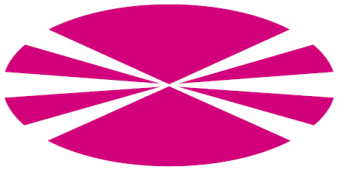


UNIVERSITY OF A CORUÑA

FACULTY OF INFORMATICS

Ph.D program in Computational Science

Doctoral Thesis

Computational learning algorithms for large-scale datasets

Author: Diego Fernández Francos
Advisors: Amparo Alonso Betanzos
Óscar Fontenla Romero

A Coruña, April 2017

April 20, 2017
UNIVERSITY OF A CORUÑA

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

Acknowledgements

Most of all, I would like to thank my advisors, Amparo and Óscar, for your patience and guidance during these years.

To go to work every day with a smile on your face and feel comfortable you need to be surrounded by good people, so a special thanks goes to all my colleagues at LIDIA Group. You are great!

And finally, last but by no means least, I am grateful to my parents José Luis and María Jesús, my brothers Pablo and David, and my girlfriend Alicia, for your support and dedication. I am also grateful to my other family members and friends who have supported me along the way.

Diego Fernández Francos
April 2017

Resumo

Hoxe en día atopámonos soterrados nunha morea de datos. Isto cambiou fundamentalmente a forma na que a información é compartida e puxo de manifesto a necesidade de desenvolver novos métodos eficientes para o procesamento e o almacenamento de grandes cantidades de datos. A aprendizaxe computacional é a área da intelixencia artificial dedicada a estudar algoritmos que poden aprender a partir dos datos, facer previsións ou crear representacións precisas con base nas observacións. Neste contexto, no cal o número de datos crece máis rápido que a velocidade dos procesadores, a capacidade dos algoritmos de aprendizaxe máquina tradicionais vese limitada polo tempo de computación e non polo tamaño da mostra. Ademais, cando se trata de grandes cantidades de datos, os algoritmos de aprendizaxe poden dexenerar o seu rendemento debido ó sobreaxuste e a súa eficiencia decae segundo o tamaño. Polo tanto, a escalabilidade dos algoritmos de aprendizaxe xa non é unha característica desexable senón que se trata de unha propiedade fundamental cando se traballa con conxuntos de datos moi grandes. Existen basicamente tres enfoques diferentes para garantir a escalabilidade dos algoritmos namentres os conxuntos de datos seguen a medrar en tamaño e complexidade: aprendizaxe en tempo real, aprendizaxe non iterativa e aprendizaxe distribuída. Esta tese presenta novos métodos de aprendizaxe computacional escalables e eficientes seguindo os tres enfoques anteriores. En concreto, desenvólvense catro novos algoritmos: (1) O primeiro método mistura selección de características e clasificación en tempo real, a través da adaptación dun filtro convencional e da modificación de un algoritmo incrementábel baseado nunha rede de neuronas de unha capa: (2) O seguinte é un novo clasificador uniclase con base nunha función de custo non iterativa para redes de neuronas autoasociativas que leva a cabo a redución da dimensionalidade na capa oculta pola técnica de Descomposición en Valores Singulares. (3) O terceiro método é un novo clasificador uniclase baseado no *convex hull* para conxuntos de datos distribuídos que reduce a dimensión dos datos do problema e, polo tanto, a complexidade, utilizando proxeccións aleatorias. (4) Por último, preséntase unha versión incremental do algoritmo de clasifición uniclase anterior.

