UNIVERSITY OF A CORUÑA

FACULTY OF INFORMATICS

Department of Computer Science

Ph.D. Thesis

Novel Methods in Distributed Machine Learning for Large Datasets

Author: Diego Peteiro Barral
Advisors: Bertha Guijarro Berdiñas
Óscar Fontenla Romero

A Coruña, September 2015

September 23, 2015
UNIVERSITY OF A CORUÑA

FACULTY OF INFORMATICS
Campus de Elviña s/n
15071 - A Coruña (Spain)

Abstract

Data is growing at an unprecedented pace. With the variety, speed and volume of data flowing through networks and databases, it has become more and more difficult to find patterns that lead to meaningful conclusions. At the same time, organizations need to find ways to make sense of all of this data. Unlocking the most value from large, varied sets of information requires a newer approach based on machine-learning. Machine learning allows a system to analyze hundreds of variables simultaneously, along with how they interconnect and it is well-suited to complex problems. However, the majority of machine learning algorithms were designed under the assumption that the data would be represented as a single memory-resident table. For large volumes of data these structures will certainly not fit in system memory. Thus, distributed computing have become essential, due to both speed and memory constraints. In this thesis, we concentrate on methods that are suitable for very large data and that have the potential for distributed implementation. Our contributions are two-fold. First, we implement methods for improving the scalability of training algorithms. Second, we develop training methods under the effect of skewed data distributions.

Resumen

La cantidad de datos almacenados actualmente está creciendo a un ritmo sin precedentes. Con la variedad, velocidad y volumen de datos transmitiéndose a través de redes de comunicación y bases de datos, encontrar patrones relevantes en estos datos que conduzcan a conclusiones significativas se ha convertido en un reto. En este contexto, el aprendizaje automático se ha convertido en una pieza fundamental para extraer el mayor valor posible de estos conjuntos de datos tan grandes y diversos. El aprendizaje automático permite analizar cientos de variables simultáneamente, así como la interacción entre ellas, y es muy adecuado para problemas complejos. Sin embargo, la mayoría de algoritmos de aprendizaje fueron diseñados con la suposición de que los datos estarían representados en la memoria principal de un computador en formato de tabla pero, con el volumen de datos actual, estas estructuras son demasiado grandes para ser almacenadas como una única tabla en memoria principal. Así, la computación distribuida se ha convertido en un paradigma esencial para enfrentar las restricciones actuales en términos de velocidad y almacenamiento. En esta tesis, nos centramos en métodos que son adecuados para trabajar con grandes volumenes de datos y que tienen el potencial de ser implementados de forma distribuida. Nuestra contribución tiene dos vertientes; en primer lugar, se implementan métodos para mejorar la escalabilidad de algoritmos de aprendizaje automático y, en segundo lugar, se desarrollan métodos de aprendizaje que muestran sesgos en las distribuciones de los datos.

Resumo

A cantidade de datos almacenados actualmente está crecendo a un ritmo sen precedentes. Coa variedade, velocidade e volume de datos transmitíndose a través de redes de comunicación e bases de datos, atopar patróns relevantes nestes datos que conduzan a conclusións significativas converteuse nun reto. Neste contexto, a aprendizaxe automática converteuse nunha peza fundamental para extraer o maior valor posible destes conxuntos de datos tan grandes e diversos. A aprendizaxe automática permite analizar centos de variables simultáneamente, así como a interacción entre elas, e é moi adecuado para problemas complexos. Con todo, a maioría de algoritmos de aprendizaxe foron deseñados coa suposición de que os datos estarían representados na memoria principal dun computador en formato de táboa pero, co volume de datos actual, estas estruturas son demasiado grandes para ser almacenadas como unha única táboa en memoria principal. Así, a computacin distribuída converteuse nun paradigma esencial para enfrontar as restricciones actuais en términos de velocidade e almacenamiento. Nesta tese, centrámonos en métodos que son adecuados para traballar con grandes volumenes de datos e que teñen o potencial de ser implementados de forma distribuída. A nosa contribución ten dúas vertentes; en primeiro lugar, impleméntanse métodos para mellorar a escalabilidade de algoritmos de aprendizaxe automática e, en segundo lugar, desenvólvense métodos de aprendizaxe que mostran sesgos nas distribucións dos datos.

Contents

List of Figures

List of Tables

Part I

Distributed Machine Learning

Chapter 1

Introduction

Machine learning aims to extract knowledge from data, relying on fundamental concepts in computer science, statistics, probability and optimization. Learning algorithms enable a wide range of applications, from everyday tasks to bleeding edge applications. In the age of *Big Data*, with datasets rapidly growing in size and complexity, machine learning techniques are fast becoming a core component of large-scale data processing pipelines. Scalability has become one of those core concept of Big Data. It's all about scaling up.

1.1 REASONS FOR SCALING UP MACHINE LEARNING TO LARGE DATASETS

The growth of data generated globally each year is 40%. According to a 2013 study, there were a massive number, over 550 billion, of documents on the Web, mostly in the invisible Web or Deep Web. With the unprecedented rate at which data is being collected today in almost all fields of human endeavor, there is an emerging economic and scientific need to extract useful information from it. Big data and big data analytics are allowing companies to gain deep insights into all aspects of business. For example, many companies already have data-warehouses in the petabyte-scale. Similarly, scientific data is reaching gigantic proportions, e.g., NASA space missions, Human Genome Project. Organizations that have the ability to handle large volumes of data in real time, undertake predictive modeling and automate decision-making and action-taking will be better able to exploit the insights gained from analyzing big data. High-performance, scalable, parallel, and distributed computing is crucial for ensuring system scalability and interactivity as datasets continue to grow in size and complexity (Zaki & Ho, 2000). In this section we showcase some of the outstanding research issues highlighted in (Zaki, 2000; Bekkerman, Bilenko, & Langford, 2011) for designing

and implementing the next generation of large-scale learning methods.

HIGH INPUT DIMENSIONALITY. In some applications, data points are represented by a very large number of features. Tasks involving natural language, images, or video can easily have several millions of input features, far exceeding the range of $10 - 1000$ considered common until recently. With many features the dimensionality of the feature space will increase, leading to a general increase in distances between data points, leading to low data density. However problems will arise from learning models in high dimensional spaces if there are insufficient training samples to learn from. In the usual case of a finite number of data points, it was shown that there is an optimal measurement dimension after which the test accuracy begins to drop. The optimum dimension is a function of the number of data points, increasing with greater volumes of data. This is a well known effect in machine learning and is known as either Hughes' phenomenon (Hughes, 1968), or the curse of dimensionality. Namely that as the number of dimensions of a measurement space increases, more data points are needed to accurately specify the probability distributions in the high-dimensional space (Bailey, 2001). As if this were not enough, current methods are only able to hand a few thousand dimensions or attributes. In general, the complexity of different mining algorithms may not be linear in the number of dimensions, and new distributed methods are needed that are able to handle large number of attributes.

Parallelizing or distributing the computation across features can thus be an effective pathway for scaling up computation to richer representations, or just for speeding up algorithms that naturally iterate over features, such as decision trees.

LARGE NUMBER OF INSTANCES. Databases continue to increase in size. In many domains, the number of instances is extremely large and is increasing at a high pace, such as Internet and finance, making single-machine processing infeasible. Also, more and more devices include sensors continuously logging information resulting in datasets of hundreds of thousands of millions of records. Even if each feature takes only 1 byte to store, datasets collected over time can easily reach the scale of terabytes (10^{12} bytes). Current methods are able to handle data in the gigabyte range, but are not suitable for terabyte-scale. Even a single scan for these databases is considered expensive. Most current algorithms are iterative, and scan data multiple times. In general, minimizing the number of data scans is paramount. Another factor limiting the scalability of most mining algorithms is that they rely on in-memory data structures. For large datasets these structures will certainly

not fit in system memory. This means that temporary results will have to be written out to disk or the dataset will have to be divided into partitions small enough to be processed in memory, entailing further data scans.

The preferred way to effectively process datasets with large number of instances is to combine the distributed storage and bandwidth of a cluster of machines. Several computation frameworks have recently emerge to ease the use of large quantities of data, such as MapReduce (Dean & Ghemawat, 2008), DryadLINQ (Yu et al., 2008), Hadoop (Apache, 2014b), and Spark (Apache, 2014a).

INCREMENTAL METHODS. Everyday new data is being collected, and existing data stores are being updated with the new data or purged of the old one. To-date there have been very few parallel or distributed algorithms that are incremental in nature, which can handle updates and deletions without having to recompute patterns or rules over the entire database.

MODEL AND ALGORITHM COMPLEXITY. A number of complex learning algorithms either rely on nonlinear models or employ computationally expensive routines, such as decision tree ensembles or multilayer neural networks. These complex learning algorithms are needed for high-accuracy learning on data that has inherently nonlinear structure with respect to the basic features. In this case, although the data may easily fit on one machine, the learning process may simply be too slow. This is the case for some learning algorithms for which the computational complexity is exponential in the number of training data points.

For problems of this nature, parallel or distributed implementations appear viable an have been employed successfully, allowing the use of complex algorithms and models for large datasets.

TIME CONSTRAINTS. In some applications, predictions have to be made in real time. Latency issues arise in any situation where systems are waiting for a prediction, making the overall performance of the system decrease.

Utilizing highly parallelized or distributed hardware architectures has been found effective (Vydyanathan, Catalyurek, Kurc, Sadayappan, & Saltz, 2007; Subhlok & Vondran, 1996; Moreira, Valente, & Bekooij, 2007).

PARAMETER TUNING AND MODEL SELECTION. In parameter tuning, the learning algorithm is run multiple times with different settings, followed by validation on a validation set. During statistical significance testing procedures such as cross-validation or bootstrapping, training and testing is

performed repeatedly on different datasets subsets, with results aggregated for subsequent measurement of statistical significance.

The practice of developing, tuning, and evaluating learning algorithms relies on workflow that is embarrassingly parallel: it requires no intercommunication between the tasks with independent executions. In so-called embarrassingly parallel problems, a computation consists of a number of tasks that can execute more or less independently, without communication. These problems are usually easy to adapt for parallel execution. Usefulness of parallel and distributed platforms is obvious for these tasks, as they can be easily performed concurrently without the need to parallelize actual learning algorithms.

The need for large-scale learning algorithms is real and immediate. Parallel and distributed computing is essential for providing scalable, incremental and interactive solutions. This field offers many interesting research directions to pursue.

1.2 THE DISTRIBUTED LEARNING SETTING

In a standard machine learning setting, the problem of learning from a single site is summarized as given a dataset $\mathcal{D}$ and a performance criterion $\mathcal{E}$, the learning algorithm $\mathcal{L}$ constructs a hypothesis h that optimizes $\mathcal{E}$. The dataset $\mathcal{D}$ consists of a single set of training examples of attribute values where one of the attributes corresponds to the desired output, in a supervised learning environment, and the others represent the inputs to the learning algorithm. In a distributed setting, each site stores only a fragment of the dataset (Caragea, Silvescu, & Honavar, 2004). There are two common types of data fragmentation —horizontal and vertical fragmentation. In horizontal fragmentation, the examples of a dataset are split across multiple sites. Each individual partition is referred to as a shard or dataset shard. Horizontal fragmentation is illustrated in the examples presented in (Kargupta, Byung-Hoon, & Johnson, 1999) and in Figure 1.1. Some prototypical examples where one could find horizontal fragmentation are the following;

> Case I. *Two financial organizations want to cooperate for preventing fraudulent intrusion into their computing systems. They need to share data relevant to fraudulent intrusion.*
>
> Case II. *A multinational corporation has thousands of establishments throughout the world and wants to analyze the customer transaction records for developing a successful business strategy quickly.*

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.1	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*

(a) Shard #1.

Sepal length	Sepal width	Petal length	Petal width	Species
5.0	3.6	1.4	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*

(b) Shard #2.

Figure 1.1: Example of horizontal fragmentation using a subset of the *Iris* dataset.

Case III. *Consider a defense organization wherein several sensor systems are monitoring a situation and collecting data. A sensor network consists of spatially distributed autonomous sensors to monitor physical or environmental conditions.*

In vertical fragmentation (see Figure 1.2), the attributes are split. Next, we present some examples of vertical fragmentation drawn from (Kargupta et al., 1999).

Sepal length	Sepal width	Species
5.1	3.5	*I. setosa*
4.9	3.0	*I. setosa*
4.7	3.2	*I. setosa*
4.6	3.1	*I. setosa*
5.0	3.6	*I. setosa*
5.4	3.9	*I. setosa*
4.6	3.4	*I. setosa*

(a) Partition #1.

Petal length	Petal width	Species
1.4	0.1	*I. setosa*
1.4	0.2	*I. setosa*
1.3	0.2	*I. setosa*
1.5	0.2	*I. setosa*
1.4	0.2	*I. setosa*
1.7	0.4	*I. setosa*
1.4	0.3	*I. setosa*

(b) Partition #2.

Figure 1.2: Example of vertical fragmentation using a subset of the *Iris* dataset.

Case IV. *Consider a group of epidemiologists, studying the spread of hepatitis C in the US. They are interested in detecting any underlying relation of the emergence of hepatitis C in US with the weather conditions. They have access to a large hepatitis C dataset at the CDC and an environmental dataset at EPA.*

Case V. *A drug manufacturing company is studying the risk factors of breast cancer. It has a mammogram image database and several databases containing patient tissue analysis results, food habits, age, etc. The company wants to find out if there is any correlation between the breast cancer markers in the mammogram images with the tissue features or the age or the food habits.*

If a dataset $\mathcal{D}$ is distributed among the sites $1, \ldots, P$ containing fragments of the dataset $\mathcal{D}_1, \ldots, \mathcal{D}_P$ it is usually assumed that the fragments contain all the information needed to rebuild the complete dataset $\mathcal{D}$. Note also that distributed datasets can exhibit mixtures of horizontal and vertical fragmentation (see Figure 1.3).

Sepal length	Sepal width	Species
5.1	3.5	*I. setosa*
4.9	3.0	*I. setosa*
4.7	3.2	*I. setosa*
4.6	3.1	*I. setosa*

(a) Partition #1.

Petal length	Petal width	Species
1.4	0.1	*I. setosa*
1.4	0.2	*I. setosa*
1.3	0.2	*I. setosa*
1.5	0.2	*I. setosa*

(b) Partition #2.

Sepal length	Sepal width	Species
5.0	3.6	*I. setosa*
5.4	3.9	*I. setosa*
4.6	3.4	*I. setosa*

(c) Partition #3.

Petal length	Petal width	Species
1.4	0.2	*I. setosa*
1.7	0.4	*I. setosa*
1.4	0.3	*I. setosa*

(d) Partition #4.

Figure 1.3: Example of mixed—hybrid fragmentation using a subset of the *Iris* dataset.

The application of standard machine learning algorithms in distributed environments as those listed above requires collecting the data from physically distributed sites to a single data site for centralized processing. However, this is usually futile for the following reasons (Tsoumakas & Vlahavas, 2009).

PRIVACY. There are many machine learning applications that learn from private and sensitive data—medical and financial records, etc. The monolithic storage of such data may put their privacy at risk because data may be delivered over unsecured networks. In some other cases, organizations may want to cooperate but they do not want to share their data because they want to maintain competitive advantage. For example, taking the previous cases,

Case I. *The two financial organizations are not allowed to share the data because they are private. Therefore, combining the databases is not feasible. Standard machine learning algorithms cannot handle this situation.*

STORAGE COST. The cost of storing the entire dataset in a single site is substantially larger than the sum of the costs of storing smaller subsets of data in several sites. In some cases, the requirements for monolithic storage are vast. A classical example is data from images of Earth and space. The size of these datasets is on the exabyte scale. The monolithic storage of such big datasets would demand an enormous data warehouse of enormous cost.

Case II. *Collecting all the data from thousands of establishments to a single site may be quite impractical.*

COMMUNICATION COST. The deliver of a huge volume of data over a network may take too much time. Even a small volume of data may cause problems in a wireless network with limited bandwidth. Note also that communication may be a continuous overhead because databases are not constant and unchangeable. On the contrary, it is common to have databases that are frequently updated with new data or data streams that constantly record information—remote sensing, sports statistics, etc.

Case III. *Fast analysis of incoming data and quick response is imperative. Communicating all the data to a central site may consume too much time. Moreover, this approach is not scalable for state-of-the-art systems with very large number of sensors.*

COMPUTATIONAL COST. The cost of learning the entire dataset is considerably larger than the sum of the costs of learning smaller shards of data that also could be executed in parallel. Furthermore, in many cases it is likely to be impossible to learn the entire dataset because most existing machine learning algorithms were not designed to handle big data. In particular, the majority were designed under the assumption that the dataset would be represented as a single memory-resident table (Provost & Kolluri, 1999). Note that large inputs that do not fit in main memory become a bottleneck because of the cost of scanning data from secondary storage.

Cases IV & V. *Data are at different places and analyzing them using a standard machine learning algorithm will require combining big datasets at a single site and learning from a very big dataset that may not fit in main memory, leading to a big computational cost and also a big storage cost.*

The increasing interweave of computing and communications is likely to demand distributed environments. In this context, distributed machine learning approaches the problem of learning from distributed data with distributed computation. In a distributed machine learning setting, the problem of learning can be summarized as follows. Given the fragments $\mathcal{D}_1, \ldots, \mathcal{D}_N$ of a entire dataset $\mathcal{D}$ distributed across the sites $1, \ldots, N$ and a performance criterion $\mathcal{E}$, the learning algorithm $\mathcal{L}_d$ constructs a hypothesis $\hbar$ that optimizes $\mathcal{E}$. In this context, a distributed machine learning algorithm $\mathcal{L}_d$ is *exact* in comparison with a standard learning algorithm $\mathcal{L}$ if the hypotheses $\hbar$ are identical. More specifically (Caragea, Silvescu, & Honavar, 2001),

HORIZONTAL FRAGMENTATION. It has the following property $\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_N = \mathcal{D}$ where $\cup$ denotes the union of sets across samples. Thus, a distributed learning algorithm $\mathcal{L}_d$ is exact if

$$\mathcal{L}_d(\mathcal{D}_1, \ldots, \mathcal{D}_N) = L(\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_N) = L(\mathcal{D})$$

that is, a distributed learning algorithm is exact if the model learned from the different subsets of samples is the same as the model learned from the union of the different subsets of samples into a single, joint dataset.

VERTICAL FRAGMENTATION. It has the following property $\mathcal{D}_1 \times \cdots \times \mathcal{D}_N = \mathcal{D}$ where $\times$ denotes the union of sets across features. Thus, a distributed learning algorithm $\mathcal{L}_d$ is exact if

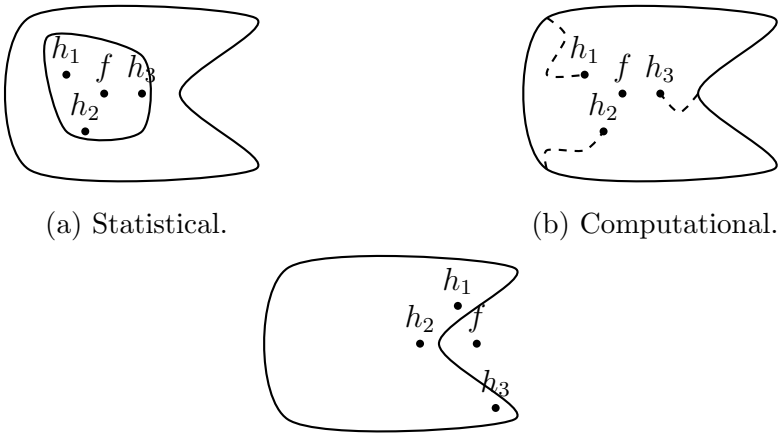
$$\mathcal{L}_d(\mathcal{D}_1, \ldots, \mathcal{D}_N) = L(\mathcal{D}_1 \times \cdots \times \mathcal{D}_N) = L(\mathcal{D})$$

that is, a distributed learning algorithm is exact if the model learning from the different subsets of features is the same as the model learned from the union of the different subsets of features into a single, joint dataset.

1.3 FOUNDATIONS OF DISTRIBUTED LEARNING

Many distributed learning algorithms have their foundations in ensemble learning. Ensemble methods have gained attention within machine learning from the late 1990s. Examples of these techniques include bagging (Breiman, 1996), boosting (Freund & Schapire, 1995, 1996), mixtures of experts (Freund, Schapire, Singer, & Warmuth, 1997; Jacobs, Jordan, Nowlan, & Hinton, 1991), etc. These methods assign partitions of the training data to different base classifiers that are independently trained. Subsequently, the individual classifiers are combined in some manner to classify new instances.

Ensembles are often more accurate than the individual base classifiers themselves. A necessary and sufficient condition for this is that the classifiers have an error rate better than random guessing and they are diverse, i.e., they make different errors on new instances (Hansen & Salamon, 1990). There are three fundamental reasons for supporting that it is possible to build effective ensembles (T. Dietterich, 2000),



(a) Statistical.

(b) Computational.

(c) Representational.

Figure 1.4: Three fundamental reasons why an ensemble may work better than a single classifier.

STATISTICAL. A learning algorithm can be viewed as searching a space of hypotheses to find the best one. The statistical problem arises when the volume of data available is too small compared to the size of the hypothesis space and the learning algorithm can find many hypotheses that are equally optimal. Thus, the method for building the ensemble can *average* their individual predictions and reduce the risk of learning the worst classifier. Figure 1.4 shows this situation. Let f denote the true function in the space $\mathcal{H}$ of hypotheses. As can be seen, it is possible to find a good approximation of f by averaging the individual hypotheses $h_1, h_2, \ldots$.

COMPUTATIONAL. Many learning algorithms perform local search that may get stuck in local minima, e.g., neural networks trained by gradient descent. In cases where there are enough data to avoid the statistical problem, it may still be very difficult for a learning algorithm to find the best hypothesis from the computational point of view. In an ensemble, the base classifiers start the local search from different points in the hypothesis space

using a smaller subset of data. Ensembles are inherently scalable because an increase in the volume of data can be compensated by increasing the number of base classifiers.

REPRESENTATIONAL. In some applications of machine learning, the true function f cannot be approximated by any of the hypotheses in $\mathcal{H}$. This issue is subtle because there are many learning algorithms that are universal approximators, i.e., given enough training data these algorithms can explore the space of all possible hypotheses (Hornik, Stinchcombe, & White, 1989; Wang, 1992). Nonetheless, these algorithms will explore only a subset of the entire hypothesis space when giving finite data.

Ensemble methods aim to reduce these three shortcomings of standard machine learning algorithms. One might assimilate ensemble methods into the framework of distributed learning, but ensemble methods are generally designed in the classical model for machine learning that assume that the training set is available on a single site. In general, the focus of ensemble learning is on the statistical and algorithmic advantages of learning with an ensemble and not on the nature of learning under communication constraints. Nonetheless, many fundamental insights into distributed learning have arisen from ensemble methods (Predd, Kulkarni, & Poor, 2006).

1.4 KNOWLEDGE TO BE COMBINED

In general, there exist two types of knowledge to be combined: the base classifiers themselves or the predictions of the base classifiers (Chan & Stolfo, 1993a). In order to combine the classifiers themselves, there is the need of defining a uniform representation to encapsulate all other representations without losing relevant knowledge. But this is difficult, e.g., it is difficult to define a uniform representation to combine the distance function of a nearest neighbor algorithm with the tree of a decision tree algorithm. Indeed, it is difficult to combine the base classifiers themselves even when they are trained with the same learning algorithm.

An alternative approach is to combine the predictions of the base classifiers. These predictions can be categorical or non-categorical, i.e., associated with some quantitative measure such that probabilities, confidence values, distances, etc. In this approach, the problem of finding a uniform representation is much less severe, e.g., quantitative measures can be treated as categorical by selecting the class with the highest confidence.

1.5 PARALLEL AND DISTRIBUTED COMPUTING

Employing parallel and distributed systems gain increased performance in machine learning applications driven by concurrent execution of tasks that are otherwise performed serially. There are two major directions in which this concurrency is realized: *data parallelism* and *task parallelism* (Bekkerman et al., 2011). For many algorithms, scaling up can be most efficient achieved by a mixture of data and task parallelism.

DATA PARALLELISM. Data parallelism refers to executing the same computation on multiple inputs concurrently. It is a natural choice for many machine learning applications and algorithms that accept input data as a batch of independent samples. There are two orthogonal directions for achieving data parallelism: horizontal fragmentation and vertical fragmentation. The most basic example of data parallelism is encountered in embarrassingly parallel algorithms, where the computation is split into concurrent subtasks requiring no intercommunication, which run independently on separate data subsets. A related simple implementation of data parallelism occurs within the *master-slave communication model*: a master process distributes the data across slave processes that execute the same computation.

TASK PARALLELISM. Unlike data parallelism defined by performing the same computation on multiple inputs simultaneously, task parallelism refers to dividing the overall algorithm into parts, some of which can be executed concurrently (see Figure 1.5). The partitioning of an algorithm into tasks can be represented by a directed acyclic graph, with nodes corresponding to individual tasks, and edges representing inter-task dependencies. Data flow between tasks occurs naturally along the graph edges. A prominent example of such paradigm is *MapReduce*.

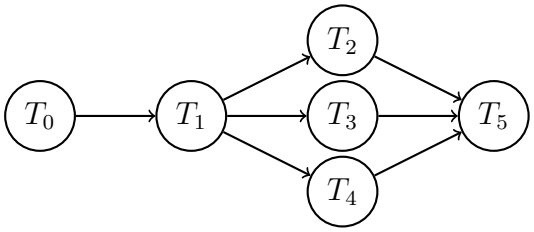


Figure 1.5: Example of task parallelism. Tasks T_2, T_3, and T_4 can be executed concurrently.

1.6 THE MAPREDUCE PARADIGM

For large datasets, it has proven particularly valuable to think about processing in terms of the same operation being applied independently to each item in the dataset, either to produce a new dataset, or to produce summary result for the entire dataset. This way of thinking often works well on parallel hardware, where each of many processors can handle one or more data items at the same time. MapReduce (Dean & Ghemawat, 2008) is a simple model for distributed computing that abstracts away many of the difficulties in parallelizing data management operations across a cluster. MapReduce provides a framework for performing a two-phase distributed computation on a (large) dataset $\mathcal{D}$. In the *Map* phase, the system partitions $\mathcal{D}$ into a set of disjoint units that are assigned to worker processes, known as *mappers*. Each mapper in parallel with the others applies a user-specified map function to its assigned data. The output of the map function is a set of key-value pairs that are collected by the *Shuffle* phase, which groups them by key. The master process redistributes the output to a series of worker processes called *reducers*, which perform the *Reducer* phase. Each reducer applies a user-specific reduce function to all the values for a key and outputs the value of the reduce function. The collection of final values from all the reducers is the final output of MapReduce.

To demonstrate how MapReduce works we bring the prototypical example of a simple MapReduce program that counts how many times different words appear in a set of documents (Bekkerman et al., 2011). In the Map phase, the set of documents is partitioned into subsets, each of which is assigned to an individual mapper. Each mapper scans its subset of documents and outputs a series of $< word_i, count_i >$ values as the key-value pair, where $count_i$ is the number of times $word_i$ occurs among the subset of documents seen by the mapper. Each reducer takes the values associated with a particular word (key) an aggregates the word counts (values) for each word. The output of the reducer phase are the counts per word across the entire set of documents.

For purposes of simplicity and clarity of explanation in the methods presented in this thesis, we will consider the most simple MapReduce architecture in which each mapper is assigned to one processor and there exist just one reducer that receives every output from the mappers as input and it outputs the final result of the method. This procedure is similar to ensemble learning where the base classifiers are the mappers and the combiner scheme is the reducer. Note that in many of the methods that will be presented and proposed in this thesis, parallelizing the reducer phase would be trivial for practical considerations.

1.7 STRUCTURE OF THIS THESIS

This thesis falls into four parts, which are relatively independent;

I Distributed machine learning. In this chapter, this thesis was intended to provide a brief and general framework of distributed machine learning, starting by walking through the reasons for scaling up machine learning to large data sets, the distributed learning setting, foundations of distributed learning and knowledge to be combined, and finishing by introducing parallel and distributed computing, and the MapReduce paradigm. In the following chapters of this part, this thesis will review the literature and current research on distributed machine learning, and assessment of algorithms, from thinking about how to define performance to evaluating the effectiveness of a distributed system.

II Scaling up learning algorithms. This part of the thesis is committed with four novel distributed learning algorithms able to learn from very large datasets. In general terms, these algorithms will aim to infer a global learner that approximates the results one would get from a single, joint data source. In many cases and applications, data is distributed across different sites for several reasons –e.g. privacy, storage cost, computational cost, etc– but the data is considered to be generated by the same, uniform process. Actually, under this view, distributed data is treated exclusively as a technical issue. Thus, these algorithms have been designed with accuracy and speed in mind.

III When Distribution is part of the semantics. In this part, the problem of learning in a distributed machine learning setting is made more challenging. Real-world distributed data sets almost always present quite strong differences between their partitions, e.g. buying patterns in different supermarkets from different countries. Under this view, the distribution of data should not be treated as a mere technical issue, just because it has deeper implications. Yet in spite of its importance, it has been no fully considered in the literature. In this part, this thesis presents different techniques for learning under these circumstances.

IV Bridging the gap. Finally, this part contains some guidelines to bridge the gap between the previous two parts. It is devoted to the study of distributed systems where the problem is not only the semantics of the different partitions of data, but also the volume of data at each partition.

Chapter 2

Review of Existing Research in Distributed Learning

Many existing methods for distributed learning combine the predictions of the base classifiers and define general frameworks where any base learning algorithm can be used. In this section, we present an overview of several outstanding methods for distributed learning. It is assumed that the distributed datasets have the same set of attributes with homogeneous schema. This assumption is generally true when the datasets belong to the same organization.

2.1 FIXED RULES

Fixed rules are functions that take several classifications as input and give a single classification as output (Kittler, Hatef, Duin, & Matas, 1998; Kittler, 1998). Consider a problem where instance x is to be assigned to one of the J possible classes $c_1, \ldots, c_J$. Let $y_p, p = 1, \ldots, P$ denote the output of the pth classifier. In the output space, each class c_j is modeled by the probability density function $p(y_p \mid c_j)$ and its a priori probability of occurrence is denoted by $p(c_j)$. According to the Bayesian theory, given outputs y_p, the instance x should be assigned to class c_k provided the a posteriori probability of that interpretation is maximum, i.e., assign $x \rightarrow c_k$ if

$$p(c_k \mid y_1, \ldots, y_P) = \max_j p(c_j \mid y_1, \ldots, y_P) \tag{2.1}$$

Although this is a correct statement of the classification problem but it may not be a practicable proposition. 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 $p(y_1, \ldots, y_P \mid c_k)$ which

would be difficult to infer. Therefore, to avoid this problem and overcome complexity of computation, the above rule shall be simplified and expressed in terms of decision support computations performed by the individual classifiers.

When a measure of belief, confidence, or certainty is available, a posteriori probability is estimated as $y = p(c_j \mid x)$. This approach will provide a framework for the development of a range of efficient classifier combination rules. Let $y_{p,j}(x)$ denote the output of the pth classifier in the class jth for the instance x, provided that the outputs are normalized $y_{p,j} = y_{p,j} / \sum_j y_{p,j}$. Some of the most common rules are defined as follows,

— The *product* rule quantifies the likelihood of a hypothesis by combining the a posteriori probabilities generated by the individual classifiers by using a product operation. Thus, assign $x \rightarrow c_k$ if

$$\prod_p y_{p,k}(x) = \max_j \prod_{p=1}^{P} y_{p,j}(x) \tag{2.2}$$

— In some applications it may be appropriate to assume that the a posteriori probabilities will not deviate dramatically from the prior probabilities. Thus, the *sum* rule is obtained as

$$\sum_p y_{p,k}(x) = \max_j \sum_{p=1}^{P} y_{p,j}(x) \tag{2.3}$$

— The *max* rule approximates the sum rule under the assumption of equal a priori probabilities of occurrence of classes,

$$\max_p y_{p,k}(x) = \max_j \max_p y_{p,j}(x) \tag{2.4}$$

where the sum operation will be dominated by the output which provides the maximum support for a particular hypotheses.

— The *min* rule approximates the product rule under the assumption of equal a priori probabilities of occurrence of classes,

$$\min_p y_{p,k}(x) = \max_j \min_{p=1}^{P} y_{p,j}(x) \tag{2.5}$$

where the product operation will be dominated by the output which provides the minimum support for a particular hypotheses.

— Under the assumption of equal priors, the sum rule can be viewed as the average a posteriori probabilities for each class. A robust estimate of the mean is the median. Thus, the *median* rule assigns an instance x to the class that maximizes the average of the a posteriori probabilities for each class,

$$Mdn_p y_{p,k}(x) = \max_j Mdn_{p=1}^{P} y_{p,j}(x) \tag{2.6}$$

where Mdn stands for median.

— The *majority voting* rule is based on the sum rule under the assumption of equal priors and by hardening of the a posteriori probabilities to produce binary valued functions $\Delta_{p,j}(x)$ as

$$\Delta_{p,k}(x) = \begin{cases} 1 & \text{if } y_{p,k}(x) = \max_{j=1}^{J} y_{p,j}(x) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

results in combining decision outcomes rather than combining a posteriori probabilities. This approximation leads to the following rule,

$$\Delta_{p,k}(x) = \max_{j=1}^{J} \sum_{p=1}^{P} \Delta_{p,j}(x) \tag{2.8}$$

which simply counts the votes received for the classes from the individual classifiers. The class which receives the largest number of votes is selected.

As the combination schemes max rule and majority voting are related to the sum rule, they are less sensitive to estimation errors, and are therefore likely to perform better than the min rule which can be derived from the product rule (Kittler et al., 1998).

2.2 META-LEARNING

Meta-learning is a general technique to combine the results of multiple classifiers. Rather than following fixed rules, the approach introduced in (Chan & Stolfo, 1993b) is to meta-learn a set of classifiers whose training data are based on predictions of a set of base classifiers. A base classifier is the outcome of learning directly on *raw* data. A meta-classifier is a classifier that is trained on the predictions produced by a set of base classifiers. Thus, meta-learning can be loosely defined as learning from information generated

by classifiers or it can also be viewed as the learning of meta-knowledge (Prodromidis, Chan, & Stolfo, 2000; Chan & Stolfo, 1993a, 1995).

The authors experimented with three types of meta-learning schemes for combining predictions that we discuss further below. Let $\mathcal{D} = \{(x_n, c_n), n = 1, \ldots, N\}$ denote a dataset where x_n is a vector representing the feature values of the nth instance and c_n is the desired class, randomly split the dataset into $P + 1$ shards $\mathcal{D} = \mathcal{D}_1, \ldots, \mathcal{D}_{P+1}$ where P is the number of processors. Let $\mathcal{D}_{P+1}$ denote an independent subset of data not used for training the base classifiers.

COMBINER. In the combiner scheme, the predictions of the base classifiers on the independent dataset form the meta-level training examples. A *composition* rule determines the input features of these examples from which the meta-classifier is trained. The authors proposed three strategies for the composition rule,

— Form the meta-level training examples with the predictions of the base classifiers on the independent dataset, and the desired class

$$\{y_1(x_n), \ldots, y_P(x_n), c_n\}$$

where $x_n \in \mathcal{D}_{P+1}$, y_p is the output of the pth classifier, and c_n is the desired class of the nth instance. This strategy is similar to the one proposed in Wolpert's stacked generalization (Wolpert, 1992). Stacked generalization is a general method for combining multiple classifiers by learning the way that their output correlates with the desired class. It works by deducing the biases of the classifiers with respect to an independent dataset. This deduction proceeds by generalizing in a second space whose features are the predictions of the classifiers for the instances of the independent dataset, and the desired outputs are the true class for those instances (Ting & Witten, 1999).

— Similar to the previous rule with the addition of the input features of the base training example x_n,

$$\{x_n, y_1(x_n), \ldots, y_P(x_n), c_n\}$$

— Similar to the first composition rule except that the predictions of the base classifiers are binary,

$$\{y_{1,1}(x_n), \ldots, y_{1,J}(x_n), \ldots, y_{P,1}(x_n), \ldots, y_{P,J}(x_n), c_n\}$$

where $y_{p,j}$ denotes the output of the pth classifier in the jth class. Thus, this strategy uses more specialized base classifiers in an attempt to learn the correlation between the binary predictions and the desired class.

Note that these rules are also used to classify new instances. Given a new instance, first the base classifiers make their predictions. Then, the composition rule is applied to form a meta-level example which is classified by the combiner to obtain the final output class.

ARBITER. In the arbiter scheme, the meta-level training examples are drawn from the independent dataset based on the predictions of the base classifiers on this dataset, i.e., the meta-level training examples rely on a particular distribution of the independent dataset. A *selection* rule determines the subset of examples from the independent dataset that the meta-level dataset will contain. The purpose of this rule is to choose examples that are confusing. The authors proposed two strategies for the selection rule,

— Select the instances in which the base classifiers disagree on their classification,

$$\{x_n \mid y_1(x_n) \neq y_2(x_n) \vee \cdots \vee y_{P-1}(x_n) \neq y_P(x_n)\}$$

— Similar to the previous rule with the addition of the instances in which the base classifiers agree but the classification is wrong,

$$\{x_n \mid y_1(x_n) \neq y_2(x_n) \vee \cdots \vee y_{P-1}(x_n) \neq y_P(x_n) \\ \vee (y_1(x_n) = \cdots = y_P(x_n) \wedge y_1(x_n) \neq c_n)\}$$

Once the meta-level training set is formed, an arbiter is trained on this dataset. Then, given a new instance, first the base classifiers and the arbiter make their predictions. Then, the instance is classified by majority voting, breaking ties in favor of the arbiter.

HYBRID. The hybrid scheme merges the combiner and the arbiter schemes. Given the predictions of the base classifiers on the independent dataset, a selection rule draws examples as in the arbiter scheme. However, the meta-level training examples are formed by a composition rule from the distribution of the independent dataset as in the combiner scheme. Thus, the hybrid scheme attempts to improve the arbiter scheme by correcting the predictions on the controversial examples.

2.3 KNOWLEDGE PROBING

Knowledge probing is first proposed in (Guo, Rueger, Sutiwaraphun, & Forbes-Millott, 1997) as a method to extract descriptive knowledge from black-box models like neural networks. The idea behind this method is to obtain a descriptive model based on the predictions of the black-box model on an unlabeled dataset. However, this idea can be easily extended to learn in a distributed environment. The authors of knowldge probing mention some limitations of meta-learning (see previous section for more details on meta-learning) that were overcome with their model. The first one is the problem of knowledge representation. Meta-learning serves the purpose of prediction but lacks in the combination of statistics rather than knowledge from the base classifiers. Moreover, it is noted in (Chan & Stolfo, 1997) that the bias induced by a particular distribution of data has an effect upon the performance of the algorithm.

Knowledge probing can be used to train a descriptive model which learns the meta-knowledge of the base classifiers (Guo & Sutiwaraphun, 1999). The key step is to use an independent dataset to probe the knowledge from the base classifiers. Thus, the meta-classifier will be trained from the dataset formed from the independent dataset as follows,

$$\{x_n, \mathcal{S}(y_1(x_n), \ldots, y_P(x_n))\} \tag{2.9}$$

where $\mathcal{S}$ is a scheme that combines the outputs $y_p, p = 1, \ldots, P$ of the base classifiers. Since the meta-classifier is trained from a dataset whose class values are assigned by a prediction scheme—which integrates the predictions of the base classifiers—it can be considered as an approximation of the combination of the base classifiers.

2.4 PASTING VOTES

Pasting votes was proposed to build ensembles of classifiers from small shards or *bites* of data. In (Breiman, 1999), two schemes were proposed: *importance* vote (Ivote) and *random* vote (Rvote). Ivote act sequentially to generate subsets of data by sampling with replacement used to train classifiers, such that each new train subset of data has more instances that were more likely to be misclassified by the ensemble of classifiers already trained. Thus, the selection of the instances that form the small training dataset of the subsequent classifier relies on the combination of the hypotheses of the previous classifiers. The sampling probabilities depend on the out-of-bag error (Breiman, 1998), that is, a classifier is tested on instances that do not belong to its

training set. This estimation provides a good approximation of the generalization error. Ivote is very similar to boosting, but the bites are much smaller in size than the original dataset. Rvote requires the creation of many random bites of a very small size. Rvote is faster and simpler than Ivote at the expense of accuracy.

In (Chawla, Hall, Bowyer, Moore Jr, & Kegelmeyer, 2002), the authors proposed DIvote and DRvote as a distributed implementation of Ivote and Rvote. The steps of the procedure are summarized as follows,

1. Divide the dataset into P shards and assign each shard to a single processor. Note that P is the number of processors.

2. Build the first bite of data by sampling with replacement on each processor and train a classifier on each bite.

3. Build the subsequent bites of data by selecting the instances that are misclassified by the majority voting of the out-of-bag classifiers, and train a classifier on each bite. These instances are drawn at random from the shards of data.

4. Repeat until the desired number of classifiers have been trained or some convergence criteria are met.

Given a new instance, classify the instance by combining the predictions of the base classifiers by majority voting to obtain the classification.

2.5 EFFECTIVE STACKING

Effective stacking (Tsoumakas & Vlahavas, 2002a, 2002b) is motivated by the problem that arise in stacking-based methods when dealing with large-scale distributed architectures, that is the increase in complexity of the meta-level training examples when the number of processors is very large. Note that the number of input features of the meta-level instances is directly proportional to the number of processors (see section 2.2). Moreover, in meta-learning it is necessary to retain independent instances to train the meta-classifier. The problem is twofold because the base classifiers are deprived of some training examples, and the meta-classifier is only trained on a small subset of the total available data. Effective stacking proposed to circumvent these problems by adopting the following procedure,

1. Divide the dataset into P shards and assign each shard to a single processor. Train a base classifier on each shard and broadcast it to all other processors, i.e., at the end of this step each processor will hold every base classifier.

2. Form the meta-level training examples with the *average* predictions of the out-of-bag base classifiers on the local shard of data, and the desired class—the meta-level instances are formed in every processor by the predictions of all base classifiers except for the local classifier that was trained on such data,

$$\left\{ \frac{1}{P} \sum_{p} y_{p,1}(x_n), \ldots, \frac{1}{P} \sum_{p} y_{p,J}(x_n) \right\}$$

where $y_{p,j}$ is the output of the pth classifier in the jth class.

3. Train P meta-classifiers on the meta-level instances that describes the knowledge of all base classifiers except for the local one in each processor.

Given a new instance, classify the instance by combining the predictions of the meta-classifiers by using the *sum* rule (see section 2.1) to obtain the classification.

2.6 DISTRIBUTED BOOSTING

In (Fan, Stolfo, & Zhang, 1999; Lazarevic & Obradovic, 2002), the authors proposed a method for combining classifiers from multiple sites using a boosting approach (Freund & Schapire, 1996). The main idea of boosting is that the algorithm should focus on the instances that are difficult to classify. In boosting, the instances are drawn using adaptive sampling according to the performance of the previous classifiers to build an accurate ensemble of many weak classifiers. In the distributed version, the classifiers are trained from disjoint partitions of the data set. This algorithm proceeds in a series of rounds. In every round, a base classifier is trained with a different distribution D_t that is modified by weighting the instances differently. Specifically, the distribution is updated to give larger weights to misclassified instances and smaller weights otherwise. Assume there are P distributed sites—processors— where site p stores dataset $\mathcal{D}_p, p = 1, \ldots, P$ with N_p instances. During the boosting rounds t, the processor p maintains a local distribution $\Delta_{p,t}$ and the local weights $w_{p,t}$ that reflect the prediction accuracy on that site. The goal is to emulate the global distribution Δ_t obtained through iterations when standard boosting is applied to a single dataset $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_P$. To approximate the global sampling distribution, the weight vectors $w_{p,t}$ from all distributed sites are merged into a joint weight vector w_t. The weight vector w_t is used to update the global distribution Δ_t. In order to reduce the communication

cost, instead of the entire vectors $w_{p,t}$ that depend on the size of the datasets, only the sums of all their elements are broadcast because there is no need to know the exact values of the elements in w. At the end, the base classifiers are combined into a final hypothesis.

2.7 OTHER METHODS FOR DISTRIBUTED LEARNING

The combination schemes presented in the previous section contemplate data distribution as a technical issue and treat distributed datasets as if they were shards of a single dataset. This has often been regarded as a narrow view of distributed machine learning (Provost, 2000; Wirth, Borth, & Hipp, 2001). Datasets that are inherently distributed frequently show data skew. Data skew primarily refers to a non-uniform distribution in a dataset, when the initial distribution of instances varies between partitions, e.g., datasets that store diseases and causes of death from hospitals around the world, or datasets that store shopping patterns in supermarkets in different regions.

The traditional approach of combining classifiers trained from distributed datasets in an attempt to infer a single global classifier is not appropriate for these cases. There may not exist a single model that optimally describes these distributed datasets. There may exist two or more models. Thus, a straightforward combination of classifiers is not recommended before exploring the relationship between the distributed datasets and classifiers. There is not much work on this area. In (Parthasarathy & Ogihara, 2000), an algorithm to measure the similarity between homogeneous datasets is presented. This measure compares the dataset in terms of how they correlate with the attributes in the database. More relevant for our purposes is the framework for clustering local classifiers trained on distributed datasets presented in (Tsoumakas, Angelis, & Vlahavas, 2004). In that research, the authors introduced the notion of *classifier distance* as a measure of how different two classifiers are based on their predictions. Let $\hbar_1$ and $\hbar_2$ denote two classifiers. The disagreement measure is computed as

$$d(\hbar_1, \hbar_2) = \frac{\sum_n \delta(x_n)}{N} \tag{2.10}$$

where

$$\delta(x) = \begin{cases} 1 & \text{if } \hbar_1(x) = \hbar_2(x) \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

Thus, only the base classifiers are exchanged between distributed sites safeguarding the privacy of the raw data.

Chapter 3

Assessment of the Algorithms

In the past, 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. Nowadays, several sources produce data creating environments with several distributed data sets. Also big datasets collected in a central repository in which processing imposes quite high computing requirements. Then one actually thinks in distributed processing of the data as a way to have a more powerful computing platform.

3.1 EVALUATING THE EFFECTIVENESS OF A DISTRIBUTED SYSTEM

First of all, some definitions and assumptions will be needed (Kumar & Gupta, 1994);

— Parallel system: The combination of a parallel architecture and a parallel algorithm implemented on it. We assume that the parallel computer being used in a homogeneous ensemble of processors, i.e., all processors and communication channels are identical in speed.

— Problem size W: The size of the problem is measured in terms of the size of the dataset. The number of records will the number of input features multiplied by the number of samples.

— Serial Fraction s: The ratio of the serial component of an algorithm to its execution time on one processor. The serial component of the algorithm is that part of the algorithm which cannot be parallelized and has to be executed on a single processor.

— Parallel Execution Time T_p: The time elapsed from the moment a parallel computation starts, to the moment the last processor finishes execution. For a given parallel system, T_p is normally a function of the problem size W and the number of processors p

— Total Parallel Overhead T_o: The sum total of all the overhead incurred due to parallel processing by all the processors. It includes communication costs, non-essential work and idle time due to synchronization and serial components of the algorithm. Mathematically, $T_o = p \times T_P - T_S$, where T_s is the serial execution time.

In what follows we will use the term *runtime* to refer to the elapsed time taken by the entire system to complete a specified task. We will use the term *workload* of a processor to mean the number of instances held in its associated memory. We will assume that the workload is the same for each processor that is in use in the network. Finally, we will use the term *total workload* to mean the sum of the workloads for each of the processors in use in the network, again measured as a number of instances.

A distributed system can be evaluated in terms of three kinds of performance: its *scale-up*, its *size-up*, and its *speed-up* (Bramer, 2013).

