100 locations and 10 classes in the domain places the size of the *meta*-knowledge level in $100 \times 10 = 1,000$ features. Training a classifier on a data set with $1,000$ features could be a problem depending on the number of samples. Moreover, notice that we may also have to deal with much more complex environments and domains. On the other hand, Devonet-*Imprv1* shows a good scalability due to the GA is computed in a distributed manner. Thus, we believe that Devonet-*Imprv1* is a more suitable algorithm for real-world problems.

7 Conclusions

In this research, three different improvements of the distributed learning algorithm Devonet were proposed in order to overcome several issues of the original algorithm. With this aim, a distributed computation of the fitness function of the GA, an integration method for using all data available for training, and a combination of both approaches were followed. Even when we simulated distributed environments in a single computer, like most papers in the literature, we took into account inherent issues regarding distributed data which usually do not drawn the attention of researchers; e.g. data privacy, communications costs or data skewness. Devonet allows privacy-preserving classification, since no raw data is exchanged across locations, as well it minimizes communications, since only the classifiers which have a negligible size with respect to the data are exchanged. In this research, we shed light on the distributions of data across the locations. Up to the authors' knowledge, this is a novel approach in the literature though real-world application usually have to deal with heterogeneous distributions of data. In particular, the first improvement shows the best tradeoff between performance on different distributions of data and simplicity, while maintaining privacy-preserving classification and reduced communication costs. Comparisons with other distributed algorithms show the validity of Devonet, in particular of its first improvement.

As future work, we plan to extend this research in several directions, including the assessment of distributed algorithms in terms of computational and communication costs or scalability; the assessment of distributed algorithms when new data do not follow the distribution of the training samples; or the development of a clustering method wrapped in Devonet for combining only similar knowledge in order to build the global classifier. We believe that the last point will be particularly useful in heterogeneous distributions of data in which integrating *all* local classifiers with noticeable diversity knowledge may not be the answer.

Acknowledgements

This work was supported in part by the Spanish Ministry of Science and Innovation (MICINN), Grant code TIN2009-10748, and by the Xunta de Galicia, projects PGIDT-08TIC012105PR and CN2011/007, both partially supported by FEDER funds. Diego Peteiro-Barral acknowledges the support of Xunta de Galicia under *Plan I2C* Grant Program.

References

[1] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data mining and knowledge discovery*, 3(2):131–169, 1999.

[2] J. Catlett. *Megainduction: machine learning on very large databases*. PhD thesis, School of Computer Science, University of Technology, Sydney, Australia, 1991.

[3] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20:161–168, 2008.

[4] S. Sonnenburg, G. Ratsch, and K. Rieck. Large scale learning with string kernels. *Large Scale Kernel Machines*, pages 73–104, 2007.

[5] C. Moretti, K. Steinhaeuser, D. Thain, and N.V. Chawla. Scaling up classifiers to cloud computers. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)*, pages 472–481, 2008.

[6] S. Krishnan, C. Bhattacharyya, and R. Hariharan. A randomized algorithm for large scale support vector learning. In *Proceedings of Advances in Neural Information Processing Systems*, pages 793–800, 2008.

[7] R. Raina, A. Madhavan, and A.Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 873–880, 2009.

[8] D. Sculley. Large scale learning to rank. In *NIPS 2009 Workshop on Advances in Ranking*, 2009.

[9] L. Tang, S. Rajan, and V.K. Narayanan. Large scale multi-label classification via metalabeler. In *Proceedings of the 18th international conference on World Wide Web*, pages 211–220. ACM, 2009.

[10] F.J. Huang and Y. LeCun. Large-scale learning with svm and convolutional for generic object categorization. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 284–291. IEEE, 2006.

[11] S. Sonnenburg, G. Ratsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN Machine Learning Toolbox. *Journal of Machine Learning Research*, 11:1799–1802, 2010.

[12] G. Tsoumakas. Distributed Data Mining. *Database Technologies: Concepts, Methodologies, Tools, and Applications*, pages 157–171, 2009.

[13] P.K. Chan and S.J. Stolfo. Toward parallel and distributed learning by meta-learning. In *AAAI workshop in Knowledge Discovery in Databases*, pages 227–240, 1993.

[14] W. Davies and P. Edwards. Dagger: A new approach to combining multiple models learned from disjoint subsets. *Machine Learning*, 2000:1–16, 2000.

[15] A. Lazarevic and Z. Obradovic. The distributed boosting algorithm. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, page 316. ACM, 2001.

[16] G. Tsoumakas and I. Vlahavas. Effective stacking of distributed classifiers. In *ECAI 2002: 15th European Conference on Artificial Intelligence, July 21-26, 2002, Lyon France: including Prestigious Applications of Intelligent Systems (PAIS 2002): proceedings*, page 340. Ios Pr Inc, 2002.

[17] N. Chawla, L. Hall, K. Bowyer, T. Moore, and W. Kegelmeyer. Distributed pasting of small votes. *Multiple Classifier Systems*, pages 52–61, 2002.

[18] B. Guijarro-Berdiñas, D. Martínez-Rego, and S. Fernández-Lorenzo. Privacy-Preserving Distributed Learning Based on Genetic Algorithms and Artificial Neural Networks. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 195–202, 2009.

[19] E. Castillo, O. Fontenla-Romero, B. Guijarro-Berdiñas, and A. Alonso-Betanzos. A global optimum approach for one-layer neural networks. *Neural Computation*, 14(6):1429–1449, 2002.

[20] O. Fontenla-Romero, B. Guijarro-Berdiñas, B. Pérez-Sánchez, and A. Alonso-Betanzos. A new convex objective function for the supervised learning of single-layer neural networks. *Pattern Recognition*, 43(5):1984–1992, 2010.

[21] G. Carayannis, N. Kalouptsidis, and D.G. Manolakis. Fast recursive algorithms for a class of linear equations. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30:227–239, 1982.

[22] A. Bojańczyk. Complexity of solving linear systems in different models of computation. *SIAM Journal on Numerical Analysis*, 21(3):591–603, 1984.

[23] S.M. Weiss and C.A. Kulikowski. *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann, San Francisco, 1991.

[24] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[25] J. Kittler. Combining classifiers: A theoretical framework. *Pattern Analysis & Applications*, 1(1):18–27, 1998.

[26] D.H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[27] M. Hollander and D.A. Wolfe. Nonparametric statistical methods. 1999.

[28] J.C. Hsu. *Multiple comparisons: theory and methods*. Chapman & Hall/CRC, 1996.

[29] The MathWorks. MATLAB - The Language Of Technical Computing. http://www.mathworks.com/products/matlab/, 2010. [Online; accessed 15-August-2010].