Resumen

Actualmente nos encontramos sumidos en una avalancha de datos. Este hecho ha modificado fundamentalmente la manera en que se comparte la información y ha puesto de manifiesto la necesidad de desarrollar nuevos métodos eficientes para procesar y almacenar grandes cantidades de datos. El aprendizaje computacional es el área de la inteligencia artificial dedicada a estudiar algoritmos que puedan aprender a partir de los datos, hacer predicciones o crear representaciones exactas basadas en las observaciones. En este contexto, en el que el número de datos crece más rápido que la velocidad de los procesadores, la capacidad de los algoritmos tradicionales de aprendizaje máquina se encuentra limitada por el tiempo de computación y no por el tamaño de la muestra. Además, al tratar con gran cantidad de datos, los algoritmos de aprendizaje pueden degenerar su rendimiento debido al sobreajuste y su eficiencia decae de acuerdo con el tamaño. Por lo tanto, la escalabilidad de los algoritmos de aprendizaje ha dejado de ser una característica deseable de los algoritmos de aprendizaje para convertirse en una propiedad crucial cuando se trabaja con conjuntos de datos muy grandes. Existen, básicamente, tres enfoques diferentes para asegurar la escalabilidad de los algoritmos a medida que los conjuntos de datos continúan creciendo en tamaño y complejidad: aprendizaje en tiempo real, aprendizaje no iterativo y aprendizaje distribuido. Esta tesis desarrolla nuevos métodos de aprendizaje computacional escalables y eficientes siguiendo los tres enfoques anteriores. Específicamente, se desarrollan cuatro nuevos algoritmos: (1) El primero combina selección de características y clasificación en tiempo real, mediante la adaptación de un filtro clásico y la modificación de un algoritmo de aprendizaje incremental basado en una red neuronal de una capa. (2) El siguiente consiste en nuevo clasificador uniclase basado en una función de coste no iterativa para redes neuronales autoasociativas que lleva a cabo la reducción de dimensionalidad en la capa oculta mediante la técnica de Decomposición en Valores Singulares. (3) El tercer método es un nuevo clasificador uniclase basado en el cierre convexo para entornos de datos distribuidos que reduce la dimensionalidad del problema y, por lo tanto, la complejidad, mediante la utilización de proyecciones aleatorias. (4) Por último, se presenta una versión incremental del anterior algoritmo de clasificación uniclase.

Abstract

Nowadays we are engulfed in a flood of data. This fact has fundamentally changed the ways that information is shared, and has made it clear that efficient methods for processing and storing vast amounts of data should be put forward. Computational learning theory is the area of artificial intelligence devoted to study algorithms aim at learning from data, making accurate models based on observations. In this context, where data has grown faster than the speed of processors, the capabilities of traditional machine learning algorithms are limited by the computational time rather than by the sample size. Besides, when dealing with large quantities of data, learning algorithms can degenerate their performance due to over-fitting and their efficiency declines in accordance with size. Therefore, the scalability of the learning algorithms has turned from a desirable property into a crucial one when very large datasets are envisioned. There exists, basically, three intersecting approaches to ensure algorithms scalability as datasets continue to grow in size and complexity: online learning, non-iterative learning and distributed learning. This thesis develops new efficient and scalable machine learning methods following the three previous approaches. Specifically, four new algorithms are developed. (1) The first one performs online feature selection and classification at the same time, by the adaptation of a classical filter method and the modification of an online learning algorithm for one-layer neural network. (2) The next one is a new fast and efficient one-class classifier based on a non-iterative cost function for autoassociative neural networks that performs dimensionality reduction in the hidden layer by means of Singular Value Decomposition. (3) The third method is a new one-class convex hull-based classifier for distributed environments that reduces the dimensionality of the problem and hence the complexity by means of Random Projections. (4) Finally, an online version of the previous one-class classification algorithm is presented.

Contents

List of figures

List of tables

CHAPTER 1

Introduction

In 2011, according to the International Data Corporation (IDC), the overall data volume in the world was 1.800 exabytes. From then until 2020, this amount will double every other two years [29, 42]. Nowadays we are engulfed in a flood of data. In general, this explosive increase of data volume has fundamentally changed the ways that information is shared, and has made it clear that efficient methods for processing and storing vast amounts of data should be put forward. Besides, its automatic analysis (data analytics) has become an emerging economic and scientific opportunity. For example, big data is fundamentally changing the way businesses operate. Companies that invest in and successfully acquire value from their data have an obvious advantage over their competitors. Scientific research has also been revolutionized in many fields such as astronomy, bioinformatics, intrusion detection in computer networks, text classification or engineering problems in which the size and number of available datasets is increasing exponentially.

Computational learning theory is an area of artificial intelligence devoted to study the design, development and analysis of machine learning algorithms. In particular, such algorithms aim at learning from data, making accurate predictions or representations based on observations. In this context, where data has grown faster than the speed of processors, the capabilities of machine learning algorithms are limited by the computational time rather than the sample size [16]. A database is considered of large size when: the number of samples is very high; the number of features is very high; or the number of samples and features are very high. Learning methods struggle when dealing with databases with around 10.000.000 data (being data samples $\times$ features). In this thesis we will adopt the term *large-scale* datasets to refer to those high dimensional databases where the number of samples is considerably higher than the number of features.