— Scale-up experiments evaluate the performance of the system with respect to the number of processors for a fixed workload per processor. We keep the workload per processor constant and measure the runtime as additional processors are added. Ideally, the runtime measured this way would remain constant. Figure 3.1 plots the runtime against the number of processors. We can see that rather than remaining horizontal, each plot increases as the number of processors increases. This is caused by an additional communications overhead in the network as more processors need to communicate information.

— Size-up experiments evaluate the performance of the system with respect to the total workload for a fixed configuration of processors. We keep the number of processors constant and measure the runtime as the total number of training instances is increased. Figure 3.2 shows a graph of relative runtime against number of instances. Each plot shows and approximately linear size-up, i.e., the runtime is approximately a linear function of the size of the training data.

— Speed-up experiments evaluate the performance of the system with respect to the number of processors for a fixed total workload. We keep the total workload of the system constant and measure the runtime

as the number of processors is increased. This shows how much a distributed algorithm is faster than the serial (one processor) version, as a large dataset is distributed to more and more processors. We can define two performance metrics associated with speed-up. The *speedup factor* S_p is the ratio of the serial execution time T_s to the parallel execution time T_p on p processors. This measures how much the runtime is faster using p processors than just one. An ideal case is that $S_p = p$, but the usual situation is that $S_p < p$ because of communication or other overheads in the system. Mathematically, the total parallel overhead is defined by $T_0 = p \times T_p - T_s$. Figure 3.3 shows a graph of speedup factor against number of processors. This form of display makes it straightforward to see the largest number of processors that has a positive impact on the runtime, for a fixed workload. On the other hand, the *efficiency* E_p is the ratio of speed-up S to the number of processors p. Thus, $E_p = \frac{T_s}{p \times T_p} = \frac{1}{1+\frac{T_p}{T_s}}$. E_p is usually a number between 0 and 1 but can occasionally by a value greater than one, in the case of what is known as *superlinear* speedup.

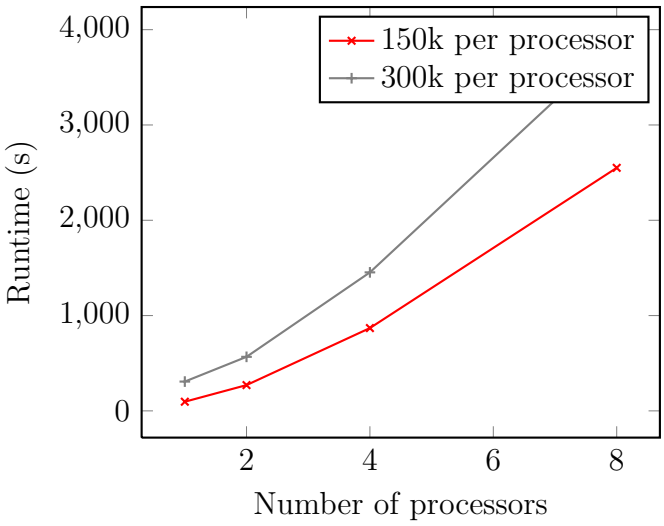


Figure 3.1: Scale-up

Given a parallel architecture and a problem instance of a fixed size, the speedup of a parallel algorithm does not continue to increase with increasing number of processors. The speedup tends to saturate or peak at a certain value. In 1967, Amdahl (Amdahl, 1967) made the observation that if s is the serial fraction in an algorithm, then its speedup is bounded by $1/s$, no matter how many processors are used. For a fixed problem size, the

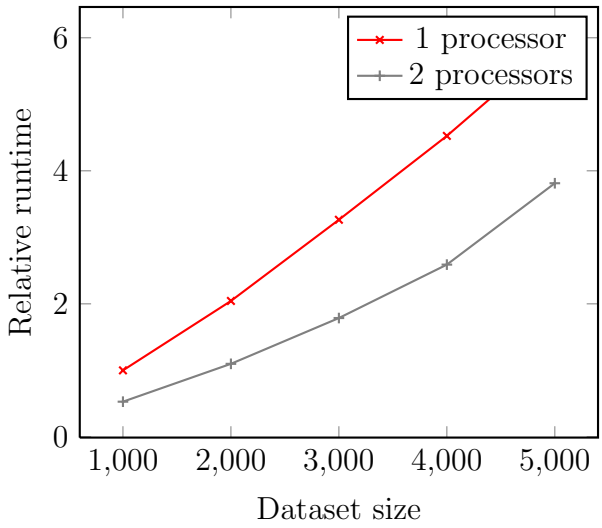


Figure 3.2: Size-up

speedup saturates either because the overheads grow with increasing number of processors or because the number of processors eventually exceeds the degree of concurrency inherent in the algorithm. Kumar and Rao (Kumar & Rao, 1987) developed a scalability metric relating the problem size to the number of processors necessary for an increase in speedup in proportion to the number of processors. This metric is known as the *isoefficiency function* (see Figure 3.4). If a parallel system is used to solve a problem of a fixed size, then the efficiency decreases as p increases. The reason is that T_o increases with p. For many parallel systems, if the problem size W is increased on a fixed number of processors, then the efficiency increases because T_o grows slower than W. For these parallel systems, the efficiency can be maintained at some fixed value (between 0 and 1) for increasing p, provided that W is also increased. For some parallel systems, the maximum obtainable efficiency E_{max} is less than 1. Even such parallel systems are considered scalable if the efficiency can be maintained at a desirable value between 0 and E_{max}. We call such systems scalable parallel systems.

3.2 THINKING ABOUT PERFORMANCE

The measures introduced in the previous section provide a good starting point for evaluating distributed algorithms. Moreover, we could add to this evaluation framework the classical analysis of algorithms' complexity based on O-notation to bound and quantify computational costs. However, something is still missing. Both approaches meet difficulties with many machine

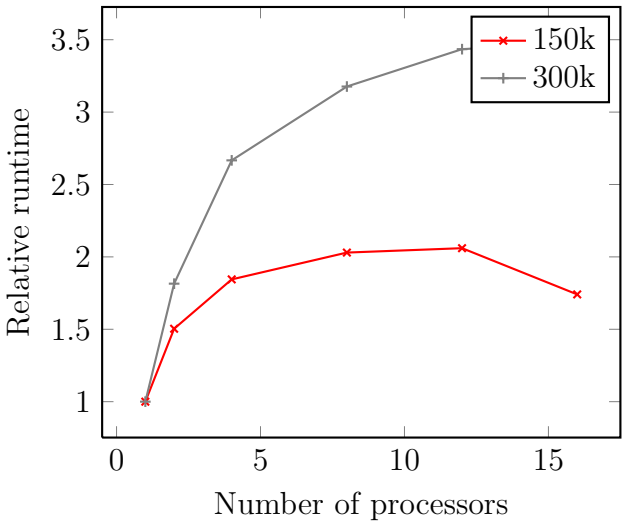


Figure 3.3: Speed-up

learning algorithms, as they often include optimization-based termination conditions for which no formal analysis exists. For example, a typical early stopping algorithm may terminate when predictive error measured on a hold-out test set begins to rise —something that is difficult to analyze because the core algorithm does not have access to this test set by design. The term "performance" is deeply ambiguous for parallel and distributed learning algorithms, as it includes both predictive accuracy and computational speed, each of which can be measured by a number of metrics. The variety of learning problems addressed in the literature makes the presented approaches generally incomparable in terms of predictive performance: the algorithms are designed to optimize different objectives in different settings. Even in those cases where the same problem is addressed, differences in application domains and evaluation methodology typically lead to incomparability in accuracy results (Bekkerman et al., 2011).

In this novel situation, classical evaluation methods and metrics (Gama, Rodrigues, & Sebastião, 2009) 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 (Urban, Défago, & Schiper, 2001). 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 (Tsoumakas et al., 2004), e.g.,

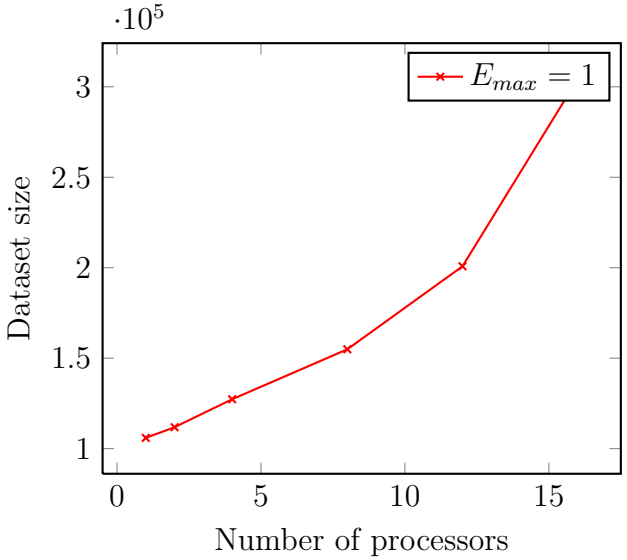


Figure 3.4: Isoefficiency function

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 measured 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 (Sonnenburg, Franc, Yom-Tov, & Sebag, 2009) 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 three

quite innovative plots measuring "*training time* vs. *area over the precision recall curve*", "*data set size* vs. *area over the precision recall curve*", and "*data set size* vs. *training time*". Additionally, it may be useful to borrow some ideas from (Peteiro-Barral, Bolon-Canedo, Alonso-Betanzos, Guijarro-Berdinas, & Sanchez-Marono, 2012) in which the authors are concerned with the scalability and efficiency of existing feature selection methods. All these concepts will be explained in detail in the next section.

3.3 PASCAL LARGE SCALE LEARNING CHALLENGE AND ITS PERFORMANCE MEASURES

The ideal goal would be to determine the best algorithm in terms of learning accuracy, depending on the time budget allowed. Accordingly, the score of an algorithm is computed as the average rank of its contribution with regard to six scalar measures which will be calculated based in the following figures.

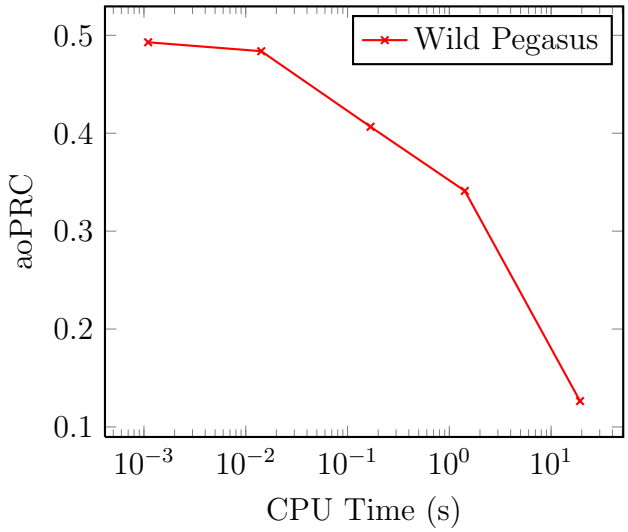


Figure 3.5: Training time vs. area over the precision recall curve (aoPRC)

Figure 3.5 measures *training time* vs. *area over the precision recall curve* (aoPRC). It is obtained by displaying the different time budgets and their corresponding aoPRC on the biggest dataset. We compute the following scores based on that figure:

— Minimum aoPRC

— Area under time vs. aoPRC curve

— The time t for which the aoPRC x falls below a threshold

$$\frac{x - \text{overall minimum aoPRC}}{x} < 0.05$$

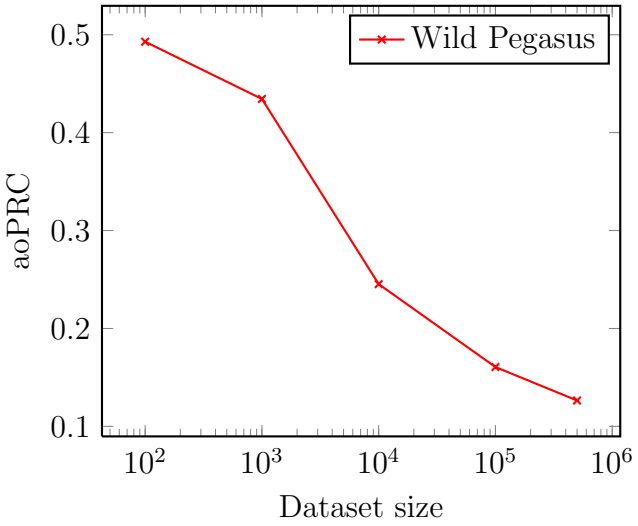


Figure 3.6: Dataset size vs. area over the precision recall curve (aoPRC)

Figure 3.6 measures *dataset size* vs. *area over the precision recall curve* (aoPRC). It is obtained by displaying the different dataset sizes and their corresponding aoPRC that the methods achieve. We compute the following scores based on that figure:

— Area under size vs. aoPRC curve

— The size s for which the aoPRC x falls below a threshold

$$\frac{x - \text{overall minimum aoPRC}}{x} < 0.05$$

Figure 3.7 measures *dataset size* vs. *training time*. It is obtained by displaying the different dataset sizes and the corresponding training time that the methods achieve. We compute the following scores based on that figure:

— Slope of the curve b using a least squares fit to $a \times x^b$.

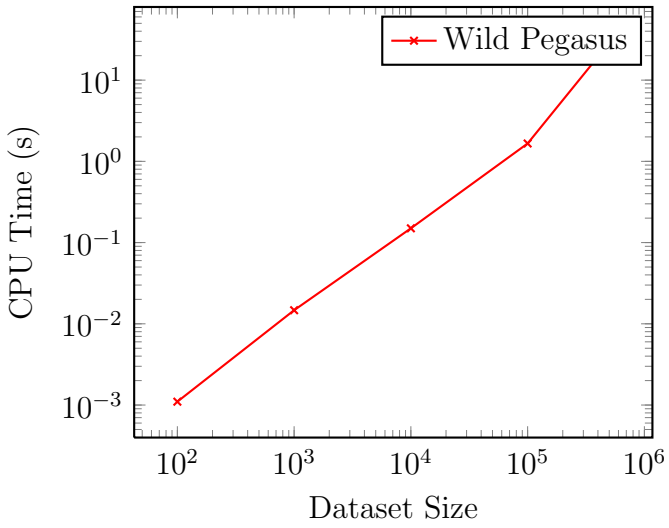


Figure 3.7: Dataset size vs. training time

A CASE STUDY ON THE SCALABILITY OF TRAINING ALGORITHMS FOR NEURAL NETWORKS

In this section, a case study on the scalability of five popular training algorithms for neural networks using the measures defined during the PASCAL challenge is presented (Peteiro-Barral, Guijarro-Berdiñas, Pérez-Sánchez, & Fontenla-Romero, 2013). The aim of the experiments is to get a good grasp of the methods proposed by PASCAL for assessing the performance of algorithms in terms of scalability and not simply in terms of error as is the case in the majority of papers in the literature. The six scalar measures defined on the three performance figures show some of the most important issues to the end-user with regard to the scalability of learning algorithms: how long does a learning algorithm take to reach a given performance? what is the amount of data needed for reaching a given performance? and, how does the computational effort increase with dataset size? For this case study, the measures defined in the PASCAL Large Scale Learning Challenge (see Section 3.3 for more details) have been used in order to assess the performance of five of the most popular training algorithms for artificial neural networks (ANNs). Three of these algorithms are gradient descent (GD) (Bishop, 2006), gradient descent with momentum and adaptive learning rate (GDX) (Bishop, 2006) and stochastic gradient descent (SGD) (Bottou, 1991), whose computational complexity is $O(n)$. The other algorithms are scaled conjugated gradient (SCG) (Møller, 1993) and Levenberg-Marquardt (LM) (Moré, 1978), whose complexities are $O(n^2)$ and $O(n^3)$, respectively.

Dataset	Inputs	Outputs	Training	Test	Task
Connect-4	42	3	60,000	7,557	Classification
KDD Cup 99	42	2	494,021	311,029	Classification
Covertype	54	2/1	100,000	50,620	Class. / Regr.
MNIST	748	2/1	60,000	10,000	Class. / Regr.
Friedman	10	1	1,000,000	100,000	Regression
Lorenz	8	1	1,000,000	100,000	Regression

Table 3.1: Brief description of each dataset.

These algorithms were applied to the most common tasks in machine learning: classification and regression. Table 3.1 shows the datasets[1] used in this experimentation along with a brief description of them (number of inputs, outputs, training samples and test samples; and task). Covertype and MNIST datasets, which are originally classification tasks, were also transformed into a regression task (Collobert & Bengio, 2001) by using only one output neuron to predict -1 for samples of class 1; and $+1$ for samples of class 2 . Friedman and Lorenz are artificial datasets. Friedman is defined by the equation $y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \sigma(0,1)$ where the input attributes $x_1, \ldots, x_{10}$ are generated independently from a uniform distribution on the interval $[0, 1]$. On the other hand, Lorenz is defined by the simultaneous solution of three equations $\frac{dX}{dt} = \delta Y - \delta X$, $\frac{dY}{dt} = -XZ + rX - Y$, $\frac{dZ}{dt} = XY - bZ$, where the systems exhibits chaotic behavior for $\delta = 10$, $r = 28$ and $b = \frac{8}{3}$. In order to choose the training algorithm showing best scalability the following procedure was applied on each dataset.

— for n = 1 to N –i.e. different simulations were carried out for accurately estimating the scalability of algorithms.

  – Divide the dataset using *holdout validation*. This kind of validation is suitable because the size of the datasets is very large.

  – Train a model setting its parameters to default values. Set the number of hidden units of the ANN to $2 \times number\_of\_inputs + 1$ (Hecht-Nielsen, 1990). It is important to remark that the aim here is not to investigate the optimal topology of an ANN for a given dataset, but to check the scalability of learning algorithms on large networks.

[1]Connect-4 and Covertype datasets are available on http://archive.ics.uci.edu/ml/-datasets.html; KDD Cup 99 on http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.-html; and MNIST on http://yann.lecun.com/exdb/mnist/

  - Run the algorithms to compute the six scalar measures defined in Section 3.3.
  - Rank the algorithms for each of these measures and calculate the *Score* of each algorithm as its average position with regard to the six rankings. For example, an algorithm that ranks first in three measures and second in the remaining three will obtain a final score of $\frac{1+1+1+2+2+2}{6} = 1.5$.

— Compute the mean overall score of the algorithms using the N results.

Regarding the desired output for training, for classification tasks, it comprises as many components as classes in the domain, where the value +1 is assigned to the desired class and -1 to the others. Thus, the hyperbolic tangent, which is bounded in $[-1, +1]$, was used as the transfer function of the output units of the ANNs. In the case of prediction tasks, the desired output comprises only one component, where its value is unbounded. Thus, the linear function was used as transfer function of the output units. In all cases, hyperbolic tangent function was used for the hidden units.

The results obtained are summarized in Tables 3.2 and 3.3. Notice that the lower the score, as defined in Section 3.3, the higher the scalability. Unless otherwise specified, learning algorithms were trained using all available samples for every dataset (see Table 3.1 for further details). However, some learning algorithms are not able to do this, mostly due to their spatial complexity. In this case, these measures are computed on the largest dataset the learning algorithms are able to process. If this occurs, it is specified along with the results.

As can be inferred from the results, these five popular training algorithms for ANNs present, in general, two opposite behaviors: a lower error at the expense of a longer training time (e.g. *SCG* or *LM*), or a shorter training time at the expense of a higher error (e.g. *GD* or *GDX*). On the other hand, *SGD* shows a good balance between training time and error, ranking the best overall with regard to the general measure of scalability (*Score*). However, notice that the current performance measures and the aggregation of the ranks are detrimental to learning algorithms which are accurate but slow. In this manner, the ranking could be scrambled by using a naive but fast training algorithm. For example, in the regression task of *MNIST* dataset, the algorithm *GD* ranks better than *SCG*, *GDX* and *SGD* due to its short training time, early stopping, in spite of obtaining a huge error. Further experiments showed convergence problems with regard to the training process of the algorithm *GD*.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD	2.67	0.38	$5.16e1$	$1.08e2$	0.97	$1.00e2$	0.43
GDX	2.17	0.31	$3.71e1$	$7.98e1$	0.92	$6.00e4$	0.40
SGD	2.67	0.16	$5.32e1$	$2.36e2$	0.54	$6.00e4$	0.54
SCG	2.67	0.21	$7.01e1$	$2.62e2$	0.77	$1.00e4$	0.50
LM†	3.5	0.23	$3.79e2$	$7.80e2$	0.77	$1.00e4$	0.77

(a) Connect-4.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD	2.67	0.38	$1.24e2$	$2.78e2$	1.20	$1.00e3$	0.49
GDX	2.50	0.42	$4.74e1$	$1.01e2$	1.32	$1.00e4$	0.41
SGD	2.50	0.13	$1.21e2$	$7.83e2$	0.62	$1.00e5$	0.58
SCG	2.83	0.20	$1.64e2$	$5.80e2$	0.81	$1.00e5$	0.55
LM†	3.83	0.24	$6.41e2$	$1.74e3$	0.94	$1.00e4$	0.84

(b) Covertype.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD‡	2.00	0.13	$4.29e1$	$5.53e1$	0.43	$1.00e2$	0.50
GDX‡	2.50	0.15	$2.55e1$	$5.93e1$	0.46	$1.00e3$	0.44
SGD	2.67	0.00	$8.85e0$	$1.35e3$	0.07	$4.94e5$	0.59
SCG‡	3.50	0.14	$1.10e2$	$3.54e2$	0.51	$1.00e4$	0.55
LM†	3.67	0.11	$2.21e2$	$1.24e3$	0.46	$1.00e4$	0.80

(c) KDD Cup 99.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD†	2.17	0.36	$1.41e2$	$2.26e2$	0.85	$1.00e2$	0.65
GDX†	2.50	0.22	$2.30e2$	$6.91e2$	0.66	$1.00e3$	0.72
SGD	2.50	0.02	$1.06e3$	$2.85e4$	0.34	$6.00e4$	0.99
SCG†	2.83	0.05	$2.85e2$	$1.62e3$	0.40	$1.00e4$	0.81
LM	–	–	–	–	–	–	–

(d) MNIST.

Table 3.2: Performance measures for classification tasks. Largest training set it can deal with: $^\dagger 1e4$ or $^\ddagger 1e5$ samples.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD	2.83	0.90	1.26e3	5.38e2	3.62	1.00e4	0.55
GDX	2.33	0.68	1.01e3	4.54e2	4.17	1.00e5	0.53
SGD	2.50	0.45	1.07e3	1.77e3	2.17	1.00e5	0.65
SCG	2.67	0.57	1.64e3	9.86e2	2.72	1.00e5	0.60
LM†	3.5	0.60	1.02e4	1.35e3	3.42	1.00e4	0.82

(a) Covertype.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD†	3.00	8.33	2.19e3	7.51e1	36.77	1.00e3	0.37
GDX†	2.50	4.41	1.83e3	7.20e1	24.57	1.00e5	0.37
SGD	3.17	0.21	2.24e4	1.12e4	6.88	1.00e5	0.68
SCG†	2.50	0.79	1.67e3	1.71e2	10.33	1.00e5	0.44
LM†	2.33	0.11	1.11e3	8.74e2	8.57	1.00e5	0.59

(b) Friedman.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD†	2.33	0.74	4.82e2	6.17e1	2.98	1.00e2	0.36
GDX†	2.33	2.66	2.45e2	2.04e1	13.63	1.00e4	0.26
SGD	4.00	0.01	6.54e3	9.96e3	0.69	1.00e6	0.67
SCG†	2.50	0.01	5.61e2	1.38e2	0.05	1.00e4	0.43
LM†	2.83	0.00	3.26e3	5.19e2	0.00	1.00e5	0.54

(c) Lorenz.

Name	Score	Err	AuTE	Te5%	AuSE	Se5%	Eff
GD†	2.00	303.12	1.66e3	6.60e0	903.14	1.00e2	0.44
GDX†	2.33	9.25	7.49e4	9.71e2	66.06	1.00e4	0.75
SGD	3.17	0.14	8.74e4	6.50e4	221.79	6.00e4	1.02
SCG†	2.17	3.06	3.10e4	1.82e3	41.52	1.00e4	0.82
LM	–	–	–	–	–	–	–

(d) MNIST

Table 3.3: Performance measures for regression tasks. Largest training set it can deal with: $^\dagger 1e4$ or $^\ddagger 1e5$ samples.

3.4 SOME FINAL CONSIDERATIONS

Note that, in the previous case study, a learning algorithm showing convergence problems is able to beat other algorithms in terms of scalability. This is an isolated case and the conclusions of this case study are not affected by it but, and this is the point, the performance measures must be revised for specific applications. By no means this fact makes PASCAL evaluation framework useless. Again, it would be an exercise of deciding which measure is more appropriate for the specific domain. And maybe for a specific scenario, the measures defined by the PASCAL challenge are a perfect match. This chapter was intended to provide some guidelines of assessing machine learning algorithms in terms of scalability and distribution. Of course, one can find many other measures in the literature. The key point for any application would be to find the most appropriate measure to assess the performance of the algorithms. In this thesis, we borrowed some of the ideas of the measures presented in this chapter in order to show the validity of the different algorithms that will be introduced in the following chapters, but we will keep the evaluation framework as simple and clear as possible.

Part II

Scaling Up Learning Algorithms

Chapter 4

Distributed Single-Layer Neural Networks

It is a fact that most traditional learning algorithms cannot look at very large datasets and plausibly find a good solution with reasonable requirements of computation. 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 sites wherein the data is processed. This chapter introduces a novel distributed training algorithm based on single-layer neural networks and genetic algorithms (Peteiro-Barral, Guijarro-Berdinas, & Pérez-Sánchez, 2012).

4.1 BACKGROUND: A FAST TRAINING ALGORITHM FOR SINGLE-LAYER NEURAL NETWORKS

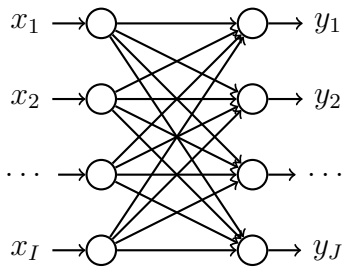


Figure 4.1: Single-layer neural network.

This section describes the method introduced in (Castillo, Fontenla-Romero, Guijarro-Berdiñas, & Alonso-Betanzos, 2002) for learning the weights of a

single-layer neural network. A single-layer neural network is the simplest kind of neural network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs. Figure 4.1 illustrates what is meant by a single-layer neural network. Inputs $x_1, \ldots, x_I$ are shown as circles, which are connected by the weights $w_{j1}, \ldots, w_{jI}$ to the output y_j where I is the number of inputs and J is the number of outputs. Notice that $y_j(\mathbf{x}; \mathbf{w})$ represents the output of unit j as a function of the input vector $\mathbf{x}$ and the weight vector $\mathbf{w}$. The independent variables have been omitted for purposes of clarity. Each line connecting an input i to an output j corresponds to a weight parameter w_{ji} The biases $w_{10}, \ldots, w_{J0}$ are represented as weights from an extra input x_0 which is permanently set to 1 (Bishop, 1995). We can express the network output in terms of the components of the input vector $\mathbf{x}$ and the weight vector $\mathbf{w}$ to give

$$y_j = f_j \left(\sum_{i=0}^{I} w_{ji} x_i \right) \tag{4.1}$$

where $j = 1, \ldots, J$, and f_j is the activation function of the jth output unit. The set of equations over all the samples in the training set can be written as

$$y_{jn} = f_j \left(\sum_{i=0}^{I} w_{ji} x_{in} \right) \tag{4.2}$$

where $n = 1, \ldots, N$. System 4.2 has $J \times N$ equations in $J \times (I+1)$ unknowns. In practice, the number of samples N is much larger than the number of inputs I which leads to an incompatible system of equations. Thus we shall consider some errors. The most common approach is to use the sum of squares error function which is given by a sum over all samples in the training set, and over all the outputs, of the form

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2} \sum_{n=1}^{N} \sum_{j=1}^{J} (d_{jn} - y_{jn})^2 \\ &= \frac{1}{2} \sum_{n=1}^{N} \sum_{j=1}^{J} \left(d_{jn} - f_j \left(\sum_{i=0}^{I} w_{ji} x_{in} \right) \right)^2 \end{aligned} \tag{4.3}$$

where d_{jn} represents the desired value for output unit j when the input vector is $\mathbf{x}_n$. This error function can be minimized by a variety of standard techniques. Among them, a method proposed in (Castillo et al., 2002) for learning the weights in single-layer neural networks is remarkable. It leads to the existence of a global optimum that is easily obtained solving a system

of linear equations. If we consider activation functions $f_1, \ldots, f_J$ which are invertible $f_1^{-1}, \ldots, f_J^{-1}$, the system of equations 4.3 that measures the error in the outputs can be rewritten as

$$Q(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{j=1}^{J} \left(f_j^{-1}(d_{jn}) - \sum_{i=0}^{I} w_{ji} x_{in} \right)^2 \tag{4.4}$$

which measures the error in the input of the activation functions. Notice that most commonly used activation functions are invertible. The solution for the weight values at the minimum of error function can therefore be found exactly by deriving Q with respect to the weights for each output $j = 1, \ldots, J$

$$\frac{\partial Q}{\partial w_{jp}} = \sum_{n=1}^{N} \left(f_j^{-1}(d_{jn}) - \sum_{i=0}^{I} w_{ji} x_{in} \right) x_{pn} = 0 \tag{4.5}$$

which leads to the system of linear equations

$$\sum_{i=0}^{I} \left(\sum_{n=1}^{N} x_{in} x_{pn} \right) w_{ji} = \sum_{n=1}^{N} \left(f_j^{-1}(d_{jn}) - w_{j0} \right) x_{pn} \tag{4.6}$$

which can be written as

$$\sum_{i=0}^{I} A_{pi} w_{ij} = b_{pj} \tag{4.7}$$

where

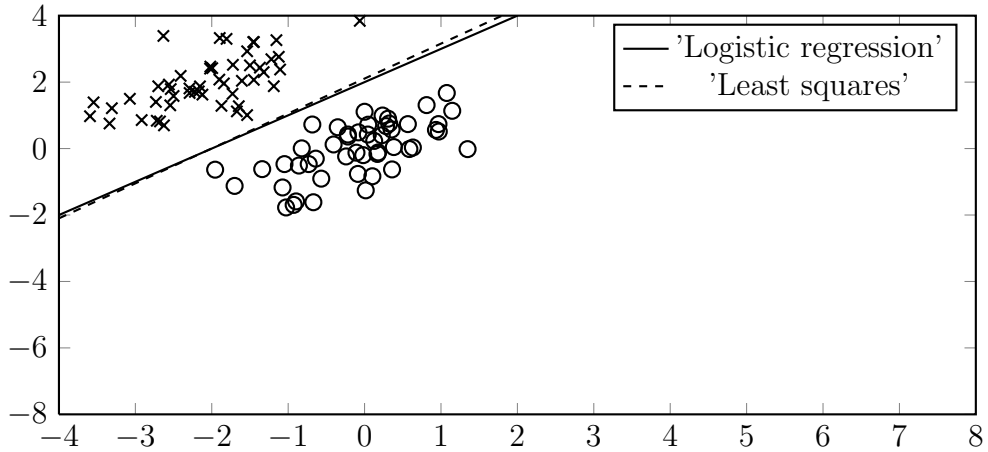
$$A_{pi} = \sum_{n=1}^{N} x_{in} x_{pn} \qquad\qquad b_{pj} = \sum_{n=1}^{N} f_j^{-1}(d_{jn}) x_{pn} \tag{4.8}$$

For every output unit j, Equation 4.7 has $I+1$ linear equations and $I+1$ unknowns. Hence, there exist only one solution that corresponds to the global optimum of the objective function 4.4. The global optimum can be easily obtained by well-known linear programming techniques. Computationally efficient methods for solving systems of linear equations with complexity $\mathcal{O}(J \times (I+1)^2)$ can be found in the literature (Carayannis, Kalouptsidis, & Manolakis, 1982; Bojańczyk, 1984). These techniques require much less computational resources and computational time than those involved in minimizing E in Equation 4.3.

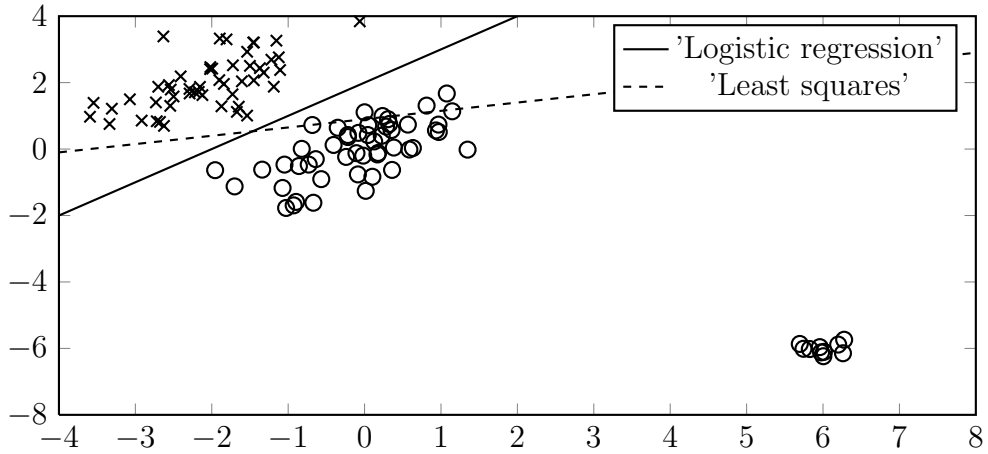
4.2 SOME CONSIDERATIONS ON THE MINIMIZATION OF THE SUM-OF-SQUARES ERROR FUNCTION

The minimization of the sum-of-squares error function (see Equation 4.3) led to a simple solution for the parameter values. Consider a general classification problem with J classes, with a 1-of-J binary coding scheme for the desired output $\mathbf{d}$. One justification for using least squares in a classification problem is that it approximates the conditional expectation $E(\mathbf{d} \mid \mathbf{x})$ of the desired output values $\mathbf{d}$ given the input vector $\mathbf{x}$, where this conditional expectation is given by the posterior class probabilities (Bishop, 2006). However, these probabilities are approximated rather poorly. If we use a 1-of-J coding scheme for J classes, then the predictions made by the model will have the property that the elements of the output $\mathbf{y}(\mathbf{x})$ will sum to 1 for any value of $\mathbf{x}$. However, this summation constraint alone is not sufficient to interpret the model outputs as probabilities. In the classical linear regression model the outputs are not bounded to lie within the interval $[0, 1]$. In the single-layer neural network presented above, the outputs are bounded by the transfer function that can be plugged in the outputs units. In particular, the logistic function is appropriate because its output values lie within the interval $[0, 1]$. But it is still not sufficient. Even as a discriminant model that one can use to make decisions with a probabilistic interpretation, it suffers from several problems. Least-squares solutions lack robustness to outliers and class imbalance problems. As illustrated in Figure 4.2, the additional data points in the bottom figure produce a significant change in the location of the decision boundary, even though these data points would be correctly classified by the original decision boundary in the top figure, showing that least squares is highly sensitive to outliers and class imbalance problems.

Least squares corresponds to maximum likelihood under the assumption of a Gaussian conditional distribution, whereas binary desired output vectors clearly have a distribution that is far from Gaussian. By adopting more appropriate models, one shall obtain classification models with much better properties than least squares. However, least squares seems an efficient method to train a single-layer neural network as demonstrated in (Castillo et al., 2002) especially for regression problems. Moreover, the matrices of coefficients $\mathbf{A}$ and $\mathbf{b}$ (see equations 4.8) are computed as a sum of terms along the data points in the training set. Therefore, the following equalities



(a) Data from two classes, denoted by crosses and circles, together with the decision boundary found by the logistic regression model (solid line) and also by least squares (dashed line).



(b) The corresponding results obtained when extra data points are added at the bottom left to the class denoted by circles.

Figure 4.2: Least squares is highly sensitive to outliers.

hold

$$A_{pi} = \sum_{n=1}^{N} x_{in}x_{pn} = \sum_{\mathcal{D}_1} x_{in}x_{pn} + \cdots + \sum_{\mathcal{D}_P} x_{in}x_{pn} \tag{4.9}$$

$$b_{pj} = \sum_{n=1}^{N} f_j^{-1}(d_{jn})x_{pn} = \sum_{\mathcal{D}_1} f_j^{-1}(d_{jn})x_{pn} + \cdots + \sum_{\mathcal{D}_P} f_j^{-1}(d_{jn})x_{pn} \tag{4.10}$$

for arbitrary non-overlapping partitions of the data $\mathcal{D}_p, p = 1, \ldots, P$. That is, the weight vector $\mathbf{w}$ can be computed incrementally regardless of the order of the data points because of the commutative property of the sum. This task is embarrassingly parallel and can be easily used in the development of a distributed training algorithm. Assuming that the training dataset is partitioned in P subsets $\mathcal{D}_1, \ldots, \mathcal{D}_P$, we can take different considerations regarding to different tasks,

REGRESSION TASK. The solutions provided by least squares in the P partitions are locally optimal, and the global optimum is obtained by simply summing the corresponding matrices of coefficients $\mathbf{A}_p$ and $\mathbf{b}_p$ where $p = 1, \ldots, P$.

CLASSIFICATION TASK. The solutions provided by least squares in the P partitions are locally suboptimal in terms of classification performance, and summing the corresponding matrices of coefficients $\mathbf{A}_p$ and $\mathbf{b}_p$ will lead to a suboptimal solution.

4.3 COMBINING MODELS USING GENETIC ALGORITHMS

As explained in the previous section, combining single-layer neural networks by means of a sum will lead to suboptimal solutions for classification problems. However, this combination may be improved by adding another step in process that optimizes a more appropriate cost function that approximate the actual loss we are trying to minimize, e.g. the standard loss functions for classification is zero-one-loss, misclassification rate, and the ones used for training classifiers are approximations of that loss. Thus, we propose to combine the single-layer neural networks trained in each partition using a genetic algorithm rather than a simple sum. The hypothesis here is that genetic algorithms will make good use of the suboptimal solutions as the starting point of the optimization method and then they will drive the model towards a better solution based on a more appropriate cost function.

Genetic algorithms (Holland, 1975; Goldberg & Holland, 1988) are stochastic search methods that operate on multiple solutions to a problem. Genetic algorithms work well on continuous and discrete combinatorial problems. They are less susceptible to getting stuck at local optima than gradient search methods at the expense of being computationally intensive. To use genetic algorithms, one must encode a solution to the problem as a *genome* or *chromosome*. Then, the genetic algorithm generates a population of feasible solutions and applies genetic operators such as mutation and crossover to evolve the individuals of the population in order to find the best solution to the problem. The objective function determines how good each individual is. The three most important parts of a genetic algorithm are

Definition of the **fitness function**, or objective function, $f : \mathcal{X} \to \mathbb{R}$ and the general problem is to find $\min_{\mathbf{x} \in \mathcal{X}} f$ where $\mathbf{x}$ is a vector of features and $\mathcal{X}$ is the search space.

Definition of the **genetic representation**. One can use any representation for the individual genomes. Pioneers worked primarily with strings of bits but one can use arrays, trees, lists, or any other data structure. The important issue is that each individual must represent a complete solution to the problem at hand.

Definition of the **genetic operators**; initialization, selection, crossover, and mutation. Initialization is the method that builds the initial population of individuals. The selection method uses the fitness of each individual to choose which individuals are allowed to breed more often. Typically crossover is defined as the manner in which two individuals (parents) are combined to produce two more individuals (children or offspring). The fundamental purpose of the crossover operator is to get genetic material from the previous generation to the subsequent generation. The mutation operator introduces a certain amount of randomness in the search. It can help to find solutions that crossover alone might not find (Verma, Llora, Goldberg, & Campbell, 2009).

A general formulation of a genetic algorithm is presented as follows where many different selection, crossover, and mutation operators can be accommodated, and a new population is generated in each generation.

1. Choose an initial population of individuals.

2. Select a subset of individuals for breeding by using the fitness function.

3. Combine the parents to generate the offspring by using the crossover operator.

4. Perform mutation on the offspring generated from crossover.

5. Repeat steps 2 and 3 until any termination condition is satisfied.

4.4 PROPOSED DISTRIBUTED LEARNING MODEL FOR SINGLE-LAYER NEURAL NETWORKS

During the local, *Map* phase (see Section 1.6 for more information on the MapReduce paradigm), single-layer neural networks are trained on the different partitions of the dataset $\mathbf{x}_p \in \mathcal{D}_p \subset \mathcal{D}$, by computing

$$A_{pi} = \sum_n x_{in} x_{pn}, i = 1, \ldots, I; \qquad b_{pj} = \sum_n f_j^{-1}(d_{jn}) x_{pn}, j = 1, \ldots, J$$

and outputting the pair $(p, < A_p, b_p >)$ where p is the index of the partition. Then, during the aggregation, *Reduce* phase, the different single-layer NNs $< A_p, b_p >$ are combined using a genetic algorithm, as described in the previous section. In particular;

FITNESS FUNCTION. As explained in the previous section, combining single-layer neural networks by means of a sum will lead to suboptimal solutions only for classification problems. Thus, for classification, the fitness function is defined as the logistic regression function

$$J(\mathbf{w}) = -\frac{1}{M}\left[\sum_m d_m \log h(y_m) + (1 - d_m)\log(1 - h(y_m))\right]$$

or the softmax regression function if the class label can take more than two possible values

$$J(\mathbf{w}) = -\frac{1}{M}\left[\sum_m \sum_j 1\{d_m = j\} \log \frac{e^{h(y_m^j)}}{\sum_j e^{h(y_m^j)}}\right]$$

where

$$h(y) = \frac{1}{1 + e^{-y}}$$

is the sigmoid function and $1\{\cdot\}$ is the indicator function, so that $1\{\text{a true statement}\} = 1$, and $1\{\text{a false statement}\} = 0$, and y_m^j is the $j th$ output for sample m.

GENETIC REPRESENTATION. Each individual p is represented by the pair of matrices $< A_p, b_p >$ (see Equation 4.8) that contains a solution to the partition p of the data.

GENETIC OPERATORS.

Initialization. The initial population comprises the set of neural networks $< A_p, b_p >, p = 1, \ldots, P$ trained on the P partitions of the data during the Map phase.

Selection. We implement tournament selection without replacement (Goldberg, Korb, & Deb, 1989). Tournament selection is a method of selecting an individual from a population of individuals where tournaments are run among T (tournament size) randomly chosen individuals and the winner (the one with the best fitness) is selected for breeding. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected. This process is repeated P (population size) times.

Crossover. For individuals p and q, the child is derived from the sum of the matrices $\mathbf{A}$ and $\mathbf{b}$ of the parents $< A_p + A_q, b_p + b_q >$ (see equations 4.9 and 4.10).

Mutation. The mutation operator adds a random number taken from a Gaussian distribution with mean 0 to random entries of the parent matrices $\mathbf{A}$ and $\mathbf{b}$. The standard deviation of the distribution is determined by the range of values of the inputs.

4.5 EXPERIMENTAL STUDY

The objective of this section is to experimentally evaluate the performance of the distributed training algorithm for single-layer neural networks proposed in this chapter.

4.5.1 MATERIALS AND METHODS

In this experimentation, we compare the performance in terms of error and training time of the original linear algorithm for single-layer neural networks trained in batch mode in a single machine, against the distributed version proposed in this chapter. In distributed scenarios, training data have been

scattered across 2, 4, and 8 different nodes. The evaluation of the methods has been done using holdout, 90% for training, 10% for testing. Experiments were run 100 times with random partitions of the datasets.

The distributed algorithm is evaluated on four regression datasets: Friedman, Lorenz, Covertype, and MNIST; and four classification datasets: Connect-4, KDD Cup 99, Covertype, and MNIST. This is the same group of datasets as the experimentation presented in Section 3.3 as a case study on the scalability of training algorithms for neural networks –a more detailed description of the datasets can be found there. Notice that we are using Covertype and MNIST as regression and classification problems, using the same procedure described in Section 3.3.

As stated before in Section 4.2, in regression tasks, the solutions provided by least squares in the different partitions are locally optimal, and the global optimum is obtained by simply summing the corresponding matrices of coefficients, i.e. there is no need of optimising the solution by means of the genetic algorithm just because the solution is already optimal. Thus, the weights of the neural network will be the same regardless of the setup –batch or distributed– and regardless of the number of nodes –because it is just a sum of terms–, so it is guaranteed that the performance of this model will be the same in the distributed implementation. The reason for including regression problems in this experimental study is to show the potential of the algorithm for improving training time when it is executed in a distributed environment.

4.5.2 RESULTS

In the four regression tasks considered in this study, the results in terms of mean squared error of the single-layer neural network on the four regression datasets are: 9.83 for Friedman, 0.76 for Covertype, 0.08 for Lorenz, and 1.89 for MNIST (these results are calculated as the average of 100 executions). As mentioned in the previous section, these results hold for the different setups: batch and distributed training, as the formulation of the algorithm as a sum of terms makes the solution independent of the number of nodes. The most interesting result for regression tasks is related to the training time for the different setups. Table 4.1 shows the training time of the original algorithm where the number of nodes is equal to 1, and the different configurations for the proposed distributed algorithm, when the number of nodes is set to 2, 4, and 8. The training times showed in this table are the sum of the training times for the 100 repetitions of the experiment. Also, Figure 4.3 plots the speed-up of the algorithm for the different number of processors and regression datasets.

Data set	Number of nodes			
	1	2	4	8
Friedman	33.67	17.66	12.11	12.32
Covertype	71.01	38.13	22.69	21.50
Lorenz	24.39	13.49	9.95	10.02
MNIST	5856.69	2953.36	1522.51	848.05

Table 4.1: Training time (s) of the proposed distributed algorithm for single-layer neural networks on regression tasks.

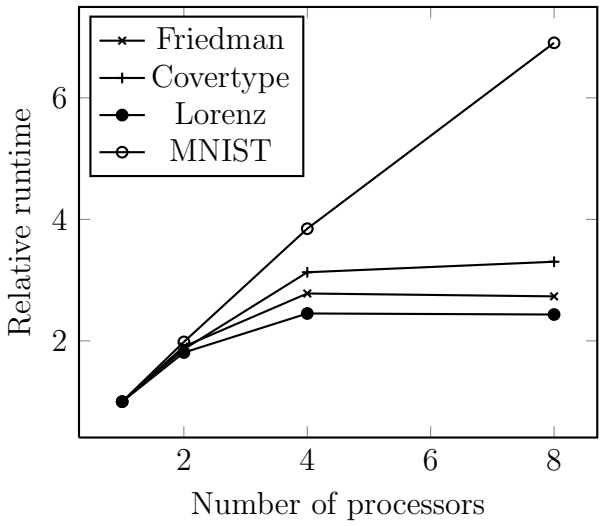


Figure 4.3: Speed-up of the proposed distributed algorithm for single-layer neural networks on regression tasks.

On the other hand, regarding classification tasks, as stated in the previous section, the solution provided by least squares in the different partitions are locally suboptimal in terms of classification performance, and summing the corresponding matrices of coefficients $\mathbf{A}$ and $\mathbf{b}$ will lead to a suboptimal solution. Thus, we would expect an improvement in performance by using a genetic algorithm for merging the different models trained in the distributed locations. Table 4.2 shows the mean test classification error of the proposed algorithm on the four classification tasks considered in this experimentation, and for the different setups of the training algorithm using different number of nodes. Finally, regarding the training time performance of the algorithm, Table 4.3 shows the training time of the proposed algorithm for classification tasks, and Figure 4.4 plots the speed-up of the distributed algorithm for the different number of processors.

Data set	Number of nodes			
	1	2	4	8
Connect-4	27.52	27.73	27.39	26.89
Covertype	26.94	26.76	25.27	24.45
KDD Cup 99	25.73	24.72	24.61	23.46
MNIST	13.34	13.45	12.54	12.05

Table 4.2: Test classification error (%) of the proposed distributed algorithm for single-layer neural networks on classification tasks.