In theory, it seems logical that having more information leads to better results. However, this is not always the case due to the so-called curse of dimensionality [5]. This phenomenon happens when the dimensionality increases and the time required by the computational learning algorithm to train increases drastically. Besides, when dealing with high amounts of data,

learning algorithms can degenerate their performance due to over-fitting and its efficiency declines in accordance with size. Therefore, the scalability of the learning algorithms has turned from a desirable property into a crucial one when one envisions very large datasets.

Two of the most common tasks in machine learning, the ones in which this thesis is focused, are [10]: 1) classification, where the algorithm assigns unseen samples to a series of categories; and 2) dimensionality reduction, where samples are downsized by mapping them to lower dimensional spaces. According to the nature of the available learning datasets, the previous tasks can also be classified in: (a) supervised learning, where all data is labeled and the algorithms learn to predict the output from the input data; (b) unsupervised learning, where all data is unlabeled and the algorithms learn the inherent structure from the input data; and (c) semi-supervised learning, where some data is labeled but most of it is unlabeled and a variety of supervised and unsupervised techniques can be employed.

When dealing with large databases, an essential aspect is to prepare the data adequately to be processed by the learning algorithms. This has triggered the use of dimensionality reduction procedures as a preprocessing step in order to reduce the dimensionality of data and improve the performance of learning algorithms. It may sound strange to apply dimensionality reduction techniques when datasets have a high number of samples but not of features. However, this methods have proven to be effective in this situation, since they aim at reducing the processing time as well as improving the performance of machine learning algorithms. There are basically two types of dimensionality reduction techniques:

- Feature extraction [46, 78] is a process that creates a new set of features from the original ones through some functional mapping into lower dimensions. It is usually considered as a preprocessing transformation that alters the dimensionality of the problem by reducing it. The main advantages of using feature extraction are the reduction of the algorithm complexity and the savings in computational time. Besides, the new compact set of features helps in the visualization of the data and can provide stronger discriminating power. However, the transformation applied to the original set of features limits the applicability of this techniques to problems where model accuracy is more important than model interpretability. One of the most widely used method is Principal Components Analysis (PCA). It is a statistical tool, which is useful to extract dominant features (principal components) from a set of multivariate data [121]. Feature extraction is common in applications such as signal processing, information retrieval and image processing. In the third and fourth chapters of this thesis two general purpose dimensionality reduction techniques such as Singular Value Decomposition (SVD) [34] and Random Projections

[7] have been used, respectively.

- Feature selection [149] is intended for detecting the relevant features and removing the irrelevant and redundant ones, which implies savings in storage requirements and computational complexity, improving the efficiency of learning algorithms. Feature selection methods can be classified in many ways. One of the most common is the classification into filters, wrappers, embedded, and hybrid methods. There exists a large amount of feature selection methods for supervised and unsupervised types of problems. Examples of very popular algorithms (mainly filters) are: Information Gain [104], Correlation-based Feature Selection (CFS) [49], ReliefF [70] and SVM-RFE (embedded method) [47]. These techniques have been widely used in many application fields such as text mining, image processing, fault diagnosis and bioinformatics [59]. Unlike feature extraction, feature selection preserves the original features, which is interesting when these are needed for interpreting the learned model. In the first part of this thesis we will focus on the well-known χ^2 filter [77], as it is inherently incremental and algorithm-independent. These properties make it computationally simple and fast, being able to handle extremely large-scale datasets.

Classification, the other problem treated in this thesis, is an area of machine learning concerned with identifying the category of a new observation. Traditional methods have employed all data classes to build discriminatory models. Some well-know examples of these techniques are: Artificial Neural Networks (ANNs), Decision Trees and Support Vector Machines (SVMs) [71]. Such supervised methods assume previous knowledge of the class for every dataset sample used to learn the model. However, in real world scenarios that is not always the case. Nowadays, in many fields, the large amounts of data make it impossible for humans to manually label the class for each sample. Hence, unsupervised or semi-supervised classification is needed. A particular case that has seen a rise in the last years is one-class classification, where it is only possible to have data from one class (normal or target class) to train; data from other classes, outlier classes, is very difficult to obtain. The scarcity of outlier examples can be due to several reasons, such as high costs to gather them or the low frequency at which this kind of events occur. This situation is common in a wide variety of real problems such as fault detection in industrial machinery, medical diagnostic, intrusion detection in security systems, video surveillance and document classification, for example. As a result, one-class learning algorithms only use data from a single class to build a model and their goal is to identify data from that class and reject data from all other classes. Examples of classical one-class classification methods are: Parzen Density Estimation (PDE) [84], Minimum Spanning Trees (MST) [60] and Support Vector Data Description (SVDD) [128].

A common scenario in machine learning is the application, as a preprocessing step, of feature selection or feature extraction techniques to classification problems. That is one of the premises for the methods developed in this thesis. The other premise is that this new methods should be scalable and should improve the performance of classical approaches in this new big data scenario. There exists a field in machine learning devoted to this task: large-scale learning.

1.1 Large-scale learning

Traditional machine learning algorithms were designed to extract the most information from the limited amount of data available. However, nowadays, data is being collected at an unprecedented fast pace that the new limiting factor is the inability of learning algorithms to deal with such amounts of data, sometimes distributed in several databases, in a reasonable computational time. For the sake of handling this new situation a new field in machine learning has emerged: large-scale learning [16, 119], concerned with the development of scalable learning algorithms with regard to requirements of computational complexity, memory and time. In recent years, large-scale learning has attracted a great deal of attention. Many publications [111, 143] and even workshops, such as the *PASCAL Large Scale Learning Challenge* [118], were concerned with the efficiency and scalability of classical learning algorithms with respect to computational and storage resources.

There exists, basically, three intersecting approaches to ensure algorithms scalability as datasets continue to grow in size and complexity [146]: 1) online learning, 2) non-iterative learning and 3) distributed learning. These three topics will be covered independently in each of the three main parts in which this thesis is divided and so they have been made as self-contained as possible. Before diving into the specific details of each one, in this introduction a brief insight of the main intentions of the present work in each field is given.

1.1.1 Online learning

Network event logs, telephone call records and surveillance video streams are examples of data streams that flow constantly. Traditional batch learning approaches cannot manage this kind of data, as they create their models by learning on the entire dataset at once. Thus, in order to deal with this situation, algorithms capable of learning one instance at a time are needed. The artificial intelligence field devoted to learn on streaming data is called online learning.

This field has become a trending area in the last few years since it allows to learn a model in an incremental manner. It has been mostly used in those situations in which data becomes available sequentially or it is computationally unfeasible to train over the entire dataset [136]. It is also a useful technique in situations where it is necessary for the algorithm to dynamically adapt to changes in the underlying distribution of data (concept drift) or when data itself is generated as a function of time, e.g. stock price prediction. For these reasons, advances in this field have recently appeared.

Novel online dimensionality reduction techniques are been proposed and research fields such as online feature selection (OFS) are hot topics at the present time [95, 137, 141]. OFS is very useful when a concept drift appears, as this phenomenon may provoke that the relevant set of features changes over time. In spite of the recent progress, online feature selection has not evolved in line with online classification methods. Therefore, in the second chapter of the thesis, we review the problematic of online learning, especially regarding the combination of online feature selection and classification, and propose a novel pipeline that efficiently address both tasks. The new method performs online feature selection and classification at the same time, by the adaptation of a classical filter method and the modification of an online algorithm for one-layer neural network.

1.1.2 Non-iterative learning

Traditionally, learning algorithms operate with all data stored in main memory. When the complexity of the algorithm surpasses the computational resources then the algorithm does not scale well. Furthermore, many classical machine learning algorithms, specially those based on Artificial Neural Networks (ANNs), adjust their free parameters through an iterative training procedure that is repeated over time in order to reduce the model error to a minimum. Most of these iterative methods are based on backpropagation [108] and second-order approaches [93, 116]. Nowadays, large-scale datasets render some iterative training algorithms inapplicable due to their high computational requirements and low convergence speed.

In such cases, the most straightforward solution is to create more efficient methods or increase the efficiency of the existing ones by, for example, simplifying the functions that are trying to be optimized by the learning process. Another way to reduce the complexity of classical iterative-based methods relies on the development of new non-iterative training algorithms to adjust the parameters. However, non-iterative approaches are scarce in the literature, with just a few salient contributions for ANNs [105, 112, 135]. One of those traditional learning

methods that suffer the consequences of very large datasets is the autoencoder [138]. This is a neural network approach for one-class classification that has been successfully applied in many applications fields in the past. However, this classifier relies on traditional iterative learning algorithms to train the network, which is quite time-consuming.

In the third chapter of the thesis, to tackle the slow learning speed in traditional autoencoder, a new fast and efficient one-class classifier based on a non-iterative cost function is presented. Another interesting characteristic of the proposed method is that it allows dimensionality reduction in a very efficient way, extracting meaningful low-dimensional representation by means of the Singular Value Decomposition technique. Furthermore, this method can be parallelized, which makes it more scalable and efficient when dealing with large-scale datasets.