Data set	Number of nodes			
	1	2	4	8
Connect-4	26.91	15.69	13.79	20.24
KDD Cup 99	237.74	112.69	61.363	43.01
Covertype	64.97	33.65	22.488	22.98
MNIST	6623.88	3187.24	1594.74	927.46

Table 4.3: Training time (s) of the proposed distributed algorithm for single-layer neural networks on classification tasks.

4.5.3 DISCUSSION AND CONCLUSIONS

Regarding regression tasks, as can be seen in Table 4.1 and Figure 4.3, for MNIST the algorithm achieves almost linear speed up, i.e. doubling the number of processors doubles the speed –reduces the training time by half. For the remaining datasets, the speed-up saturates when the number of processors is set to 4. This result is reasonable because the size of the datasets is not big enough. If we compute the size of a dataset as the number of features times the number of samples, MNIST is more than 2 times larger than KDD Cup 99; more than 4, 5, and 8 times larger than Friedman, Lorenz, and Covertype, respectively; and more than 17 times larger than Connect-4. Thus, for these datasets apart from MNIST, increasing the number of processors to a large number will not improve performance just because the time the learning algorithm takes to learn in the local partitions is not very different from a larger dataset. Notice, however, than these results are just simulations. In real-world environments, the size of the datasets could be many times larger than the datasets presented in this experimentation, so we could expect almost linear speed-up in for a very large dataset. In any case, these results show that the algorithm is able to train faster when we increase the number of processors whilst maintaining error performance.

On the other hand, regarding classification tasks, as can be seen in Table

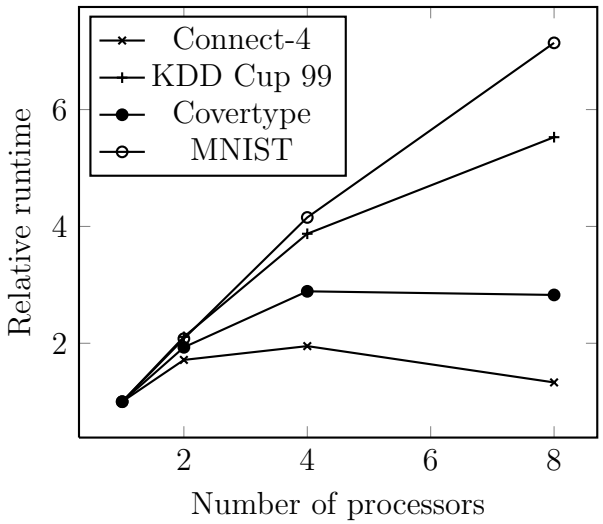


Figure 4.4: Speed-up of the proposed distributed algorithm for single-layer neural networks on classification tasks.

4.2, refining the solution achieved by least squares by optimising a classification error function improves results when increasing the number of processors. When the number of processors is higher, the diversity of the models during the execution of the genetic algorithm is potentially higher, leading to a higher chance of finding a better solution of the problem at hand. In average, for the four classification datasets, training on 8 nodes improves, in relative terms, the batch algorithm (1 node) by 8.23%. However, this improvement in accuracy comes at the cost of making the training step slightly slower (see Table 4.3 and Figure 4.4). Note that, unlike the training process for regression tasks, in classification, the distributed algorithm does not achieve linear speedup because the genetic algorithm creates an overhead during the integration stage. With a large population size, the genetic algorithm searches the solution space more thoroughly, thereby reducing the chance that the algorithm returns a local minimum that is not a global minimum. However, a large population size also causes the algorithm to run more slowly. However, notice that this overhead is quite small and the speed of the training algorithm is still remarkable. Note also that in classification tasks, the output layer of the single-layer neural networks has the size of the number of clases, i.e. the number of computations is higher than in regression tasks for a dataset with the same number of features and samples. In terms of the performance of the algorithm in specific datasets, as it happened with regression tasks, the distributed algorithm is almost able to achieve linear speed-up on MNIST. Also, the performance of the algorithm on KDD Cup

99 increases substantially when we increase the number of processors. On the contrary, in the case of Covertype and Connect-4, we can observe a very interesting effect; increasing the number of processors leads to a longer training time. In these two cases, the datasets are not very large so increasing the number of processors to a large number will not improve performance just because the time the learning algorithm takes to learn in the local partitions is not very different from a larger dataset, and on top of this, the genetic algorithm will take longer because the number of models to combine is larger. In any case, with large enough datasets as the ones we can find in real-world scenarios, these results show that the distributed algorithm is able to train faster on classification tasks when we increase the number of processors, whilst maintaining error performance.

In light of the above, we can conclude that the distributed training algorithm for single-layer neural networks shows a good performance in terms of both training speed and error. In particular, regarding classification, the genetic algorithm is able to improve the results obtained by just summing the matrices of coefficients, only adding a relatively small overhead to the process. Thus, the good scalability of the algorithm makes it very appropriate for large scale learning problems and big data.

Chapter 5

Distributed Two-Layer Neural Networks

Single-layer neural networks are only capable of approximating linear problems, which may be enough in some scenarios, but not in some others. However, the universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated to any desired degree of accuracy by a multi-layer perceptron with just one hidden layer (Hornik et al., 1989). In this chapter, this thesis expands the concepts seen in the previous chapter to neural networks with one hidden layer (Peteiro-Barral, Guijarro-Berdinas, Pérez-Sánchez, & Fontenla-Romero, 2011).

5.1 BACKGROUND: A SENSITIVITY-BASED LINEAR LEARNING METHOD FOR TWO-LAYER NEURAL NETWORKS

Consider the single-layer neural network introduced in Section 4.1. As stated in the previous chapter, the most common approach to compute the optimal set of weights is to use the sum of squares error function which is given by a sum over all samples in the training set, and over all the outputs. This error function can be minimized by a variety of standard techniques. In particular, the method proposed in (Castillo et al., 2002) leads to the existence of a global optimum that is easily obtained solving a system of linear equations. If we consider activation functions of the units of the neural network are are invertible. Thus, under this framework, rather than minimizing the sum of squares error in the output of the of the activation functions, one can minimize the sum of squares error in the input of these activation functions,

that leads to the equation we presented before, that is;

$$Q(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{j=1}^{J}\left(f_j^{-1}(d_{jn}) - \sum_{i=0}^{I} w_{ji}x_{in}\right)^2 \tag{5.1}$$

The solution for the weight values at the minimum of error function Q can be found exactly by deriving Q with respect to the weights $\mathbf{w}$ which leads to a system of linear equations which global optimum can be easily obtained by well-known linear programming techniques. All these concepts were introduced in the previous sections (see Section 4.1 for more details). Now, we introduce the concept of sensitivity, which will be useful to expand the formulation seen so far to two-layer neural networks as proposed in (Castillo, Conejo, Pedregal, Garcia, & Alguacil, 2011). Sensitivity analysis is the study of how the uncertainty in the output of a model can be apportioned to different sources of uncertainty in its inputs. Sensitivity analysis is useful here because it increases the understanding of the relationships between input and output variables in a model. In other words, sensitivity analysis answer the question of how sensitive are the optimal values in a minimization problem to the data points, i.e. what is the marginal increment in the optimum resulting from changes in the data. The main implication is that sensitivities measure the rate of change of the objective function in a small neighborhood around the optimal solution (Castillo, Hadi, Conejo, & Fernández-Canteli, 2004). Thus, in this context, the sensitivities are computed as the derivative of the cost function Q with respect to the input samples $\mathbf{x}$, that is;

$$\frac{\partial Q}{\partial x_{pq}} = -\sum_{j=1}^{J}\left(f_j^{-1}(d_{jq}) - \sum_{i=0}^{I} w_{ji}x_{iq}\right)w_{jr}; \forall p, q \tag{5.2}$$

and with respect to the output data $\mathbf{d}$ can be computed as

$$\frac{\partial Q}{\partial d_{pq}} = \frac{f_p^{-1}(d_{pq}) - \sum_{i=0}^{I} w_{pi}x_{iq}}{f_p'(d_{pq})}; \forall p, q \tag{5.3}$$

Based on the previous equations, one can develop a training algorithm for two-layer neural networks based on sensitivity analysis. Fist, consider the two-layer neural network shown in Figure 5.1. Inputs $x_1, \ldots, x_I$ are connected by the weights $w_{k1}, \ldots, w_{kI}$ to the output of the hidden layer $z_k(\mathbf{x})$ where I is the number of inputs and K is the number of hidden units. Hidden units $z_1, \ldots, z_K$ are connected by the weights $w_{j1}, \ldots, w_{jK}$ to the output $y_j(\mathbf{z})$ where J is the number of outputs. A two-layer neural network can be considered to be composed of two single-layer neural networks. Thus, using

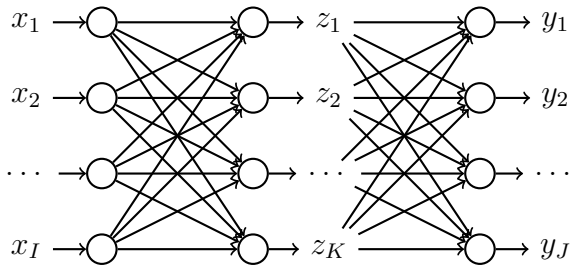


Figure 5.1: Two-layer neural network.

Equation 5.1 and assuming $\mathbf{z}$ is known, the cost function can be defined as follows;

$$\begin{aligned} Q(\mathbf{z}) &= Q^{(1)}(\mathbf{z}) + Q^{(2)}(\mathbf{z}) \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K}\left(g_k^{-1}(z_{kn}) - \sum_{i=0}^{I} w_{ki}^{(1)} x_{in}\right)^2 \\ &\quad + \sum_{n=1}^{N}\sum_{j=1}^{J}\left(f_j^{-1}(d_{jn}) - \sum_{k=0}^{K} w_{jk}^{(2)} z_{kn}\right)^2 \end{aligned} \tag{5.4}$$

where superscript (1) refers to the first layer and superscript (2) refers to the second layer, and the activation functions in the hidden units are denoted by g_k and the activation functions in the output units are denoted by f_j. The weights $w_{ki}^{(1)}$ and $w_{jk}^{(2)}$ can be learn independently for each layer using z_{ks} by solving a system of linear equations. Thus, the sensitivities (see equations 5.2 and 5.3) with respect to w_{ks} are computed as follows

$$\begin{aligned} \frac{\partial Q}{\partial z_{kn}} &= \frac{\partial Q^{(1)}}{\partial z_{kn}} + \frac{\partial Q^{(2)}}{\partial z_{kn}} \\ &= \frac{g_k^{-1}(z_{kn}) - \sum_{i=0}^{I} w_{ki}^{(1)} x_{in}}{g_k'(z_{kn})} \\ &\quad - \sum_{j=1}^{J}\left(f_j^{-1}(d_{jn}) - \sum_{k=0}^{K} w_{jr}^{(2)} z_{rn}\right) w_{jk}^{(2)} \end{aligned} \tag{5.5}$$

Then, the values of $\mathbf{z}$ are updated using Taylor series

$$Q(\mathbf{z} + \Delta\mathbf{z}) = Q(\mathbf{z}) + \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{\partial Q(\mathbf{z})}{\partial z_{kn}} \approx 0 \tag{5.6}$$

which leads to the following update rule

$$\Delta \mathbf{z} = -\rho \frac{Q(\mathbf{z})}{\|\nabla Q\|^2} \nabla Q \tag{5.7}$$

where ρ is a relaxation factor or step size. The concepts introduced so far take shape in the subsequent algorithm, called SBLLM and introduced in (Castillo, Guijarro-Berdiñas, Fontenla-Romero, & Alonso-Betanzos, 2006). The input to the algorithm consists of the input data points $\mathbf{x}_n$ and desired outputs $\mathbf{d}_n$.

INITIALIZATION. Assign $\mathbf{z}$ to be the output computed on the input $\mathbf{x}$ and some random weights $\mathbf{w}^{(1)}$ plus a small random error

$$z_{kn} = g_k \left(\sum_{i=0}^{I} w_{ki}^{(1)} x_{in} \right) + \epsilon_{kn} \tag{5.8}$$

where $\epsilon_{kn} \in U(-\eta, \eta)$ and η is set to a small value.

SOLUTION. Compute the weights and sensitivities by solving the systems of linear equations

$$\sum_{i=0}^{I} A_{ri}^{(1)} w_{ki}^{(1)} = b_{rk}^{(1)} \tag{5.9}$$

$$\sum_{k=0}^{K} A_{qk}^{(2)} w_{jk}^{(2)} = b_{qj}^{(2)} \tag{5.10}$$

where

$$A_{ri}^{(1)} = \sum_{n=1}^{N} x_{in} x_{rn} \qquad b_{rk}^{(1)} = \sum_{n=1}^{N} g_k^{-1}(z_{kn}) x_{rn} \tag{5.11}$$

$$A_{qk}^{(2)} = \sum_{n=1}^{N} z_{ks} z_{qn} \qquad b_{qj}^{(2)} = \sum_{n=1}^{N} f_j^{-1}(d_{jn}) z_{qn} \tag{5.12}$$

where $r = 1, \ldots, I$ and $q = 1, \ldots, K$. These systems of linear equations are the same introduced in the previous section (see Equation 4.7), in this case, for each of the two layers.

EVALUATION. Evaluate Q by

$$\begin{aligned} Q(\mathbf{z}) &= Q^{(1)}(\mathbf{z}) + Q^{(2)}(\mathbf{z}) \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K}\left(g_k^{-1}(z_{kn}) - \sum_{i=0}^{I} w_{ki}^{(1)} x_{in}\right)^2 \\ &\quad + \sum_{n=1}^{N}\sum_{j=1}^{J}\left(f_j^{-1}(d_{jn}) - \sum_{k=0}^{K} w_{jk}^{(2)} z_{kn}\right)^2 \end{aligned} \tag{5.13}$$

and mean squared error E by

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{j=1}^{J}\left(d_{jn} - f_j\left(\sum_{i=0}^{I} w_{ji} x_{in}\right)\right)^2 \tag{5.14}$$

Stop the training process if $|Q_{iter} - Q_{iter-1}| < \epsilon$ or $|E_{iter} - E_{iter-1}| < \epsilon$ where ϵ is a predefined parameter.

UPDATE. Compute the sensitivities

$$\begin{aligned} \frac{\partial Q}{\partial z_{kn}} &= \frac{\partial Q^{(1)}}{\partial z_{kn}} + \frac{\partial Q^{(2)}}{\partial z_{kn}} \\ &= \frac{\sum_{i=0}^{I} w_{ki}^{(1)} x_{in} - g_k^{-1}(z_{kn})}{g_k'(z_{kn})} \\ &\quad - \sum_{j=1}^{J}\left(\sum_{r=0}^{K} w_{jr}^{(2)} z_{rn} - f_j^{-1}(y_{jn})\right) w_{jk}^{(2)} \end{aligned} \tag{5.15}$$

and update $\mathbf{z}$ using Taylor series

$$\mathbf{z} = \mathbf{z} - \rho\frac{Q(\mathbf{z})}{\|\nabla Q\|^2}\nabla Q \tag{5.16}$$

where ρ is a predefined parameter. Then, go back to the *solution* step for another iteration of the method.

5.2 PROPOSED DISTRIBUTED LEARNING MODEL FOR TWO-LAYER NEURAL NETWORKS

The matrices of coefficients in the first layer $\mathbf{A}^{(1)}$ and $\mathbf{b}^{(1)}$ (see Equation 5.11) and the matrices of coefficients in the second layer $\mathbf{A}^{(2)}$ and $\mathbf{b}^{(2)}$ (see

Equation 5.12) are computed as a sum of terms along the data points in the training set. Therefore, the following equalities hold

$$A_{ri}^{(1)} = \sum_{n=1}^{N} x_{in}x_{rn} = \sum_{\mathcal{D}_1} x_{in}x_{rn} + \cdots + \sum_{\mathcal{D}_p} x_{in}x_{rn} \tag{5.17}$$

$$b_{pk}^{(1)} = \sum_{n=1}^{N} g_k^{-1}(z_{ks})x_{ps} = \sum_{\mathcal{D}_1} g_k^{-1}(z_{kn})x_{rn} + \cdots + \sum_{\mathcal{D}_p} g_k^{-1}(z_{kn})x_{rn} \tag{5.18}$$

and

$$A_{qk}^{(2)} = \sum_{n=1}^{N} z_{kn}z_{qn} = \sum_{\mathcal{D}_1} z_{kn}z_{qn} + \cdots + \sum_{\mathcal{D}_p} z_{kn}z_{qn} \tag{5.19}$$

$$b_{qj}^{(2)} = \sum_{n=1}^{N} f_j^{-1}(d_{js})z_{qs} = \sum_{\mathcal{D}_1} f_j^{-1}(d_{jn})z_{qn} + \cdots + \sum_{\mathcal{D}_p} f_j^{-1}(d_{jn})z_{qn} \tag{5.20}$$

for arbitrary partitions of the data $\mathcal{D}_p, p = 1, \ldots, P$. That is, the weight vectors $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ can be computed incrementally regardless of the order of the data points because of the commutative property of the sum. Using this property, it is possible to take a similar approach to the previous algorithm for single-layer ANNs, and implement the combination of different two-layer ANNs by means of a genetic algorithm. The implementation of this genetic algorithm is very similar to the one proposed in the previous section, just taking into account the different equations introduced in this section. Thus, most of the considerations stated in the previous chapter are also valid here.

However, there is an important difference for regression problems in connection with the training algorithm for single-layer neural networks. Note that least squares is a good approximation of the cost function for regression problems, but in this case the solutions provided by least squares in the different partitions are not guaranteed to be locally optimal, just because adding a new layer to the neural network makes the problem not convex, i.e. the optimization process could get stuck in a local minima. Thus, adding a genetic algorithm as an integration step could lead to better solutions in this context, since the combination of different models increments the chances of finding the global minima. On the other hand, for classification problems, the solutions provided by least squares is not optimal in terms of classification performance, so we will define a more appropriate cost function that approximate the actual loss we are trying to minimize, e.g. logistic regression function, zero-one-loss, misclassification rate, and the ones used for training classifiers are approximations of that loss.

Regarding the genetic representation, for both regression and classification tasks, each individual will be represented with the four matrices $\mathbf{A}_p^{(1)}$, $\mathbf{b}_p^{(1)}$, $\mathbf{A}_p^{(2)}$, and $\mathbf{b}_p^{(2)}$ that contains a solution to the partition p of the data. Finally, the genetic operators will be the same used in the previous chapter, just taking into account the representation of the two-layer neural networks using the four matrices of coefficients.

5.3 EXPERIMENTAL STUDY

The objective of this section is to experimentally evaluate the performance of the distributed training algorithm for two-layer neural networks proposed in this chapter.

5.3.1 MATERIALS AND METHODS

The experimental setup will be the same as the one described in Section 4.5 for evaluating the performance of the distributed training algorithm for single-layer neural networks. Thus, the algorithms are evaluated on four regression datasets: Friedman, Lorenz, Covertype, and MNIST; and four classification datasets: Connect-4, KDD Cup 99, Covertype, and MNIST. In this experimentation, we compare the performance in terms of error –mean squared error for regression tasks and classification error for classification tasks– and training time of the original algorithm trained in batch in a single machine, against the distributed version proposed in this chapter. In distributed scenarios, training data have been scattered across 2, 4, and 8 different nodes. Again, the evaluation of the methods has been done using holdout, 90% for training, 10% for testing. Experiments were run 100 times with random partitions of the datasets. Regarding the number of hidden units in the hidden layer, we set the number of the neural network to two times the number of inputs. Note the aim here is not to investigate the optimal topology of the neural network for a given dataset, but to evaluate the performance of the training algorithms on large networks. Finally, we will use the sigmoid function as the activation function in the hidden layer, and linear and sigmoid functions in the output layer for regression and classification, respectively.

5.3.2 RESULTS

In the first place, we will show the results for regression tasks. Table 5.1 shows the mean squared error of the original algorithm where the number of nodes is equal to 1, and the different configurations for the proposed

Data set	Number of nodes			
	1	2	4	8
Covertype	0.70	0.70	0.68	0.67
Friedman	7.92	7.94	7.05	6.43
Lorenz	0.03	0.02	0.00	0.00
MNIST	1.91	1.84	1.87	1.78

Table 5.1: Test mean squared error of the proposed distributed algorithm for two-layer neural networks on regression tasks.

Data set	Number of nodes			
	1	2	4	8
Friedman	889.62	389.71	209.27	129.67
Covertype	1762.94	914.16	626.70	670.05
Lorenz	526.68	267.51	145.25	92.92
MNIST	29587.73	14668.46	7550.82	4277.828

Table 5.2: Training time (s) of the proposed distributed algorithm for two-layer neural networks on regression tasks.

distributed algorithm for two-layer neural networks, using 2, 4, and 8 nodes (the number of samples in each node will be approximately the same); on the four different regression tasks considered in this experimentation. Regarding the training time performance, Table 5.2 shows the training time, and Figure 5.2 plots the speed-up of the distributed algorithm for the different number of nodes. As in the experimental study for single-layer neural networks, the training times showed in this table are the sum of the training times for the 100 repetitions of the experiment.

On the other hand, regarding classification tasks, Table 5.3 shows the test classification error of the proposed algorithm on the four classification datasets. With respect to the training time of the algorithm in classification tasks, Table 5.4 shows the training time performance of the proposed distributed algorithm for two-layer neural networks in classification tasks. Finally, Table 5.3 plots the speed-up of the distributed algorithm for two-layer neural networks for the different number of nodes.

5.3.3 DISCUSSION AND CONCLUSIONS

Regarding regression tasks, as can be seen in Table 5.1, the trend is towards a smaller mean squared error when increasing the number of nodes. If we compare the performance of the original algorithm (number of nodes equal

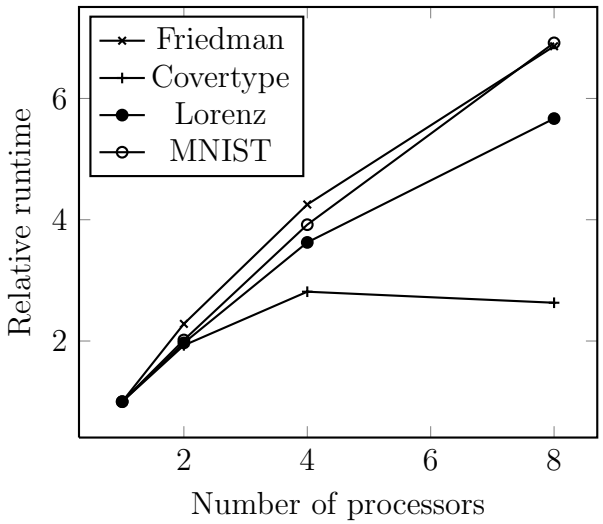


Figure 5.2: Speed-up of the proposed distributed algorithm for two-layer neural networks on regression tasks.

Data set	Number of nodes			
	1	2	4	8
Connect-4	25.93	25.67	25.08	23.63
Covertype	24.11	24.17	22.84	21.73
KDD Cup 99	10.12	10.03	8.93	8.82
MNIST	14.91	15.13	13.87	11.67

Table 5.3: Test classification error (%) of the proposed distributed algorithm for two-layer neural networks on classification tasks.

to 1) against the performance of the distributed algorithm when the number of nodes is 8, the relative improvement in performance is around 11.75%. This shows the effectiveness of the proposed approach when merging the solutions calculated at the different locations using a genetic algorithm. However, decreasing mean squared error comes the cost of a overhead because of running a genetic algorithm in the integration step, making the algorithm perform slightly worse than linear speedup (see Table 5.2 and Figure 5.2). However, the overhead of the genetic algorithm is quite small, and these results show that the algorithm is able to train faster when we increase the number of processors whilst maintaining, or in many cases improving, error peformance. In particular, for the largest datasets: Friedman, Lorenz, and MNIST; the algorithm achieve sublinear speed-up, but not that far from being linear. In particular, the speedup for Friedman and MNIST is almost

Data set	Number of nodes			
	1	2	4	8
Connect-4	714.23	402.41	320.42	414.33
KDD Cup 99	5793.57	2911.17	1634.84	992.50
Covertype	1678.39	911.91	639.08	680.06
MNIST	34256.23	16696.99	8853.41	4800.44

Table 5.4: Training time (s) of the proposed distributed algorithm for two-layer neural networks on classification tasks.

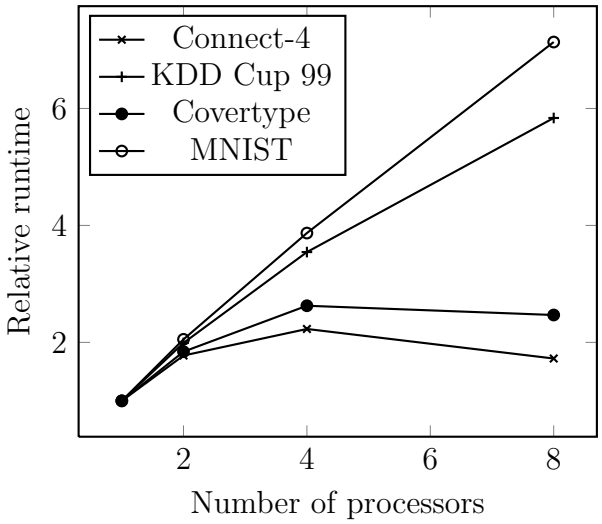


Figure 5.3: Speed-up of the proposed distributed algorithm for two-layer neural networks on classification tasks.

7 when the number of nodes or processors is 8. For the smallest dataset of this group, Covertype saturates in speed-up when the number of processors is set to 4, even running slower when we increase the number of processors to 8. As explained in the experimental section of the distributed learning algorithm for single-layer neural networks (see Section 4.5 for more details), Covertype is not a very large dataset so increasing the number of processors to a large number will not improve performance just because the time the learning algorithm takes to learn in the local partitions is not very different from a larger dataset. Moreover, the overhead of the genetic algorithm is higher because the number of models to combine is larger, so the time the genetic algorithm will take to find a solution is longer.

On the other hand, regarding classification tasks, if we compare the performance of the original algorithm (number of nodes equal to 1) against the

performance of the distributed algorithm when the number of nodes is 8, the relative improvement in performance is around 18.81%, which is remarkable. This result shows the effectiveness in terms of classification error of the proposed approach when merging the solutions calculated at the different locations using a genetic algorithm. With respect to the performance of the algorithm in terms of training time, the results are very similar as the ones reported for regression tasks (see Table 5.4 and Figure 5.3). As can be seen, using a genetic algorithm in the integration step makes the algorithm perform slightly worse than linear speedup, but this overhead is again quite small. The behaviour of the algorithm follows the same pattern it showed in regression tasks. When the datasets are large, the distributed algorithm is almost able to reach linear speedup. For smaller datasets, the speedup of the algorithm saturates when dividing the dataset in more partitions is not worth it, especially because of the small overhead of the genetic algorithm during the integration stage.

Finally, it is also relevant to compare the performance of the distributed training algorithm for single and two-layer neural networks. The results of the single-layer neural network can be found in Section 4.5. For regression tasks, using a two-layer neural network reduces, in average, the mean squared error by 13.42% for Covertype, Friedman, and MNIST. For Lorenz, the two-layer neural network is able to reach an error of zero, whilst the result using a single-layer neural network was 0.08. Regarding classification, in average for the four datasets, the classification error decreases by 22.18% when using a two-layer neural network rather than a single-layer neural network. However, the improvement in terms of error comes with the cost of the training time. In average, for both regression and classification, training a two-layer neural network takes around 15 times longer than training a single-layer neural network. Note than the training algorithm for two-layer neural networks need to iterate until convergence and also solve a system of linear equations in each layer -being the hidden layer larger than the input layer in this experimentation. As can be seen, scalability is a balance between error and speed. If reaching the lowest possible error is not that important but learning as fast as possible, most likely a single-layer neural network is a good choice. On the contrary, if the speed of the training process is not critical but the error of the model, most likely a two-layer neural network is a good choice.

Chapter 6

Distributed Frontier Vector Quantization Based on Information Theory

In this chapter, a novel distributed learning algorithm built upon the Frontier Vector Quantization based on Information Theory (FVQIT) method is introduced (Peteiro-Barral & Guijarro-Berdiñas, 2013). The FVQIT is very effective in classification problems but it shows poor training time performance. Thus, distributed learning is appropriate here to speed up training.

6.1 BACKGROUND ON FRONTIER VECTOR QUANTIZATION AND INFORMATION THEORY

The frontier vector quantization based on information theory (FVQIT) method (Porto-Díaz, Martínez-Rego, Alonso-Betanzos, & Fontenla-Romero, 2012) is a supervised learning algorithm based on information theoretic learning (ITL) (J. C. Principe, 2010) and vector quantization (Lehn-Schiøler, Hegde, Erdogmus, & Principe, 2005). ITL is a framework to adapt systems nonparametrically that has developed cost functions for machine learning based on entropy and divergence expressed in terms of Renyi's entropy and nonparametric probability density function estimators. It has proven that these cost functions build robust systems and obtain optimal parametrization (J. C. Principe, 2010). The development of the FVQIT is based on vector quantization using information theoretic learning (VQIT). The principle of vector quantization is to represent a dataset with a smaller number processing elements (PEs). The central idea of the VQIT is to minimize the free energy of an information potential function. It was shown that minimizing free energy is equivalent to minimizing the divergence between the

Parzen estimator of the distributions of data and the Parzen estimator of the distribution of PEs.

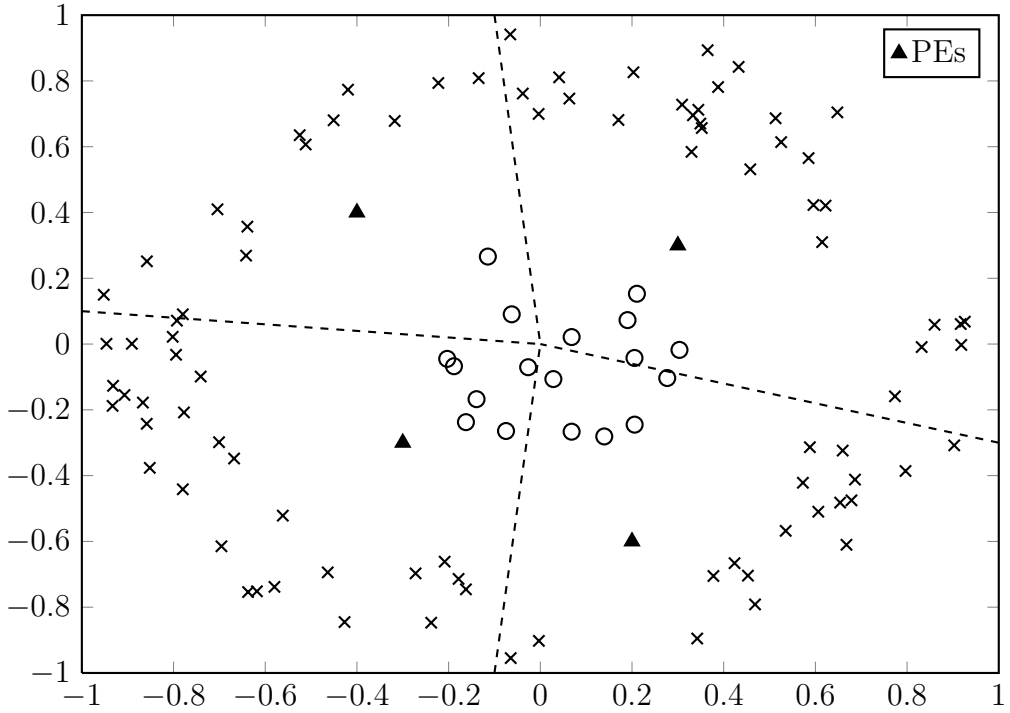


Figure 6.1: Example of a partition of the input space into four parts.

A physical interpretation considers two types of particles from which potential fields with opposite polarities are created. The data points are one set of particles that occupy a fixed position in the input space. The PEs are the other set of particles that are free to move. They will move according to the forces that act on them by other particles and eventually minimizing the free energy. The attracting forces from data to PEs will tend to place PEs in the neighborhood of data points whilst the repulsion forces between PEs will tend to spread the PEs in the input space. In the original formulation of the VQIT, data points and PEs have opposite polarities. In the FVQIT, data points from different classes have different polarities. The most frequent class in the neighborhood of the PE will attract it whilst the second most frequent class of the PE will repel it. Thus, multiple attraction and repulsion forces converge on each PE. The idea is to place the PEs in the frontier between data samples from two different classes (see Figure 6.1). In this situation, the Parzen density estimators of the distribution of data points $f(\mathbf{x})$ and PEs

$g(\mathbf{x})$ are defined as

$$f\left(\mathbf{x}\right) = \frac{1}{N}\sum_{n=1}^{N} G\left(\mathbf{x} - \mathbf{x}_n, \sigma_f^2\right) \tag{6.1}$$

$$g\left(\mathbf{x}\right) = \frac{1}{K}\sum_{k=1}^{K} G\left(\mathbf{x} - \mathbf{c}_k, \sigma_g^2\right) \tag{6.2}$$

where N is the number of data points, K is the number of PEs, G is the Gaussian kernel, σ_f^2 and σ_g^2 are the variances of the Gaussian functions, $\mathbf{x} \in \Re^d$ are the data points, and $\mathbf{c}_k \in \Re^d$ are the positions of the PEs. The function of energy $J(\mathbf{c})$ that calculates the divergence between the Parzen estimators of the data points from two classes is computed as

$$\begin{aligned} J(\mathbf{c}, \mathbf{x}) = \log \int f^2(\mathbf{x})\, dx + 2\log \int f^+(\mathbf{x}) g(\mathbf{x})\, dx \\ - 2\,\log \int f^-(\mathbf{x}) g(\mathbf{x})\, dx + \log \int g^2(\mathbf{x})\, dx \end{aligned} \tag{6.3}$$

where $f^+(\mathbf{x})$ is the estimator of the distribution of data points from the positive class and $f^-(\mathbf{x})$ is the estimator of the distribution of data points from the negative class. According to (J. Principe, Xu, Zhao, & Fisher, 2000) and (Porto-Díaz et al., 2012), the first term of Equation 6.3 is the information potential of the data points. This term will be zero for stationary data. The second term is the cross correlation between the distribution of data of the positive class and the PEs. The third term is the cross correlation between the distribution of data of the negative class and the PEs. Last, the fourth term is the information potential of the PEs. Assuming this formulation, the PEs will be situated on the frontier between the two classes when the energy function $J(\mathbf{c})$ is minimized. Gradient descent is used to solve the minimization problem. The solution for the position of the PEs $\mathbf{c}$ at the minimum of error function can therefore be found exactly by deriving J (see Equation 6.3) with respect to $\mathbf{c}$. For purposes of simplicity it seem best to develop J by means of its three relevant terms: positive class, negative class, and PEs (entropy). Remember that the first term of the equation will be zero, as the data is stationary. First, the term for the positive class is defined

as

$$
\begin{aligned}
C^{+} &= \int f^{+}(\mathbf{x}) g(\mathbf{x})\, dx \\
&= \frac{1}{KN^{+}} \int \sum_{n}^{N^{+}} G(\mathbf{x} - \mathbf{x}_n^{+}, \sigma_f^2) \sum_{k}^{K} G(\mathbf{x} - \mathbf{c}_k, \sigma_g^2)\, dx \\
&= \frac{1}{KN^{+}} \sum_{n}^{N^{+}} \sum_{k}^{K} \int G(\mathbf{x} - \mathbf{x}_n^{+}, \sigma_f^2) G(\mathbf{x} - \mathbf{c}_k, \sigma_g^2)\, dx \\
&= \frac{1}{KN^{+}} \sum_{k}^{K} \sum_{n}^{N^{+}} G(\mathbf{c}_k - \mathbf{x}_n^{+}, \sigma_a^2)
\end{aligned}
\tag{6.4}
$$

where K is the number of PEs, N^{+} is the number of data points from the positive class, $\mathbf{x}_n^{+}$ are the data points from the positive class, $\mathbf{c}_k$ are the positions of the PEs and $\sigma_a^2 = \sigma_f^2 + \sigma_g^2$ is the variance of the Gaussian function. Second, the term for the negative class is defined as

$$
\begin{aligned}
C^{-} &= \int f^{-}(\mathbf{x}) g(\mathbf{x})\, dx \\
&= \frac{1}{KN^{-}} \int \sum_{n}^{N^{-}} G(\mathbf{x} - \mathbf{x}_n^{-}, \sigma_f^2) \sum_{k}^{K} G(\mathbf{x} - \mathbf{c}_k, \sigma_g^2)\, dx \\
&= \frac{1}{KN^{-}} \sum_{n}^{N^{-}} \sum_{k}^{K} \int G(\mathbf{x} - \mathbf{x}_n^{-}, \sigma_f^2) G(\mathbf{x} - \mathbf{c}_k, \sigma_g^2)\, dx \\
&= \frac{1}{KN^{-}} \sum_{k}^{K} \sum_{n}^{N^{-}} G(\mathbf{c}_k - \mathbf{x}_n^{-}, \sigma_a^2)
\end{aligned}
\tag{6.5}
$$

where N^{-} is the number of data points from the negative class and $\mathbf{x}_n^{-}$ are the data points from the negative class. Third, the term for the PEs, entropy, is defined as

$$
\begin{aligned}
V &= \int g(\mathbf{x})^2\, dx \\
&= \frac{1}{K^2} \sum_{k}^{K} \sum_{q}^{K} G(\mathbf{c}_k - \mathbf{c}_q, \sqrt{2}\sigma_g)
\end{aligned}
\tag{6.6}
$$

Then, based on the three previous terms, the contribution to the gradient

can be found by deriving J with respect to the positions $\mathbf{c}$

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{c}_q} &= \frac{\partial}{\partial \mathbf{c}_q} 2\log C^+ - \frac{\partial}{\partial \mathbf{c}_q} 2\log C^- + \frac{\partial}{\partial \mathbf{c}_q} V \\ &= 2\frac{\nabla C^+}{C^+} - 2\frac{\nabla C^-}{C^-} + \frac{\nabla V}{V}\end{aligned} \tag{6.7}$$

where ∇ denotes the derivative of the following terms

$$\nabla C^+ = -\frac{1}{KN^+} \sum_n^{N^+} G(\mathbf{c}_q - \mathbf{x}_n^+, \sigma_a)\sigma_a^{-1}(\mathbf{c}_q - \mathbf{x}_n^+) \tag{6.8}$$

$$\nabla C^- = -\frac{1}{KN^-} \sum_n^{N^-} G(\mathbf{c}_q - \mathbf{x}_n^-, \sigma_a)\sigma_a^{-1}(\mathbf{c}_q - \mathbf{x}_n^-) \tag{6.9}$$

$$\nabla V = -\frac{1}{K^2} \sum_k^{K} G(\mathbf{c}_k - \mathbf{c}_q, \sqrt{2}\sigma_g)\sigma_g^{-1}(\mathbf{c}_q - \mathbf{c}_k) \tag{6.10}$$

Equation 6.3 can then be minimized using gradient descent optimization by means of Equation 6.7. In the nth iteration the position of the PEs $\mathbf{c}_k$ will be updated with the following rule

$$\mathbf{c}_k(n) = \mathbf{c}_k(n-1) - \eta \left(\frac{\nabla C^+}{C+} - \frac{\nabla C^-}{C^-} + \frac{\nabla V}{V} \right) \tag{6.11}$$

where η is the step size. As with self-organizing maps, a good starting point is to choose kernels such that all PEs interact with each other. Large Gaussian variances and large step sizes allow a fast initial distribution of PEs in the feature space. These values will by decreased or annealed later in the training process to obtain stability and smooth convergence.

The operation of the FVQIT is summarized as follows. The input to the algorithm consists of the input data points $\mathbf{x}_n$ and desired outputs $\mathbf{d}_n$. Initialize the positions $\mathbf{c}_k, k = 1, \ldots, K$ of the PEs randomly in the subspace defined by the input space and repeat the following steps until convergence

1. Compute the k-nearest neighbors (Cover & Hart, 1967) of every PE. The most frequent class will be the repelling class of the PE. The second most frequent class will be the attracting class of the PE. Note that the parameter k of the nearest neighbor algorithm is unrelated with the counter k and the number of PEs K. We maintain the notation for purposes of consistency with the literature.

2. Compute the cross information potential C^+ between every PE and the data points from the repelling class (see Equation 6.4), the cross

information potential C^- between every PE and the data points from the attracting class (see Equation 6.5), and the entropy V between PEs (see Equation 6.6).

3. Compute the derivative of the function of energy J (see Equation 6.7).

4. Update the positions $\mathbf{c}_k$ of the PEs (see Equation 6.11).

Ideally, the PEs will find themselves well distributed on the frontiers between classes. Recalling the example illustrated in Figure 6.1 that showed a simple two-class bi-dimensional example, each PE handles a region in the feature space defined by proximity—the local model associated to each PE is composed of the nearest samples according to Euclidean distance. At this moment, the goal is to construct a classifier for each local model. The FVQIT utilizes the single-layer neural network presented in Section 4.1, a lightweight classifier trained with the efficient algorithm proposed in (Castillo et al., 2002). As explained before, the weights of the neural network are computed as follows

$$\sum_{i=0}^{I} A_{pi} w_{ij} = b_{pj} \tag{6.12}$$

where

$$A_{pi} = \sum_{n=1}^{N} x_{in} x_{pn} \qquad\qquad b_{pj} = \sum_{n=1}^{N} f_j^{-1}(d_{jn}) x_{pn} \tag{6.13}$$

allowing rapid supervised training and requires less computational resources than classic methods (see Section 4.1 for more details). This classifier will be in charge of classifying samples in the region assigned to its local model and will be trained only with the points of the training set in this region (see Figure 6.2). Note that the decision boundaries do not necessarily pass through the position of the PEs and do not necessarily intersect in the region boundaries.

6.2 PROPOSED DISTRIBUTED LEARNING MODEL: DFVQIT

The combination of the distributed, partial solutions of the FVQIT in a comprehensive, global solution is not a straightforward process. A simple idea may be to combine every PE from every node but this approximation may lead to redundant, crowded areas in the input space. Figure 6.3 shows an example of the combination of the PEs from two different sites. As can be seen, several PEs are too close one each other. This will affect the predictive

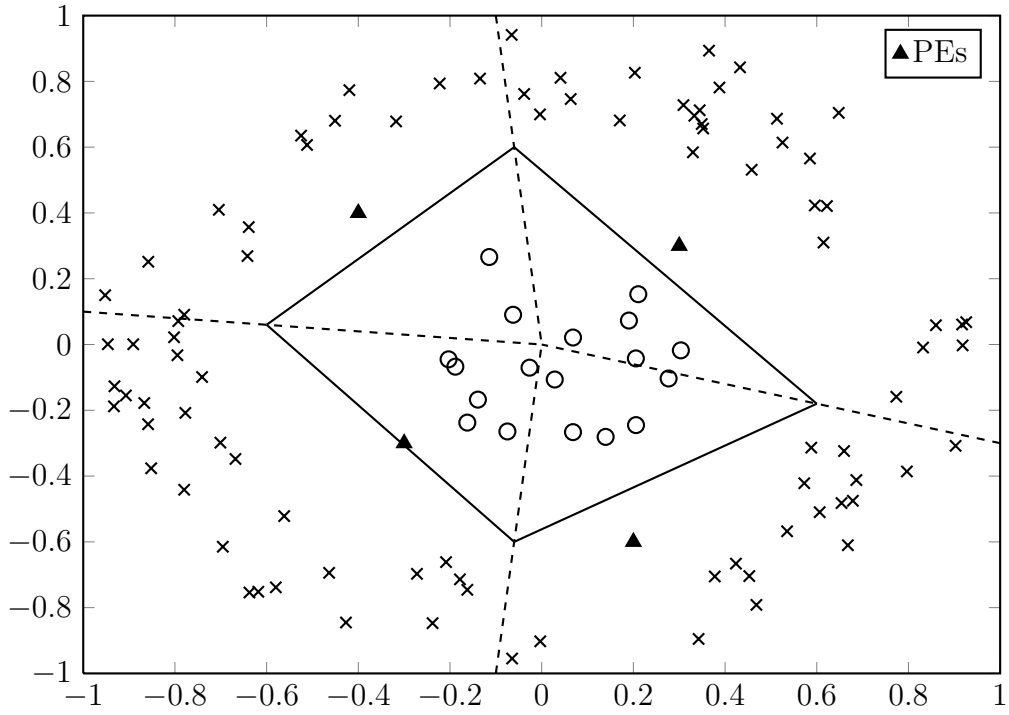


Figure 6.2: Example of a partition of the input space into four parts (dashed lines) and the decision boundary associated to each region (solid lines).

ability of the model in the presence of noise. In order to overcome this difficulty, we propose three strategies to optimize the combination of models. The first one will be based on genetic algorithms, the second one will be based on clustering, and the third one will be a combination of the previous two methods in order to exploit the advantages of both.

Before going into the details of the methods, we would like to introduce how to combine two PEs. Note that a PE is defined by its position in the space and the neural network, discriminant function, associated to the partition of the input space assigned to it. Consider the situation illustrated in Figure 6.4. As can be seen, the two PEs in the left figure can be merged in one with no loss in accuracy.

The procedure for combining two process elements is composed of two steps. On the one hand, the position $\mathbf{c}$ of the resultant PE is located in the midpoint of the positions $\mathbf{c}_1$ and $\mathbf{c}_2$ of the two original PEs, that is, $\mathbf{c} = \frac{\mathbf{c}_1+\mathbf{c}_2}{2}$. On the other hand, using the incremental property of the neural network employed in the FVQIT, the decision boundary of the resultant PE is the sum of the matrices of coefficients of the two original PEs, that is, $\mathbf{A} = \mathbf{A}_1 + \mathbf{A}_2$ and $\mathbf{b} = \mathbf{b}_1 + \mathbf{b}_2$.

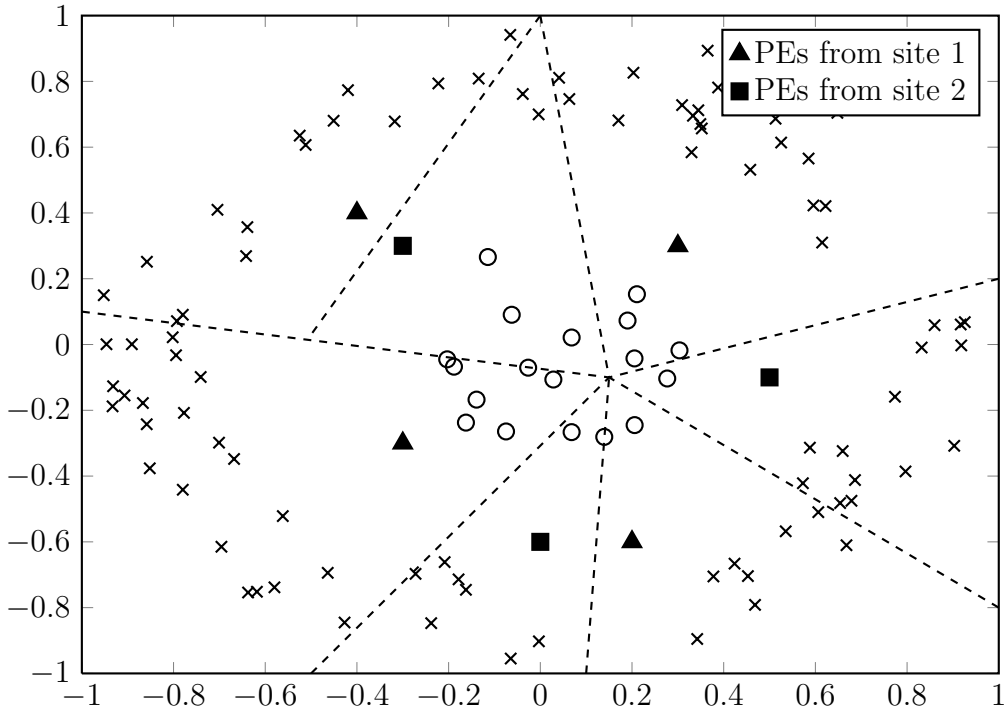
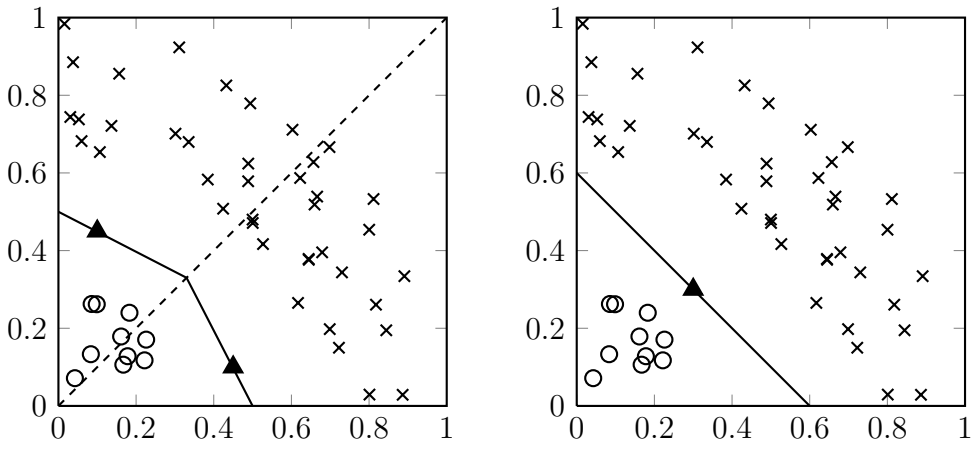


Figure 6.3: Example of a straightforward combination of the process elements from two sites.