1.1.3 Distributed learning

In general, the advent of big data has contributed to the proliferation of big databases, usually distributed, which automatic analysis is of high interest. However, most existing learning algorithms cannot handle this circumstance. The great majority of them require gathering the several partitions of data in a single node for central processing. However, there exist situations in which this approximation is unfeasible or ineffective [131] since:

- *Storage cost*. The necessary capacity of storage could not be available. For example, the central storage of the data of all the hospitals from one country (medical images, patient records, etc) would require a enormous data warehouse of huge cost.
- *Communication cost.* The cost of efficiently transfer through a network this enormous volume of data.
- *Computational cost*. Learning algorithms can be unable to deal with such volumes of data due to their memory or computational requirements.
- *Private and sensitive data*. The need of preserving the privacy of data makes impossible to share them between distinct locations. Again, patient's records are an example of data that if shared through a network would put privacy into risk.

In order to give solutions to these new problems, a new field of research, distributed learning, has arisen. Distributed learning is drawing a lot of attention lately and is becoming one

of the most promising research lines in machine learning. The problem with most of the existent distributed algorithms is that practically no one of them considers all already mentioned restrictions and conditions that arise when working in these type of environments. That is even more true in the case of one-class learning algorithms.

Therefore, in the fourth chapter of the thesis a new one-class convex hull-based classification algorithm that can scale out over distributed architectures is proposed. This method approximates the high dimensional convex hull decision by means of a dimensionality reduction technique (i.e. random projections) and an ensemble of convex hull decisions in very low dimensions. Furthermore, a method to eliminate the less relevant and redundant random projections in order to obtain a lightweight ensemble model, more scalable and efficient. is also proposed.

1.2 Objectives

The aim of this thesis is to develop new scalable and efficient algorithms that tackle two of the most important stages of machine learning, i.e. data preprocessing by means of dimensionality reduction and classification (especially one-class classification), so that they resolve all the mentioned challenges that big data has introduced. The thesis is divided in three main parts that are covered in the following three self-contained chapters. The objectives for each one of them are described as follows:

1. Online learning.
    - To analyze the fields of online feature selection and online classification.
    - To fill the gap found in the literature and propose a method that efficiently combines online feature selection and classification.
    - To assess the performance and the characteristics of the proposed method.
    - To analyze the characteristics of the proposed algorithm, as for example, the influence in the order of appearance of the data.
2. Non-iterative learning.
    - To study the field of one-class classification.
    - To adapt a classical iterative one-class algorithm for dealing with large-scale datasets.

- To achieve an improvement in the overall computational performance of the method, while maintaining or improving the classification accuracy.
- To analyze the behavior of the algorithm, comparing its performance and scaling properties.

3. Distributed learning.

- To study the new situation where large amounts of data are originally distributed.
- To analyze the state-of-the-art of one-class classification methods for distributed environments.
- To propose a new distributed one-class algorithm that avoids all the problems that gathering all the data into a unique node for processing will cause.
- To maintain or improve the classification performance obtained with the classical monolithic approach.
- To assess the performance and the characteristics of the proposed method, comparing its performance and scaling properties.

1.3 Outlook of this thesis

In this brief introduction we have presented the main topics to be discussed in this dissertation. Chapter 2 gives an introduction to the online learning field and presents a novel unified pipeline for online learning. Chapter 3 addresses the problem of one-class classification with large-scale datasets and proposes a new scalable algorithm based on the classical autoencoder neural network that efficiently performs dimensionality reduction and classification. Chapter 4 introduces the problem of distributed learning and presents a new one-class convex hull-based classification algorithm for distributed environments. Finally, in chapter 5 the main conclusions and contributions of this work are summarized.

CHAPTER 2

A Unified Pipeline for Online Learning: Feature Selection and Classification

During the last years and with increasing frequency, real-time production systems generated tremendous amount of data at unprecedented rates, such as network event logs, telephone call records or sensoring and surveillance video streams. To deal with data streams that flow continuously, classical batch learning algorithms cannot be applied and it is necessary to employ online approaches. Online data mining consists of continuously revise and refine a model by incorporating new data as they arrive [136]. Note that any online method is inherently incremental. This type of learning has been applied in fields such as classification of textual data streams, financial data analysis, credit card fraud protection, traffic monitoring and predicting customer behavior [36, 62, 136].

Most of these applications present a great challenge for machine learning researches due to the high amount of data available. Theoretically, it could seem logical that having more features could lead to better results, but this is not always the case due to the so-called *curse of dimensionality* [5]. This phenomenon happens when the dimensionality increases and the time required by the machine learning algorithm to train the data increases exponentially. To overcome these problems, feature selection is a well-known dimensionality reduction technique. *Feature selection* consists of selecting the relevant features and discarding the irrelevant ones to obtain a subset of features that describes properly the problem with a minimum degradation of performance [46].

A special case of feature selection is known as *online feature selection* [45, 95, 100, 137, 141], which can be very useful, being one of the most interesting when a *concept drift* appears. This phenomenon is present in situations where the underlying data distribution changes. These changes make the model built on old data inconsistent with the new data, and regular updating of the model is necessary [132]. Applied to feature selection, a concept drift may cause that the subset of relevant features changes over the time. In other words, as time goes by, different sets of features become important for classification and some totally new features with high

predictive power may appear. Online feature selection has been faced mostly individually, i.e. by selecting features previously in a single step independent of the online machine learning step, or performing online feature selection without performing online classification afterwards. Notice that after an online feature selection process, where the set of relevant features changes across the time, the subsequent classification algorithm has to be capable of updating its model according not only to new samples but also to new features, making it harder to find available methods that can cope with both requirements.

Therefore, in this chapter a new method that covers both online feature selection and online learning is proposed. This proposal includes an algorithm that performs online feature selection and classification at the same time, by modifying a classical well-known feature selection algorithm and introducing a novel implementation for a classification learning algorithm. Among the different feature selection methods available, a representative of so-called *filter methods* was chosen [46] since they are known for being fast, simple, classifier-independent and having a low computational cost [15]. Specifically, we reimplemented the χ^2 metric [77], chosen because of its simplicity and effectiveness, as well as having some characteristics that make it inherently incremental. However, this filter requires data to be discrete, and thus, well-known k-means discretizer [81, 130, 134] was also adapted to make it incremental.

The last step of the proposed online pipeline requires an incremental classifier, however, those available in the literature are incremental in the instance space, but not in the feature space. Up to the authors' knowledge, a complete pipeline as the one introduced here has not been presented elsewhere. In fact, the popular machine learning tool Weka [48] provides methods able to receive new instances, but they do not support different sets of features, perhaps with different sizes, in each iteration. Thus, an online training algorithm for one-layer artificial neural networks ANNs is also introduced in this chapter, which continuously adapts the input layer to those features, that remind might vary in number, selected at each time. In order to achieve this, a new implementation of a previously proposed algorithm [39], which reaches a minimum error in a few epochs of training and exhibits a higher speed when compared to other classical methods is presented. Moreover, the structure of this algorithm makes it suitable for a dynamic input space, as happens when selecting features on-line. In this chapter, a novel implementation, which continuously adapts the size of the input layer to those features selected at each time is proposed. The contents of this chapter have been published in [13].

The remainder of this chapter is organized as follows: Section 2.1 summarizes the state of the art in the field of online machine learning. Section 2.2 describes the method proposed in detail. Section 2.3 and Section 2.4 describe the experimental settings and the obtained results, respectively. Section 2.5 is focused on a case study about the influence of the order of

occurrence of the samples (data order) on the performance of the pipeline. Finally, Section 2.6 sums up the contents of the chapter.

2.1 Background

Online learning has become a trending area in the last few years since it allows to solve difficult problems such as concept drift or managing extremely high-dimensional datasets. For this reason, advances in this field have recently appeared. However, online feature selection has not evolve in line with online learning. Zhang et al. [148] proposed an incremental computation feature subset selection algorithm which, originated from Boolean matrix technique, selects useful features for the given data objective efficiently. Nevertheless, the efficiency of the feature selection method has not been tested with an incremental machine learning algorithm. Keerthika and Priya [63] examined various feature reduction techniques for intrusion detection, where training data arrive in a sequential manner from a real time application. Katakis et al. [62] mentioned the idea of a dynamic feature space. The features that are selected based on an initial collection of training documents are the ones that are subsequently considered by the learner during the operation of the system. However, these features may vary over time and in some applications an initial training set is not available. In the approach presented inhere, we are interested in flexible feature selection methods able to modify the selected subset of features as new training samples arrive, in both subset size and specific features selected. It is also desirable that these methods can be executed in a dynamic feature space that would be empty at the beginning and add features when new information arrives (e.g. documents in their text categorization application). Katakis et al. [62] applied incremental feature selection combined with what they called a feature based learning algorithm to deal with online learning in high-dimensional data streams. This framework is applied to a special case of concept drift inherent to textual data streams, which is the appearance of new predictive words over time. The problem with this approach is that they assume that features have discrete values. Perkins et al. [99] presented a novel and flexible approach, called grafting, which treats the selection of suitable features as an integral part of learning a predictor in a regularized learning framework. To make it suitable for large problems, grafting operates in an incremental iterative fashion, gradually building up a feature set while training a predictor model using gradient descent. Perkins and Theiler [100] tackle the problem in which, instead of all features being available from the start, features arrive one at a time. Online Feature Selection (OFS) assumes that, for any reason, is not affordable to wait until all features have arrived before learning begins, therefore one needs to derive a mapping function f from the inputs to the outputs that is as "good as possible" using a subset of just the features seen so far. In [141], a promising alternative method, Online Streaming Feature Selection (OSFS), to online select strongly relevant and non-redundant features is presented. Glocer et al. [45] demonstrated the power of OFS in the image processing domain by applying it to the problem of edge detection. Mao et al. [83] proposed a real-time compressive tracking algorithm based on online feature selection to