6.2.1 COMBINING MODELS USING GENETIC ALGORITHMS

The hypothesis now is that the genetic algorithm will optimize the initial population of PEs by pruning the crowded areas in the input space whilst also optimizing the decision boundaries (see Section 4.2 for more details in the problems of least-squares). Moreover, it is expected that including some global knowledge by means of a genetic algorithm, will expand the search space and obtain a simpler and more general classifier. The challenge here is the encapsulation of a candidate solution. The piecewise representation of the FVQIT leads to complex crossover operators between different solutions. Thus, we turn to consider each PE as an individual of the population in order to have a simple yet powerful method. This change in approach will cause that the optimization process will be driven by fitness-maximization of the population rather than best individual. Note that in the FVQIT, each individual (process element, PE) is defined by its location $\mathbf{c}$ in the input space and its decision boundary contained in the matrices of coefficients $\mathbf{A}$ and $\mathbf{b}$. Under these conditions, the genetic algorithm we propose in this research will be implemented as follows.



(a) Two process elements. (b) Combination of the two PEs in one.

Figure 6.4: Example of partition of the input space (dashed line) and discriminant function (solid line) in the FVQIT.

FITNESS FUNCTION. The fitness function will become the class accuracy of the entire population because of the piecewise representation of the FVQIT. Note that the initial population contains every PE from every node.

SELECTION OPERATOR. Selection is the step in which individuals are chosen from the population for breeding. Note that individuals are usually selected based on their fitness but, in this implementation, the fitness function comprehend the entire population. However, we can take advantage of the locality of the FVQIT in order to propose an effectual selection method. In this research, the population is selected using roulette-wheel selection (Goldberg, 1989) by pairwise Euclidean distance between PEs; the closer the PEs the larger the probability of being selected for breeding.

CROSSOVER OPERATOR. The crossover operator defines how two individuals of the population are combined together to produce the offspring. The crossover operator is computed as already explained before of how to combine two PEs: the two parents generate a child which its location is the midpoint of their locations $\mathbf{c} = \frac{\mathbf{c}_1+\mathbf{c}_2}{2}$ and its decision boundary is the sum of their matrices of coefficients $\mathbf{A} = \mathbf{A}_1 + \mathbf{A}_2$ and $\mathbf{b} = \mathbf{b}_1 + \mathbf{b}_2$ which exploits the incremental features of the algorithm described above. Due to the incremental nature of the crossover operator, the child is expected to substitute both parents if this improves the fitness function.

MUTATION OPERATOR. The mutation operator defines how an individual is altered to produce a new individual. In this research, small random disturbances are used to change the location $\mathbf{c}$ and the matrices of coefficients $\mathbf{A}$ and $\mathbf{b}$ of the individuals.

6.2.2 EXPERIMENTAL STUDY

The objective of this section is to experimentally evaluate the performance of the method based on genetic algorithms proposed in the previous section.

MATERIALS AND METHODS

Note that the genetic algorithm implemented in the previous section can be also seen as a pruning method for the FVQIT algorithm, inasmuch as it diminishes the number of PEs whilst trying to maintain, or even improve, performance. Thus, we consider four distinct scenarios in the evaluation step:

— The original FVQIT

— The FVQIT pruned –the original FVQIT algorithm but adding a pruning step after training by means of the genetic algorithm introduced in the previous section

— The distributed FVQIT –the distributed version of the FVQIT, keeping all the PEs created at every distributed location, without pruning

— The distributed FVQIT pruned –the same distributed method as before but adding a pruning step using the method based on genetic algorithms introduced in the previous section as pruning method

The proposed algorithms are evaluated on twelve data sets of diverse kinds of tasks. Table 6.1 summarizes the number of input features, samples, and output classes of the data sets. A more detailed description of the twelve data sets can be found in (Frank & Asuncion, 2010). The number of input features range from 6 to 54, the number of samples from $4,177$ to $101,241$, and the number of classes from 2 to 28.

In distributed scenarios, training data have been scattered across 5 different nodes in which each node contains 10 PEs. In batch, monolithic scenarios where all the data in stored in a single location, the learning algorithm uses the entire data set in which the number of PEs is set to 50. The evaluation of the methods has been done using holdout, 90% for training, 10% for testing. When pruning is enabled, 10% of the training data have been used for validation. Experiments were run 100 times with random partitions of the

Name	Features	Samples	Classes
Abalone	8	4,177	28
Adult	33	30,162	2
Chess	6	28,056	18
Connect4	42	67,557	3
Forest	54	101,241	7
Letter	16	20,000	26
Magic	10	19,020	2
Mushroom	22	8,124	2
Nursery	16	12,960	5
Poker	10	25,010	10
Shuttle	9	43,500	7
Waveform	21	5,000	3

Table 6.1: Brief description of the data sets.

data set. We use the Kruskal-Wallis test to check if there are significant differences, then we apply a multiple comparison procedure to find the methods which are not significantly different.

RESULTS

Table 6.2 shows the test classification error of the four implementations: the original FVQIT, the FVQIT pruned, the distributed FVQIT, and the distributed FVQIT pruned. The best result, or those not significantly different from the best one, are underlined for each data set. Table 6.3 presents the number of PEs at the end of training. Note that for the original FVQIT algorithm and the DFVQIT, the number of PEs is not always 50 even when pruning is not enabled. This is due to the fact that PEs are removed during the execution of the algorithm if they are not assigned to any sample in the training dataset. Finally, Table 6.4 shows the average training time of the four configurations of the algorithm.

DISCUSSION AND CONCLUSIONS

As can be seen in Table 6.2, regarding classification error, the pruned algorithms obtain the best result, or not significantly different from the best one, in 10 out of 12 data sets (when compared to DFVQIT) and 12 out of 12 (when compared to FVQIT). The original FVQIT is only competitive with the proposed algorithms in 4 out of 12 data sets. In average, pruning decreases classification error by 5.43% and 5.96% with respect to the original FVQIT and plain DFVQIT, respectively. This shows the effectiveness of the

Data set	Original FVQIT	Pruned FVQIT	DFVQIT	Pruned DFVQIT
Abalone	84.00 ± 2.01	79.23 ± 2.49	82.37 ± 1.24	77.06 ± 2.58
Adult	24.24 ± 1.06	17.03 ± 0.66	25.18 ± 1.39	17.25 ± 0.83
Chess	62.95 ± 1.01	64.68 ± 1.51	73.27 ± 1.43	68.39 ± 1.17
Connect4	30.65 ± 0.41	25.99 ± 0.64	32.23 ± 0.79	25.96 ± 0.82
Forest	37.89 ± 1.27	36.31 ± 1.24	37.64 ± 1.14	34.87 ± 0.75
Letter	13.14 ± 1.01	14.31 ± 0.67	22.45 ± 0.96	22.84 ± 1.16
Magic	20.94 ± 0.83	15.79 ± 0.70	19.79 ± 1.65	15.69 ± 0.83
Mushroom	17.14 ± 2.29	11.53 ± 1.22	16.15 ± 2.57	11.34 ± 0.78
Nursery	7.24 ± 0.89	7.54 ± 1.09	8.74 ± 0.90	8.31 ± 0.87
Poker	72.74 ± 0.90	50.26 ± 1.01	72.71 ± 0.91	51.60 ± 1.13
Shuttle	0.51 ± 0.30	0.32 ± 0.13	1.92 ± 1.35	0.54 ± 0.12
Waveform	20.88 ± 2.09	16.14 ± 2.18	18.10 ± 1.86	17.16 ± 1.73

Table 6.2: Test classification error (%) for the four different methods proposed in this section: the original FVQIT algorithm and the distributed version, with and without pruning step

Data set	Original FVQIT	Pruned FVQIT	DFVQIT	Pruned DFVQIT
Abalone	47.10 ± 0.99	7.60 ± 2.22	49.90 ± 0.32	7.00 ± 1.70
Adult	49.90 ± 0.32	4.10 ± 2.47	50.00 ± 0.00	2.40 ± 1.35
Chess	49.90 ± 0.32	34.50 ± 2.92	50.00 ± 0.00	13.30 ± 2.54
Connect4	50.00 ± 0.00	6.70 ± 2.06	50.00 ± 0.00	2.40 ± 1.65
Forest	50.00 ± 0.00	35.89 ± 2.62	50.00 ± 0.00	17.89 ± 1.90
Letter	50.00 ± 0.00	48.40 ± 0.70	50.00 ± 0.00	27.40 ± 2.99
Magic	49.90 ± 0.32	9.80 ± 2.39	49.90 ± 0.32	7.10 ± 1.66
Mushroom	34.20 ± 2.70	13.50 ± 2.76	46.10 ± 1.45	11.30 ± 1.25
Nursery	41.60 ± 0.52	16.80 ± 3.39	48.40 ± 0.70	9.80 ± 2.44
Poker	43.80 ± 1.81	1.60 ± 0.70	50.00 ± 0.00	1.50 ± 0.71
Shuttle	44.20 ± 1.48	21.40 ± 2.59	49.00 ± 0.67	15.10 ± 2.56
Waveform	49.80 ± 0.42	6.21 ± 3.26	49.80 ± 0.42	6.20 ± 1.32

Table 6.3: Number of PEs for the four different methods proposed in this section: the original FVQIT algorithm and the distributed version, with and without pruning step.

proposed method based on genetic algorithms as a pruning method and as a combination method when the algorithm is trained in a distributed fashion. On the other hand, regarding the number of PEs of the models, in average, only the 36.63% and the 20.51% of the PEs are retained after pruning in the original FVQIT and DFVQIT, respectively. Note that the FVQIT algorithm is parametrized with the maximum number of PEs. If a PE do not cover any training sample, it will be deleted. It is important to remark that a smaller number of PEs is related with better generalization performance and faster execution. Finally, regarding training time performance, in average, DFVQIT performs 13.56 times faster than FVQIT. Furthermore, the larger the data set the larger the difference. In Connect4 and Forest data sets, DFVQIT is 19.24 and 22.72 times faster. If the genetic algorithm is used, pruned DFVQIT trains 4.26 and 4.63 times faster than the original FVQIT and the pruned FVQIT, respectively. Also, note that in the distributed setup, the number of nodes is set to 5, so even when the genetic algorithm is used during the integration step, the speedup of the distributed algorithm is close to linear.

6.3 IMPROVING THE SCALABILITY OF THE DISTRIBUTED FVQIT

The DFVQIT outperformed the original FVQIT algorithm in terms of time, complexity of the resultant model and test accuracy. However, the use of a genetic algorithm as integration method in the DFVQIT adds a temporal and computational overhead that may result in lower scalability for some particular datasets. In order to push the scalability of the DFVQIT even further, two novel proposal are presented in this section. In the first place, we propose the use of a clustering method rather than a genetic algorithm during the integration stage. In the second place, we propose to prune the distributed instances of the FVQIT during the map, parallel stage in order to shrink the search space for the integration method during the reduce phase. As demonstrated in the previous section, the integration method of the DFVQIT can be also seen as a pruning method of the FVQIT.

6.3.1 COMBINING MODELS USING HIERARCHICAL CLUSTERING

Flat clustering is efficient and conceptually simple but it has a number of drawbacks. The most important drawback for our purposes is that the number of clusters must be provided in advance—even when there is no clue of the structure of the input space. Hierarchical clustering does not require us

to specify in advance the number of clusters. This advantage of hierarchical clustering come at the cost of lower efficiency. The most common hierarchical clustering algorithms have a complexity that is at least quadratic in the number of data points compared to the linear complexity of K-means (Ceri et al., 2013). Hierarchical clustering groups data by creating a cluster tree or dendrogram. The tree is not a single set of clusters, but rather a multilevel hierarchy, where clusters at one level are joined as clusters at the next level (see Figure 6.5 for an example of the dendrogram drawn from Figure 6.3). This allows to decide the level or scale of clustering that is most appropriate for the specific application.

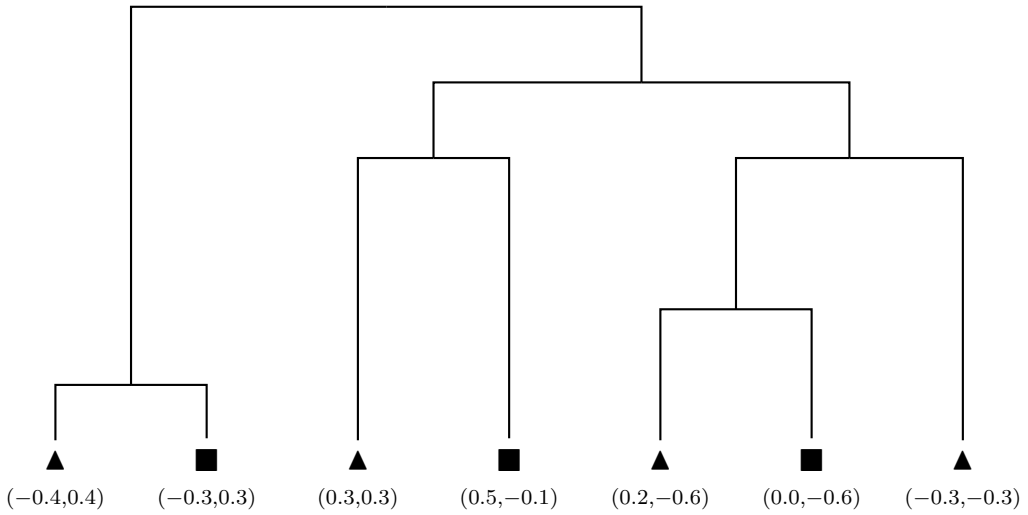


Figure 6.5: Example of a dendrogram.

There are two types of strategies for hierarchical clustering: agglomerative and divisive. Agglomerative clustering is a *bottom-up* approach in which each data point starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. Divisive clustering is a *top-down* approach in which all data points start in one cluster, and splits are performed recursively as one moves down the hierarchy. In the general case, the complexity of agglomerative clustering is $\mathcal{O}(n^3)$ and the complexity of divisive clustering with an exhaustive search is $\mathcal{O}(2^n)$, which makes them too slow for large data sets. However, note that the number of data points for the problem at hand is the number of PEs, which is far smaller than the number of samples in the training dataset. The general procedure proposed for using hierarchical clustering in the DFVQIT consists of the following steps:

1. Compute the pairwise distance between PEs, i.e., find the similarity or dissimilarity between every pair of PEs. There are many ways to

calculate this distance information. The Euclidean distance has been adopted for the purposes of this model, although other distance functions are also possible such as *Minkowski*, *Chebychev*, *Mahalanobis*, *Hamming*, etc.

2. Group the PEs into a binary, hierarchical cluster tree. In this step, the method links pairs of PEs that are in close proximity, using the same combination method explained in the previous section for merging PEs during the execution of the genetic algorithm. As objects are paired into binary clusters, the newly formed clusters are grouped into larger clusters until a hierarchical tree, dendrogram is formed. As explained before, a dendrogram consists of many U-shaped lines that connect PEs in a hierarchical tree. The height of each U represents the distance between the two PEs being connected. The new cluster will represent a new PE calculated as explained in the previous section, in the same manner as for the method based on genetic algorithms.

3. Determine where to cut the hierarchical tree into clusters. In this step, one prunes branches off the bottom of the hierarchical tree, and assign all the objects below each cut to a single cluster. This creates a partition of the PEs. Find the optimal set of clusters by cross-validation against the performance of the method for the different combinations of clusters.

It is expected that the output model resulting from grouping PEs will be a more general representation of the input space with fewer PEs.

6.3.2 A TWO-LEVEL APPROACH USING GENETIC ALGORITHMS AND HIERARCHICAL CLUSTERING

The combination method of the FVQIT can be seen as a pruning method of the topology of the model inasmuch as the combination method diminishes the number of PEs whilst maintaining or improving classification performance. Thus, one can think in a two-level pruning approach with a first, local pruning and a second, global pruning during the combination stage. The hypothesis here is that the first, local pruning will quickly shrink the search space for the second, global pruning. Note that the two-level approach does not depend on the combination algorithm so it will be suitable in both hierarchical clustering and genetic algorithms approaches, i.e. both hierarchical clustering and genetic algorithms may be used in any of the two levels of pruning.

6.3.3 EXPERIMENTAL STUDY

The goal of this section is to experimentally evaluate the different approaches proposed in this section. In particular, the following four implementations of the DFVQIT are compared in this section: one-level GA-based approach (DFVQIT proposed in the previous section, "gold standard" for comparison), one-level cluster-based approach, two-level GA-based approach, and two-level cluster-based approach.

MATERIALS AND METHODS

The same datasets used to assess the performance of the approach based on genetic algorithms for the DFVQIT proposed in Section 6.2.1 are used here. In distributed scenarios, training data have been scattered across 5 different nodes in which each node contains 10 PEs. Thus, the number of PEs after applying the reduce method may be up to 50. The evaluation of the methods has been done using holdout validation: 90% of data for training and 10% for testing. Additionally, 10% of the training data have been used for validation. Experiments were run 100 times with random partitions of the data set to ensure reliable results. Again, we use the Kruskal-Wallis test to check if there are significant differences for a level of significance $\alpha = 0.05$. If there are differences, then we apply a multiple comparison procedure to find the methods which are not significantly different.

RESULTS

The results for the test classification error of the four implementations presented in this section (one-level GA-based approach, one-level cluster-based approach, two-level GA-based approach, and two-level cluster-based approach) are shown in Table 6.5. The best result, or those not significantly different from the best one, are underlined for each data set. Table 6.6 presents the number of PEs at the end of training for the four models. Note that the original number of PEs before pruning was 50. Finally, the average training time of the four implementations using one and two-level approaches is shown in Table 6.7.

DISCUSSION AND CONCLUSIONS

As can be seen in Table 6.5, the differences in terms of error between the one and two-level approaches are negligible ($< 0.23\%$ in average). In the question concerning the performance of GA-based against cluster-based methods, the former outperform the latter by 3.77%. Significant differences are found

Data set	Original FVQIT	Pruned FVQIT	DFVQIT	Pruned DFVQIT
Abalone	9.82 ± 0.08	13.77 ± 0.82	1.10 ± 0.00	6.41 ± 0.51
Adult	96.06 ± 1.16	103.61 ± 2.56	6.33 ± 0.38	11.58 ± 2.00
Chess	24.39 ± 0.12	44.60 ± 1.21	2.15 ± 0.01	17.23 ± 1.61
Connect4	275.92 ± 5.08	294.09 ± 7.85	14.34 ± 0.10	33.80 ± 8.66
Forest	388.78 ± 9.41	517.17 ± 9.42	17.11 ± 0.40	142.67 ± 8.97
Letter	25.81 ± 0.13	57.88 ± 0.57	2.13 ± 0.01	32.41 ± 1.49
Magic	33.24 ± 2.32	41.68 ± 2.21	2.62 ± 0.17	7.51 ± 0.75
Mushroom	37.44 ± 3.16	32.80 ± 3.12	3.90 ± 0.30	7.63 ± 0.37
Nursery	25.96 ± 0.68	30.13 ± 1.51	2.13 ± 0.08	7.69 ± 0.76
Poker	38.06 ± 0.52	34.28 ± 2.00	3.15 ± 0.19	5.43 ± 2.11
Shuttle	52.60 ± 0.84	55.94 ± 1.38	3.17 ± 0.05	15.89 ± 0.69
Waveform	27.42 ± 3.77	26.93 ± 2.51	2.72 ± 0.48	5.27 ± 0.58

Table 6.4: Training time (s) for the four different methods proposed in this section: the original FVQIT algorithm and the distributed version, with and without pruning step.

Data set	One-level pruning		Two-level pruning	
	GA-based	*Cluster-based*	*GA-based*	*Cluster-based*
Abalone	77.06 ± 2.58	80.12 ± 1.92	77.32 ± 1.46	79.69 ± 1.03
Adult	17.25 ± 0.83	17.20 ± 1.08	16.71 ± 0.61	17.35 ± 1.02
Chess	68.39 ± 1.17	69.57 ± 1.00	69.42 ± 1.38	69.98 ± 1.06
Connect4	25.96 ± 0.82	27.89 ± 0.81	25.99 ± 0.70	27.28 ± 1.01
Forest	34.87 ± 0.75	35.77 ± 0.68	34.71 ± 0.64	36.68 ± 1.07
Letter	22.84 ± 1.16	22.50 ± 1.73	23.45 ± 1.12	22.84 ± 1.28
Magic	15.69 ± 0.83	18.10 ± 0.76	15.87 ± 0.49	17.31 ± 1.23
Mushroom	11.34 ± 0.78	13.79 ± 0.84	12.44 ± 1.34	13.35 ± 1.90
Nursery	8.31 ± 0.87	8.87 ± 0.74	8.52 ± 1.04	8.73 ± 1.25
Poker	51.60 ± 1.13	52.56 ± 1.99	51.46 ± 1.44	51.68 ± 1.95
Shuttle	0.54 ± 0.12	1.28 ± 0.56	0.57 ± 0.24	1.10 ± 0.35
Waveform	17.16 ± 1.73	17.12 ± 2.53	17.26 ± 2.49	16.72 ± 1.39

Table 6.5: Test classification error (%) for the four different methods proposed in this section: the genetic-algorithm- and clustering-based methods, using one- and two-level pruning.

Data set	One-level pruning		Two-level pruning	
	GA-based	*Cluster-based*	*GA-based*	*Cluster-based*
Abalone	7.00 ± 1.70	7.80 ± 2.30	4.70 ± 1.42	9.50 ± 5.68
Adult	2.40 ± 1.35	1.40 ± 0.84	1.30 ± 0.67	2.10 ± 1.14
Chess	13.30 ± 2.54	19.60 ± 6.95	11.90 ± 1.73	14.30 ± 5.02
Connect4	2.40 ± 1.65	7.00 ± 3.43	1.50 ± 0.85	8.10 ± 3.88
Forest	17.89 ± 1.90	23.90 ± 6.30	15.22 ± 3.63	21.70 ± 7.11
Letter	27.40 ± 2.99	32.60 ± 5.85	29.00 ± 3.20	33.50 ± 3.67
Magic	7.10 ± 1.66	8.30 ± 4.57	5.00 ± 1.05	12.40 ± 6.12
Mushroom	11.30 ± 1.25	34.50 ± 11.19	9.30 ± 1.49	31.40 ± 8.10
Nursery	9.80 ± 2.44	18.90 ± 14.68	4.60 ± 1.58	19.30 ± 5.32
Poker	1.50 ± 0.71	1.60 ± 0.84	1.10 ± 0.32	1.00 ± 0.00
Shuttle	15.10 ± 2.56	32.00 ± 8.74	12.20 ± 2.62	27.40 ± 8.19
Waveform	6.20 ± 1.32	7.20 ± 4.57	5.10 ± 1.66	11.00 ± 8.76

Table 6.6: Number of PEs for the four different methods proposed in this section: the genetic-algorithm- and clustering-based methods, using one- and two-level pruning.

Data set	One-level pruning		Two-level pruning	
	GA-based	*Cluster-based*	*GA-based*	*Cluster-based*
Abalone	6.41 ± 0.51	1.68 ± 0.02	4.55 ± 0.58	1.37 ± 0.09
Adult	11.58 ± 2.00	8.17 ± 0.68	7.57 ± 2.66	5.25 ± 0.30
Chess	17.23 ± 1.61	3.08 ± 0.02	13.96 ± 1.06	3.03 ± 0.13
Connect4	33.80 ± 8.66	17.97 ± 0.22	19.40 ± 8.99	10.97 ± 0.25
Forest	142.67 ± 8.97	29.56 ± 0.30	104.69 ± 9.06	24.61 ± 0.88
Letter	32.41 ± 1.49	3.37 ± 0.04	24.46 ± 1.81	3.53 ± 0.03
Magic	7.51 ± 0.75	2.84 ± 0.01	4.69 ± 0.42	2.46 ± 0.06
Mushroom	7.63 ± 0.37	4.38 ± 0.40	5.81 ± 0.52	3.05 ± 0.26
Nursery	7.69 ± 0.76	2.46 ± 0.02	5.36 ± 0.78	2.26 ± 0.11
Poker	5.43 ± 2.11	3.29 ± 0.02	2.52 ± 1.52	2.49 ± 0.07
Shuttle	15.89 ± 0.69	3.86 ± 0.06	12.60 ± 1.35	3.72 ± 0.13
Waveform	5.27 ± 0.58	3.05 ± 0.33	3.76 ± 0.64	2.18 ± 0.24

Table 6.7: Training time (s) for the four different methods proposed in this section: the genetic-algorithm- and clustering-based methods, using one- and two-level pruning.

between GA-based and cluster-based methods, but not between one and two-level approaches. However, in 6 out of 12 data sets, GA-based methods are not significantly different from cluster-methods. On the other hand, regarding the number of PEs in the final model (see Table 6.6), both GA-based and cluster-based pruning methods are able to significantly reduce the number of PEs (around 81% the former and 68% the latter). As conjectured, the stochastic behavior of GAs plays in favor of flexible configurations of PEs against the deterministic behavior of clustering. Moreover, the two-level methods reduce the number of PEs by and extra 16.86% and 1.59% in GA and cluster-based algorithms, respectively, in comparison with one-level pruning methods. However, these differences are not statistically significant. Finally, regarding the training time, in average, cluster-based methods perform 3.51 and 3.23 times faster than GA-based methods in the one and two-level pruning approaches, respectively. As expected, these differences are significant with respect to the former implementation of the FVQIT. Moreover, the two-level approach is worth it in terms of time. This approach reduces by 28.67% the training time of GA-based method and by 22.45% of cluster-based methods. These results support the initial assumption that states that the first, local pruning would quickly reduce the search space for the second, global pruning. Note also that this response is expected to be boosted for larger numbers of PEs.

In summary, all these results suggest that cluster-based methods are faster than those based on genetic algorithms at the expense of slight higher number of PEs and performance error. If we compare one-level against two-level approaches, the latter are faster and simpler –less number of PEs– whilst maintains performance error. In light of the above, we can say that if the learning process is strongly constrained by the training time then the two-level cluster-based DFVQIT is recommendable. Contrarily, if the learning process is constrained by maximum-accuracy then the two-level genetic algorithm-based DFVQIT is the most appropriate method.

Chapter 7

Distributed One-Class Support Vector Machine

This chapter presents a novel distributed one-class classification approach based on an extension of the ν-SVM method. In this novel method (Castillo, Peteiro-Barral, Berdiñas, & Fontenla-Romero, 2015), several models will be considered, each one determined using a given local data partition on a processor, and the goal is to find a global model. The cornerstone of this method is the novel mathematical formulation that makes the optimization problem separable whilst avoid some data points considered as outliers in the final solution. This is particularly interesting and important because the decision region generated by the method will be unaffected by the position of the outliers and will fit the data in a more natural manner.

7.1 INTRODUCTION TO ONE-CLASS CLASSIFICATION

In classical supervised classification problems the discriminating models are trained using positive and negative examples. Nevertheless, for a number of practical problems, counter-examples are either rare or entirely unavailable. One-class classifiers (Moya & Hush, 2013) have emerged as a technique for situations where labeled data exists for only one of the classes in a two-class problem. One-class classification is also called outlier (or novelty) detection because the learning algorithm is being used to differentiate between data that appears normal and abnormal with respect to the distribution of the training data (Chandola, Banerjee, & Kumar, 2009). This type of classifiers are relevant in many real applications, for example, machine fault detection (Mahadevan & Shah, 2009; Fernández-Francos, Martínez-Rego, Fontenla-Romero, & Alonso-Betanzos, 2013), text classification (Manevitz & Yousef, 2007) and image analysis (Bilgin, Erturk, & Yildirim, 2011; Cyganek, 2008;

Lai, Tax, Duin, Pekalska, & Paclík, 2004). Earlier research in this area was directed on density estimation with parametric generative models, such as Gaussian mixture models, however they usually make assumptions about the nature of the underlying distribution. One of the most widespread strategy is to enclose the provided training data by a boundary using a hypersphere (D. M. J. Tax & Duin, 1999; Schölkopf, Platt, Shawe-Taylor, Smola, & Williamson, 2001), a set or ellipsoids (Martínez-Rego, Castillo, Fontenla-Romero, & Alonso-Betanzos, 2013), the convex hull (Casale, Pujol, & Radeva, 2011) or a convex polytope (Casale, Pujol, & Radeva, 2014), and to measure the distance to the estimated surface. New lines of recent research are also opening up in the field of Gaussian Processes (Kemmler, Rodner, Wacker, & Denzler, 2013) and Random Forests (Désir, Bernard, Petitjean, & Heutte, 2013). Two of the most well-known techniques are the Support Vector Domain Description (D. M. J. Tax & Duin, 1999) and one-class ν-SVM (Schölkopf et al., 2001). Although they are successful tools for one-class classification, they applicability for large data sets is quite restricted due to their high computational demand. Some attempts have been proposed to reduce the training time and memory consuming such us the work in (Zhu, Ye, Yu, Xu, & Li, 2014) that reduces the training set by selecting useful samples for training; the work in (Cabral & Oliveira, 2011, 2012) based on prototype reduction by creating artificial prototypes outside the normal description of the class; or the work in (Clifton et al., 2014) which proposes a method that allows interpreting the SVM output as a conditional class probability which brings the advantage, among many others, that a cross-validation process is avoided in order to select appropriate values for the SVM parameters, thus saving computational time.

In this chapter, we will present an extension of the ν-SVM model to be applied in a *distributed one-class* scenario, called DOC-SVM, thus permitting its application to large datasets. In this distributed context, Das et col. (Das, Bhaduri, & Votava, 2011) presented a one-class SVM for anomaly detection when the dataset is vertically distributed. That is, when only a subset of features is available at any physical partition. In this approach, a high accuracy compared to complete centralization is obtained with only centralizing a very small sample of the subsets. In addition, some approaches for horizontal distribution have also been proposed. In (Krawczyk, Woźniak, & Cyganek, 2014; Krawczyk & Woźniak, 2014) a highly parallel architecture is proposed for creating ensembles of one-class classifiers based on the idea of data clustering in the feature space into smaller partitions. The proposed framework, called the one-class clustering-based ensemble (OCClustE), is very flexible and places no restrictions on the clustering, the one-class classifier and the ensemble fusion methods to be used. Another interesting proposal for hor-

izontal distribution can be found in (To & Elati, 2013), where the authors applied an island model for genetic programming for one-class classification in which only the trees generated at each island are exchanged among them.

7.2 PROPOSED MODEL: DOC-SVM

This section describes, mathematically, the distributed one-class support vector machine presented in this chapter (DOC-SVM). In this novel algorithm, the extension of the ν-SVM is inspired by the formulation of one-class Support Vector Machines (SVM) proposed by (Schölkopf et al., 2001). It employs the idea of projecting the training samples to a higher dimensional feature space and then separating most of the samples from the origin, as far as possible, using a maximum margin hyperplane. This is equivalent to find the smallest region (sphere) enclosing the data in the original space. However, in our distributed method we will consider several regions, each one determined using a given local data partition on a processor, and the goal is to find a global classifier. The cornerstone of this method is the novel mathematical formulation that makes the optimization problem separable, thus allowing to learn in parallel from each partition on different processors, whilst avoiding some data samples considered as outliers in the final solution. This is particularly interesting and important because the decision region generated by the method will be unaffected by the position of the outliers and that will contribute to obtain the smallest volume region that fits the data.

7.2.1 BACKGROUND: THE BASIC ONE-CLASS CLASSIFIER MODEL

For the sake of comprehension, in this section we reproduce the basis of the formulation of the one-class Support Vector Machine (SVM) proposed by (Schölkopf et al., 2001) that will be used as the root model in our proposal.

Let $X_m = \{\mathbf{x}_i\}, i = 1, \ldots, m$ be a set of m training samples of a single class, where $\mathbf{x}_i$ is a sample in $\mathbb{R}^d$, the classifier estimates the boundary region of a minimum volume that captures an appropriate fraction of data, so that a if new sample lies within this boundary it is labeled as a normal class, otherwise it is labeled as an outlier. The ν-SVM constructs the boundary using the main ideas of support vector kernel methods. Therefore, first the training data is mapped into a higher dimensional feature space induced by a kernel function. Afterwards, on the assumption that the origin in the transformed space belongs to the outlier class, these transformed samples are separated from the origin by a maximum-margin hyperplane which is as far away as possible from the origin. To separate the data set from the origin,

the authors propose the following optimization problem in order to find the parameters, normal weight vector $\boldsymbol{w}$ and a threshold ρ, of the hyperplane:

$$\underset{\boldsymbol{w}, \boldsymbol{\xi}, \rho}{\text{Minimize}} \quad \frac{1}{2}||\boldsymbol{w}||^2 - \rho + \frac{1}{\nu m}\sum_{i=1}^{m}\xi_i \tag{7.1}$$

subject to

$$\langle \boldsymbol{w}, \phi(\mathbf{x}_i)\rangle \geq \rho - \xi_i, \xi_i \geq 0 \tag{7.2}$$

where $\nu \in (0, 1]$ is an upper bound on the fraction of data that could be outliers, ξ_i are the slack variables (errors) that allows relaxing the constraints in some cases, $\phi(\cdot)$ is a non-linear transformation, $\langle \cdot, \cdot \rangle$ stands for the inner product, and $||\boldsymbol{w}||$ stands for the norm of $\boldsymbol{w}$. This primal problem can be solved by introducing the Lagrange multipliers α_i and finding the solution for the corresponding dual problem:

$$\underset{\boldsymbol{\alpha}}{\text{Minimize}} \quad \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j\langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle \tag{7.3}$$

subject to

$$0 \leq \alpha_i \leq \frac{1}{\nu m}, \sum_{i=1}^{m}\alpha_i = 1 \tag{7.4}$$

$$\tag{7.5}$$

which, using the kernel trick and a kernel function $k(\cdot, \cdot)$, allows to obtain the final decision by:

$$f(\boldsymbol{x}) = sgn\left(\sum_{i=1}^{m}\alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) - \rho\right) \tag{7.6}$$

7.2.2 MATHEMATICAL FORMULATION OF THE PROPOSED MODEL

Now, assume that the data points are distributed into K partitions, which are referred by index k. These partitions could be artificially created or the data can be distributed in origin. To determine the maximum-margin hyperplanes associated with the different K partitions, we need to deduce their normal vector $\boldsymbol{w}_k$ and thresholds ρ_k by solving the following optimization problem:

$$\underset{\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\rho}, \boldsymbol{\eta}, \mathbf{u}}{\text{Minimize}} \quad \sum_{k=1}^{K}\left(\frac{1}{2}||\boldsymbol{w}_k||^2 - \rho_k\right) + \frac{1}{\nu m}\sum_{i=1}^{m}u_i\xi_i \tag{7.7}$$

subject to

$$\begin{aligned}
\sum_k \eta_i^k (A_i^k + \rho_k - \xi_i) &\leq 0 : \alpha_i \quad \forall i && (7.8)\\
\sum_k \eta_i^k &= 1 : \beta_i; \quad \forall i && (7.9)\\
\sum_i u_i &= m - n_{out} : \gamma; && (7.10)\\
-\xi_i &\leq 0 : \lambda_i; \quad \forall i && (7.11)\\
\eta_i^k &\leq 1 : \sigma_i^k \quad \forall i, k. && (7.12)\\
-\eta_i^k &\leq 0 : \tau_i^k \quad \forall i, k && (7.13)\\
u_i &\leq 1 : \pi_i \quad \forall i && (7.14)\\
-u_i &\leq 0 : \chi_i \quad \forall i, && (7.15)
\end{aligned}$$

where u_i are variables that take value 0 for outliers and 1, otherwise; η_i^k are variables that take value 1 if the sample point i is assigned to partition k; $\alpha_i, \beta_i, \gamma, \lambda_i, \sigma_i^k, \tau_i^k, \pi_i$, and χ_i are dual variables associated with the corresponding constraints (7.8)-(7.15); and $\phi()$ is a non-linear transformation such that

$$A_i^k = -\langle \boldsymbol{w}_k, \phi_k(\mathbf{x}_i) \rangle. \tag{7.16}$$

Equation (7.8), similarly to Equation (7.2) in the original non-distributed problem, forces the points to be inside the corresponding decision regions; Equation (7.9) forces each point i to be assigned to one and only one partition; Equation (7.10) forces the number of outliers to be n_{out}; Equation (7.11) forces the non-negativity of ξ_i; Equations (7.12) to (7.15) define variables u_i and η_i^k to be continuous and defined on the range $[0, 1]$. In fact, the variables u_i and η_i^k should be binary variables, but we will demonstrate later on that these variables can be replaced by continuous variables in the range $[0, 1]$, such as presented in the above formulation, to obtain an equivalent solution.

To facilitate the solution of Equation (7.7), the problem can be transformed to a dual space representation using positive Lagrangian multipliers. The primal Lagrangian function $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\rho}, \boldsymbol{\eta}, \mathbf{u}; \boldsymbol{\alpha}, \boldsymbol{\beta}, \gamma, \boldsymbol{\lambda}, \boldsymbol{\sigma}, \boldsymbol{\tau}, \boldsymbol{\pi}, \boldsymbol{\chi})$ of

Equation (7.7) is then given by

$$\begin{aligned}
\mathcal{L} &= \sum_k \left(\frac{1}{2} ||\boldsymbol{w}_k||^2 - \rho_k \right) + \frac{1}{\nu m} \sum_i u_i \xi_i \\
&\quad + \sum_i \alpha_i \left(\sum_k \eta_i^k \left(A_i^k + \rho_k - \xi_i \right) \right) \\
&\quad + \sum_i \beta_i \left(\sum_k \eta_i^k - 1 \right) \\
&\quad + \gamma \left(\sum_i u_i - m + n_{out} \right) - \sum_i \lambda_i \xi_i \\
&\quad + \sum_{i,k} \sigma_i^k (\eta_i^k - 1) - \sum_{i,k} \tau_i^k \eta_i^k \\
&\quad + \sum_i \pi_i (u_i - 1) - \sum_i \chi_i u_i. \qquad (7.17)
\end{aligned}$$

and the Karush-Kuhn-Tucker (KKT) conditions are:

$$0 = w_k^j - \sum_i \alpha_i \eta_i^k \phi_k^j(\mathbf{x}_i); \quad \forall k, j \qquad (7.18)$$

$$0 = \frac{1}{\nu m} u_i - \sum_k \alpha_i \eta_i^k - \lambda_i; \quad \forall i \qquad (7.19)$$

$$0 = \sum_i \alpha_i \eta_i^k - 1; \quad \forall k \qquad (7.20)$$

$$0 = \alpha_i (A_i^k + \rho_k - \xi_i) + \beta_i + \sigma_i^k - \tau_i^k; \quad \forall i, k \qquad (7.21)$$

$$0 = \frac{1}{\nu m} \xi_i + \gamma + \pi_i - \chi_i; \quad \forall i \qquad (7.22)$$

$$\sum_k \eta_i^k (A_i^k + \rho_k - \xi_i) \leq 0; \quad \forall i \tag{7.23}$$

$$\sum_k \eta_i^k = 1; \quad \forall i \tag{7.24}$$

$$\sum_i u_i = m - n_{out}; \tag{7.25}$$

$$-\xi_i \leq 0; \quad \forall i \tag{7.26}$$

$$\eta_i^k \leq 1; \quad \forall i, k. \tag{7.27}$$

$$-\eta_i^k \leq 0; \quad \forall i, k \tag{7.28}$$

$$u_i \leq 1; \quad \forall i \tag{7.29}$$

$$-u_i \leq 0; \quad \forall i \tag{7.30}$$

$$0 = \alpha_i \left(\sum_k \eta_i^k \left(A_i^k + \rho_k - \xi_i \right) \right); \quad \forall i \tag{7.31}$$

$$0 = \lambda_i \xi_i; \quad \forall i \tag{7.32}$$

$$0 = \sigma_i^k (\eta_i^k - 1); \quad \forall i, k \tag{7.33}$$

$$0 = \tau_i^k \eta_i^k; \quad \forall i, k \tag{7.34}$$

$$0 = \pi_i (u_i - 1); \quad \forall i \tag{7.35}$$

$$0 = \chi_i u_i; \quad \forall i \tag{7.36}$$

$$0 \leq \sigma_i^k, \tau_i^k; \quad \forall i, k \tag{7.37}$$

$$0 \leq \alpha_i, \lambda_i, \pi_i, \chi_i; \quad \forall i. \tag{7.38}$$

where w_k^j and $\phi_k^j(\mathbf{x}_i)$ are the j components of $\boldsymbol{w}_k$ and $\phi_k(\mathbf{x}_i)$, respectively. In the following sections, several results are given in order to finally obtain a solution to the original optimization problem in Equations (7.7)-(7.15).

7.2.3 AVOIDING BINARY VARIABLES

In what follows we provide some results showing that the optimization problem presented in the previous subsection, with continuous variables in the interval $[0, 1]$, and its binary form (being u_i and η_i^k binary variables) share the same optimal value. This change in the formulation implies a substantial reduction in CPU time, since the equivalent mixed integer program has a higher computational complexity. This is because many combinations of specific integer values for the variables must be tested, and each combination requires the solution of a nonlinear optimization problem. Thus, the number

of combinations rise exponentially with the size of the problem.

Theorem 7.2.1 (Binary and relaxed problems equivalence) *The problem proposed previously in (7.7)-(7.15) and its binary form (making u_i and η_i^k binary) share the same optimal value of the objective function. In addition, for any optimal solution of the continuous problem there exists binary solutions for u_i and η_i^k which are also optimal solutions of the same problem. Consequently, the u_i and η_i^k binary values of the binary problem can be immediately obtained from the u_i and η_i^k values of the relaxed (continuous) problem.*

Proof *First we deal with the u_i variables.* If all the resulting values of u_i in the Problem (7.7)-(7.15) are zeros or ones, we have a binary solution and then the relaxed and the binary problems provide the same solution. Otherwise, all data points i with $0 < u_i < 1$ must share the same $\xi_i = -\gamma\nu m$ value, as will be demonstrated later on in Theorem 7.2.2. This implies that these points are just in the boundary between outliers and regular points and since they must be n_{out} outliers, we can freely choose arbitrary data points among then to complete the n_{out} outliers. In order to accomplish this, we reassign the u_i values to zeros, for the selected outliers, and ones, for those points that we can keep as regular points. This change does not alter either the sum $\sum_i u_i$ in the constraint (Equation (7.10)) or the term $\sum_i u_i\xi_i$ in the objective function, i.e. we obtain a feasible binary solution that provides the same value of the objective function (Equation (7.7)). Since this can always be done, we have proved that the relaxed and the binary problems reach the same optimal value.

Next, we deal with the η_i^k variables. If all the resulting values of η_i^k are zeros or ones, we have a binary solution and then the relaxed and the binary problems provide the same solution. Otherwise, all data points i with $0 < \eta_i^k < 1$ must share the same $A_i^k + \rho_k$ value. In this case, we can reassign the η_i^k values to binary values by keeping its sum (Equation (7.9)) without changing the solution regions because we do not change $\sum_k \eta_i^k(A_i^k + \rho_k - \xi_i)$ (see Equation (7.8)). Since this change in the η_i^k values does not alter the objective function (Equation (7.7)) we obtain a feasible binary solution that provides the same value of the objective function. Since this can be always done, we have proved that the relaxed and the binary problems reach the same optimal value.

7.2.4 CRITERIA FOR OUTLIER DETERMINATION AND PARTITION ASSIGNATION

The theorems included bellow establish the rules needed to select which data are outliers and to reassign data samples to partitions.

Theorem 7.2.2 (Outlier selection rule) *The n_{out} outliers in Problem (7.7)-(7.15) correspond to the n_{out} data points with the largest values of ξ_i. In case of tie, which occurs only when $\xi_i = -\gamma\nu m$, any selection among tied cases is a valid solution and all of them provide the same objective function value.*

Proof From Equation (7.22) we get

$$\frac{1}{\nu m}\xi_i + \gamma + \pi_i - \chi_i = 0; \quad \forall i, \tag{7.39}$$

and then we have:

1. If $0 < u_i < 1$, from Equations (7.35) and (7.36) we obtain $\pi_i = \chi_i = 0$, respectively, and from Equation (7.39) we get $\xi_i = -\gamma\nu m$, that is, this case is possible only when the ξ_i share the same value for all the corresponding i with $\pi_i = \chi_i = 0$. This proves that

$$\xi_i \neq -\gamma\nu m \quad \Leftrightarrow \quad u_i = 0 \text{ or } u_i = 1. \tag{7.40}$$

2. If $u_i = 0$ because of Equation (7.35) then $\pi_i = 0$ and due to Equation (7.39) we have
$$\frac{1}{\nu m}\xi_i + \gamma - \chi_i = 0,$$
which implies $\xi_i = -\gamma\nu m$ if $\chi_i = 0$ and $\xi_i > -\gamma\nu m$ if $\chi_i > 0$.

3. If $u_i = 1$ because of Equation (7.36) then $\chi_i = 0$ and due to Equation (7.39) we have
$$\frac{1}{\nu m}\xi_i + \gamma + \pi_i = 0,$$
which implies $\xi_i = -\gamma\nu m$ if $\pi_i = 0$ and $\xi_i < -\gamma\nu m$ if $\pi_i > 0$.

This means that the ξ_i and the threshold value $-\gamma\nu m$ allow us to determine whether or not a data point is an outlier, and that the three considerations above imply the outlier selection rule indicated by the theorem. Consequently, point i is selected as an outlier if its ξ_i is among the n_{out} largest values of ξ_i.

Theorem 7.2.3 (Partition assignment rule) *The optimal solution of problem (7.7)-(7.15) assigns data points to partitions using the following rule: Point i is assigned to the partition k iff $k = \arg\min_k(A_i^k + \rho_k)$.*

Proof From Equation (7.21) we get

$$\alpha_i(A_i^k + \rho_k - \xi_i) + \beta_i + \sigma_i^k - \tau_i^k = 0; \quad \forall i, k, \tag{7.41}$$

and then, we have:

1. If $0 < \eta_i^k < 1$, from Equations (7.33) and (7.34) we obtain $\sigma_i^k = \tau_i^k = 0$, respectively, and then from Equation (7.41) we obtain $\xi_i - \beta_i/\alpha_i = A_i^k + \rho_k$, that is, this case is possible only when the corresponding data points share the same value for all k. This proves that

$$(A_i^k + \rho_k) \neq \xi_i - \beta_i/\alpha_i \quad \Leftrightarrow \quad \eta_i^k = 0 \text{ or } \eta_i^k = 1.$$

2. If $\eta_i^k = 0$ because of Equation (7.33) then $\sigma_i^k = 0$ and due to Equation (7.41) we have
$$\alpha_i(A_i^k + \rho_k - \xi_i) + \beta_i - \tau_i^k = 0,$$
which implies $A_i^k + \rho_k = \xi_i - \beta_i/\alpha_i$ if $\tau_i^k = 0$ and $(A_i^k + \rho_k) > \xi_i - \beta_i/\alpha_i$ if $\tau_i^k > 0$.