address the problems of drifting and tracking lost caused by changes in the appearance of the tracked object. The discriminating features selected are then integrated to construct a classifier to carry out the tracking process. Nguyen et al. [95] presented an online unsupervised feature selection method for background suppression in video sequences, that allows them to prune the feature set avoiding any combinatorial search.

Finally, some other researches have been found in the literature comprising online feature selection and classification. Kalkan and Çetisli [61] presented an online learning algorithm for feature extraction and classification, implemented for impact acoustics signals to sort hazelnut kernels. Levi and Ullman [74] proposed to classify images by ongoing feature selection. However, their approach only uses at each stage a small subset of the training data. Carvalho and Cohen [24] performed online feature selection based on the weights assigned to each input of the classifiers. Note, however, that this method is highly dependent on the classifier. Another method that is strongly dependent on the classifier was presented by Wang et al. [137]. They addressed two different tasks of OFS: 1) learning with full input, where the learner is allowed to access all the features to decide the active ones, and 2) learning with partial input, where only a limited number of features is allowed to be accessed for each instance by the learner. In a recent work, Roy [106] proposed an interesting algorithm for streaming big data and for highly parallel implementation on Apache Spark based on Kohonen networks. It examines some streaming data to select the features with a high discriminative power and then uses those features to learn pattern classifiers. Kohonen networks trained in the first phase are discarded once features are selected.

Therefore online feature selection has been dealt with mostly on an individual basis, or by performing online feature selection without subsequent online classification. In the few researches that comprise online feature selection and classification, the methods proposed are highly dependent on the classifier. Therefore, achieving real-time analysis and prediction for high-dimensional datasets remains a challenge for computational intelligence on portable platforms. The question now is to find flexible feature selection methods capable of modifying the selected subset of features as new training samples arrive [12]. It is also desirable for these methods to be executed in a dynamic feature space that would initially be empty but would add or remove features as new information arrived (e.g., documents in their text categorization application).

In light of the above, a unified pipeline that tries to fill the gap detected in the literature is presented. On the one hand, this proposal is able to modify the selected subset of features as new samples arrive (in both the number and the specific features selected) and, on the other hand, the classifier included can be updated according not only to new samples but also to new

features.

2.2 Description of the method

The proposed method consists of three independent stages that can be used alone or in a pipeline, bearing in mind that the filter requires discrete data. Besides, we propose a general method that could be applied to a wide range of problems. As explained before, the method introduced in this research consists of three online stages: discretizer, filter and classifier. Figure 2.1 shows the flowchart of this method (parameters k and λ will be explained in the corresponding subsections). Each step of the methodology and the reimplementation of the algorithms will be following described in depth. Notice that not all the existing algorithms can be reimplemented to tackle online data, as they need to have some properties that make them inherently incremental (remember that the classifier must be incremental not only in the samples space but also in the feature space). For this reason, the methods chosen to be reimplemented are the k-means discretizer, the χ^2 filter, and a one-layer artificial neural network.