3. If $\eta_i^k = 1$ because of Equation (7.34) then $\tau_i^k = 0$ and due to Equation (7.41) we have
$$\alpha_i(A_i^k + \rho_k - \xi_i) + \beta_i + \sigma_i^k = 0,$$
which implies $(A_i^k + \rho_k) = \xi_i - \beta_i/\alpha_i$ if $\sigma_i^k = 0$ and $(A_i^k + \rho_k) < \xi_i - \beta_i/\alpha_i$ if $\sigma_i^k > 0$.

This means that $(A_i^k + \rho_k)$ and the threshold values $\xi_i - \beta_i/\alpha_i$ allow us to determine whether or not a data point should be in partition k, and that the three considerations above imply the assignment rule indicated by the theorem. Consequently, point i is assigned to partition k if its $(A_i^k + \rho_k)$ value is the smallest value of $(A_i^k + \rho_k); k = 1, 2, \ldots, K$.

7.2.5 ADDITIONAL PROPERTIES

Furthermore, the following properties hold:

1. If $\tau_i^k > 0$ then $(A_i^k + \rho_k) \geq \xi_i - \beta_i/\alpha_i$.

2. If $\sigma_i^k > 0$ then $(A_i^k + \rho_k) \leq \xi_i - \beta_i/\alpha_i$.

3. If $\chi_i > 0$ then $\xi_i \geq -\gamma\nu m$.

4. If $\pi_i > 0$ then $\xi_i \leq -\gamma\nu m$.

These properties can be derived from the Karush-Kuhn-Tucker conditions in Equations 7.18-7.38 as follows:

1. If $\tau_i^k > 0$ because of Equation (7.34) then $\eta_i^k = 0$ and due to Equation (7.33) $\sigma_i^k = 0$. In this case, from Equation (7.41) we get

$$\begin{aligned} &\alpha_i(A_i^k + \rho_k - \xi_i) + \beta_i - \tau_i^k = 0; \\ \Leftrightarrow \quad &(A_i^k + \rho_k) \geq \xi_i - \beta_i/\alpha_i, \end{aligned}$$

2. If $\sigma_i^k > 0$ because of Equation (7.33) then $\eta_i^k = 1$ and due to Equation (7.34) $\tau_i^k = 0$. In this case, from Equation (7.41) we get

$$\begin{aligned} &\alpha_i(A_i^k + \rho_k - \xi_i) + \beta_i + \sigma_i^k = 0; \\ \Leftrightarrow \quad &(A_i^k + \rho_k) \leq \xi_i - \beta_i/\alpha_i, \end{aligned}$$

3. If $\chi_i > 0$ because of Equation (7.36) then $u_i = 0$ and due to Equation (7.35) $\pi_i = 0$. In this case, from Equation (7.39) we get

$$\frac{1}{\nu m}\xi_i + \gamma - \chi_i = 0; \quad \Leftrightarrow \quad \xi_i \geq -\gamma\nu m.$$

4. If $\pi_i > 0$ because of Equation (7.35) then $u_i = 1$ and due to Equation (7.36) $\chi_i = 0$. In this case, from Equation (7.39) we get

$$\frac{1}{\nu m}\xi_i + \gamma + \pi_i = 0; \quad \Leftrightarrow \quad \xi_i \leq -\gamma\nu m.$$

7.2.6 SOLUTION TO THE PROBLEM: SEPARABLE OPTIMIZATION APPROACH AND BI-LEVEL ALGORITHM

Since from the KKT conditions we were able to decide which data points are outliers and to derive the rules to decide which data points must be assigned to each partition, we can reconsider the Problem (7.7)-(7.15) and use a bi-level approach in which the first level solves the Problem (7.7)-(7.15) with known η_i^k and u_i variables, and the second level simply decides about outliers and assigns data points to partitions. This process can be repeated until convergence is attained.

Note that, if the assignment of the data points has been done and the outliers have been identified, that is, variables η_i^k and u_i are known, the Problem (7.7)-(7.15) becomes the Problem:

$$\underset{\boldsymbol{w},\boldsymbol{\xi},\boldsymbol{\rho}}{\text{Minimize}} \quad \sum_{k=1}^{K}\left(\frac{1}{2}||\boldsymbol{w}_k||^2 - \rho_k + \frac{1}{\nu m}\sum_{i=1|\bar{u}_i\bar{\eta}_i^k=1}\xi_i\right) \tag{7.42}$$

subject to

$$\begin{aligned}(A_i^k + \rho_k - \xi_i) &\leq 0 : \alpha_i \quad \forall i|\bar{\eta}_i^k = 1 &(7.43)\\ -\xi_i &\leq 0 : \lambda_i; \quad \forall i|\bar{\eta}_i^k = 1, &(7.44)\end{aligned}$$

where the bar above the variables η_i^k and u_i refer to the fact that they are known values.

Problem (7.42)-(7.44) is separable in k, that is, we can distribute the calculations over the k partitions into k different processors, such that the kth processor solves the problem:

$$\underset{\boldsymbol{w},\boldsymbol{\xi},\boldsymbol{\rho}}{\text{Minimize}} \quad \frac{1}{2}||\boldsymbol{w}_k||^2 - \rho_k + \frac{1}{\nu m}\sum_{i=1|\bar{u}_i\bar{\eta}_i^k=1}\xi_i \tag{7.45}$$

subject to

$$\begin{aligned}(A_i^k + \rho_k - \xi_i) &\leq 0 : \alpha_i \quad \forall i|\bar{\eta}_i^k = 1 &(7.46)\\ -\xi_i &\leq 0 : \lambda_i; \quad \forall i|\bar{\eta}_i^k = 1. &(7.47)\end{aligned}$$

The dual of the previous problem can be obtained by considering its Lagrangian (for simplicity we remove the k index):

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}\sum_j w_j^2 - \rho + \frac{1}{\nu m}\sum_i \xi_i \\ &\quad + \sum_i \alpha_i(A_i + \rho - \xi_i) - \sum_i \lambda_i\xi_i, &(7.48)\end{aligned}$$

minimizing with respect to the primal variables, and the resulting objective function maximized with respect to the dual variables subject to non-negative values. Thus, obtaining the partial derivatives of the Lagrangian with respect to the primal variables and considering (7.16), we get:

$$\begin{aligned}0 = \frac{\partial\mathcal{L}}{\partial w_j} &= w_j - \sum_i \alpha_i\phi_j(x_i); \quad \forall j &(7.49)\\ 0 = \frac{\partial\mathcal{L}}{\partial\xi_i} &= \frac{1}{\nu m} - \alpha_i - \lambda_i; \quad \forall i &(7.50)\\ 0 = \frac{\partial\mathcal{L}}{\partial\rho} &= -1 + \sum_i \alpha_i. &(7.51)\end{aligned}$$

Replacing (7.49) to (7.51) into (7.48) we get

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j\phi_i(\mathbf{x}_i)\phi_j(\mathbf{x}_j) - \rho + \frac{1}{\nu m}\sum_i \xi_i \\
&\quad - \sum_{i,j}\alpha_i\alpha_j\phi_i(\mathbf{x}_i)\phi_j(\mathbf{x}_j) + \rho\sum_i \alpha_i \\
&\quad - \sum_i \alpha_i\xi_i + \sum_i \alpha_i\xi_i - \frac{1}{\nu m}\sum_i \xi_i \\
&= -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j\phi_i(\mathbf{x}_i)\phi_j(\mathbf{x}_j) \\
&= -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j). \qquad (7.52)
\end{aligned}$$

Thus, each of these k Problems can be solved by considering their duals:

$$\underset{\boldsymbol{\alpha}}{\text{Minimize}} \quad \frac{1}{2}\sum_{i,j}^{m_k}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j) \qquad (7.53)$$

subject to

$$\begin{aligned}
-\alpha_i &\leq 0 : \psi_i; \quad \forall i = 1,2,\ldots,m_k \qquad (7.54)\\
\alpha_i &\leq \frac{1}{\nu m} : \xi_i; \quad \forall i = 1,2,\ldots,m_k \qquad (7.55)\\
\sum_{i=1}^{m_k}\alpha_i &= 1 : \rho, \qquad (7.56)
\end{aligned}$$

where m_k is the number of training samples in partition k, $K(\mathbf{x}_i,\mathbf{x}_j) = \phi_i(\mathbf{x}_i)\phi_j(\mathbf{x}_j)$, and Equation (7.56) comes from Equation (7.50).

7.3 IMPLEMENTATION CONSIDERATIONS

The previous section presented the mathematical foundations of the proposed DOC-SVM method. In order to shed light on the implementation of the method, the pseudocode of the algorithm is shown next, where i goes through the data points, n_{out} is set to the desired number of outliers in the dataset, and k goes through the partitions. In order to implement the bi-level approach, a master-slave architecture is used in which every processor is a slave that learns the data of its partition and one of them acts also as the master responsible for determining the global outliers and moving data between partitions.

1. Randomly initialize u_i to value 0 for n_{out} data and 1 for the rest, such that $\sum_i u_i = n_{out}$;

2. Initialize η_i^k to value 1 if the sample point i is assigned to partition k and 0 otherwise;

3. Repeat while convergence is not attained;

   (a) At each partition: solve the problem (7.45)-(7.47) with fixed variables u and η;

   (b) Outlier selection: Update u

      i. Broadcast to the master the n_{out} largest ξ's from every partition;

      ii. Master: Determine n_{out} outliers that correspond to the n_{out} data points with largest value of ξ. In case of tie, random selection is a valid solution. Inform the partitions about the selection;

      iii. At each partition update u accordingly;

   (c) Partition assignment: Update η

      i. Broadcast A_i^k and ρ_k among partitions;

      ii. At each partition, for every data point i, calculate

$$k = \arg\min_k (A_i^k + \rho_k), k = 1 \dots K$$

      and move data i to partition k, if needed, updating η accordingly;

Convergence is attained when variables u and η remain unchanged between iterations. Experimentation showed that the initialization of η is keystone to yield rapid convergence. Random initialization lead to uniform distribution across partitions over the input space. Thus, the redistribution rule $k = \arg\min_k(A_i^k + \rho_k)$ may hop data points repeatedly between partitions taking a long time to converge with no further improvement with the subsequent iterations. In an attempt to overcome this difficulty, the initialization of η could be directed by clustering the input space into K pieces. In the experimental part of this research, the K-means method has been shown

to be effective in producing good values for η, where K is set to the number of partitions. It is also worth noting that the k partitions can be distributed over k processors, that can or cannot belong to the same machine. In this sense, in what follows, the term *partition* could be interpreted as *processor*. Finally, notice that n_{out} is a lower bound on the number of global outliers: when ν tends to zero, the number of total outliers tends to n_{out}, otherwise, the bound grows as a function of ν.

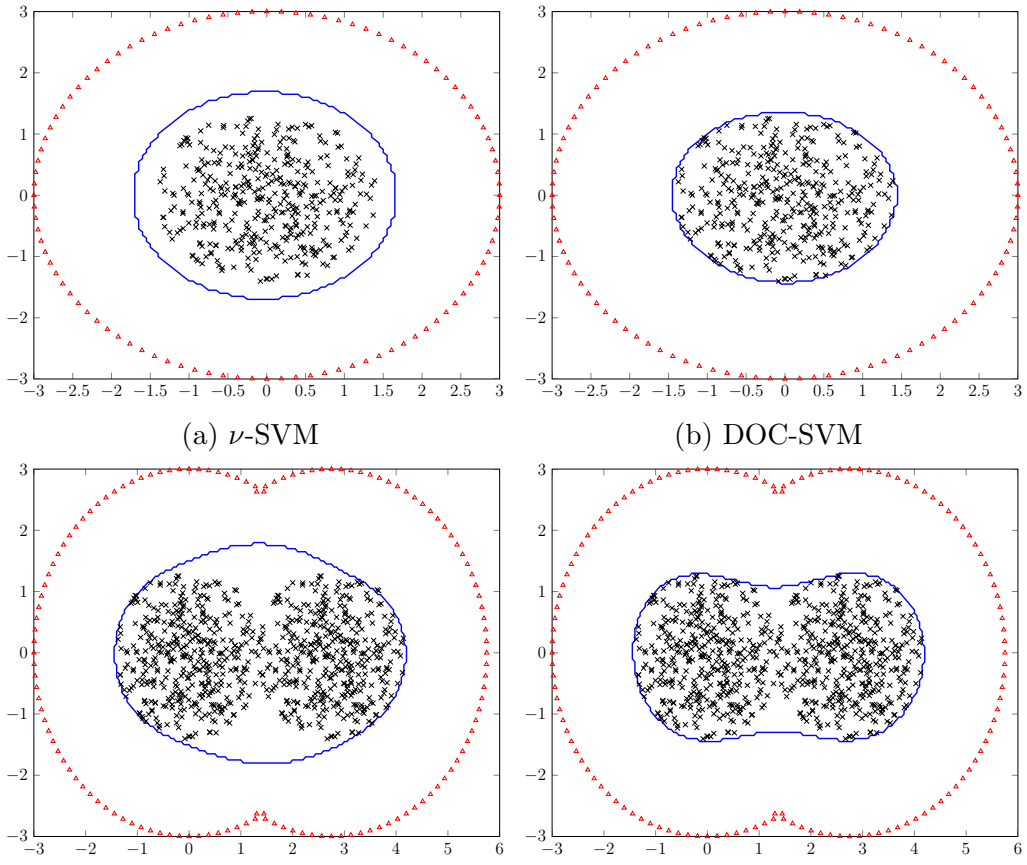
7.4 EXPERIMENTAL RESULTS

In this section, we present some results to investigate the performance of the DOC-SVM. Firstly, we will present some results using artificial data in order to illustrate the influence, on the decision region, of the position of the real outliers, the number of partitions and the value of the n_{out} parameter, as well as, to study the evolution of the training time. Secondly, the proposed method is compared to other known approaches to one-class classification using real datasets.

7.4.1 EXPERIMENTS USING ARTIFICIAL DATA: ANALYZING THE PERFORMANCE OF DOC-SVM

In this first experiment, we compare the decision regions produced by the ν-SVM and DOC-SVM and the influence of the position of the outliers on these regions. With this aim, we have generated two simple datasets, as illustrated in Figure 7.1. The figures in the top row correspond to a single cloud of normal data points uniformly distributed with center $(0, 0)$ and a single ring of outliers separated from the border of the cloud by 1.5 units. The figures in the bottom row correspond to two overlapping clouds of normal data points uniformly distributed with centers $(0, 0)$ and $(2.5, 0)$, and a single ring of outliers separated from the border of the two clouds by 1.5 units. Results were obtained obtained using a Gaussian Kernel and $\gamma = 0.2$. As can be observed, DOC-SVM fits the shape of the clouds much more accurately than the ν-SVM. According to the formulation, this is based on the fact that the decision region produced by the proposed method is unaffected by the position of the outliers as the slack variables ξ_i do not add to the cost function when variables u_i take value zero for outliers (see Equation (7.7)).

In the second experiment, we analyze the influence of the number of partitions on the decision regions produced by DOC-SVM by considering architectures of 1, 2, and 4 partitions. Also, we analyze what happens when the number of outliers n_{out} is overestimated. With this aim we have run the method over several clouds of data points, using again a Gaussian Kernel and



(a) ν-SVM (b) DOC-SVM

(c) ν-SVM (d) DOC-SVM

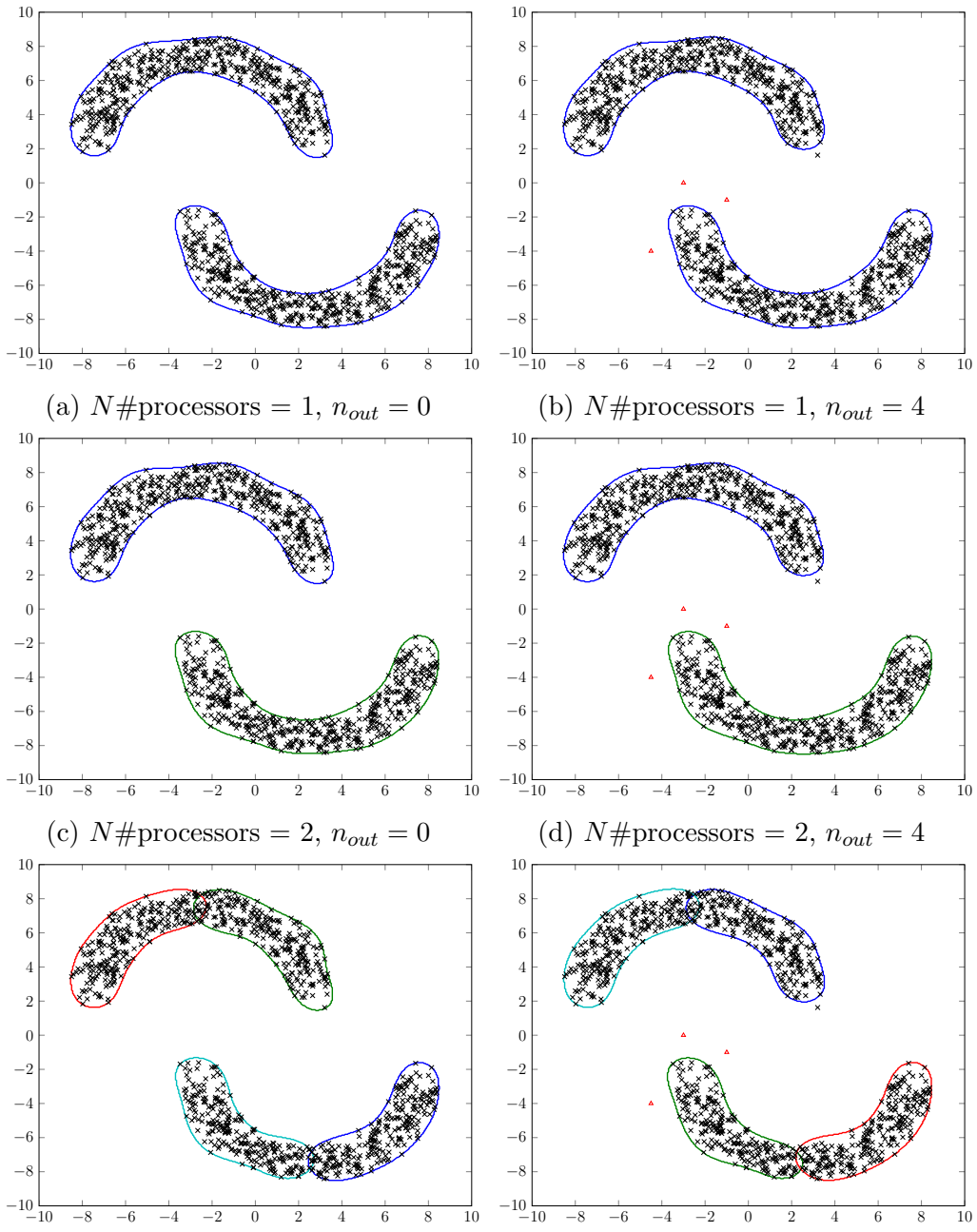
Figure 7.1: Decision region (*solid line*) produced by the ν-SVM and DOC-SVM for a cloud of normal data points (*crosses*) and a ring of data outliers (*triangles*).

$\gamma = 0.2$. Figures 7.2—7.5 shows a representative example of the obtained results. The figures in the left column show clouds of data samples, in which every data sample is considered as normal and DOC-SVM was trained to cover every point ($n_{out} = 0$). The figures in the right column show the same cloud of data points plus three artificially generated outliers. In this case, the proposed method was trained to leave four data points out of the decision region ($n_{out} = 4$). It is expected that it will cover every sample but the three artificial outliers and the *furthest* data point from the cloud.

Regarding the effect of the number of partitions on the output, graphically illustrated, the decision regions are displayed with different enclosing balls for different partitions. One can see how the method splits up the inputs space in several subspaces –equal to the number of partitions– and adjusts the decision region to these subspaces. Note that the datasets are disjoint between partitions even when the decision regions may be slightly overlapped in the frontiers because of the tails of the Gaussian kernel functions coalesce. Also it can be observed that relatively equal-size datasets are assigned for each partition. This result relies on a good initialization of the method since no constraints are imposed on the number of data points per partition.

In addition, the quality of the solution when the number of partitions increases needs some discussion. On the left column of Figures 7.2—7.5, the clouds of data points are accurately covered by DOC-SVM regardless of the number of partitions. On the right column of Figures 7.2)—(7.5, the three artificially generated outliers are marked as such in every configuration. As the number of outliers was overestimated, the fourth outlier will be some point of the data cloud. For example, Figures 7.2—7.4 show that the same data point is marked regardless of the number of partitions. This is an expected result that occurred in our experiments whenever this data point seem to be *the* outlier in the cloud. Conversely, Figure 7.5 shows a quite packed set of data points wherein the fourth outlier will depend on the initialization of the method and the frontiers between partitions.

Finally, the training time performance of the proposed method is analyzed over the four artificial datasets. First, the evolution of the average training time against the number of training samples was calculated when 1, 2, 4 and 8 processors (partitions) are used. Results are shown in Figure 7.6a. As expected, the parallel time decreases with the increase of the number of processors. Also, the speedup factor S_p was calculated, defined by $S_p = R_1/R_p$, where R_1 and R_p are the training times of DOC-SVM on a single processor (partition) and on p processors (partitions), respectively. This measures how much the training time is faster using p processors rather than one. Figure 7.6b shows the speedup factor against the number of processors (1, 2, 4 and 8) for a training set from $8,192$ to $524,288$ samples. This

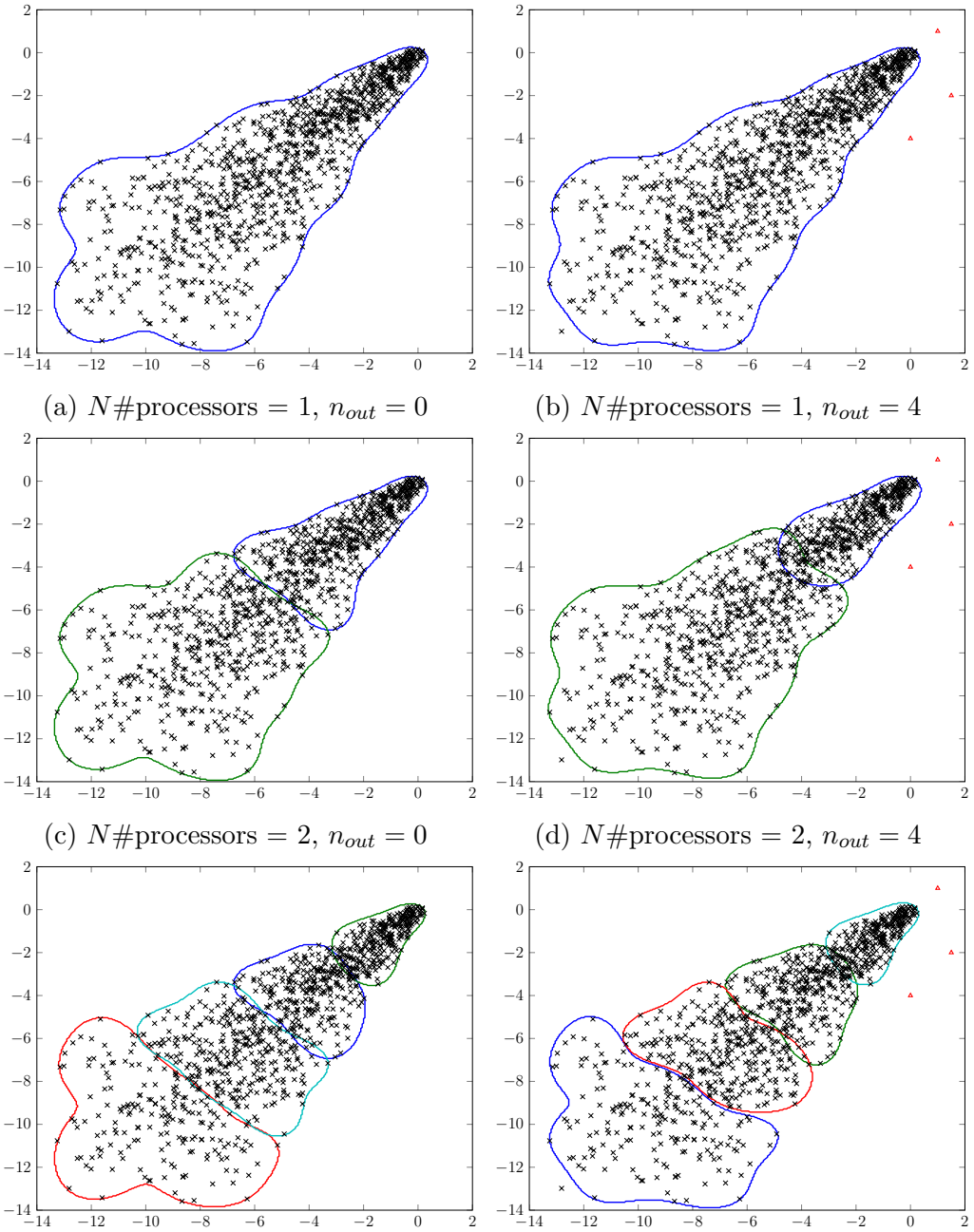


(a) $N\#\text{processors} = 1$, $n_{out} = 0$ (b) $N\#\text{processors} = 1$, $n_{out} = 4$

(c) $N\#\text{processors} = 2$, $n_{out} = 0$ (d) $N\#\text{processors} = 2$, $n_{out} = 4$

(e) $N\#\text{processors} = 4$, $n_{out} = 0$ (f) $N\#\text{processors} = 4$, $n_{out} = 4$

Figure 7.2: *Banana*. Decision regions (*solid curves* –every closed curve corresponds to one partition) produced by DOC-SVM for a cloud of normal data samples (*crosses*) and three artificial outliers (*triangles*).



(a) $N\#\text{processors} = 1$, $n_{out} = 0$

(b) $N\#\text{processors} = 1$, $n_{out} = 4$

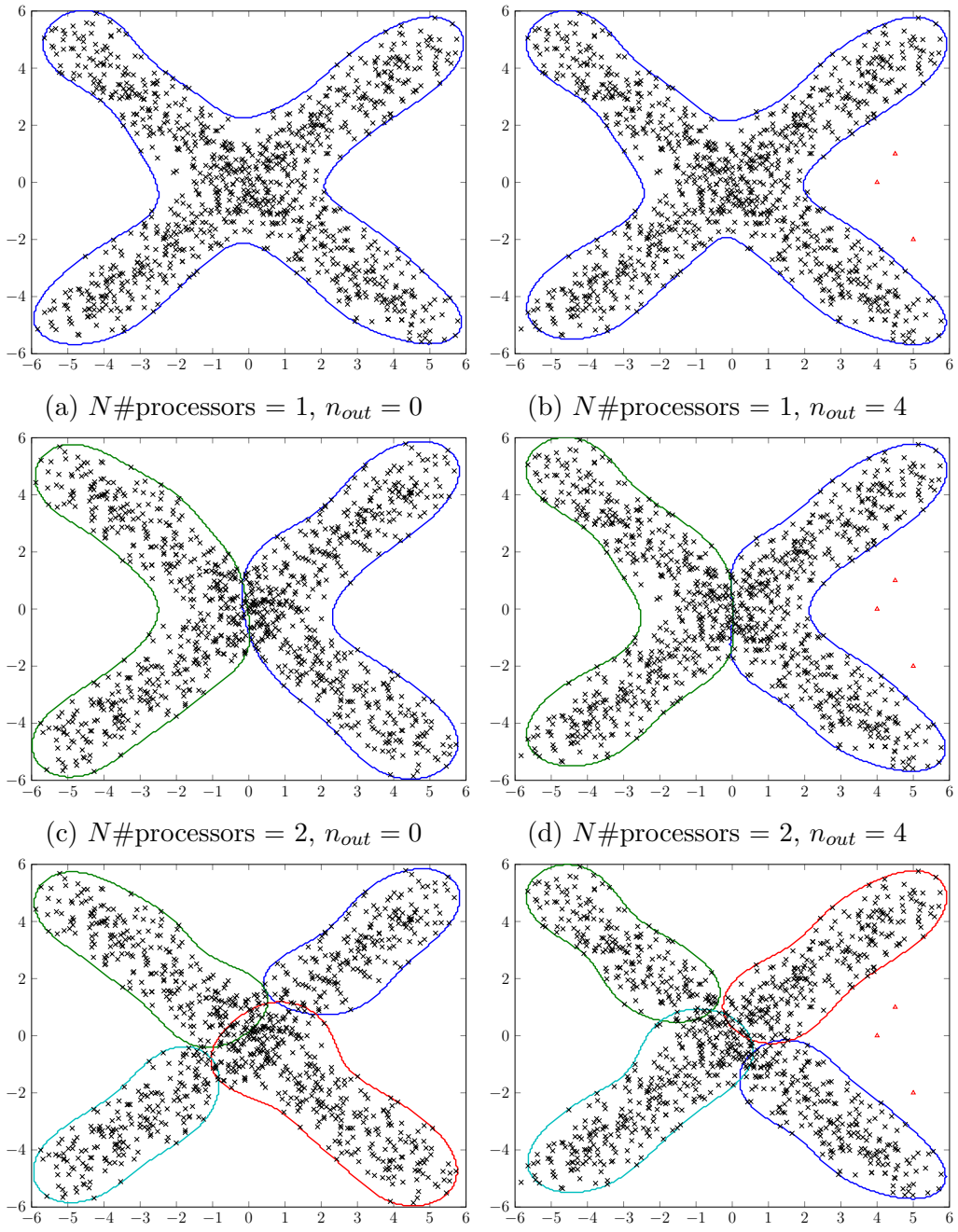
(c) $N\#\text{processors} = 2$, $n_{out} = 0$

(d) $N\#\text{processors} = 2$, $n_{out} = 4$

(e) $N\#\text{processors} = 4$, $n_{out} = 0$

(f) $N\#\text{processors} = 4$, $n_{out} = 4$

Figure 7.3: *Comet*. Decision regions (*solid curves* –every closed curve corresponds to one partition) produced by DOC-SVM for a cloud of normal data samples (*crosses*) and three artificial outliers (*triangles*).



(a) $N\#\text{processors} = 1$, $n_{out} = 0$

(b) $N\#\text{processors} = 1$, $n_{out} = 4$

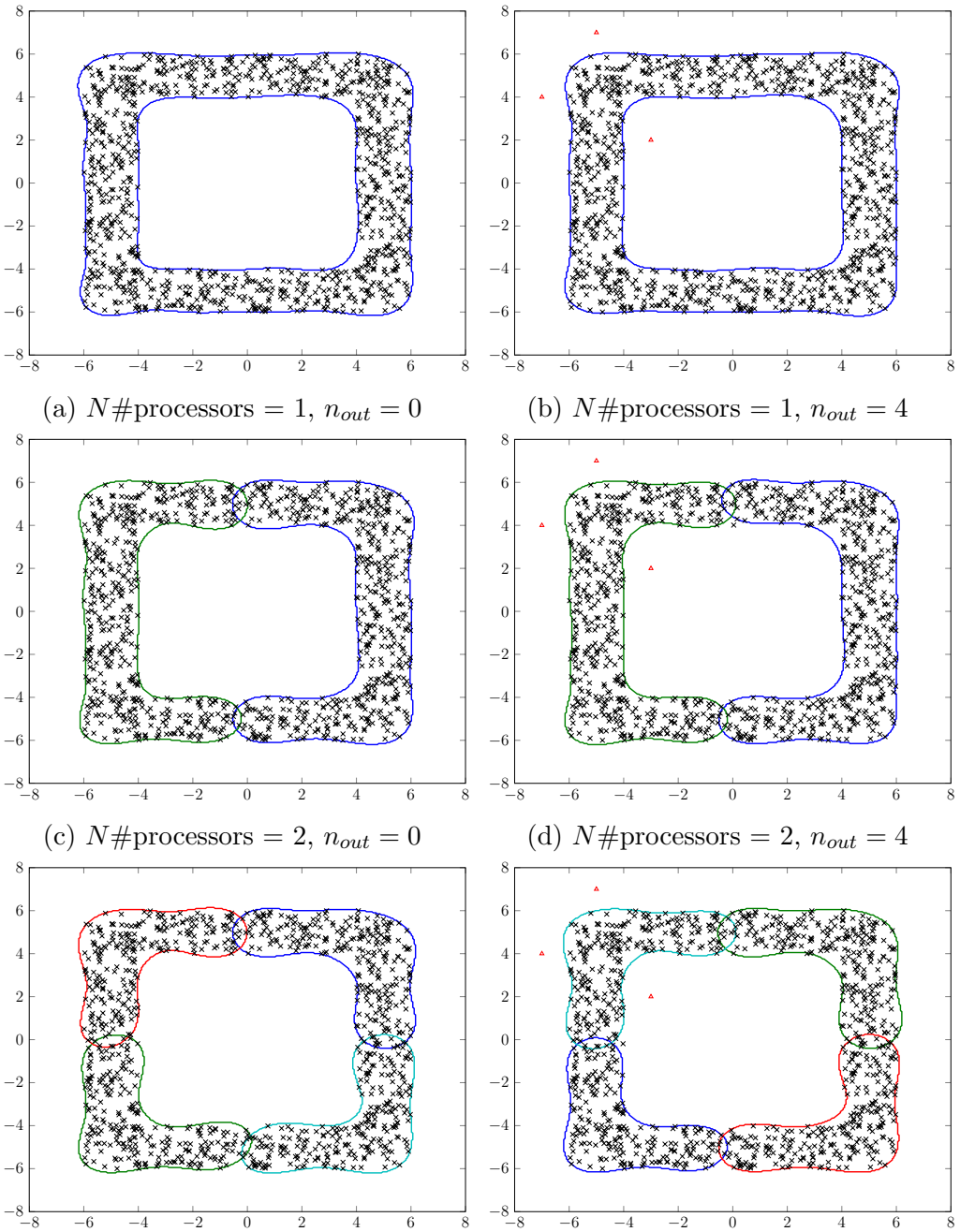
(c) $N\#\text{processors} = 2$, $n_{out} = 0$

(d) $N\#\text{processors} = 2$, $n_{out} = 4$

(e) $N\#\text{processors} = 4$, $n_{out} = 0$

(f) $N\#\text{processors} = 4$, $n_{out} = 4$

Figure 7.4: *Ex.* Decision regions (*solid curves* –every closed curve corresponds to one partition) produced by DOC-SVM for a cloud of normal data samples (*crosses*) and three artificial outliers (*triangles*).



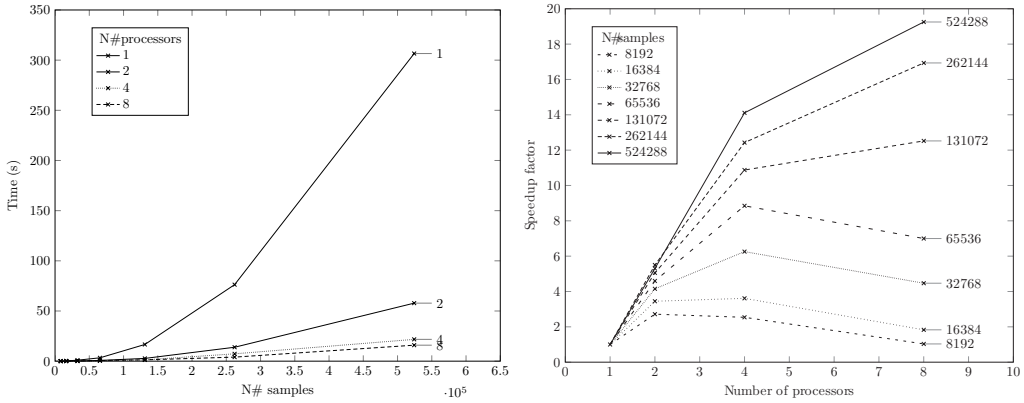
(a) $N\#\text{processors} = 1$, $n_{out} = 0$ (b) $N\#\text{processors} = 1$, $n_{out} = 4$

(c) $N\#\text{processors} = 2$, $n_{out} = 0$ (d) $N\#\text{processors} = 2$, $n_{out} = 4$

(e) $N\#\text{processors} = 4$, $n_{out} = 0$ (f) $N\#\text{processors} = 4$, $n_{out} = 4$

Figure 7.5: *Square*. Decision regions (*solid curves* –every closed curve corresponds to one partition) produced by DOC-SVM for a cloud of normal data samples (*crosses*) and three artificial outliers (*triangles*).

form of display is complementary to the more obvious plot of training time versus number of processors, as it makes straightforward to see the largest number of processors that has a positive impact on the training time, for a fixed number of samples. We can see from Figure 7.6b that having more that 4 processors reduces the speedup factor for the four smallest training set sizes (up to $65,536$ samples) –i.e. the proposed method is slower using 8 processors rather than 4 because of communication and other overheads. However, using a larger number of processors (up to 8 in these experiments) is beneficial for the three largest training sizes.



(a) Training time (s). (b) Speedup factor.

Figure 7.6: Performance measures of the proposed method.

7.4.2 EXPERIMENTS WITH BENCHMARKS: A COMPARATIVE STUDY

In this section, the aim of the experiments was to assess the classification quality of DOC-SVM in comparison with the base classifier ν-SVM and also with other 18 known approaches for one-class classification. For the ν-SVM the LIBSVM implementation (Chang & Lin, 2011) was used whilst for the rest of the methods the implementation included in the DDtools toolbox (D. Tax, 2014) was employed. To assess the performance of the methods the mean Area Under the Curve (AUC) for the test sets was calculated. In all the kernel methods, a radial basis was used as the kernel function with a γ parameter. The optimal n_{out} parameter was selected as the one that provided the best result of the AUC. For DOC-SVM, 4 partitions were used.

This comparative study was done using real world data sets from the UCI repository (Frank & Asuncion, 2010) that contains outliers artificially added. These data sets were created by the Pattern Recognition Laboratory of the

Name	ID	Target objects	Outlier objects	Features
Wisconsin Breast Cancer	#505	458	241	9
Ionosphere	#588	225	126	34
Liver-disorders	#591	200	145	6
Waveform Data Generator	#598	300	600	21
Dataset Vehicle saab	#613	217	629	18
Haberman's Survival Data > 5yr	#616	225	81	3

Table 7.1: Data sets employed in the experimental comparative study.

	Dataset ID					
	#505 ($\gamma = 0.1$, $n_{out} = 70$)	*#588* ($\gamma = 0.5$) $n_{out} = 6$)	*#591* ($\gamma = 0.001$) $n_{out} = 19$)	*#598* ($\gamma = 0.01$) $n_{out} = 220$)	*#613* ($\gamma = 0.001$) $n_{out} = 5$)	*#616* ($\gamma = 0.1$) $n_{out} = 24$)
ν-SVM	99.5 (0.3)	96.4 (1.4)	54.2 (7.3)	79.9 (5.2)	62.4 (5.7)	68.7 (4.6)
DOC-SVM	99.6 (0.4)	95.0 (3.7)	55.3 (6.2)	84.3 (2.9)	60.8 (5.6)	67.5 (5.2)

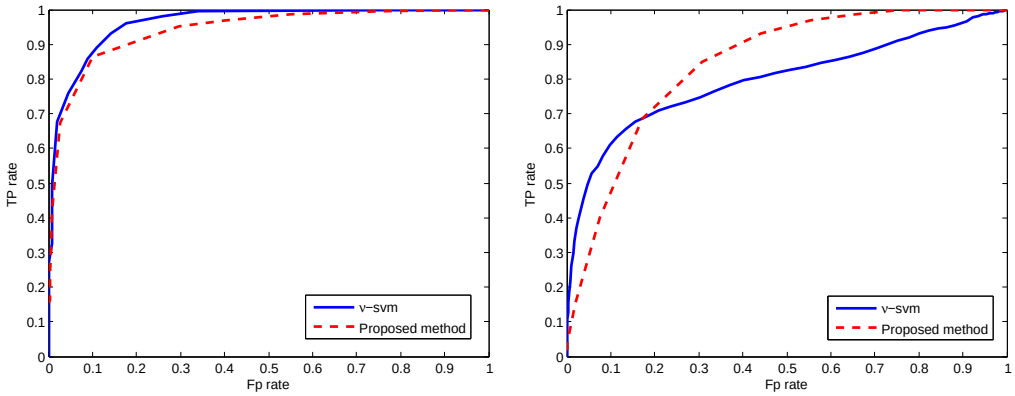
Table 7.2: Mean AUC (x100) and the standard deviation (in brackets) for the test sets in 50 random simulations.

Delft University of Technology[1] and Table 7.1 contains their main characteristics. Every method was applied to these data sets using 100 random simulations. In each simulation a random training set was built composed of 90% of the normal samples and 10% of the outliers. The rest of the data was used as test set.

Table 7.2 contains the results obtained by DOC-SVM and the ν-SVM used as the base classifier. Applying a statistical test (t-test) with a 5% of significance level it was determined that only for data sets #588 and #598 the results of both classifiers are different. As an example, the ROC curves for two data sets are shown in Figure 7.7. As can be seen, the curves are slightly different for both models confirming that their behavior is not just the same.

Moreover, in order to check how DOC-SVM copes with cases where each partition has all possible data (global distribution), or nearly all, we accomplished a new experiment for the data sets #588 and #598. In this new experiment two scenarios were compared: the first one consisting of replicating the global data set in each node, and the second one in which the data set is replicated, as many times as nodes, but performing a random assignment of the data in each partition. For this experiment we used the same parameter configuration as in the experiments in Table 7.2 but setting, arbitrarily, the number of outliers (n_{out}) to 30 (data set #588) and 280 (data set #598). Nevertheless, the optimal value of n_{out} is not relevant in this case,

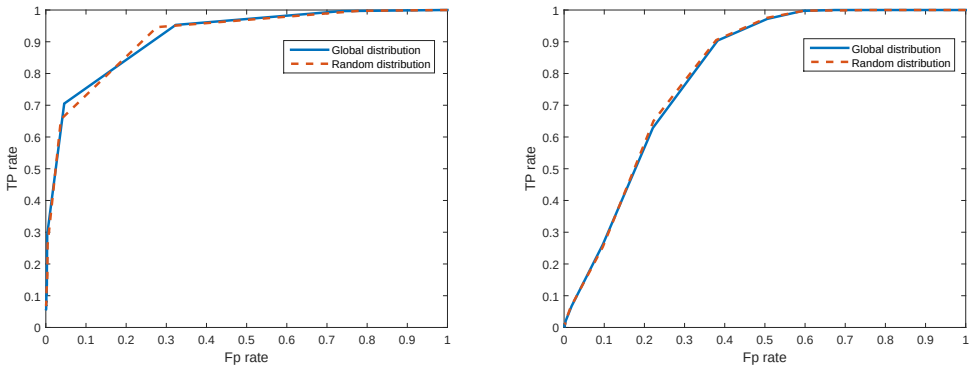
[1]Data available at: http://homepage.tudelft.nl/n9d04/occ/index.html



(a) Dataset #588. (b) Dataset #598.

Figure 7.7: ROC curves for the datasets #588 and #598.

as the aim is to compare if the performance of the method is similar in both scenarios (global and random distribution). Figure 7.8 contains the obtained ROC curves that demonstrate that the results are almost equivalent for both scenarios.



(a) Dataset #588 (b) Dataset #598

Figure 7.8: ROC curves for a random or a global distribution of the dataset.

In addition, Table 7.3 contains the comparative study with 18 one-class methods. The table shows the mean AUC values for each test set and, in brackets, the ranking (1 corresponding to the best result). Analyzing the table it can be observed that the proposed algorithm obtains, in all the cases, better results than the mean classifier and is on the top 5 classifiers in most of the cases. Also, summing the rankings the proposed method obtains the first position with a sum equal to 34, the second place being for the Mixture

of Gaussians data description ($sum = 42$).

Finally, we carried out an experiment using a large data set: MNIST (LeCun, Bottou, Bengio, & Haffner, 1998). This is a well-known benchmark problem for handwritten digits classification. The data set is formed by 70,000 images of 28×28 pixels. We transformed this data set to a one class problem using as the outlier class the data for the digit 0 and as the target class the data for the other digits (digits 1 to 9). In this case 56,925 data points were randomly selected as training set (0.25% of outliers), using data from the target and the outlier class, and 13,075 data points were chosen as the test set (51% of outliers). In this case the experiment was performed using 16 nodes and $\gamma = 0.01$. The number of outliers n_{out} was varied using the values in the set $\{200, 1000, 4000, 6000\}$. Figure 7.9 contains the results (ROC curves) for the test data set. As can be observed the performance is improved as the value of the n_{out} parameter is increased.

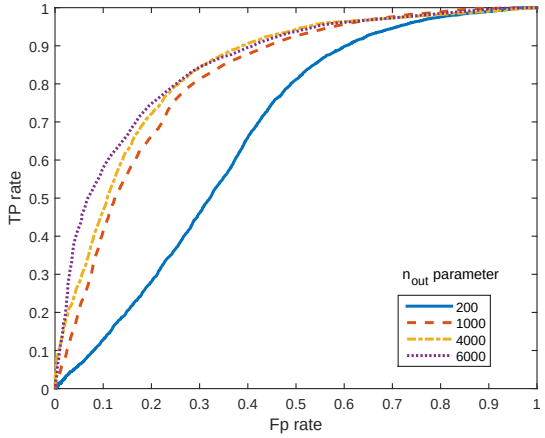


Figure 7.9: ROC curves for the test set of the MNIST problem varying the value of the $n_{out} parameter$.

One-class Classifier	Data set					
	#505	*#588*	*#591*	*#598*	*#613*	*#616*
Multivariate Gaussian probability density function	98,5 (14)	96,3 (5)	50,7 (9)	84,3 (2)	79,4 (2)	59,2 (11)
Min. Covar. Determinant Robust Gaussian data des.	NaN (19)	NaN (19)	51,8 (7)	84,3 (3)	86,1 (1)	69,5 (1)
Mixture of Gaussians data description	98,3 (15)	96,6 (3)	50,5 (10)	83,0 (6)	74,6 (3)	65,2 (5)
Naive Parzen density data description	98,7 (11)	93,2 (16)	48,4 (18)	83,8 (5)	68,4 (5)	64,8 (7)
Parzen density data description	99,2 (3)	95,6 (8)	49,6 (17)	82,5 (7)	60,6 (13)	66,5 (4)
k-means data description	99,1 (7)	97,0 (2)	50,0 (15)	82,5 (8)	64,8 (9)	64,9 (6)
Nearest neighbor based data description	99,2 (4)	95,5 (9)	50,1 (12)	81,3 (14)	49,2 (16)	50,8 (15)
k-nearest neighbor data description	99,2 (5)	95,5 (10)	50,1 (13)	81,3 (15)	49,2 (17)	50,8 (16)
Distance k-nearest neighbor data description	86,7 (18)	86,4 (17)	54,4 (3)	76,4 (17)	62,1 (10)	47,2 (19)
Principal component data description	95,1 (17)	97,8 (1)	55,9 (1)	74,2 (18)	68,7 (4)	53,9 (12)
Self-Organizing Map data description	99,1 (8)	94,7 (14)	54,3 (5)	82,4 (9)	61,6 (11)	62,8 (9)
Auto-encoder neural network data description	96,4 (16)	95,2 (12)	52,5 (6)	81,2 (16)	66,2 (7)	59,3 (10)
Minimum spanning tree data description	99,1 (9)	95,5 (11)	49,9 (16)	81,7 (11)	66,5 (6)	49,0 (17)
L_p-ball data description	98,7 (12)	65,8 (18)	54,4 (4)	81,8 (10)	55,7 (14)	51,0 (14)
Distance k-center data description	98,7 (13)	94,0 (15)	51,5 (8)	62,5 (19)	54,9 (15)	47,3 (18)
Support vector data description	99,0 (10)	96,6 (4)	0,7 (19)	81,4 (13)	0,0 (19)	53,4 (13)
Minimax probability machine data description	99,2 (6)	96,0 (7)	50,1 (14)	81,5 (12)	65,2 (8)	66,9 (3)
Distance-linear programming data description	99,3 (2)	96,2 (6)	50,3 (11)	86,7 (1)	49,0 (18)	63,0 (8)
Mean of the previous classifiers	97,9	93,4	48,6	80,7	60,1	58,1
DOC-SVM	99,6 (1)	95,0 (13)	55,3 (2)	84,3 (4)	60,8 (12)	67,5 (2)

Table 7.3: Comparative study using the mean AUC (x100) for the test set. The number in brackets is the ranking for each data set.

Part III

When Distribution is Part of the Semantics

Chapter 8

Distribution of the Distributions

In this chapter, the problem of learning in a distributed machine learning setting is made more challenging. Thus, given the fragments $\mathcal{D}_1, \ldots, \mathcal{D}_P$ of a entire dataset $\mathcal{D}$ distributed across the sites $1, \ldots, P$, a performance criterion $\mathcal{E}$, and a set of constraints $\mathcal{Z}$, the learning algorithm $\mathcal{L}_d$ constructs a hypothesis $\hbar$ that optimizes $\mathcal{E}$ and meets $\mathcal{Z}$. Clearly, the problem of learning in a standard setting where $P = 1$, and the problem of learning in a distributed setting with no constraints ($\mathcal{Z} = \emptyset$) are special cases of this setting.

8.1 DISTRIBUTION MATTERS

In general terms, distributed learning algorithms aim to infer a global learner that approximates the results one would get from a single, joint data source. Note that this approach assumes that there is a single, global model that could be induced from the distributed data sites. Under this view, distribution is treated exclusively as a technical issue. However, there are deeper implications (Wirth et al., 2001). Real-world distributed data sets almost always present quite strong differences between their partitions, e.g. buying patterns in different supermarkets from different countries. Yet in spite of its importance, there has been no fully considered in distributed learning settings. In (Zaki, 2000), Mohammed J. Zaki puts the focus on today's outstanding search issues and open problems for developing the next generation of distributed machine learning algorithms. Two of them are of particular interest to our purposes in this chapter.

DATA LOCATION. Big datasets are usually logically and physically distributed. They may even belong to different organizations that want to

share knowledge but do not want to share raw data. The datasets may also have heterogeneous schemes. In any case, it seems like having all the data in a single site is no longer the model.

DATA SKEW. One of the problems adversely affecting load balancing in parallel mining algorithms is sensitivity to data skew. Most methods partition the database horizontally in equal-sized blocks. However, assuming rule-based systems for argument's sake, the number of rules generated from each block can be heavily skewed, i.e., while one block may contribute many, the other may have very few rules, implying that the processor responsible for the latter block will be idle most of the time. Randomizing the blocks is one solution, but it is still not adequate, given the dynamic and interactive nature of mining. The effect of skewness on different algorithms needs to be further studied. For example, in (Cheung & Xiao, 1999), data skew is defined in terms of its effect on parallel mining of associations. In that paper, the data skewness of a partitioned database is high if the supports of most large itemsets –groups of items, frequently appearing together in transactions– are clustered in a few partitions and it is low if the supports of the most large itemsets are distributed evenly across the processors. On the other hand, in (Walton, Dale, & Jenevein, 1991), data skew primarily refers to a non-uniform distribution in a database and its effect on parallel joins. In that research, the authors classify the effects of skewed data distribution on a distributed execution, distinguishing intrinsic skew from partition skew. Intrinsic skew is skew inherent in the dataset, e.g., there are more citizens in Paris than in Waterloo (Liu & Özsu, 2009). Partition skew occurs on distributed implementations when the workload is not evenly distributed among nodes, even when the input data is uniformly distributed. In particular, *tuple placement skew* is the skew introduced when the initial distribution of data examples varies between partitions, e.g., tuples may be partitioned by clustering attribute in user specified ranges. In this thesis, the term *data skew* will be used in the related sense of *dataset shift between partitions*. But first, what is meant by *dataset shift*?

8.2 LESSONS FROM DATASET SHIFT IN STANDARD MACHINE LEARNING

In the real world, the conditions in which one uses systems may differ from the conditions in which they were developed. Given some data, and some modeling framework, a model can be learned. This model can be used for making predictions $p(y|\mathbf{x})$ for some output y given some new input $\mathbf{x}$. How-

ever, what if there is a possibility that something may have changed between training and test situations? Would this model be considered reliable?

The term *dataset shift* is defined as cases where the joint distribution of inputs and outputs differs between training and test data (Storkey, 2009). That is, when

$$p_{tr}(Y, X) \neq p_{tst}(Y, X) \tag{8.1}$$

The problem of dataset shift is closely related to another area of study known by various terms such as *transfer learning* or *inductive transfer*. Transfer learning deals with the general problem of how to transfer information from a variety of previous different environments to help with learning, inference, and prediction in a new environment. Dataset shift is more specific: it deals with the business of relating information in (usually) two closely related environments to help with the prediction in one given the data in the other (Quionero-Candela, Sugiyama, Schwaighofer, & Lawrence, 2009). Faced with the problem of dataset shift one need to know if it possible to characterize the types of changes that occur from training to test situations. We present three kinds of dataset shift that may appear in a classification problem (Moreno-Torres, Raeder, Alaiz-Rodríguez, Chawla, & Herrera, 2012).

COVARIATE SHIFT. One simple assumption one can make about the connection between the distributions of the training and the test data is that given the same observation $X = x$, the conditional distributions of Y are the same in the two datasets. However, the marginal distributions of X may be different in the training and the test data. This difference is called *covariate shift* (Shimodaira, 2000). Formally, it is defined as the case where

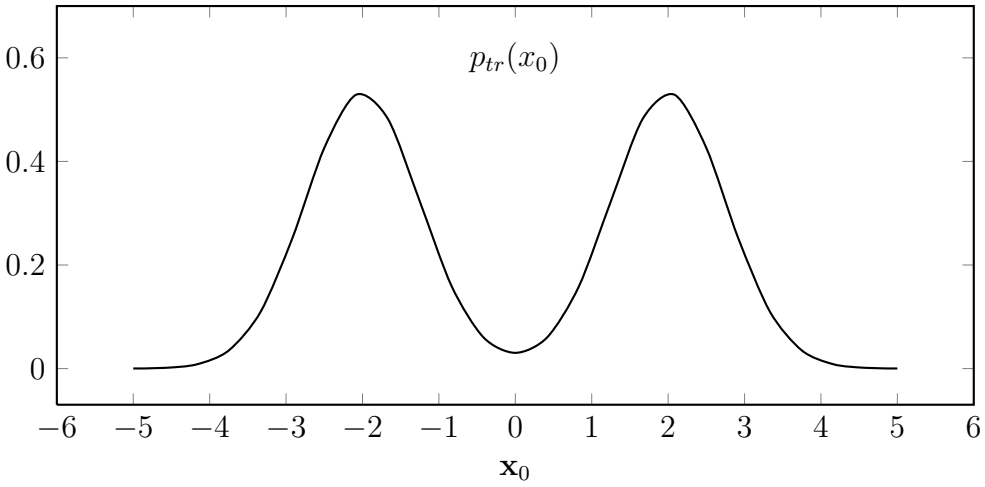
$$p_{tr}(Y|X = \mathbf{x}) = p_{tst}(Y|X = \mathbf{x}), \forall \mathbf{x} \in \mathcal{X} \tag{8.2}$$

$$p_{tr}(X) \neq p_{tst}(X) \tag{8.3}$$

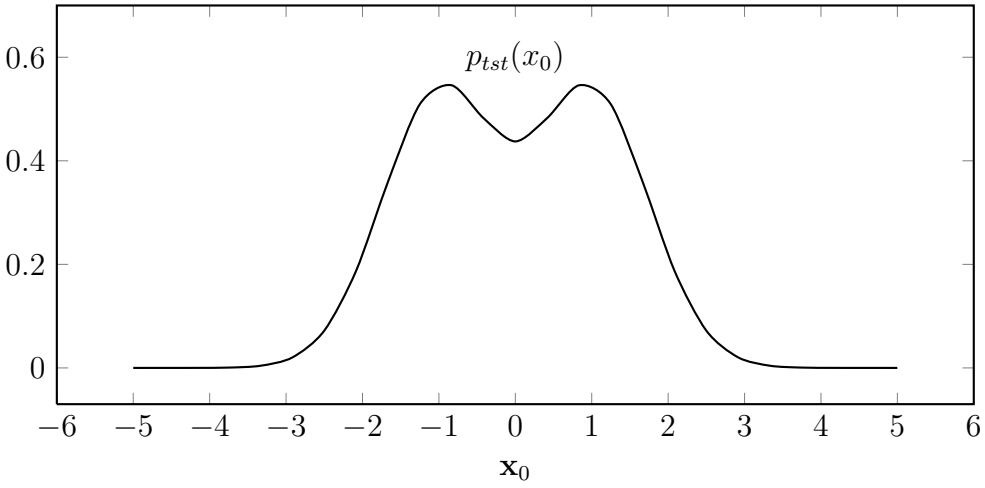
Let us take as example of covariate shift a problem where there is one covariate (feature) $\mathbf{x}_0$ and a target y. The training data distribution $p_{tr}(\mathbf{x}_0)$ by the union of two Gaussian distributions $\mathcal{N}(-2, 0.75)$ and $\mathcal{N}(2, 0.75)$. The conditional distribution is defined as

$$p_{tr}(y|x_0) = \frac{1}{1 + \exp(-\frac{x_0}{0.2})}$$

Consider now that the conditional distribution in the test data remains unchanged $p_{tst}(y|\mathbf{x}_0) = p_{tr}(y|x_0)$, but the Gaussian distributions that compose the marginal distributions of X are $\mathcal{N}(-1, 0.75)$ and $\mathcal{N}(1, 0.75)$. Figure 8.1 illustrates this simple example of covariate shift.



(a) Training data.



(b) Test data.

Figure 8.1: Example of covariate shift.

PRIOR PROBABILITY SHIFT. Another assumption is that given the same class label, the conditional distributions of X are the same in the two data sets. However, the class distributions may be different in the training and the test domains. This difference is referred to as *prior probability shift* (Jiang, 2008). Formally, it is defined as the case where

$$\begin{aligned} p_{tr}(X|Y=y) = p_{tst}(X|Y=y), \forall y \in \mathcal{Y} \\ p_{tr}(Y) \neq p_{tst}(Y) \end{aligned} \tag{8.4}$$

As an example, assume one has a problem with one feature x_0 and a target y that may take the class values $y = 0$ and $y = 1$. In the training data, $p_{tr}(y = 0) = p_{tr}(y = 1) = 0.5$ and $p_{tr}(x_0|y)$ is defined as

$$x_0 = \begin{cases} \mathcal{N}(-2, 0.75) & \text{if } y = 1 \\ \mathcal{N}(2, 0.75) & \text{otherwise} \end{cases}$$

Consider now that in the test data, $p_{tst}(\mathbf{x}_0|y = 0)$ and $p_{tst}(\mathbf{x}_0|y = 1)$ remain unchanged, but the class prior probabilities vary, taking the values $p_{tst}(y = 0) = 0.25$ and $p_{tst}(y = 1) = 0.75$. Figure 8.2 illustrates this simple example of prior probability shift.

CONCEPT DRIFT. It happens when the relationship between the input features and output classes changes. Formally, it is defined as

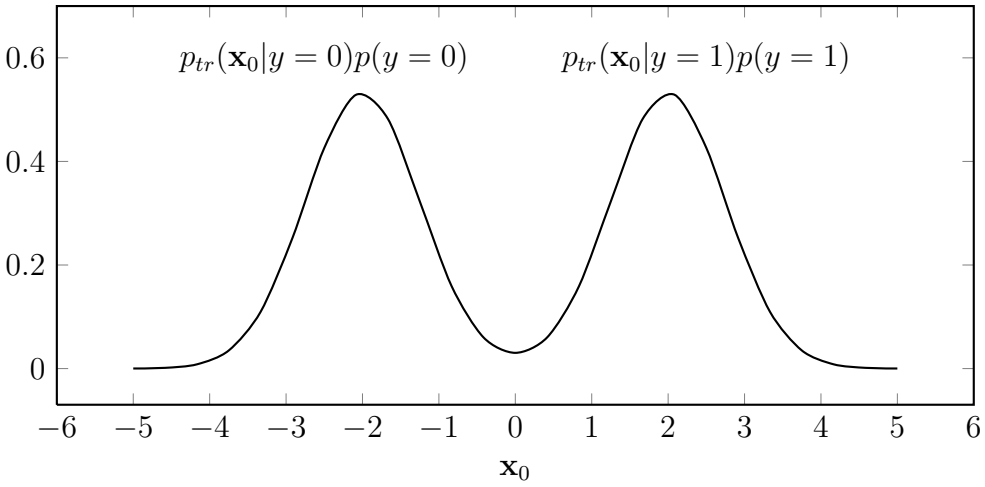
$$\begin{aligned} p_{tr}(X = \mathbf{x}) = p_{tst}(X = \mathbf{x}), \forall \mathbf{x} \in \mathcal{X} \\ p_{tr}(Y|X) \neq p_{tst}(Y|X) \end{aligned} \tag{8.5}$$

As an example, consider the problem already used to illustrate the covariate shift problem. If a concept drift appears, the test set distribution $p_{tst}(\mathbf{x}_0)$ remains constant, but $p_{tst}(y|x_0)$ changes (see Figure 8.3), for instance

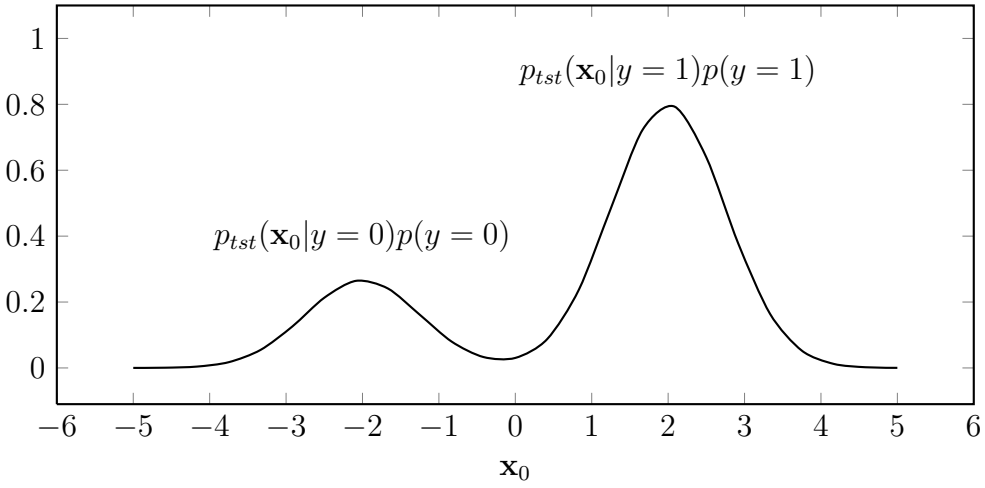
$$p_{tr}(y|x_0) = \frac{1}{\left(1 + \exp(-\frac{2+x_0}{0.2})\right)\left(1 + \exp(-\frac{2-x_0}{0.2})\right)}$$

8.3 DATA SKEW IN DISTRIBUTED MACHINE LEARNING

The three types of dataset shift in standard machine learning considered in this thesis (covariate shift, prior probability shift, and concept drift) are easily amended to make them useful in a distributed setting. In distributed learning, we define training and test datasets to be local and remote datasets,

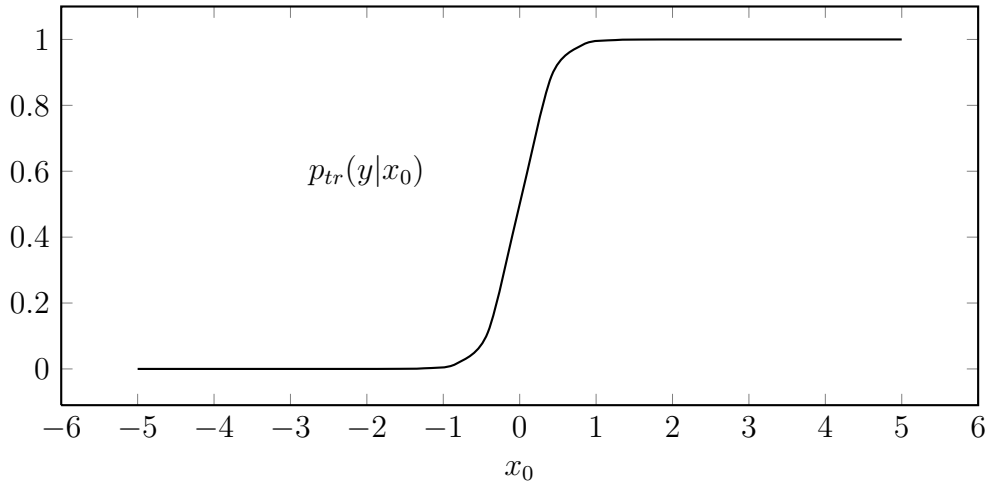


(a) Training data.

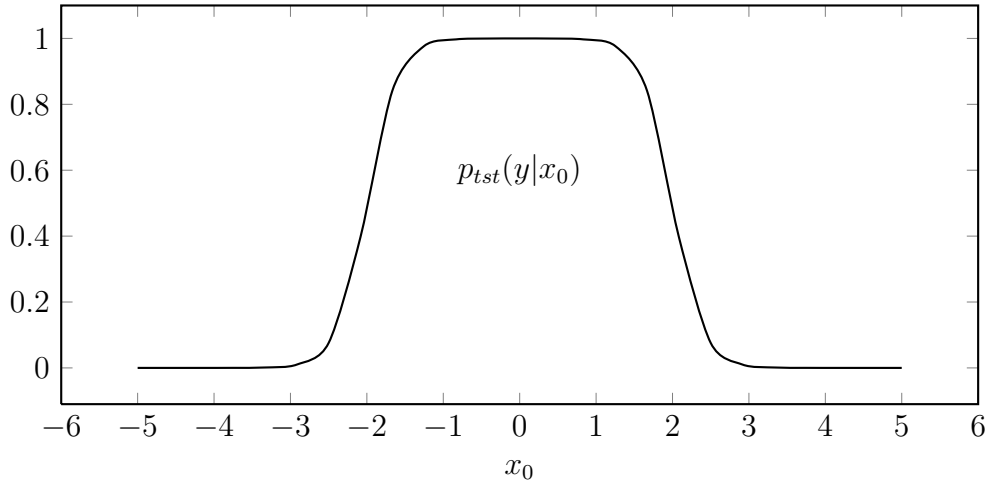


(b) Test data.

Figure 8.2: Example of prior probability shift.



(a) Training data.



(b) Test data.

Figure 8.3: Example of concept drift.

respectively. From the point of view of a processor, it will maintain its own local, source data whilst the remainder processors will be seen to maintain remote, target data. More formally, in the distributed setting, we define the three types of dataset shift as follows:

COVARIATE SHIFT BETWEEN PARTITIONS. It is defined as

$$\begin{aligned} p_t(Y|X=\mathbf{x}) = p_s(Y|X=\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}; \forall \mathcal{D}_t, \mathcal{D}_s \subset \mathcal{D} \\ p_t(X) \neq p_s(X); \exists \mathcal{D}_t, \mathcal{D}_s \subset \mathcal{D} \end{aligned} \tag{8.6}$$

where $p_t(Y|X=\mathbf{x})$ and $p_s(Y|X=\mathbf{x})$ are the conditional distributions of Y in the partitions $\mathcal{D}_t$ and $\mathcal{D}_s$, respectively.

PRIOR PROBABILITY SHIFT BETWEEN PARTITIONS. It is defined as

$$\begin{aligned} p_t(X|Y=y) = p_s(X|Y=y), \forall y \in \mathcal{Y}; \forall \mathcal{D}_t, \mathcal{D}_s \subset \mathcal{D} \\ p_t(Y) \neq p_s(Y); \exists \mathcal{D}_t, \mathcal{D}_s \subset \mathcal{D} \end{aligned} \tag{8.7}$$

where $p_t(X|Y=y)$ and $p_s(X|Y=y)$ are the conditional distributions of X in the partitions $\mathcal{D}_t$ and $\mathcal{D}_s$, respectively.

CONCEPT DRIFT BETWEEN PARTITIONS. It is defined as

$$\begin{aligned} p_t(X=\mathbf{x}) = p_s(X=\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}; \forall \mathcal{D}_t, \mathcal{D}_s \subset \mathcal{D} \\ p_t(Y|X) \neq p_s(Y|X); \exists \mathcal{D}_t, \mathcal{D}_s \subset \mathcal{D} \end{aligned} \tag{8.8}$$

where $p_t(X=\mathbf{x})$ and $p_s(X=\mathbf{x})$ are the distributions of X in the partitions $\mathcal{D}_t$ and $\mathcal{D}_s$, respectively.

Note that the three types of dataset shift in standard machine learning are defined in terms of two sets of data. Here, in a distributed setting, they are defined in terms of the P partitions of the dataset. Thus, it may often happen that these *shifts* occur between subsets of partitions and not others. It may also happen that subsets of partitions show different types of shifts between them.

8.4 LEARNING FROM SKEWED DATA

Assume given a labeled data set $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N = X \times Y \in \mathcal{X} \times \mathcal{Y} = \mathbb{R}^m \times \{1, 2, \dots\}$. A data sample is a pair $(\mathbf{x_i}, y_i)$ consisting of an input sample $\mathbf{x}_i$ and its associated class label y_i. In a distributed setting, data are assumed

to be distributed across multiple sites $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_P$. The union of all these sets constitutes the complete data set: $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_P = \mathcal{D}$.

A classifier is a function h that maps from inputs to classes. The goal of the learning process is to find an h that correctly predicts the class $y = h(\mathbf{x})$. This is accomplished by searching some space $\mathcal{H}$ of possible classifiers (T. G. Dietterich, 2002). Nearly all of the works in distributed machine learning make *uniformity* assumptions: samples are uniformly distributed to sites. Then, the challenge is to obtain the learner one would get from a single, joint data set:

$$h(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_P) = h(\mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_P) \tag{8.9}$$

without providing the distributed learning algorithm h with simultaneous access to $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_P$ (Caragea et al., 2001). However, there is considerable evidence that data skew—the non-uniform distribution of samples to sites—exists. Taking this into account, does this equality remain unchanged under data skew?

Data skew force us to revisit the earlier correspondence to make it fit into this novel scenario. Thus, Equation 8.9 can be rewritten as follows $\forall \mathbf{x} \in \mathcal{X}$,

$$h(\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_P) = h_1(\mathcal{D}_1) \cup h_2(\mathcal{D}_2) \cup \cdots \cup h_P(\mathcal{D}_P)$$

where h_p, $p = 1, \ldots, P$, is the local hypothesis inferred from processor p. Thus, when a new sample arrives for classification at one processor, it will be this processor solely responsible for its classification. Note that it is assumed that the new data will follow the distribution of the training data. However, the problem with this approach is that it ignores the knowledge generated in other processors even if the hypotheses from some other processors—but maybe not all of them—are in some way *compatible* with its own local hypothesis. A better approach may be to introduce the concept of *similarity* between sites. Thus, the relevance of the local hypotheses can be weighted by the probability of occurrence of the new sample $\mathbf{x}$ in the different sites, that is, covariate shift,

$$\begin{aligned} h(\mathbf{x}) = & \, h_1(\mathbf{x})\, p(\mathcal{D}_1 = \mathbf{x}) \\ & \cup h_2(\mathbf{x})\, p(\mathcal{D}_2 = \mathbf{x}) \\ & \cup \ldots \\ & \cup h_P(\mathbf{x})\, p(\mathcal{D}_P = \mathbf{x}) \end{aligned} \tag{8.10}$$

where $p(\mathcal{D}_p = \mathbf{x})$ is the probability of occurrence of data $\mathbf{x}$ in the site p. If we consider prior probability shift, then the local hypotheses will be weighted

by the changes in class distributions with respect to the site t, that is,

$$\begin{aligned} h(\mathbf{x}) = & \, h_1(\mathbf{x})\, sim(p(Y_1), p(Y_t)) \\ & \cup h_2(\mathbf{x})\, sim(p(Y_2), p(Y_t)) \\ & \cup \ldots \\ & \cup h_P(\mathbf{x})\, sim(p(Y_P), p(Y_t)) \end{aligned} \tag{8.11}$$

where $sim(p(Y_s), p(Y_t))$ is a similarity function that measures the similarity between the class prior probability $p(Y_s)$ of partition s and the class prior probability $p(Y_t)$ of partition t. The similarity function sim outputs 1 if the two probabilities take exactly the same values, and 0 otherwise. Values between 0 and 1 quantify the degree of similarity.

Finally, if we consider concept drift with respect to the site t, then the local hypotheses will be weighted by the changes in the relationship between the input and class variables, that is,

$$\begin{aligned} h(\mathbf{x}) = & \, h_1(\mathbf{x})\, sim(p(Y_1|\mathcal{D}_1 = \mathbf{x}), p(Y_t|\mathcal{D}_t = \mathbf{x})) \\ & \cup h_2(\mathbf{x})\, sim(p(Y_2|\mathcal{D}_2 = \mathbf{x}), p(Y_t|\mathcal{D}_t = \mathbf{x})) \\ & \cup \ldots \\ & \cup h_P(\mathbf{x})\, sim(p(Y_P|\mathcal{D}_P = \mathbf{x}), p(Y_t|\mathcal{D}_t = \mathbf{x})) \end{aligned} \tag{8.12}$$

In a more general situation, one can interpret Equations 8.10, 8.11, and 8.12 as particular implementations of a reducer function $\mathcal{R}$ such that

$$h(\mathbf{x}) = \mathcal{R}\left(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_P(\mathbf{x})\right)$$

assuming that h_p is the local hypothesis inferred from dataset $\mathcal{D}_p$. According to Equations 8.10, 8.11, and 8.12, one can think in two different situations:

RANDOM PARTITIONS. If the dataset is randomly divided into P partitions then all these equalities hold (approximately),

1. $p(\mathcal{D}_s = \mathbf{x}) \approx p(\mathcal{D}_t = \mathbf{x}), \forall s, t.$
2. $sim(p(Y_s), p(Y_t)) \approx 1, \forall s, t.$
3. $sim(p(Y_s|\mathcal{D}_s = \mathbf{x}), p(Y_t|\mathcal{D}_t = \mathbf{x})) \approx 1, \forall s, t.$

Moreover, if the reducer function $\mathcal{R}$ is *exact* with respect to the hypothesis h then $\mathcal{R}\left(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_P(\mathbf{x})\right) = h(\mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_P)$.

SKEWED PARTITIONS. If the dataset shows partition skew then each local hypothesis h_p will contribute to the final hypothesis depending on its weight factor. Note that the presence of data skew is revealed if any of the previous equalities is violated.

The question now is how to quantify the skewness between partitions without communicating private data between nodes. The next section proposes several approaches to deal with this problem. Either way, the output of this process will be a dissimilarity matrix (also called distance matrix) that will describe the pairwise distinction between the P partitions. The matrix that defines the skewness of a distributed setting in terms of any distance function is formulated as

$$\begin{pmatrix} 0 & d_{1,2} & \dots & d_{1,n} \\ & 0 & \dots & d_{2,n} \\ & & \ddots & \vdots \\ & & & 0 \end{pmatrix}$$

where $d_{s,t}$ denotes the distance between partitions $\mathcal{D}_s$ and $\mathcal{D}_t$, and the matrix is symmetric, i.e., $d_{s,t} = d_{t,s}$. Note that a distance function is a measure of how similar, or dissimilar, two elements are.

8.5 APPROXIMATING THE DISTRIBUTIONS

Given the former equations, the three different types of dataset shift considered –covariate shift, prior probability shift, and concept drift– are revealed based on some violations of the conditions of uniformity of distributions. However, the *extent* of the problem must be defined in a quantitative form in order to exploit possible advantages associated with alternative combination schemes. Thus, in the next sections we present some of the most common ways to evaluate the different types of dataset shift: covariate shift by approximating the input distribution, prior probability shift by approximating the class distribution, and covariate shift by approximating the relationship between the input and class variables.

8.5.1 APPROXIMATING THE INPUT DISTRIBUTION

Consider the problem of estimating the probability $p(\mathcal{D} = \mathbf{x})$ that a data point $\mathbf{x}$ in D-dimensional Euclidean space belongs to a dataset $\mathcal{D}$. Three of the most common approaches for approximating the input distribution are the following.

UNIVARIATE GAUSSIAN DISTRIBUTION. The first step would be to find the center of mass of the sample points. Intuitively, the closer the point in question is to this center of mass, the more likely it is to belong to the dataset. However, one also need to know if the dataset is spread out over a large range or a small range, so that we can decide whether a given distance from the center is noteworthy or not. The simplistic approach is to estimate the standard deviation of the distances of the sample points from the center of mass. If the distance between the test point and the center of mass is less than one standard deviation, then we might conclude that it is highly probable that the test point belongs to the set. The further away it is, the more likely that the test point should not be classified as belonging to the set. This intuitive approach can be made quantitative by defining the normalized distance between the test point and the set to be $\frac{x-\mu}{\sigma}$, where μ is the center of mass and σ is the standard deviation. By plugging this into the normal distribution we can derive the probability of the data point belonging to the dataset (see Figure 8.4). The drawback of this approach is that we assumed that the sample points are distributed about the center of mass in a spherical manner.

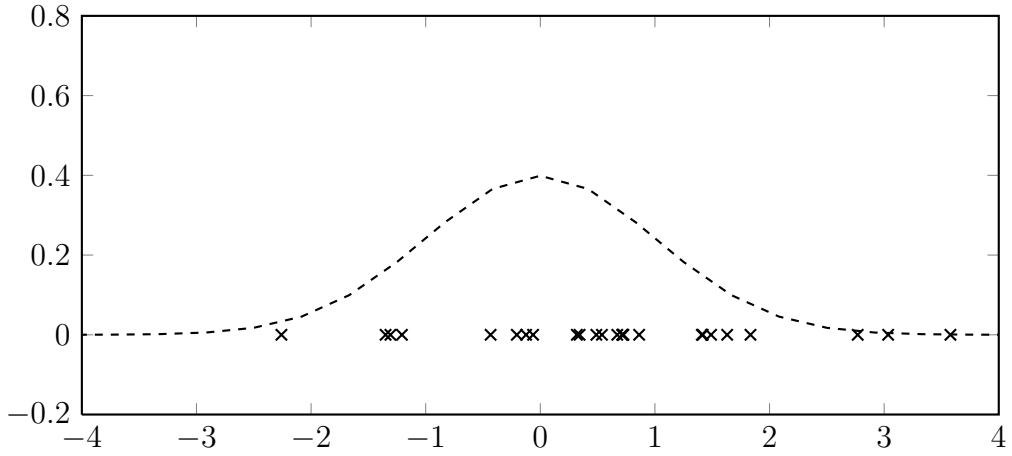


Figure 8.4: Example of a univariate Gaussian distribution.

MULTIVARIATE GAUSSIAN DISTRIBUTION. Were the distribution to be decidedly non-spherical, for instance ellipsoidal, then we would expect the probability of the data point belonging to the dataset to depend not only on the distance from the center of mass, but also on the direction. In those directions where the ellipsoid has a short axis the test point must be closer, while in those where the axis is long the test point can be further away from

the center. Putting this on a mathematical basis, the ellipsoid that best represents the set's probability distribution can be estimated by building the covariance matrix of the samples (see Figure 8.5). The Mahalanobis distance is simply the distance of the data point from the center of mass divided by the width of the ellipsoid in the direction of the data point. Mahalanobis distance is closely related to the leverage statistic. In statistics, high-leverage points are those that are outliers with respect to the independent variables. Leverage points are those that cause large changes in the parameter estimates when they are deleted. Mahalanobis distance and leverage are often used to detect outliers, especially in the development of linear regression models. A point that has a greater Mahalanobis distance from the rest of the sample population of points is said to have higher leverage since it has a greater influence on the slope or coefficients of the regression equation. Mahalanobis distance is also used to determine multivariate outliers. Regression techniques can be used to determine if a specific case within a sample population is an outlier via the combination of two or more variable scores. A point can be a multivariate outlier even if it is not a univariate outlier on any variable (consider a probability density similar to a hollow cube in three dimensions, for example).

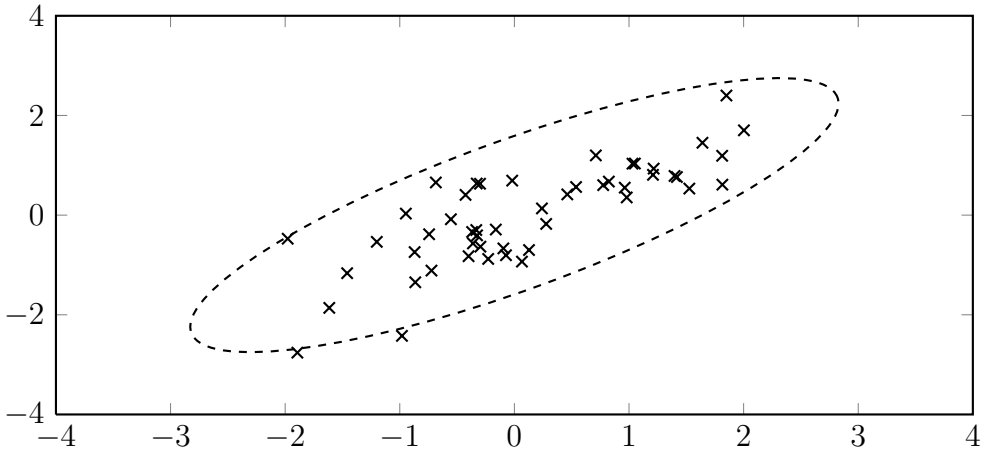


Figure 8.5: Example of a multivariate Gaussian distribution.

MIXTURE OF GAUSSIANS. A Gaussian model is commonly extended to fit multivariate Gaussian distributions (see Figure 8.6). A Gaussian mixture model is a weighted sum of K component Gaussian densities as given by the

equation,

$$p(\mathbf{x}|w, \mu, \mathbf{\Sigma}) = \sum_{k=1}^{K} w_k G(\mathbf{x}|\mu_k, \mathbf{\Sigma}_k) \tag{8.13}$$

where $\mathbf{x}$ is a D-dimensional data point, w_k is the k mixture weight, and $G(\mathbf{x}|\mu_k, \mathbf{\Sigma}_k)$ is the k Gaussian density. The complete Gaussian mixture model is parametrized by the mean vectors μ, covariance matrices $\mathbf{\Sigma}$, and mixture weights w. These parameters are usually estimated from training data using the Expectation-Maximization (EM) algorithm (Dempster, Laird, Rubin, et al., 1977). The EM algorithm is an efficient iterative procedure to compute the maximum likelihood.

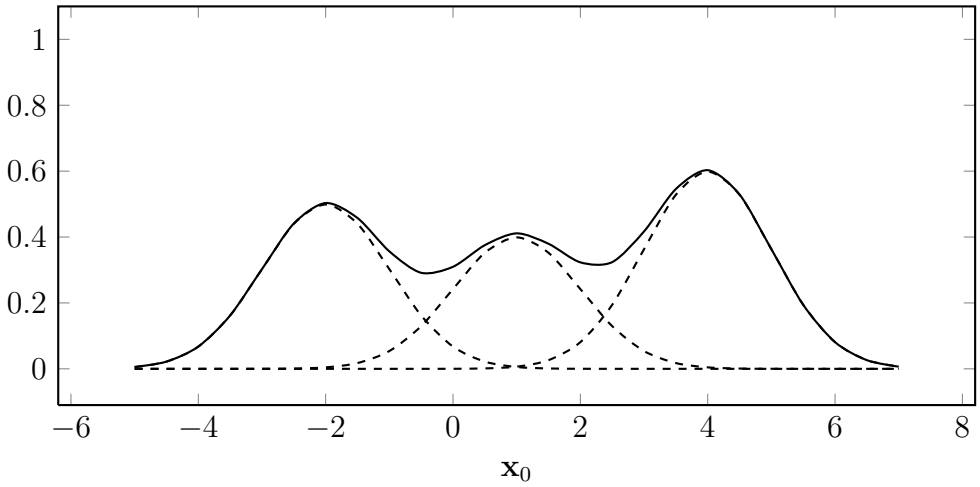


Figure 8.6: Example of mixture of Gaussians. Density function of the training data (solid line) and mixture of three Gaussians (dashed line).

8.5.2 APPROXIMATING THE CLASS DISTRIBUTION

Let $(p(Y_{1,s}), p(Y_{2,s}), \ldots, p(Y_{C,s}))$ be the prior class probabilities of the C classes in the s-th site. Note that this sample is just a data point in a C-dimensional space. The most popular metric to compute the distance between two data points—in this case representing the class prior probabilities from two sites—is the Euclidean distance. Euclidean distances are special because they conform to our physical concept of distance. But there are many other distance measures which can be defined between multivariate samples. These non-Euclidean distances are of different types: some still satisfy the basic axioms of what mathematicians call a metric, while others are not even metrics but still make very good sense as a measure of difference

between samples in the context of certain data. In mathematics, a metric d is a function that is required to satisfy the following conditions: positive definiteness ($d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$), symmetry ($d(x, y) = d(y, x)$), and triangle inequality ($d(x, z) \leq d(x, y) + d(y, z)$) (Bronshtein, Semendiaev, & Hirsch, 1985). Some of the most popular non-Euclidean distance measures are the Hamming distance, the Canberra distance, and the Czekanowski coefficient (Krzanowski & Krzanowski, 2000).

8.5.3 APPROXIMATING THE RELATIONSHIP BETWEEN THE INPUT AND CLASS VARIABLES

The notion of *classifier distance* introduced in (Tsoumakas et al., 2004) as a measure of how different two models are based on their predictions can be used to approximate the relationship between the input and class variables. The measures that one can use for computing the distance between two classifiers will depend on the type of classification. On the one hand, soft classifiers explicitly estimate the class conditional probabilities $p(y|\mathbf{x})$ and then perform classification based on the estimated probabilities, i.e., the classifier predicts membership grades for each class. In this case, the measures that can be used are the same than when approximating the distance between class distributions. The distance of two classifiers is defined as the distance of their output vectors with respect to all data points of an independent dataset. By independent we understand a dataset whose instances were not in the training set of the classifier. This will ensure unbiased results, as the predictions of classifiers on their training data tend to be optimistic.

On the other hand, hard classifiers directly target on the classification boundary without producing the probability estimation, i.e., each data point belongs to the class it most closely resembles. In an ensemble, it is intuitively accepted that classifiers to be combined should be *diverse*. If they were identical, we could not gain any improvement by combining them. Therefore, *diversity* among the ensemble has been recognized as a key issue. Because of this, diversity has been widely investigated in the literature. However, there is no single definition of diversity. There are different diversity measures available from different fields of research such as statistics and pattern recognition. Some of these measures work on the whole ensemble of classifiers whilst other measures consider the classifiers on a pairwise basis. For the purposes of clustering classifiers, we are more interested in pairwise measures.

Consider two classifiers h_s and h_t and a 2×2 table for the case of correct/incorrect (1/0) classifier outputs (oracle-type outputs) as shown in Table 8.1. There are various statistics to assess the similarity of two classifier outputs based on the intuition that two diverse classifiers perform differently on

	$\hbar_t$ correct (1)	$\hbar_t$ wrong (0)
$\hbar_s$ correct (1)	a	b
$\hbar_s$ wrong (0)	c	d

Table 8.1: Contingency table for two binary classifiers.

the same training data (Shipp & Kuncheva, 2002; Tang, Suganthan, & Yao, 2006).

— Yule's Q statistic. The Yule's Q statistic (Yule, 1900) for two classifiers $\hbar_s$ and $\hbar_t$ is computed as

$$Q_{s,t} = \frac{ad - bc}{ad + bc}$$

where $Q_{s,t} = 0$ for statistically independent classifiers.

— Correlation coefficient ρ. The correlation between two binary classifier outputs is

$$\rho_{s,t} = \frac{ad - bc}{\sqrt{(a+b)(c+d)(a+c)(b+d)}} \tag{8.14}$$

for any two classifiers, Q and ρ have the same sign.

— Disagreement measure D. In (Skalak, 1996), the authors proposed the disagreement measure to evaluate the diversity between two base classifiers. This measure is defined as

$$D = \frac{b + c}{a + b + c + d} \tag{8.15}$$

— Double-fault measure DF. (Giacinto & Roli, 2001) proposed the double-fault measure to select classifiers that are least related from a pool of classifiers. The authors claimed that the more different two classifiers are, the fewer the coincident errors between them. This measure is defined as

$$D = \frac{d}{a + b + c + d} \tag{8.16}$$

8.6 FINAL REMARKS

In this chapter, we provided some background on the distribution of the distributions of data. Borrowing ideas from the concept of dataset shift in

standard machine learning, we extended these definitions to a distributed environment and reviewed some popular ways of evaluating the distribution of the data in the input and output space of a learning model, and between the input and the output. In the next two chapters, we make use of some of these concepts and methods in order to expand the operation of the distributed learning algorithms based on single and two-layer neural networks, and the FVQIT (presented in the previous part of this thesis) when the distribution of data is skewed.

Chapter 9

Training Distributed Neural Networks on Skewed Data

In Chapters 4 and 5, we introduced two distributed learning algorithms for single-layer and two-layer artificial neural networks, respectively. These algorithms are fast and accurate, making possible privacy-preserving classification by not exchanging raw data across distributed locations but only the neural networks. Furthermore, they minimize communications by using this approach since the size of the neural networks is negligible in comparison with the size of the raw data. However, the performance of these algorithms may be affected when the distributions of data show skewness. With the aim of overcoming this issue, in this chapter we take into account this skewness in order to propose several improvements of the algorithms, based on distributing the computation of the genetic algorithm (Peteiro-Barral, Guijarro-Berdinas, & Pérez-Sánchez, 2011, 2012).

9.1 REVISITING THE DISTRIBUTED TRAINING ALGORITHMS FOR NEURAL NETWORKS

In order to learn from distributed data, the approach we followed to develop the two distributed learning algorithms for neural networks was the *MapReduce* paradigm, i.e. local learning and model integration; in the first place, the classifiers are trained on their corresponding, local, subset of data and then they are integrated using some combination method. These two algorithms took advantage of this paradigm avoiding moving raw data across the locations and therefore allows privacy-preserving classification as well as minimizing communications, as described in Chapters 4 and 5.

As local classifiers, these two distributed algorithms used single-layer and two-layer neural networks trained with efficient learning algorithms (Fontenla-

Romero, Guijarro-Berdiñas, Pérez-Sánchez, & Alonso-Betanzos, 2010; Castillo et al., 2006). 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. These training algorithms present some advantages which make them suitable for our purposes on distributed learning. On the one hand, they are very efficient since the weights of the artificial neural networks are computed analytically and, on the other hand, there are able to learn in an incremental manner. Then, the weights of a single, global, neural network representing the union of all, local, neural networks may be computed by summing their corresponding matrices of coefficients **A** and **b**, solving the new systems of linear equations. Notice that even when the knowledge of an artificial neural network is contained in its weight matrix **W** it can be computed from the coefficients **A** and **b**. However, this method may be improved by aggregating the knowledge contained in the different neural networks by means of a more appropriate cost function for classification problems. Thus, the two distributed algorithms compute the global neural network using a genetic algorithm. Basically, a genetic algorithm is defined by its fitness function, crossover operator, and mutation operator; where the set of local classifiers represented by the matrices of coefficients **A** and **b** are the initial population of the algorithm. In order to find the best combination of local neural networks, the genetic algorithm is run in a central location wherein independent, validation data is stored. A high level view of the algorithms would be the following;

1. For each location, train a local classifier on the data.

2. Broadcast each local model, represented by matrices **A** and **b**, to the central location.

3. Train the genetic algorithm 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, these algorithms are able to learn from distributed data sets without exchanging any raw, private data.

The operation of the two algorithms follows a basic *MapReduce* approach, and actually this approach works in practice for many problems. However, notice that the algorithm needs to meet two requirements;

— 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 genetic algorithm 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 genetic algorithm but the data used.

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

If we assume that the number of samples in the dataset is large, in principle, 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 factor for achieving a good performance using this algorithm. With the aim of overcoming these limitations three different improvements of this algorithm are proposed in the next section. Notice that any improvement of the algorithm must keep the advantages of the original, that is allowing privacy-preserving classification and minimizing communication costs. Notice also that this improvements may be applied to both single-layer and two-layer neural networks training algorithms presented in chapters 4 and 5.

9.2 PROPOSED IMPROVEMENTS FOR TRAINING NEURAL NETWORKS IN A DISTRIBUTED SETUP

In this section, we propose three improvements of the distributed training algorithm for both single and two-layer neural networks.

#I. The first improvement tries to overcome the limitation of the need of a central location by computing genetic algorithm 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. Notice that, in this model, any distributed location can play this role, it is

just a matter of computation, not data stored at any specific location. 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. The following procedure summarizes the operation of the algorithm and describes how the global classifier is obtained;

1. For each location, divide the data into training and validation sets, and 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 genetic algorithm at the designated location. Repeat until convergence (epochs, error...);

   (a) 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.

   (b) 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.

#II. The second improvement tries to overcome the limitation of retaining independent data for training the global model by following the approach proposed in (Tsoumakas & Vlahavas, 2002b). The key in this approach is to bring the concept of *partially*-global classifiers in the algorithm. The hypothesis here is that training both local and partially-global models on more data will improve the performance of the classifiers. The procedure is as follows;

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 genetic algorithm 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 *partially*-global classifier and broadcast it to the designated location.

4. For the designated location, build the global classifier by combining the *partially*-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.

#III. Finally, the third improvement merges the two previous improvements to some extent. The hypothesis here is that this approach will be able to exploit the strengths of both previous approaches, at the expense of the higher complexity of the model. The following procedure summarizes the operation of the algorithm;

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 a genetic algorithm 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 *partially*-global classifier and broadcast it to the designated location.

4. Execute the genetic algorithm at the designated location using the *partially*-global classifiers as initial population. Repeat until convergence (epochs, error...),

   (a) 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.

   (b) 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.

9.3 EXPERIMENTAL STUDY

The experimentation presented below is focused on the assessment of the three improvements proposed in this chapter. We will pay attention to 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. This fact may cause that the models at different locations perform quite differently from one another in terms of accuracy.

9.3.1 MATERIALS AND METHODS

Four well-known data sets selected from the *UCI Machine Learning Repository* (Frank & Asuncion, 2010) were used. For each data set, number of inputs, classes and samples are shown in Table 9.1. In order to be as representative as possible of different data sets, we included binary and multi-class medium-large size datasets. Again, 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 this experimentation, we focus our attention on prior probability shift (see Section 8.3 for more details). We chose this kind of skewness of data, in the class distributions, because it is easier to visualize and represent than the other kinds of data skewness. Notice, however, that the concepts and applications of the improvements presented in this chapter are also suitable for those other kinds, i.e. covariate shift and concept drift. For a given data set, we have defined the prior probability as the 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, distributed data environment as similar prior probability distributions for *every* location, that is, the prevalence of each class is the same among the locations. On the other hand, a nonuniform, or skewed, distributed data environment occurs when there are an dissimilar prior probability distributions for at least two locations. The latter opens the way for different degrees of skewness. Data skewness is an important factor that affects learning. Therefore, in order to assess the performance of the distributed algorithms in the most appropriate manner, we simulate several distributions of data among locations; one uniform and three skewed. The difference among the nonuniform distributions lies in the degree of skewness: slight, quite and severe.

Figure 9.1 shows the different environments for each dataset considered in this experimentation. The darker the shading the more skewed the distribution. 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

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 9.1: Properties of the datasets used for the experimentation.

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 skewed 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. With the aim of assessing the performance of the distributed learning algorithm for neural networks and the three improvements presented in this chapter, the following procedure was performed. For purposes of simplicity, the distributed algorithm uses single-layer neural networks. The same procedure would be valid for training two-layer neural networks as proposed in Chapter 5.

— For each algorithm; repeat $I = 100$ times

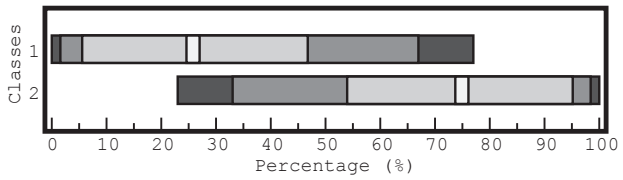
  – Divide the data set into $N = 10$ subsets. In this manner, each subset of data will represent a distributed location.

  – For each location, divide the data using *holdout 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.

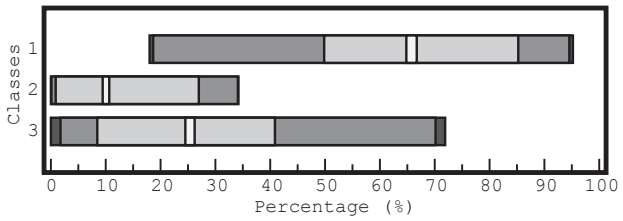
  – Train the distributed model.

  – Test the model at each location by computing the standard class accuracy (Weiss & Kulikowski, 1991) on the test data set. Also compute the global test accuracy by averaging the test accuracy among the locations.

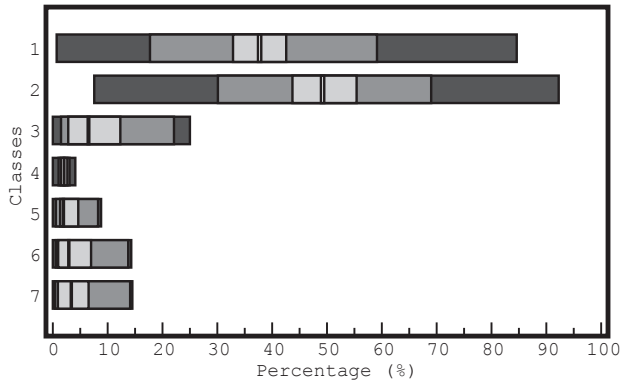
— Compute the *mean* test accuracy and *standard deviation* of each model over the 100 simulations.



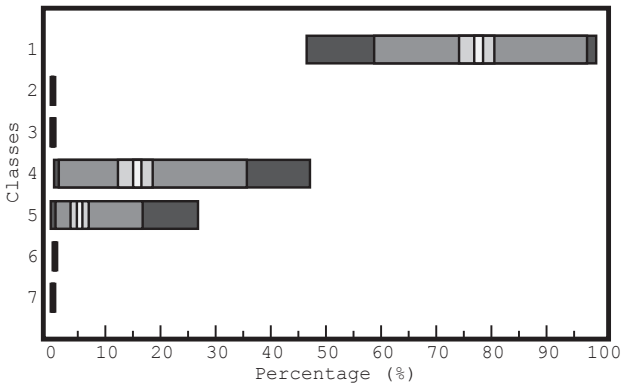
(a) Adult.



(b) Connect-4



(c) Covertype



(d) Statlog

Figure 9.1: Prior probability distributions of data. The darker the shading the more skewed the distribution.

— Apply a Kruskal-Wallis test (Hollander & Wolfe, 1999) 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 (Hsu, 1996) 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 (Hsu, 1996) was used as multiple comparison test.

Finally, apart from the test accuracy which is crucial, we believe that another interesting measure of performance in skewed environments of this distributed learning algorithm is the disparity of performance among different 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 chapter use a global model for classifying new samples, and this global model is the same for every location. However, notice that we assume a similar prior probability distribution between the training samples and the new samples received for every location. Thus if the distribution of data is not uniform 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,

$$disp = \frac{P \cdot (P-1)}{2} \sum_{i=0}^{P-1} \sum_{j=i+1}^{P} |acc_i - acc_j| \tag{9.1}$$

P being the number of locations. We believe that the measure of disparity is a good measure for having more insights about how well a distributed algorithm performs in different distributions of data. Of course, the lower the disparity the better, i.e. the more constant the performance of the model in the different locations the better.

9.3.2 RESULTS

Table 9.2 shows the mean test accuracies and standard deviations of the original distributed algorithm (see Chapter 4) and the three improvements proposed in this chapter (see Section 9.2). The best result, or those not significantly different from the best one, are underlined for each data set

and degree of skewness. On the other hand, Table 9.3 shows the disparity among locations. This is a measure of how different the locations perform by averaging all pairwise differences in accuracy among them.

Dataset	Skewness	Algorithm			
		Original	*Imprv 1*	*Imprv 2*	*Imprv 3*
Adult	None	$\underline{83.59 \pm 0.72}$	$\underline{83.22 \pm 0.94}$	81.61 ± 2.61	$\underline{83.43 \pm 0.65}$
	Slight	$\underline{82.35 \pm 1.25}$	$\underline{83.46 \pm 0.61}$	79.21 ± 5.04	$\underline{83.30 \pm 0.62}$
	Quite	79.06 ± 3.61	$\underline{83.59 \pm 0.86}$	77.71 ± 7.71	$\underline{82.86 \pm 1.10}$
	Severe	75.92 ± 5.71	$\underline{83.15 \pm 0.76}$	78.09 ± 7.59	$\underline{83.34 \pm 0.62}$
Connect4	None	$\underline{75.56 \pm 0.50}$	$\underline{75.17 \pm 0.64}$	$\underline{73.82 \pm 4.92}$	$\underline{75.16 \pm 0.52}$
	Slight	$\underline{74.43 \pm 1.83}$	$\underline{75.25 \pm 0.31}$	$\underline{73.89 \pm 3.41}$	$\underline{75.24 \pm 0.64}$
	Quite	71.56 ± 4.09	$\underline{75.25 \pm 0.53}$	61.77 ± 9.99	$\underline{75.14 \pm 0.52}$
	Severe	68.20 ± 6.29	$\underline{75.06 \pm 0.50}$	67.90 ± 9.74	$\underline{74.93 \pm 0.55}$
Covertype	None	$\underline{70.24 \pm 0.24}$	$\underline{70.18 \pm 0.20}$	$\underline{69.78 \pm 1.18}$	$\underline{70.33 \pm 0.33}$
	Slight	$\underline{69.34 \pm 1.49}$	$\underline{70.48 \pm 0.30}$	$\underline{70.10 \pm 0.54}$	$\underline{70.59 \pm 0.16}$
	Quite	66.67 ± 1.99	$\underline{70.38 \pm 0.24}$	$\underline{68.97 \pm 1.35}$	$\underline{70.09 \pm 0.98}$
	Severe	64.75 ± 4.84	$\underline{69.79 \pm 1.02}$	56.28 ± 9.99	$\underline{68.68 \pm 3.94}$
Shuttle	None	90.89 ± 0.98	90.30 ± 1.20	77.06 ± 9.99	$\underline{94.13 \pm 2.06}$
	Slight	$\underline{89.82 \pm 1.23}$	$\underline{90.63 \pm 1.36}$	86.48 ± 4.27	$\underline{90.54 \pm 1.63}$
	Quite	$\underline{89.91 \pm 1.94}$	$\underline{91.40 \pm 1.46}$	78.33 ± 9.99	$\underline{92.47 \pm 1.64}$
	Severe	88.87 ± 3.57	$\underline{91.80 \pm 0.73}$	87.46 ± 3.77	$\underline{90.98 \pm 2.53}$

Table 9.2: Mean test accuracy (%) and standard deviation of the original algorithm and its improvements for each data set and the four prior probability distributions.

9.3.3 DISCUSSION AND CONCLUSIONS

The results presented in the previous section were focused on the test accuracy and disparity among locations of the original distributed algorithm for neural networks and the three improvements introduced in this chapter. For this experimentation we used single-layer neural networks, but the concepts and procedures would be the same for two-layer neural networks. Regarding the test accuracy, as can be inferred from Table 9.2:

— If we compare the performance of the original algorithm between *none* and *severe* skewed 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

Dataset	Skewness	Algorithm			
		Original	*Imprv 1*	*Imprv 2*	*Imprv 3*
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
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
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
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 9.3: Disparity (%) among locations of the original algorithm and its improvements for each data set and prior probability distribution.

with the increasing skewness of data. Thus, it is logical that the higher the heterogeneity the lower the accuracy.

— Almost the same performance is reached by the first improvement for every distribution of data. Small variabilities can be explained as the effect of randomness. As can be seen, distributing the computation of the genetic algorithm works much better than selecting a location by random as happened in the original algorithm. This improvement is able to form an unbiased, and *distributed*, validation set.

— Notice that the performance of the second improvement is uncorrelated with the distribution of data. In fact, it shows a worse performance than the original algorithm in many cases. Notice also that the standard deviation for the second improvement 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 the third improvement for each distribution of data. The performance of the first and third

improvements are not significantly different, but the former is much simpler than the latter. 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 the purposes of this chapter are those computed on severe skewed distributions of data 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 the first and third improvements are the best algorithms and also they are not significantly different from each other. The original algorithm and the second improvement are significantly worse than these two.

Finally, regarding the disparity in accuracy of the algorithms as can be seen Table 9.3, the higher the heterogeneity the larger the disparity, this result being logical for every algorithm. 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 the first improvement is the best algorithm in terms of disparity, better or not significantly worse than the others. In view of the above, the first improvement is chosen as the best performing algorithm when the prior probability distributions of data are skewed. The first improvement shows the best trade-off between performance on different distributions of data and simplicity, while maintaining privacy-preserving classification and reduced communication costs. However, notice that the disparity in the performance among the different locations is still an issue, and it increases when the skewness of data increases. Thus, in the next chapter, we will go one step further in order to address this issue. The key will be combining only similar knowledge, since combining *all* local classifiers when there is a noticeable diversity in the knowledge contained in the different models may be improved.

Chapter 10

Training Algorithms on Skewed Data using Island Models

In general terms, distributed learning algorithms aim to infer a global learner that approximates the results one would get from a single, joint data source. Note that this approach assumes that there is a single, global model that could be induced from the distributed locations. Under this view, distribution is treated exclusively as a technical issue. However, there are deeper implications. Real-world distributed data sets almost always present quite strong differences between their partitions, e.g. buying patterns in different supermarkets from different countries. In this chapter, we introduced a novel proposal for distributed learning that requires no assumptions about the statistical properties of the partitions of data. This novel algorithm is based on the island model genetic algorithms. In the island model, several populations are kept at the same time. From time to time, individuals move from one population to another by migration. Hence, the various islands maintain some degree of independence and interdependence from each other. We believe that this idea is highly related to the topic in hand.

10.1 ISLAND MODEL GENETIC ALGORITHMS

Most genetic algorithms work with one large panmictic population. A panmictic population is one where all individuals are potential partners. Those genetic algorithms suffer from the problem that natural selection relies on the fitness distribution over the whole population, i.e. it is based on global knowledge (Gorges-Schleuter, 1991). Now we turn to the case in which the population is viewed as the concept of keeping several panmictic subpopulations at the same time. Each panmictic subpopulation evolves like a traditional genetic algorithm. The isolated populations may communicate by

migration, that is from time to time individuals move from one population to another population. This model is known under the name *island model* (Gorges-Schleuter, 1991). Early studies of Grosso (Grosso, 1985) showed that the subdivision of one large panmictic population into interacting subpopulations leads to a performance improvement that will not be fully realized at either of the two extremes of complete subpopulation interdependence (panmixia) or complete subpopulation independence (no migration). One reason for this is that the various *islands* maintain some degree of independence and thus explore different regions of the search space while at the same time sharing information by means of migration (Whitley, Rana, & Heckendorn, 1999).

10.2 SOME INSIGHTS ON SUBPOPULATION INTERDEPENDECE AND INDEPENDENCE

The following experiment will shed light on the relation between subpopulation interdependence and independence when the diversity of data across partitions increases. For this purpose, assume a distributed setting with five degrees of increasing skewness of data. Consider also the following procedure: *every site send a random individual of its subpopulation –migration– to another random site in every generation.* Figure 10.1 plots the average percentage of individuals sent between sites that result in a decreasing rate of error in the recipient site (this is just a result computed on the same setup presented in Section 6, using the same datasets and scenarios). As can be seen, the percentage of successful communications decreases according as the skewness of the scenario increases. Namely, it is more cost effective to send individuals between similar sites rather than dissimilar ones.

10.3 IMPLEMENTING THE ISLAND MODEL ON DISTRIBUTED LEARNING ALGORITHMS

Optimally, a distributed learning algorithm should be aware of the distribution of data in order to increase the rate of successful communications between sites. The idea behind the island model will be helpful to give any distributed learning algorithm some aware of the distribution of data. The rationale of this approach is based on the assumption that the migration stream can be modulated by the differences between partitions. In the distributed setting may occur different degrees of independence and interdependence between different sites. In the case of uniformly distributed data it is expected low independence and high interdependence. Otherwise high independence

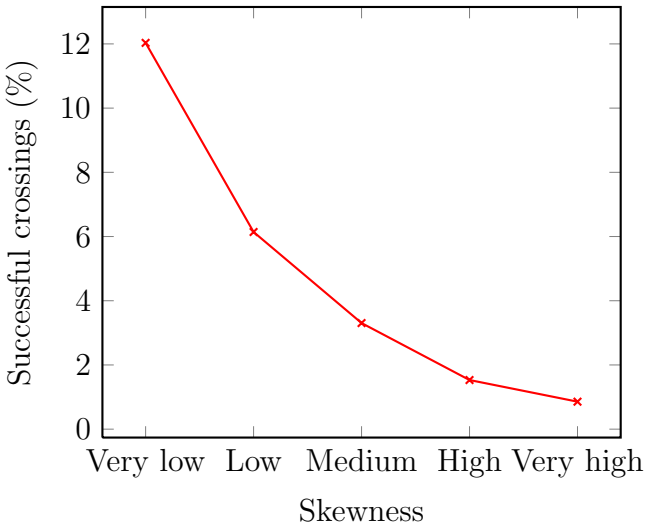


Figure 10.1: Average percentage of successful crossings between sites , i.e. successful communications.

and low interdependence. The idea that underlies this approach is to, in some manner, *cluster* distributed sources of data to boost interdependence between similar sites and promote independence –minimize communications– between dissimilar ones. For this purpose, the matrix of distances between sites proposed in Section 8.4 will be helpful. These distances can be translated into probabilities of communication between sites for migration. Thus, the migration flow is modulated by distance –the shorter the distance, the higher the chances of migration. Note that this approach permit communications between distant sites, but with low probability. This design maximizes communications between a priori related sites and minimizes those communications between a priori unrelated sites. Thus, it is expected that the number of spurious communication will be minimized whilst maximizing valuable exchanges of individuals.

10.4 EXPERIMENTAL STUDY

The objective of this section is to experimentally evaluate the performance of the island model on a distributed learning algorithm. Notice that the concept of island model could be applied to any of the algorithms proposed in this thesis. In particular, we will focus our attention of the implementation of the distributed FVQIT proposed in Chapter 6. As a quick summary, the FVQIT method is a supervised learning algorithm based on information theoretic learning and vector quantization. The principle of vector quantization is to

represent a dataset with a smaller number processing elements (PEs). The task is to place the PEs in the input space. The most frequent class in the neighborhood of the PE will attract it whilst the second most frequent class of the PE will repel it. Thus, multiple attraction and repulsion forces converge on each PE. The idea is to place the PEs in the frontier between data samples from two different classes. The combination of the distributed, partial solutions of the FVQIT in a comprehensive, global solution is not a straightforward process. In order to overcome this difficulty, we proposed a strategy to optimize the combination of models based on genetic algorithms (see Chapter 6 for all the details about the implementation of the method).

10.4.1 MATERIALS AND METHODS

We compare the distributed method based on the island model against the two extremes of complete subpopulation interdependence (panmixia, original implementation of the DFVQIT) or complete subpopulation independence (no migration). The algorithms are evaluated to classify the same twelve datasets used in the experimentation of the DFVQIT (see Section 6.2.2 for more details). In the distributed setting, training data have been scattered across 5 different sites in which each site contains 10 PEs. The evaluation of the methods has been done using holdout validation: 90% of data for training, and 10% for testing. Additionally, 10% of the training data have been used for validation. Experiments were run 10 times with random partitions of the data set to ensure reliable results. We use the Kruskal-Wallis test to check if there are significant differences for a level of significance $\alpha = 0.05$. If there are differences, then we apply a multiple comparison procedure to find the methods which are not significantly different. Five different distributed scenarios were considered. In each one of this scenarios, distributed data sets present the following properties where the differences between partitions are considered in terms of both prior probability shift and covariate shift;

— Level 1, *very weak* differences between partitions

— Level 2, *weak* differences between partitions

— Level 3, *moderate* differences between partitions

— Level 4, *strong* differences between partitions

— Level 5, *very strong* differences between partitions

Figures 10.2 and 10.3 show the different scenarios in the distributed setting in terms of differences of probability of occurrence of classes (the darker

the color of the bar the weaker the skewness). For example, in the very weak scenario in Adult data set (see top right plot in Figure 10.2), the difference across the sites in the prevalence of class 1 is under 2%. In the weak scenario, the difference rise to around 10%. The present interpretation is valid for the remaining classes and datasets included in this experimentation. Notice that the y-axis is in logarithmic scale.

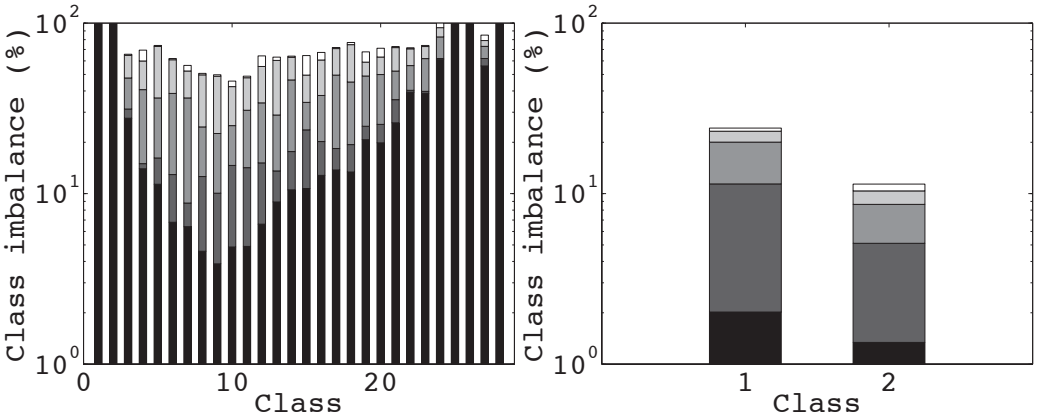
Similarly, Figures 10.4 and 10.5 show the different scenarios in the distributed setting in terms of differences of probability of occurrence of samples, i.e. covariate shift. In this case, the differences are computed in terms of the probability of every feature of a sample to belong to a site. In a very weak scenario, it is expected that a sample may belong to any site with almost equal probability. For example, in Abalone dataset (see top left plot in Figure 10.4), the difference in the feature 1 is below the 2%. On the contrary, in the very strong scenario, the difference rises to 6%. These figures together with Figures 10.2 and 10.3 enable an in depth analysis of the different data sets in the different scenarios.

10.4.2 RESULTS

Figures 10.6 and 10.7 display the test performance error in complete subpopulation interdependence (panmixia, original implementation of the DFVQIT), complete subpopulation independence (no migration), and island model. Figure 10.8 summarizes these results by showing the average test error on the twelve data sets.

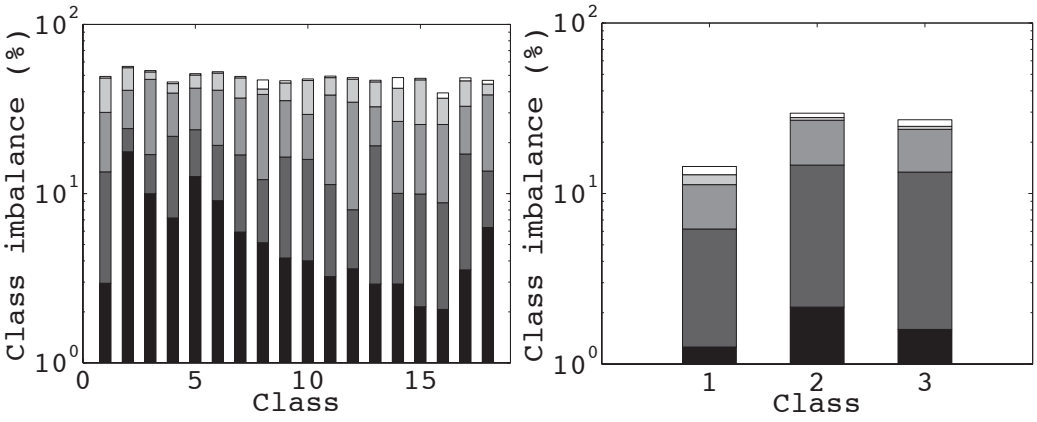
10.4.3 DISCUSSION AND CONCLUSIONS

The original implementation of the DFVQIT (panmixia) obtains promising results in basically uniform scenarios but shows a deterioration in its performance when the skewness of the dataset increases. This trend is the opposite than the DFVQIT without migration of individuals. These are logical results of the fact that random partitioning scenarios elicit more general knowledge whilst strong skewed scenarios elicit more specific knowledge. The implementation of the DFVQIT based on the island model exploits the advantages of both general and specific knowledge, by weighting interdependence and independence between sites. In average, the island model is not statistically significantly different from the original DFVQIT in the very weak (0.19%) and weak (0.13%) scenarios. However, the island model improves the former DFVQIT by 3.30% and 3.52% in strong and very strong scenarios, respectively. On the other hand, the island model improves the model with no migration by a large margin, more than 3% in every scenario. In this case,



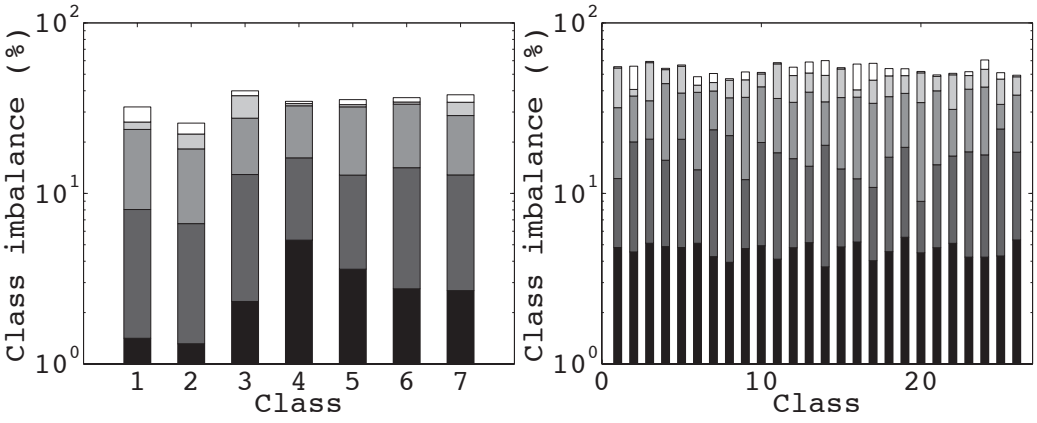
(a) Abalone.

(b) Adult.



(c) Chess.

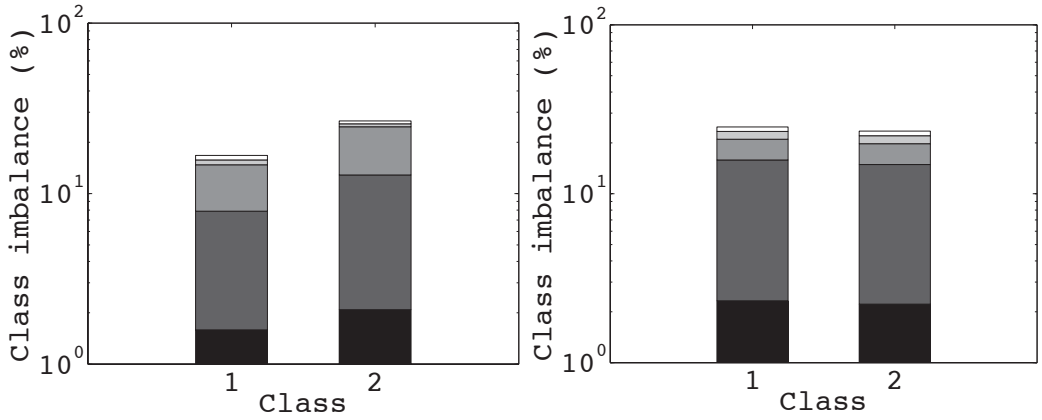
(d) Connect-4.



(e) Forest.

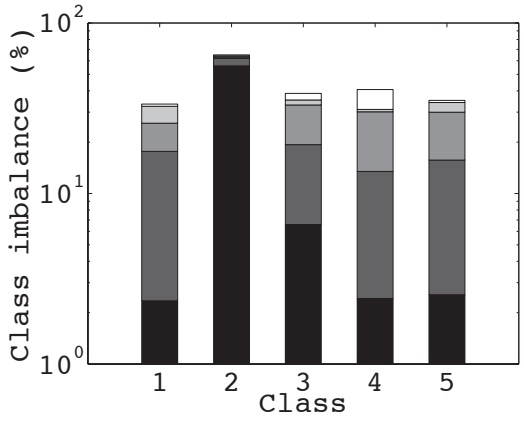
(f) Letter.

Figure 10.2: Maximum difference between sites in terms of prior probability distributions of classes; prior probability drift or class imbalance problem. The incremental skewness for the different scenarios is represented as stacked bars. Part 1 of 2.

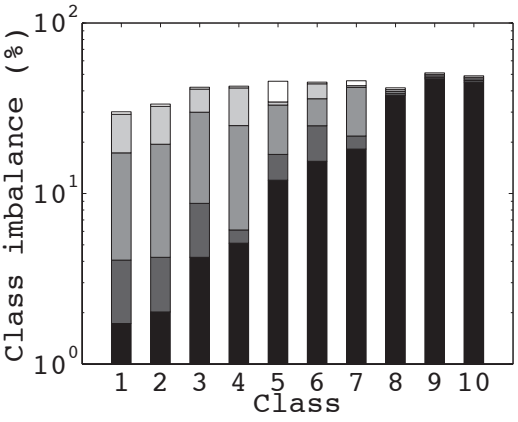


(a) Magic.

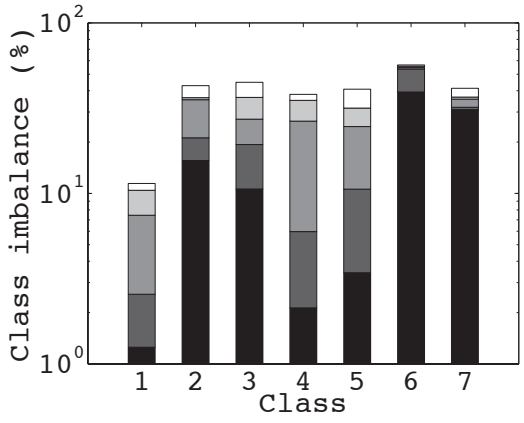
(b) Mushroom.



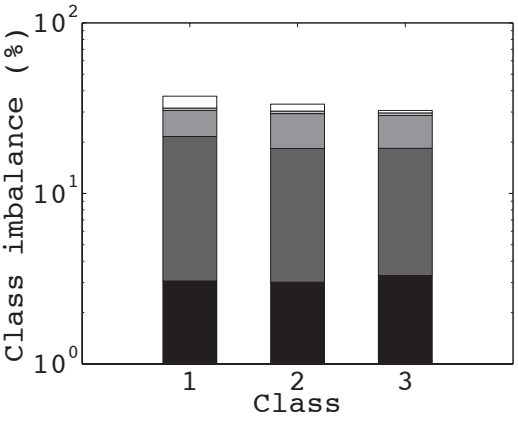
(c) Nursery.



(d) Poker.



(e) Shuttle.



(f) Waveform.

Figure 10.3: Maximum difference between sites in terms of prior probability distributions of classes; prior probability drift or class imbalance problem. The incremental skewness for the different scenarios is represented as stacked bars. Part 2 of 2.

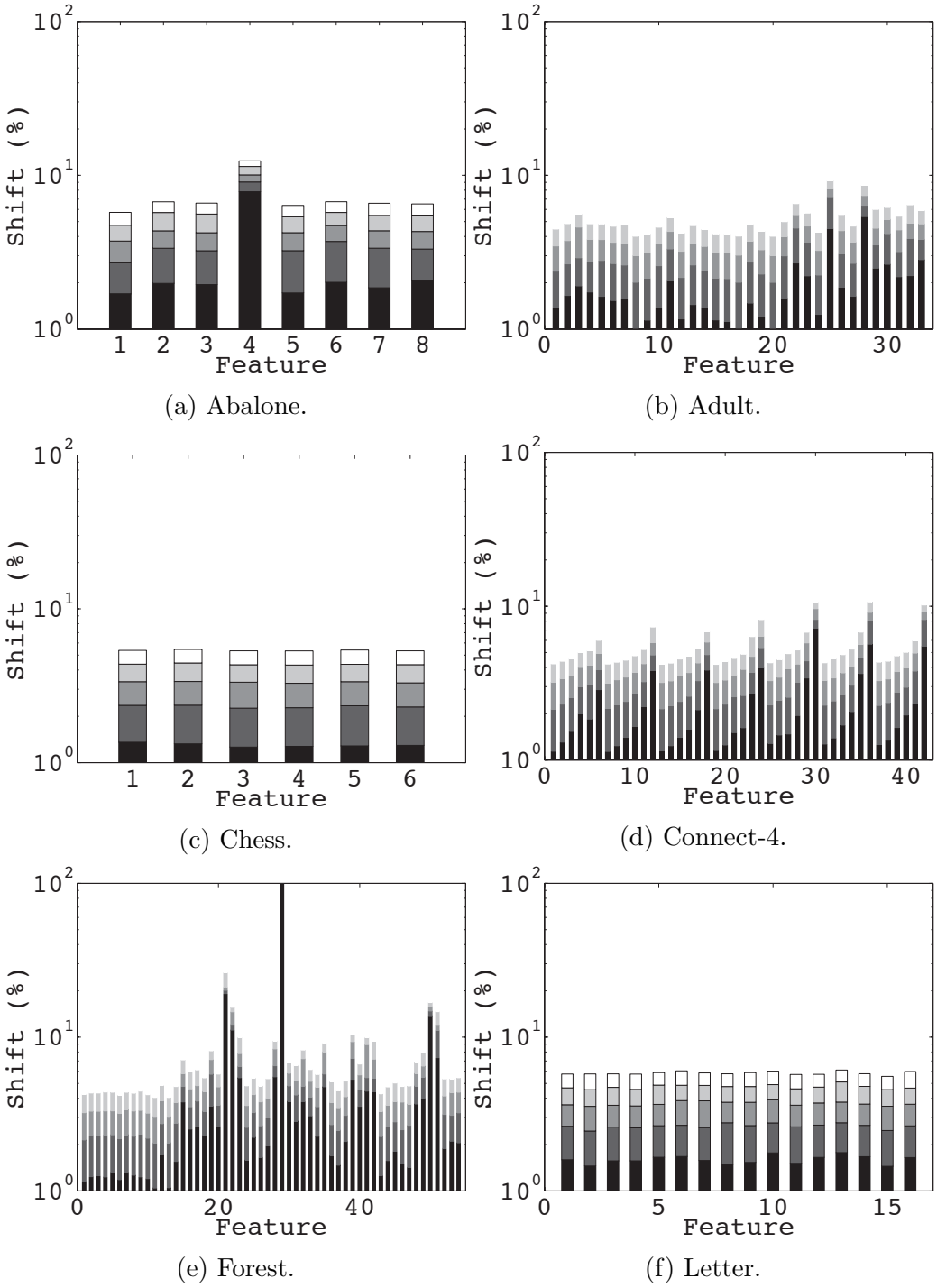
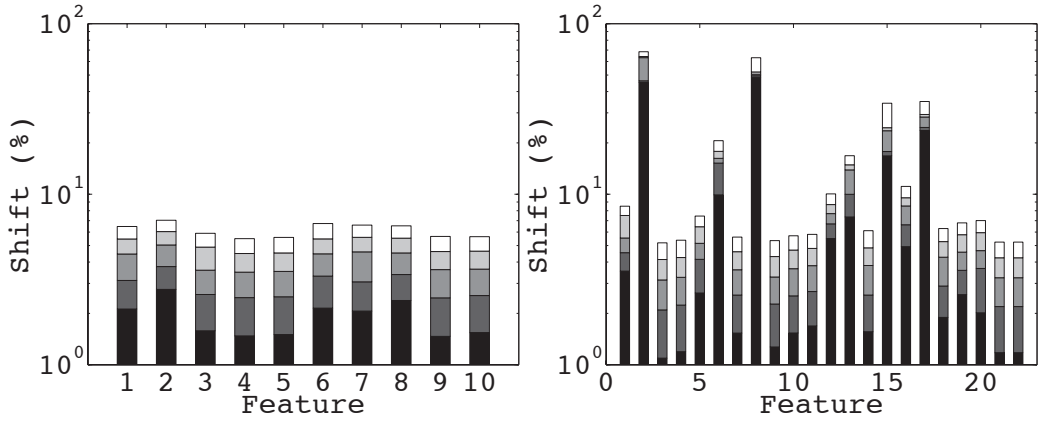
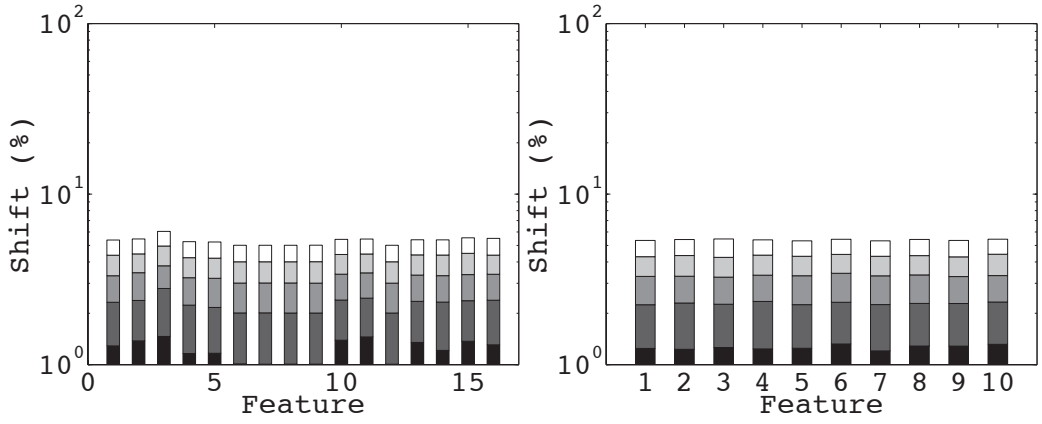


Figure 10.4: Maximum difference between sites in terms of probability of occurrence of samples; covariate shift. The incremental skewness for the different scenarios is represented as stacked bars. Part 1 of 2.



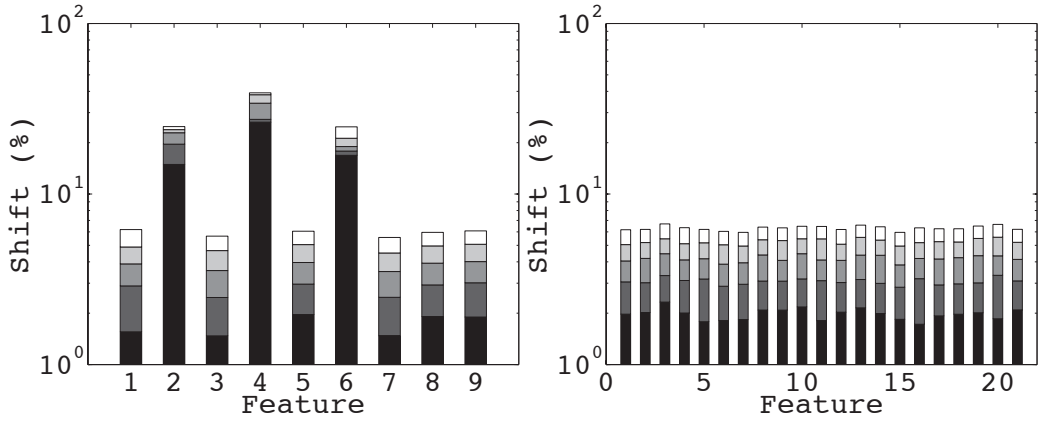
(a) Magic.

(b) Mushroom.



(c) Nursery.

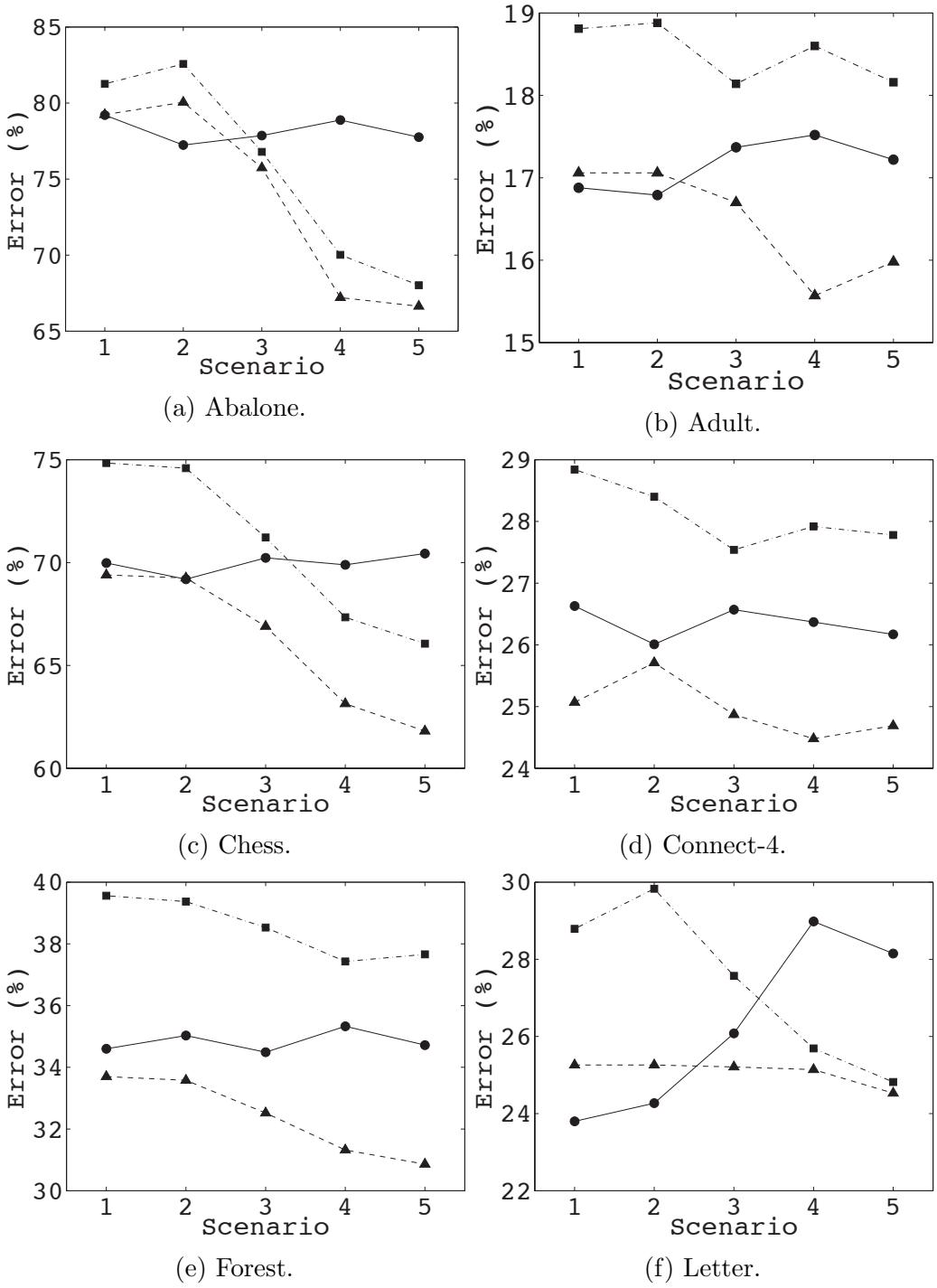
(d) Poker.



(e) Shuttle.

(f) Waveform.

Figure 10.5: Maximum difference between sites in terms of probability of occurrence of samples; covariate shift. The incremental skewness for the different scenarios is represented as stacked bars. Part 2 of 2.



(a) Abalone.

(b) Adult.

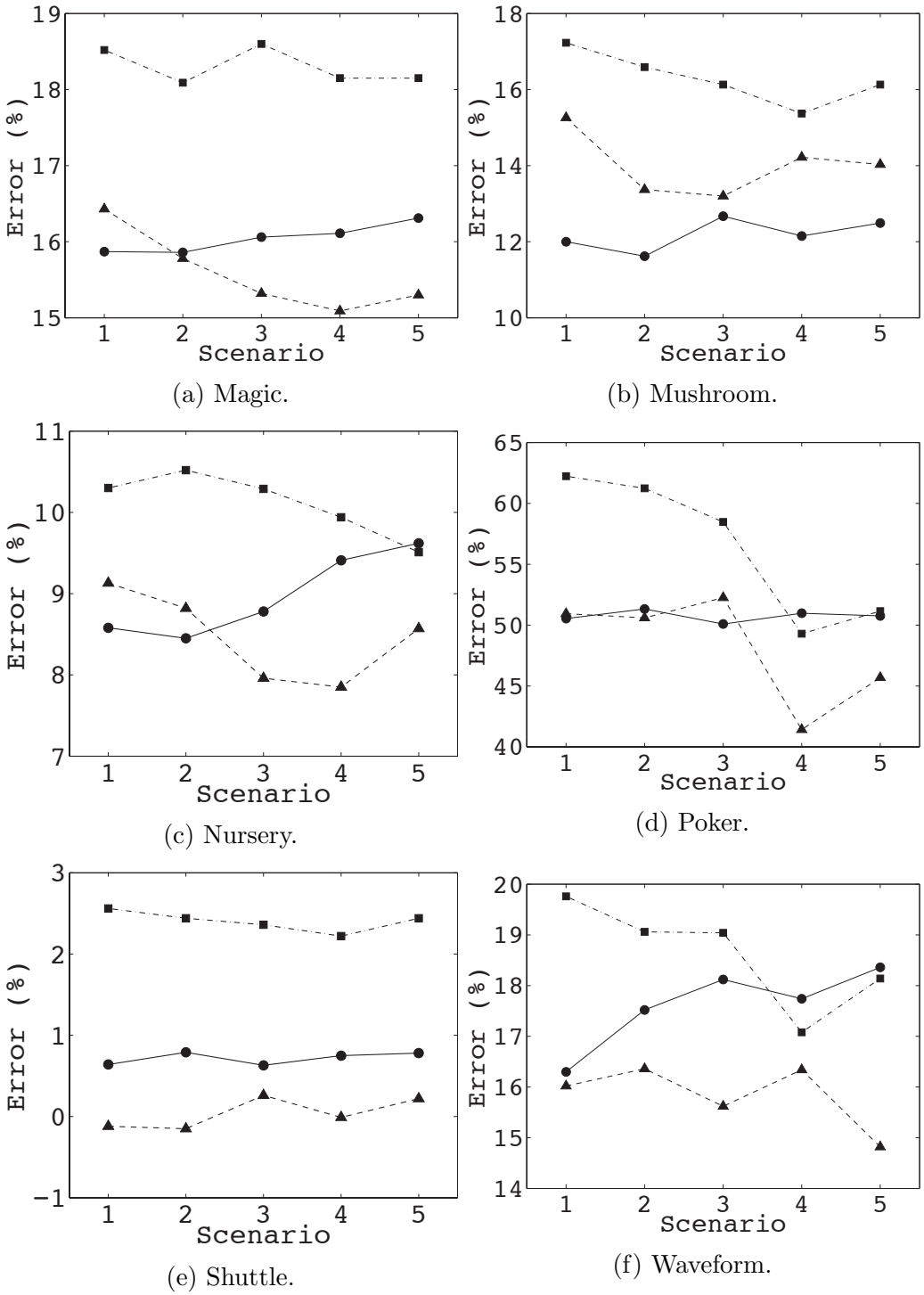
(c) Chess.

(d) Connect-4.

(e) Forest.

(f) Letter.

Figure 10.6: Test performance error in complete subpopulation interdependence (panmixia) displayed with solid lines and circle marks, complete subpopulation independence (no migration) displayed with dashed lines and square marks, and island model displayed with dotted-dashed lines and triangular marks. Part 1 or 2.



(a) Magic.

(b) Mushroom.

(c) Nursery.

(d) Poker.

(e) Shuttle.

(f) Waveform.

Figure 10.7: Test performance error in complete subpopulation interdependence (panmixia) displayed with solid lines and circle marks, complete subpopulation independence (no migration) displayed with dashed lines and square marks, and island model displayed with dotted-dashed lines and triangular marks. Part 2 or 2.

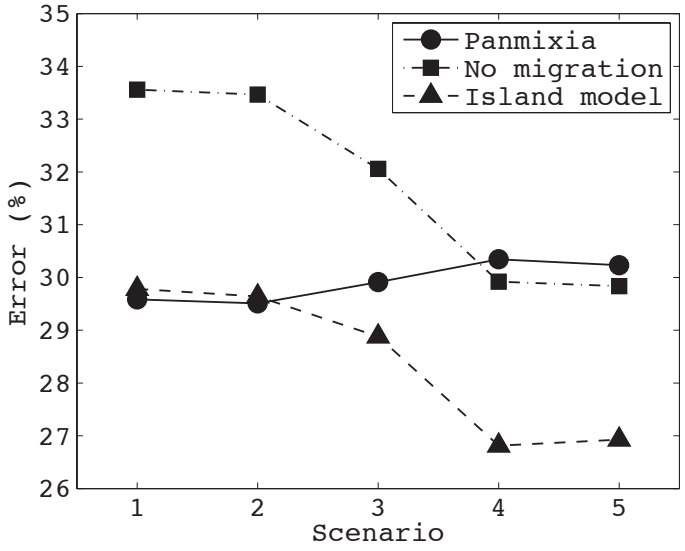


Figure 10.8: Average test performance error in complete subpopulation interdependence (panmixia), complete subpopulation independence (no migration), and island model.

combining more knowledge from different sites improves the performance of the overall model. Note that, if we would keep increasing the skewness of data to the very limit, the performance of the island model and the model with no migration would match. At that point, the skewness would be that high that sharing knowledge between sites would not be worth it, i.e. actually, no migration. Finally, note that the island model evaluated in this experimental section is just a framework for dealing with data skewness. Thus, this very same framework can be applied to any other distributed training algorithm, as the two learning algorithms based on neural networks proposed in the previous part of this thesis.

Part IV

Bridging the Gap

Chapter 11

Scaling Up Learning on Skewed Data

In this chapter, we will briefly discuss the possibility of merging some of the methods and techniques introduced in this thesis, in order to deal with distributed systems where the challenge is not only the semantics of the different partitions of data, but also the volume of data at each partition.

11.1 LEARNING FROM SKEWED DATA, FASTER

In the second part of this thesis, several techniques were introduced in order to scale up learning algorithms. In general terms, the learning algorithms presented aimed to infer a global learner that approximates the results one would get from a single, joint data source. Thus, these algorithms have been designed with accuracy and speed in mind. On the other hand, in the third part of this thesis, we introduced several concepts related to the distribution of the partitions of data. Real-world distributed data sets almost always present quite strong differences between their partitions. Under this view, the distribution of data was not treated as a mere technical issue, just because it had deeper implications. The algorithms presented in the third part have been designed with this circumstance in mind.

Thus, for argument's sake, taking the example of buying patterns in different supermarkets from different countries, we may suspect that these patterns could be different for different countries –so it would be desirable to implement a distributed learning system that takes into account this issue, using any of the techniques proposed in the third part of this thesis– but we may also suspect that the volume of data in the different nodes could be very large –so it would be desirable to implement a distributed learning system that takes into account this other issue, using any of the algorithms proposed

in the second part of this thesis. Note, however, that most of the methods proposed in this thesis are fairly generic and could be applied in combination with different strategies. Thus, using the different algorithms introduced in the previous chapters, it would be easy to extend the implementation of these algorithms for taking into account these two issues: large volume of data and data shift between partitions. The idea would be learning from skewed data, but faster –in a more scalable manner.

A general framework for achieving this would be a two-level approach, where any of the algorithms proposed for scaling up learning is used in the first, local stage; and any of the algorithms proposed for dealing with skewness of data is used in the second, global stage. For example, if we take the island model introduced in Chapter 10, in the experimental section we basically trained a batch implementation of the FVQIT in each distributed partition. However, if the volume of data had been very large, we could have used the distributed implementation of the FVQIT introduced in Chapter 6 in each local partition. Note that the output of this algorithm is equivalent in terms of model to the output of the batch algorithm. This is just an example, it would be possible to make any other combination of algorithms. The idea is that most of the techniques introduced in this thesis may be seen as layers in a learning model. If one needs to tackle some specific problem for a specific task, it would be a matter of adding the corresponding layer that tackles the specific problem. Based on the results presented in this thesis, where the distributed implementations of the different algorithms maintain or even improve the performance of the original algorithms, we would expect that adding these different layers to the model will maintain, or even improve, the overall performance of the system. In some way, by stacking these layers, we propagate the advantages of each method to the next layer.

Chapter 12

Final Conclusions of this Thesis

Machine learning aims to extract knowledge from data. Learning algorithms enable a wide range of applications, from everyday tasks to bleeding edge applications. In the age of *Big Data*, with datasets rapidly growing in size and complexity, machine learning techniques are fast becoming a core component of large-scale data processing pipelines. Scalability has become one of those core concept of Big Data. In the past, the theory and practice of machine learning have been focused on monolithic data sets from where learning algorithms generate a single model. Nowadays, several sources produce data creating environments with several distributed data sets. Also big datasets collected in a central repository in which processing imposes quite high computing requirements. Then one actually thinks in distributed processing of the data as a way to have a more powerful computing platform. This thesis was intended to provide a brief and general framework of distributed machine learning, starting by walking through the reasons for scaling up machine learning to large data sets, the distributed learning setting, foundations of distributed learning and knowledge to be combined, parallel and distributed computing, current research on distributed machine learning, and assessment of algorithms, from thinking about how to define performance to evaluating the effectiveness of a distributed system.

It is a fact that most traditional learning algorithms cannot look at very large datasets and plausibly find a good solution with reasonable requirements of computation. 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 sites wherein the data is processed. In many cases and applications, data is distributed across different sites for several reasons –e.g. privacy, storage cost, computational cost, etc– but the data is considered to be generated by the same, uniform process. Actually, under

this view, distributed data is treated exclusively as a technical issue. Thus, these algorithms should be designed with accuracy and speed in mind. In this line of research, one part of the thesis was committed with four novel distributed learning algorithms able to learn from very large datasets. The first algorithm was a novel distributed training algorithm based on single-layer neural networks and genetic algorithms. This algorithm showed good performance for practical applications, but single-layer neural networks are only capable of approximating linear problems, which may be enough in some scenarios, but not in some others. On the other hand, the universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. Thus, we expanded the concepts developed for the previous algorithm to neural networks with one hidden layer. These two algorithms are fast and accurate, making possible privacy-preserving classification by not exchanging raw data across distributed locations but only the neural networks. Furthermore, they minimize communications by using this approach since the size of the neural networks is negligible in comparison with the size of the raw data. Also, these two algorithms are able to predict regression and classification problems. The next algorithm we introduced in this thesis is specifically designed for classification; a novel distributed learning algorithm built upon the Frontier Vector Quantization based on Information Theory (FVQIT) method. The FVQIT was very effective in classification problems but it shows poor training time performance. Thus, distributed learning was appropriate to speed up training. Finally, we developed a novel distributed one-class classification approach based on an extension of the ν-SVM method. In this novel method, several models were considered, each one determined using a given local data partition on a processor, and the goal is to find a global model. The cornerstone of this method was the novel mathematical formulation that makes the optimization problem separable whilst avoid some data points considered as outliers in the final solution. This is particularly interesting and important because the decision region generated by the method will be unaffected by the position of the outliers and will fit the data in a more natural manner.

In the next part of this thesis, the problem of learning in a distributed machine learning setting was made more challenging. This part was committed to develop algorithms able to deal with skewed data as usually happens in real-world distributed datasets that quite often present quite strong differences between their partitions, e.g. buying patterns in different supermarkets from different countries. Yet in spite of its importance, we think it has not been fully considered in the literature. In this part of the thesis, we pre-

sented different techniques for learning under these circumstances. First, we expanded the operation of the two distributed learning algorithms for single-layer and two-layer artificial neural networks introduced in the previous part of the thesis. With the aim of overcoming the issue of the algorithms deteriorating their performance when the distributions of data show skewness, we took into account this skewness in order to propose several improvements of the algorithms, based on distributing the computation of the genetic algorithm. This modification of the algorithms led to better performance when the distributions of data were different across the different locations of the data. However, there was still something missing. In general terms, distributed learning algorithms aim to infer a global learner that approximates the results one would get from a single, joint data source. Note that this approach assumes that there is a single, global model that could be induced from the distributed locations. Then, we introduced a novel proposal for distributed learning that requires no assumptions about the statistical properties of the partitions of data. This novel algorithm is based on the island model genetic algorithms. In the island model, several populations are kept at the same time. From time to time, individuals move from one population to another by migration. Hence, the various islands maintain some degree of independence and interdependence from each other. This approach showed promising results for different levels of skewness of data.

Finally, in the last part of the thesis, we focused our attention on bridging the gap between the two scenarios we discussed during the thesis. For some applications, it is quite often to find skewed partitions of data where the partitions of data are also very large. In this scenarios, we proposed to combine both approaches we presented in this thesis in order to deal with the large volumes of data within partitions and the skewness of data across partitions. One of the advantages of most of the methods presented in this thesis is that they are quite generic and can be applied with a wide variety of algorithms and applications, including the combination of these methods in different layers in order to tackle different challenges of the problem at hand.

References

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the afips spring joint computing conference* (pp. 483–485).

Apache, S. F. (2014a). *Apache spark - lightning-fast cluster computing.* Retrieved from `http://spark.incubator.apache.org/`

Apache, S. F. (2014b). *Welcome to apache^{TM}hadoop®.* Retrieved from `http://hadoop.apache.org/`

Bailey, A. (2001). *Class-dependent features and multicategory classification.* Unpublished doctoral dissertation, University of Southampton.

Bekkerman, R., Bilenko, M., & Langford, J. (2011). *Scaling up machine learning: Parallel and distributed approaches.* Cambridge University Press.

Bilgin, G., Erturk, S., & Yildirim, T. (2011). Segmentation of hyperspectral images via subtractive clustering and cluster validation using one-class support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*, *49*(8), 2936-2944.

Bishop, C. M. (1995). *Neural networks for pattern recognition.* Oxford university press.

Bishop, C. M. (2006). *Pattern recognition and machine learning.* Springer.

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

Bottou, L. (1991). Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, *91*(8).

Bramer, M. (2013). Dealing with large volumes of data. In *Principles of data mining* (pp. 189–208). Springer.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L. (1998). *Out-of-bag estimation* (Tech. Rep.). University of California at Berkeley.

Breiman, L. (1999). Pasting small votes for classification in large databases and on-line. *Machine Learning*, *36*(1-2), 85–103.

Bronshtein, I. N., Semendiaev, K. A., & Hirsch, K. A. (1985). *Handbook of mathematics.* Van Nostrand Reinhold New York.

Cabral, G. G., & Oliveira, A. L. I. (2011). A novel one-class classification method based on feature analysis and prototype reduction. In *Ieee international conference on systems, man, and cybernetics (smc)* (p. 983 - 988).

Cabral, G. G., & Oliveira, A. L. I. (2012). One-class classification through optimized feature boundaries detection and prototype reduction. In

Artificial neural networks and machine learning (icann 2012). lecture notes in computer science (Vol. 7552, p. 693-700).

Caragea, D., Silvescu, A., & Honavar, V. (2001). Analysis and synthesis of agents that learn from distributed dynamic data sources. In *Emergent neural computational architectures based on neuroscience* (pp. 547–559). Springer.

Caragea, D., Silvescu, A., & Honavar, V. (2004). A framework for learning from distributed data using sufficient statistics and its application to learning decision trees. *International Journal of Hybrid Intelligent Systems*, *1*(1), 80–89.

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

Casale, P., Pujol, O., & Radeva, P. (2011). Approximate convex hulls family for one-class classification. In *10th international workshop on multiple classifier systems* (pp. 106—115).

Casale, P., Pujol, O., & Radeva, P. (2014). Approximate polytope ensemble for one-class classification. *Pattern Recognition*, *47*(2), 854 - 864.

Castillo, E., Conejo, A. J., Pedregal, P., Garcia, R., & Alguacil, N. (2011). *Building and solving mathematical programming models in engineering and science* (Vol. 62). Wiley. com.

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

Castillo, E., Guijarro-Berdiñas, B., Fontenla-Romero, O., & Alonso-Betanzos, A. (2006). A very fast learning method for neural networks based on sensitivity analysis. *The Journal of Machine Learning Research*, *7*, 1159–1182.

Castillo, E., Hadi, A. S., Conejo, A., & Fernández-Canteli, A. (2004). A general method for local sensitivity analysis with application to regression models and other optimization problems. *Technometrics*, *46*(4), 430–444.

Castillo, E., Peteiro-Barral, D., Berdiñas, B. G., & Fontenla-Romero, O. (2015). Distributed one-class support vector machine. *International journal of neural systems*, *25*(07), 1550029.

Ceri, S., Bozzon, A., Brambilla, M., Della Valle, E., Fraternali, P., & Quarteroni, S. (2013). An introduction to information retrieval. In *Web information retrieval* (pp. 3–11). Springer.

Chan, P. K., & Stolfo, S. J. (1993a). Experiments on multistrategy learning by meta-learning. In *Proceedings of the second international conference on information and knowledge management* (pp. 314–323).

Chan, P. K., & Stolfo, S. J. (1993b). Toward parallel and distributed learning by meta-learning. In *Aaai workshop in knowledge discovery in databases* (pp. 227–240).

Chan, P. K., & Stolfo, S. J. (1995). A comparative evaluation of voting and meta-learning on partitioned data. In *International conference on machine learning* (pp. 90–98).

Chan, P. K., & Stolfo, S. J. (1997). On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, *8*(1), 5–28.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3), 1-58.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*, 1–27.

Chawla, N. V., Hall, L. O., Bowyer, K. W., Moore Jr, T., & Kegelmeyer, W. P. (2002). Distributed pasting of small votes. In *Multiple classifier systems* (pp. 52–61). Springer.

Cheung, D. W., & Xiao, Y. (1999). Effect of data distribution in parallel mining of associations. *Data Mining and Knowledge Discovery*, *3*(3), 291–314.

Clifton, L. A., Clifton, D. A., Zhang, Y., Watkinson, P., Tarassenko, L., & Yin, H. (2014). Probabilistic novelty detection with support vector machines. *IEEE Transactions on Reliability*, *63*(2), 455–467.

Collobert, R., & Bengio, S. (2001). Svmtorch: Support vector machines for large-scale regression problems. *The Journal of Machine Learning Research*, *1*, 143–160.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, *13*(1), 21–27.

Cyganek, B. (2008). Color image segmentation with support vector machines: applications to road signs detection. *International Journal of Neural Systems*, *18*(04), 339–345.

Das, K., Bhaduri, K., & Votava, P. (2011). Distributed anomaly detection using 1-class SVM for vertically partitioned data. *Statistical Analysis and Data Mining*, *4*(4), 393-406.

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Dempster, A. P., Laird, N. M., Rubin, D. B., et al. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, *39*(1), 1–38.

Désir, C., Bernard, S., Petitjean, C., & Heutte, L. (2013). One class random forests. *Pattern Recognition*, *46*(12), 3490 - 3506.

Dietterich, T. (2000). Ensemble methods in machine learning. *Multiple classifier systems*, 1–15.

Dietterich, T. G. (2002). Machine learning for sequential data: A review. In *Structural, syntactic, and statistical pattern recognition* (pp. 15–30). Springer.

Fan, W., Stolfo, S. J., & Zhang, J. (1999). 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* (pp. 362–366).

Fernández-Francos, D., Martínez-Rego, D., Fontenla-Romero, O., & Alonso-Betanzos, A. (2013). Automatic bearing fault diagnosis based on one-class nu-svm. *Computers & Industrial Engineering*, *64*(1), 357 - 365.

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

Frank, A., & Asuncion, A. (2010). *UCI machine learning repository.* Retrieved from `http://archive.ics.uci.edu/ml`

Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory* (pp. 23–37).

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Icml* (Vol. 96, pp. 148–156).

Freund, Y., Schapire, R. E., Singer, Y., & Warmuth, M. K. (1997). Using and combining predictors that specialize. In *Proceedings of the twenty-ninth annual acm symposium on theory of computing* (pp. 334–343).

Gama, J., Rodrigues, P., & Sebastião, R. (2009). Evaluating algorithms that learn from data streams. In *Proceedings of the 2009 acm symposium on applied computing* (pp. 1496–1500).

Giacinto, G., & Roli, F. (2001). Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing*, *19*(9), 699–707.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning.

Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, *3*(2), 95–99.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, *3*(5), 493–530.

Gorges-Schleuter, M. (1991). Explicit parallelism of genetic algorithms through population structures. In *Parallel problem solving from na-*

ture (pp. 150–159). Springer.

Grosso, P. B. (1985). *Computer simulations of genetic adaptation: parallel subcomponent interaction in a multilocus model*. Unpublished doctoral dissertation, Ann Arbor, MI, USA. (AAI8520908)

Guo, Y., Rueger, S., Sutiwaraphun, J., & Forbes-Millott, J. (1997). Meta-learning for parallel data mining. In *Proceedings of the seventh parallel computing workshop* (pp. 1–2).

Guo, Y., & Sutiwaraphun, J. (1999). Probing knowledge in distributed data mining. In *Methodologies for knowledge discovery and data mining* (pp. 443–452). Springer.

Hansen, L., & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12*(10), 993–1001.

Hecht-Nielsen, R. (1990). Neurocomputing.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Hollander, M., & Wolfe, D. (1999). Nonparametric statistical methods.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359–366.

Hsu, J. (1996). *Multiple comparisons: theory and methods*. Chapman & Hall/CRC.

Hughes, G. (1968). On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, *14*(1), 55–63.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural computation*, *3*(1), 79–87.

Jiang, J. (2008). A literature survey on domain adaptation of statistical classifiers. *online, Mar*.

Kargupta, H., Byung-Hoon, D., & Johnson, E. (1999). Collective data mining: A new perspective toward distributed data analysis.

Kemmler, M., Rodner, E., Wacker, E.-S., & Denzler, J. (2013). One-class classification with gaussian processes. *Pattern Recognition*, *46*(12), 3507 - 3518.

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

Kittler, J., Hatef, M., Duin, R. P., & Matas, J. (1998). On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *20*(3), 226–239.

Krawczyk, B., & Woźniak, M. (2014). Diversity measures for one-class classifier ensembles. *Neurocomputing*, *126*, 36-44.

Krawczyk, B., Woźniak, M., & Cyganek, B. (2014). Clustering-based ensembles for one-class classification. *Information Sciences*, *264*, 182195.

Krzanowski, W. J., & Krzanowski, W. (2000). *Principles of multivariate analysis*. Clarendon.

Kumar, V., & Gupta, A. (1994). Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, *22*, 379–391.

Kumar, V., & Rao, V. N. (1987). Parallel depth first search. *International Journal of Parallel Programming*, *16*(6), 501–519.

Lai, C., Tax, D. M. J., Duin, R. P. W., Pekalska, E., & Paclík, P. (2004). A study on combining image representations for image classification and retrieval. *International Journal of Pattern Recognition and Artificial Intelligence*, *18*(5), 867–890.

Lazarevic, A., & Obradovic, Z. (2002). Boosting algorithms for parallel and distributed learning. *Distributed and Parallel Databases*, *11*(2), 203–229.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the ieee* (Vol. 86, p. 2278-2324).

Lehn-Schiøler, T., Hegde, A., Erdogmus, D., & Principe, J. C. (2005). Vector quantization using information theoretic concepts. *Natural Computing*, *4*(1), 39–51.

Liu, L., & Özsu, M. T. (2009). *Encyclopedia of database systems*. Springer Berlin, Heidelberg, Germany.

Mahadevan, S., & Shah, S. L. (2009). Fault detection and diagnosis in process data using one-class support vector machines. *Journal of Process Control*, *19*(10), 1627-1639.

Manevitz, L., & Yousef, M. (2007). One-class document classification via neural networks. *Neurocomputing*, *70*(7–9), 1466 - 1481.

Martínez-Rego, D., Castillo, E., Fontenla-Romero, O., & Alonso-Betanzos, A. (2013). A minimum volume covering approach with a set of ellipsoids. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *35*(12), 2997–3009.

Møller, M. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, *6*(4), 525–533.

Moré, J. J. (1978). The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis* (pp. 105–116). Springer.

Moreira, O., Valente, F., & Bekooij, M. (2007). Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proceedings of the 7th acm & ieee international conference on embedded software* (pp. 57–66).

Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., & Herrera, F. (2012). A unifying view on dataset shift in classification.

Pattern Recognition, *45*(1), 521–530.

Moya, M., & Hush, D. (2013). Network constraints and multi- objective optimization for one-class classification. *Neural Networks*, *9*(3), 463—474.

Parthasarathy, S., & Ogihara, M. (2000). Clustering distributed homogeneous datasets. *Principles of Data Mining and Knowledge Discovery*, 566–574.

Peteiro-Barral, D., Bolon-Canedo, V., Alonso-Betanzos, A., Guijarro-Berdinas, B., & Sanchez-Marono, N. (2012). Scalability analysis of filter-based methods for feature selection. *Advances in Smart Systems Research*, *2*(1), 21–26.

Peteiro-Barral, D., & Guijarro-Berdiñas, B. (2013). *A distributed learning algorithm based on frontier vector quantization and information theory.*

Peteiro-Barral, D., Guijarro-Berdinas, B., & Pérez-Sánchez, B. (2011). Distributed learning on nonuniform class-probability distributions based on genetic algorithms and artificial neural networks. In *Hybrid intelligent models and applications (hima), 2011 ieee workshop on* (pp. 54–60).

Peteiro-Barral, D., Guijarro-Berdinas, B., & Pérez-Sánchez, B. (2012). Learning from heterogeneously distributed data sets using artificial neural networks and genetic algorithms. *Journal of Artificial Intelligence and Soft Computing Research*, *2*(1), 5–20.

Peteiro-Barral, D., Guijarro-Berdinas, B., Pérez-Sánchez, B., & Fontenla-Romero, Ó. (2011). A distributed learning algorithm based on two-layer artificial neural networks and genetic algorithms. In *Esann.*

Peteiro-Barral, D., Guijarro-Berdiñas, B., Pérez-Sánchez, B., & Fontenla-Romero, O. (2013). A comparative study of the scalability of a sensitivity-based learning algorithm for artificial neural networks. *Expert Systems with Applications*, *40*(10), 3900–3905.

Porto-Díaz, I., Martínez-Rego, D., Alonso-Betanzos, A., & Fontenla-Romero, O. (2012). Information theoretic learning and local modeling for binary and multiclass classification. *Progress in Artificial Intelligence*, 1–14.

Predd, J. B., Kulkarni, S., & Poor, H. V. (2006). Distributed learning in wireless sensor networks. *Signal Processing Magazine*, *23*(4), 56–69.

Principe, J., Xu, D., Zhao, Q., & Fisher, J. (2000). Learning from Examples with Information Theoretic Criteria. *The Journal of VLSI Signal Processing*, *26*(1), 61–77.

Principe, J. C. (2010). *Information theoretic learning: Renyi's entropy and kernel perspectives.* Springer.

Prodromidis, A., Chan, P., & Stolfo, S. (2000). Meta-learning in distributed data mining systems: Issues and approaches. *Advances in distributed*

and parallel knowledge discovery, *3*.

Provost, F. (2000). Distributed data mining: Scaling up and beyond. *Advances in distributed and parallel knowledge discovery*, 3–29.

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

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (2009). *Dataset shift in machine learning*. The MIT Press.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural Computation*, *13*, 1443—1471.

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, *90*(2), 227–244.

Shipp, C. A., & Kuncheva, L. I. (2002). Relationships between combination methods and measures of diversity in combining classifiers. *Information fusion*, *3*(2), 135–148.

Skalak, D. B. (1996). The sources of increased accuracy for two proposed boosting algorithms. In *Proc. american association for artificial intelligence, aaai-96, integrating multiple learned models workshop* (Vol. 1129, p. 1133).

Sonnenburg, S., Franc, V., Yom-Tov, E., & Sebag, M. (2009). PASCAL Large Scale Learning Challenge. *Journal of Machine Learning Research*.

Storkey, A. (2009). When training and test sets are different: characterizing learning transfer. *Dataset Shift in Machine Learning (J. Candela, M. Sugiyama, A. Schwaighofer and N. Lawrence, eds.). MIT Press, Cambridge, MA*, 3–28.

Subhlok, J., & Vondran, G. (1996). Optimal latency-throughput tradeoffs for data parallel pipelines. In *Proceedings of the eighth annual acm symposium on parallel algorithms and architectures* (pp. 62–71).

Tang, E. K., Suganthan, P. N., & Yao, X. (2006). An analysis of diversity measures. *Machine Learning*, *65*(1), 247–271.

Tax, D. (2014, July). *DDtools, the data description toolbox for Matlab.* (version 2.1.1)

Tax, D. M. J., & Duin, R. P. W. (1999). Support vector domain description. *Pattern Recognition Letters*, *20*(11-13), 1191—1199.

Ting, K. M., & Witten, I. H. (1999). Issues in stacked generalization. *Journal of Artificial Intelligence Research*, *10*, 271–289.

To, C., & Elati, M. (2013). A parallel genetic programming for single class classification. In *Proceedings of the 15th annual conference companion on genetic and evolutionary computation (gecco)* (p. 1579-1586).

Tsoumakas, G., Angelis, L., & Vlahavas, I. (2004). Clustering classifiers for knowledge discovery from physically distributed databases. *Data & Knowledge Engineering*, *49*(3), 223–242.

Tsoumakas, G., & Vlahavas, I. (2002a). Distributed data mining of large classifier ensembles. In *Proceedings companion volume of the second hellenic conference on artificial intelligence* (pp. 249–256).

Tsoumakas, G., & Vlahavas, I. (2002b). Effective stacking of distributed classifiers. In *Ecai 2002: 15th european conference on artificial intelligence* (p. 340).

Tsoumakas, G., & Vlahavas, I. (2009). Distributed data mining. *Database Technologies: Concepts, Methodologies, Tools, and Applications*, 157–171.

Urban, P., Défago, X., & Schiper, A. (2001). Neko: A single environment to simulate and prototype distributed algorithms. In *15th international conference on information networking* (pp. 503–511).

Verma, A., Llora, X., Goldberg, D. E., & Campbell, R. H. (2009). Scaling genetic algorithms using mapreduce. In *Ninth international conference on intelligent systems design and applications (isda'09).* (pp. 13–18).

Vydyanathan, N., Catalyurek, U. V., Kurc, T. M., Sadayappan, P., & Saltz, J. H. (2007). Toward optimizing latency under throughput constraints for application workflows on clusters. In *Euro-par* (pp. 173–183). Springer.

Walton, C. B., Dale, A. G., & Jenevein, R. M. (1991). A taxonomy and performance model of data skew effects in parallel joins. In *Proceedings of the seventeenth international conference on very large data bases, barcelona, spain.*

Wang, L.-X. (1992). Fuzzy systems are universal approximators. In *Ieee international conference on fuzzy systems* (pp. 1163–1170).

Weiss, S., & Kulikowski, C. (1991). *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems.* Morgan Kaufmann, San Francisco.

Whitley, D., Rana, S., & Heckendorn, R. B. (1999). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, *7*, 33–48.

Wirth, R., Borth, M., & Hipp, J. (2001). When distribution is part of the semantics: A new problem class for distributed knowledge discovery. In *Proceedings of the pkdd 2001 workshop on ubiquitous data mining for mobile and distributed environments* (pp. 56–64).

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

Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, Ú., Gunda, P. K., &

Currey, J. (2008). Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *Symposium on operating systems design and implementation* (Vol. 8, pp. 1–14).

Yule, G. U. (1900). On the association of attributes in statistics: with illustrations from the material of the childhood society, &c. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, *194*(252-261), 257–319.

Zaki, M. J. (2000). Parallel and distributed data mining: An introduction. In *Large-scale parallel data mining* (pp. 1–23). Springer.

Zaki, M. J., & Ho, C.-T. (2000). *Large-scale parallel data mining* (No. 1759). Springer.

Zhu, F., Ye, N., Yu, W., Xu, S., & Li, G. (2014). Boundary detection and sample reduction for one-class support vector machines. *Neurocomputing*, *123*(10), 166-173.

Appendices

Appendix A

Resumen en Castellano

Según los últimos estudios, el volumen de datos generados globalmente aumenta un 40% cada año. Así, algunas estimaciones sitúan el número de documentos en la Web en torno a los 550 billones. El ritmo al que en la actualidad se están almacenando datos en prácticamente todas las industrias e instituciones, no tiene precedente en la historia, y ha creado la posibilidad de utilizar estos datos con el objetivo de extraer información y conocimiento de ellos. Sin embargo, el volumen de datos es tan grande que la utilización de métodos manuales para su análisis es simplemente inconcebible. Así, empresas y organizaciones han tenido que hacer uso de técnicas y tecnologías más avanzadas, siendo el aprendizaje automático una de las técnicas más populares en la actualidad.

El objetivo fundamental del aprendizaje automático es extraer conocimiento a partir de datos, y tiene sus cimientos sobre conceptos de las ciencias de la computación, estadística, probabilidad, y optimización. Los diferentes algoritmos de aprendizaje automático posibilitan y aumentan el rango de aplicaciones disponibles en la actualidad, desde aplicaciones relacionadas con tareas diarias hasta aplicaciones de vanguardia y alta tecnología. En la era del *Big Data*, o datos masivos, con conjuntos de datos aumentando frenéticamente en tamaño y complejidad, las diferentes técnicas del aprendizaje automático se han convertido, en los últimos años, en los componentes fundamentales de los grandes sistemas de procesamiento de datos. Así, la escalabilidad de estos algoritmos se ha convertido en uno de los conceptos centrales del Big Data. Sin embargo, los algoritmos de aprendizaje automático más utilizados se han desarrollado en décadas pasadas, dónde el único problema con el volumen de datos venía de la escasez de los mismos. La escalabilidad no era un problema. Así, estos algoritmos son iterativos, hacen múltiples pasadas sobre el conjunto de datos, y necesitan mantener el conjunto de datos, completo, en memoria principal. En la actualidad,

con conjuntos de datos alcanzando fácilmente la escala de los terabytes (10^{12} bytes), incluso una única pasada sobre los datos es muy costosa computacionalmente, y, definitivamente, muchos conjuntos de datos son demasiado grandes para almacenarse enteramente en memoria principal. Este hecho es un buen ejemplo que demuestra la necesidad de implementar algoritmos de aprendizaje máquina que consideren la escalabilidad de los mismos como un tema central en su desarrollo.

En los últimos tiempos, la expansión de plataformas de computación de alto rendimiento, de arquitectura distribuida y computación paralela ha representado un hecho crucial para equilibrar el nivel de recursos necesarios y cubrir la demanda de computación que ha representado el aumento masivo en volumen y complejidad de los datos. Pero los algoritmos de aprendizaje automático tienen que adaptarse a este nuevo paradigma de computación. Esta tesis aborda el reto de desarrollar nuevos algoritmos de aprendizaje distribuido y altamente escalables, y está organizada en cuatro partes relativamente independientes.

I En la primera parte, se describe un marco general para el desarollo de algoritmos de aprendizaje distribuido, empezando por las razones para mejorar la escalabilidad de algoritmos de aprendizaje automático, continuando por los conceptos básicos y fundamentales de los algoritmos distribuidos, y finalizando con una introducción de entornos de computación paralela y distribuida, en particular el paradigma *MapReduce* –un modelo de programación utilizado para dar soporte a la computación paralela sobre grandes colecciones de datos en *clústers* de computadores. Además, en esta parte, se hace una revisión de diferentes métodos y modelos de aprendizaje distribuido encontrados en la literatura científica. Por último, se hace una revisión de diferentes medidas y procedimientos para la evaluación de algoritmos distribuidos, ya que las medidas utilizadas más habitualmente para la evaluación de algoritmos de aprendizaje automático no son suficientes para cubrir los diferentes ángulos en los que se puede medir el rendimiento de un algoritmo distribuido.

II En la segunda parte, se presentan cuatro nuevos algoritmos de aprendizaje automático distribuido que son capaces de entrenar utilizando conjuntos de datos de gran volumen. En líneas generales, estos cuatro algoritmos aproximan el modelo global que se obtendría si todos los datos de entrenamiento estuviesen disponibles en un único conjunto – aprendizaje automático tradicional. En muchos casos y aplicaciones, los datos se encuentran distribuidos en diferentes localizaciones por varias razones, por ejemplo por privacidad, coste de almacenamiento, coste

computacional, etc; pero se considera que estos datos han sido generados por el mismo proceso. De hecho, con esta interpretación, la distribución de los datos se considera como un hecho exclusivamente técnico. Así, los cuatro algoritmos que se presentan en esta parte de la tesis se han diseñado e implementado principalmente para mejorar la velocidad de entrenamiento de los diferentes modelos de aprendizaje.

— El primero de los algoritmos es una implementación distribuida de un algoritmo de entrenamiento para redes de neuronas artificiales de una capa. Las redes de neuronas artificiales son un paradigma de aprendizaje inspirado en la forma en la que funciona el cerebro de los animales. Una red de neuronas artificiales de una capa está compuesta por una capa de entrada y una capa de salida, donde las entradas están conectadas a las salidas mediante conexiones que tienen asociadas pesos. Estos pesos son ajustados durante la fase de entrenamiento. La red de neuronas utilizada para el desarrollo de este algoritmo distribuido utiliza un algoritmo de aprendizaje que es capaz de calcular los pesos de la red de forma analítica, lo que hace que el proceso de entrenamiento sea muy rápido. Además, este algoritmo presenta cierta propiedades que lo hacen muy apropiado para un entorno distribuido. Sin embargo, todas estas características favorables son consecuencia de la utilización del error cuadrático medio como función de error, en el que el proceso de optimización está dirigido por mínimos cuadrados. Esta función es apropiada en el caso de problemas de regresión, pero en problemas de clasificación presenta ciertas limitaciones que impiden que el algoritmo obtenga un mejor resultado más ajustado a las fronteras de decisión entre clases. Así, con el objetivo de mejorar el rendimiento del algoritmo distribuido en problemas de clasificación, se propone la combinación de los diferentes modelos entrenados de forma local utilizando un algoritmo genético. Los algoritmos genéticos hacen evolucionar una población de individuos, en este caso particular las diferentes redes de neuronas de una capa entrenadas en cada partición del conjunto de datos, sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica: mutaciones y recombinaciones genéticas; así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven hasta la próxima generación, y cuáles los menos aptos, que son descartados. Utilizando estos conceptos el algoritmo genético es capaz de optimizar una función más apropiada para clasificación que el error cuadrático medio, como

podría ser el error de clasificación, mejorando la precisión del algoritmo distribuido para redes de una capa. La desventaja en la implementación de un algoritmo genético durante la combinación de las diferentes redes de una capa es que se añade una sobrecarga en términos de computación al algoritmo, pero esta sobrecarga es relativamente pequeña y en muchos casos no afecta de forma sustancial al rendimiento y la escalabilidad del algoritmo distribuido para redes de neuronas de una capa. Este pequeño coste computacional es el precio a pagar por mejorar la precisión del algoritmo.

— El algoritmo distribuido para redes de una capa muestra un buen rendimiento en términos de tiempo de entrenamiento y error conseguido. Sin embargo, una red de una capa es solamente capaz de aproximar problemas lineales, lo cual puede ser suficiente en algunos escenarios, pero no en todos. Con el objetivo de mejorar el rendimiento del método en términos de error, en el siguiente algoritmo distribuido que se propone en esta tesis, se utiliza una red de neuronas con una capa oculta. Según el teorema de la aproximación universal, una red de neuronas con una capa oculta es suficiente para aproximar cualquier función continua. Así, utilizando este tipo de red de neuronas artificiales y un algoritmo de entrenamiento muy rápido basado en análisis de sensibilidad, implementamos un nuevo algoritmo de aprendizaje distribuido para redes con una capa oculta, o redes de dos capas. El procedimiento que se ha seguido para desarrollar este nuevo algoritmo es muy similar al utilizado en el caso del algoritmo distribuido para redes de una capa. Así, también en este caso, se implementa un algoritmo genético como mecanismo para la mejora del rendimiento, en términos de error, del algoritmo durante la fase de combinación de modelos. Sin embargo, a diferencia del caso anterior, este algoritmo genético también será de utilidad no sólo en problemas de clasificación sino también en problemas de regresión. En el caso específico de clasificación, el problema es idéntico al presentado anteriormente al minimizar el error cuadrático medio; utilizando una función objetivo más acorde como el error de clasificación, es posible mejorar la tasa de error durante el entrenamiento. Por otra parte, añadir una capa adicional a la red de neuronas hace que la función objetivo no sea convexa, existiendo el riesgo de que el proceso de entrenamiento no encuentre el mínimo global de la función, sino simplemente un mínimo local. Este podría ser el caso tanto en clasificación como en regresión, y es aquí cuando la utilización

del algoritmo genético puede potencialmente dirigir el proceso de entrenamiento hacia el mínimo global, o al menos un mínimo local que sea más optimo que el conseguido localmente en las diferentes particiones del conjunto de datos. Los resultados conseguidos con este algoritmo demuestran que el algoritmo genético es efectivo en muchos conjuntos de datos, mejorando los resultados de la simple combinación de resultados locales. El algoritmo de aprendizaje distribuido para redes de dos capas muestra un mejor rendimiento en términos de error que el algoritmo para redes de una capa, pero el tiempo de entrenamiento de este algoritmo es considerablemente mayor que el anterior. En general, la escalabilidad de un algoritmo es un balance entre el tiempo de entrenamiento y el error cometido.

— El tercer algoritmo presentado en esta tesis es un algoritmo de clasificación que está basado en cuantificación vectorial y teoría de la información, cuyo nombre es FVQIT. La técnica de cuantificación de vectores se usa para dividir el espacio de entradas en diferentes regiones, y para cada región se define un elemento de procesado que la representa. La idea fundamental de este algoritmo es situar los diferentes elementos de procesado en la frontera de separación entre clases. Para ello se utiliza la interpretación física de diferentes partículas interactuando en el espacio de entrada, donde los elementos de procesado son atraídos por los datos pero repelidos por otros elementos de procesado. Así, el objetivo es distribuir de la forma más óptima posible estos elementos de procesados en el espacio de entradas. En el paso siguiente, una vez que los datos han sido asignados a su elemento de procesado más cercano, se entrena una red de neuronas de una capa para clasificar los diferentes ejemplos en la cercanía de los elementos de procesado. En general, la idea es representar la estructura de los datos de entrada utilizando las fronteras entre clases, y a continuación crear las fronteras de decisión entre clases utilizando modelos locales lineales, que son eficientes para entrenar. Este algoritmo tiene una fundamentación sólida y un buen rendimiento en términos de error de clasificación. Sin embargo, el tiempo de entrenamiento es muy elevado. Así, la implementación distribuida de este método es una solución apropiada para acelerar el proceso de entrenamiento. La idea del algoritmo distribuido es entrenar el algoritmo FVQIT en cada partición de datos y, a continuación, combinar las diferentes soluciones locales utilizando dos métodos: un algoritmo genético o un método de agrupamiento estático (*clustering*). La utilización

de algoritmos genéticos para la combinación de resultados obtiene mejores resultados en términos de error pero a costa de un coste temporal y computacional mayor que la utilización de un método de clustering. Así, teniendo en cuenta que los dos métodos tienen ciertas ventajas pero ciertas desventajas, también se propone la combinación de ambos métodos en dos niveles: una primera etapa en la que se reduce el número de elementos de procesado antes de la etapa combinación, y una segunda etapa en la que se combinan todos los resultados locales que han sido previamente reducidos y simplificados. Esta aproximación muestra un buen rendimiento en términos de error de clasificación y tiempo de entrenamiento.

— El cuarto algoritmo de aprendizaje distribuido que se presenta es un algoritmo de clasificación de una única clase, también conocida como detección de anomalías. En aprendizaje automático, en un problema de clasificación de una clase el objetivo es identificar objetos de una clase específica entre todos los objetos posibles, cuando solamente se tiene acceso en el conjunto de entrenamiento a ejemplos de esa clase específica. Este problema es diferente a una tarea de clasificación tradicional donde el objetivo es diferenciar ejemplos entre dos o más clases, pero el conjunto de entrenamiento contiene ejemplos de todas ellas. El algoritmo distribuido presentado en esta tesis para problemas de detección de anomalías es una extensión de una máquina de vectores de soporte que, gracias a una formulación alternativa presentada en esta tesis, permite el aprendizaje distribuido. Además, este método permite evitar algunos ejemplos que se consideran anómalos en la solución final, haciendo que la región de decisión construida por este algoritmo no se vea afectada por la posición de estos ejemplos anómalos y contribuyendo a una región de decisión más ajustada a los datos de entrenamiento.

III La tercera parte de esta tesis toma una dirección ligeramente diferente a los algoritmos presentados en la segunda parte. Estos algoritmos habían sido diseñados e implementados para mejorar la escalabilidad, reduciendo el tiempo de entrenamiento y manteniendo, o mejorando, el rendimiento de los algoritmos en términos de error. En esta tercera parte, el problema de aprender de forma distribuida incluye nuevas restricciones y retos que se observan en muchos conjuntos de datos del mundo real. Así, estos conjuntos distribuidos normalmente presentan diferencias significativas en las distribuciones de datos entre sus diferentes particiones. Por ejemplo, los diferentes patrones de comportamiento de clientes y productos comprados en supermercados en

diferentes países. En este caso, la distribución física de los datos no puede considerarse una simple cuestión técnica; tiene más implicaciones y afecta a la semántica de las distribuciones en las diferentes particiones de datos. En esta parte de la tesis se hace una revisión a diferentes métodos para evaluar las distribuciones de los datos en las diferentes particiones: desde el porcentaje de ejemplos pertenecientes a las diferentes clases entre las diferentes particiones, hasta las distribuciones estadísticas de las características de entrada de los conjuntos de datos y, finalmente, a la relación entre las entradas y las salidas de un modelo de aprendizaje. Usando estos conceptos, en esta tesis proponemos dos estrategias generales que permiten gestionar las diferentes distribuciones de datos entre diferentes particiones de datos de una forma más eficiente en términos de error.

— La primera estrategia está relacionada con la ejecución de la etapa de combinación de los diferentes algoritmos distribuidos utilizando un enfoque distribuido. Si asumimos que las distribuciones de los datos entre las diferentes particiones no están sesgadas, la fase de combinación de los modelos locales en los algoritmos presentados en la segunda parte de esta tesis, podría ejecutarse en un subconjunto de datos en cualquiera de las particiones –ya que las propiedades estadísticas de las particiones son las mismas. Sin embargo, si las particiones están sesgadas (por ejemplo, si la proporción de ejemplos pertenecientes a una clase es muy diferente entre las particiones), elegir una partición de datos u otra dirigirá el proceso de entrenamiento hacia un conjunto sesgado de datos, es decir, no será un ejemplo representativo del universo del problema en cuestión. Para superar esta limitación, se propone la implementación distribuida de los métodos de combinación presentados en la segunda parte de esta tesis. En particular, en esta parte se presentan dos casos de estudio utilizando los algoritmos de aprendizaje distribuido desarrollados anteriormente para redes de neuronas artificiales de una y dos capas, utilizando una implementación distribuida de los algoritmos genéticos utilizados en la fase de combinación de modelos locales. Así, estos algoritmos genéticos son ejecutados sobre subconjuntos extraídos de todas las particiones de datos, no solamente de una, asegurando una distribución más uniforme de los ejemplos de entrenamiento y evitando sesgos en la fase de combinación.
— La segunda estrategia está relacionada con la optimización de algoritmos distribuidos en las diferentes particiones de datos. En la primera estrategia, todas las particiones de datos contribuían

de igual forma al modelo final de una forma uniforme. Sin embargo, si las distribuciones de datos son muy diferentes entre las diferentes particiones, el proceso de aprendizaje podría optimizarse entrenando modelos específicos en cada *grupo* de particiones. Así, las particiones que son más similares entre ellas, intercambiarán más conocimiento que aquellas particiones que sean muy diferentes. Esta estrategia se implementa en esta tesis utilizando una versión de algoritmo genético conocida como modelo de isla. En este modelo, se ejecuta un algoritmo genético en cada partición, y los cruces entre diferentes individuos se producen solamente entre ciertas islas; aquellas similares entre ellas. El modelo de islas es un balance entre dos modelos más extremos: un algoritmo con conocimiento global como el implementado en la estrategia previa, y un modelo con solo conocimiento local y sin intercambio de conocimiento con ninguna otra partición del conjunto de datos. En esta tesis, demostramos la efectividad del modelo de islas utilizando como caso de estudio el algoritmo distribuido implementado sobre el FVQIT. Los resultados muestran el buen rendimiento de esta estrategia tanto en términos de error como de tiempo de entrenamiento.

IV Finalmente, la última parte de la tesis presenta algunas reglas generales para la utilización de los algoritmos, métodos y técnicas presentadas en las dos partes anteriores en un único marco común. Así, si suponemos que las diferentes particiones del conjunto de datos tienen un gran volumen pero además existen diferencias notables entre la semántica de las diferentes particiones, será necesario utilizar varios de los algoritmos presentados en esta tesis. Por ejemplo, si las diferentes particiones del conjunto de datos están sesgadas, será necesario utilizar alguna de las técnicas presentadas en la tercera parte de esta tesis; pero si además las diferentes particiones tienen un volumen de datos demasiado grande, será necesario utilizar alguno de los algoritmos presentados en la segunda parte de esta tesis (distribuyendo las diferentes particiones en datos en más particiones sin fuese necesario).

En resumen, la implementación de algoritmos de aprendizaje distribuido abre la puerta a multitud de aplicaciones. En una época en la que los conjuntos de datos están creciendo a un ritmo acelerado en tamaño y complejidad, los algoritmos de aprendizaje automático se han convertido en partes fundamentales de los sistemas de procesamiento de datos a gran escala. El propósito de esta tesis ha sido desarrollar un marco general para el desarrollo de algoritmos de aprendizaje distribuido, cubriendo las diferentes partes

involucradas en la implementación eficiente de estos algoritmos, desde su concepción hasta su evaluación.

Appendix B

Publications Supporting This Thesis

DISTRIBUTED MACHINE LEARNING SETTING

— Peteiro-Barral, D., Guijarro-Berdiñas, B., & Pérez-Sánchez, B. (2011). Dealing with "Very Large" Datasets-An Overview of a Promising Research Line: Distributed Learning. In International Conference on Agents and Artificial Intelligence (1) (pp. 476-481).

— Peteiro-Barral, D., & Guijarro-Berdiñas, B. (2013). A survey of methods for distributed machine learning. Progress in Artificial Intelligence, 2(1), 1-11.

— Peteiro-Barral, D., Guijarro-Berdiñas, B., Pérez-Sánchez, B., & Fontenla-Romero, O. (2013). A comparative study of the scalability of a sensitivity-based learning algorithm for artificial neural networks. Expert Systems with Applications, 40(10), 3900-3905.

— Peteiro-Barral, D., & Guijarro-Berdiñas, B. (2013, January). A Study on the Scalability of Artificial Neural Networks Training Algorithms Using Multiple-Criteria Decision-Making Methods. In Artificial Intelligence and Soft Computing (pp. 162-173). Springer Berlin Heidelberg.

— Bolón-Canedo, V., Peteiro-Barral, D., Alonso-Betanzos, A., Guijarro-Berdiñas, B., & Sánchez-Maroño, N. (2011). Scalability analysis of ANN training algorithms with feature selection. In Advances in Artificial Intelligence (pp. 84-93). Springer Berlin Heidelberg.

— Peteiro-Barral, D., Bolón-Canedo, V., Alonso-Betanzos, A., Guijarro-Berdiñas, B., & Sánchez-Maroño, N. (2013). Toward the scalability

of neural networks through feature selection. Expert Systems with Applications, 40(8), 2807-2816.

— Peteiro-Barral, D., Bolon-Canedo, V., Alonso-Betanzos, A., Guijarro-Berdinas, B., & Sanchez-Marono, N. (2012). Scalability analysis of filter-based methods for feature selection. Advances in Smart Systems Research, 2(1), 21-26.

SCALING UP LEARNING ALGORITHMS

— Peteiro-Barral, D., Guijarro-Berdinas, B., Pérez-Sánchez, B., & Fontenla-Romero, Ó. (2011). A distributed learning algorithm based on two-layer artificial neural networks and genetic algorithms. In European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (pp. 471-476).

— Peteiro-Barral, D., & Guijarro-Berdiñas, B. (2013). A Distributed Learning Algorithm Based on Frontier Vector Quantization and Information Theory. In International Conference on Artificial Neural Networks (pp. 122-129). Springer Berlin Heidelberg.

— Castillo, E., Peteiro-Barral, D., Berdiñas, B. G., & Fontenla-Romero, O. (2015). Distributed One-Class Support Vector Machine. International journal of neural systems, 25(0), 1550029 (17 pages).

WHEN DISTRIBUTION IS PART OF THE SEMANTICS

— Peteiro-Barral, D., Guijarro-Berdiñas, B., & Pérez-Sánchez, B. (2011). On the effectiveness of distributed learning on different class-probability distributions of data. In Advances in Artificial Intelligence (pp. 114-123). Springer Berlin Heidelberg.

— Peteiro-Barral, D., Guijarro-Berdinas, B., & Pérez-Sánchez, B. (2011, April). Distributed learning on nonuniform class-probability distributions based on genetic algorithms and artificial neural networks. In IEEE Workshop on Hybrid Intelligent Models And Applications (pp. 54-60). IEEE.

— Peteiro-Barral, D., & Guijarro-Berdiñas, B. (2012). An Analysis of Clustering Approaches to Distributed Learning on Heterogeneously Distributed Datasets. In International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (pp. 69-78).

— Peteiro-Barral, D., Guijarro-Berdinas, B., & Pérez-Sánchez, B. (2012). Learning from heterogeneously distributed data sets using artificial neural networks and genetic algorithms. Journal of Artificial Intelligence and Soft Computing Research, 2(1), 5-20.