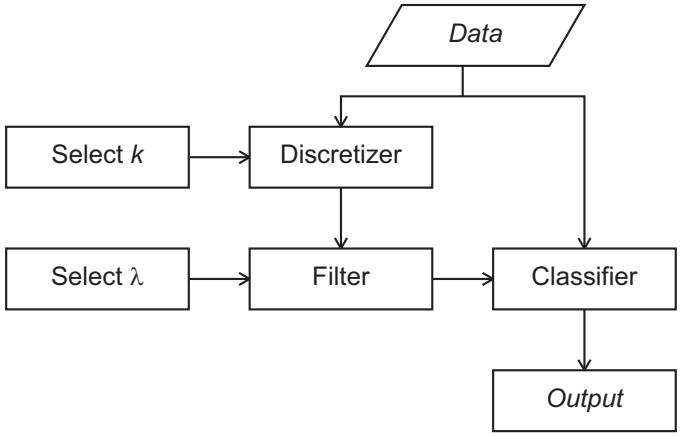


Figure 2.1: Flowchart of the proposed method.

2.2.1 Discretizer

Many feature selection algorithms are shown to work on discrete data, as it is the case of the filter selected in this work (χ^2), therefore the first stage of the proposed method is devoted to online discretization. However, due to the incremental nature of online learning, we cannot assume a range of input values for each feature a priori. This fact prevents the use of well-

known discretization algorithms such as entropy minimization discretization (EMD), equal width discretization (EWD) or equal frequency discretization (EFD). For this reason, the *k*-means discretization algorithm [130, 134] was chosen. *K*-means has been selected in Wu et al. [142] as one of the most influential algorithms in data mining. This algorithm moves the representative weights of each cluster along an unrestrained input space, making it suitable for our purposes. It is important to note that each feature is discretized independently. This clustering algorithm operates on a set of data points and assumes that the number of clusters to be determined (k) is given. The partition is done based on certain objective function. The most frequently used criterion function in k-means is minimizing the squared error ε criterion between the centroids μ_i of clusters $c_i, i = 1, \ldots, k$ and the samples x in those clusters

$$\varepsilon = \sum_{x \in c_i} |x - \mu_i|^2$$

Let C be the set of clusters and $|C|$ its cardinality. For each new sample x, the discretizer works as follows,

- If $|C| < k$ and $x \notin C$ then $C = \{x\} \cup C$, i.e. if the maximum number of cluster was not already reached and the new sample is not in C, then create a new cluster with its centroid in x.
- (*else*)
    1. Find the closest cluster to x.
    2. Update its centroid μ as the average of all values in that cluster (including x).

The method assigns at most k clusters. Notice that the number of clusters is the minimum between the parameter k and the number of different values in the feature. It is important to remark that in online methods there is no convergence criterion. The system is continuously adapted while data arrives. In section 2.4 a discussion on the impact of k on the algorithm is presented.

2.2.2 Filter

The χ^2 metric [77] was chosen because it evaluates each feature based on cumulative statistics concerning the number of times that it appears for a different class, which render it inherently

incremental. So, in our reimplementation, when a new instance appears, the statistics are updated and the evaluation can be calculated without the need of re-processing past data.

The χ^2 method evaluates features individually by measuring their chi-squared statistic with respect to the classes. The χ^2 value of an attribute is defined as:

$$\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{c} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \tag{2.1}$$

where

$$E_{ij} = R_i * C_j / S \tag{2.2}$$

k being the number of intervals (number of different values in a feature), c the number of classes, A_{ij} the number of samples in the i-th interval for the j-th class, R_i the number of samples in the i-th interval, C_j the number of samples in the j-th class, S the total number of samples, and E_{ij} the expected frequency of A_{ij}. Note that the size of the matrices is related to the number of intervals. In this manner, a very large k in the discretizer will lead to a very large size of the matrices A and E. A very large matrix is computationally expensive to update and should be taken into account for real-time applications.

After calculating the χ^2 value of all considered features, these values can be sorted with the largest one at the first position, as the larger the χ^2 value, the more important the feature is. This will provide an ordered ranking of features. To automatically select the important features in an online manner a threshold needs to be added to the original algorithm. The problem of selecting a threshold for rankers is still one of the open issues in Feature Selection research. At present, there is not yet a general and automatic method that allows researchers to establish a threshold for any given data set [15]. Some authors have tried some kind of automatic threshold that is related with the means and variance of the weights obtained for the features in the rankers [92], others with the largest gap between two consecutive attributes [89]. However, the most frequent approach is to test the results of a classifier after retaining different percentages of the ranked features [67], and thus the threshold should be tailored for the specific problem being studied. In this research, we propose a threshold λ which works in the following way. On each iteration, given the χ^2 value for each feature, the mean and the standard deviation of these values are computed. For each feature i and iteration t:

- if $\chi_i^2 < mean - \lambda \cdot std$ then the feature i is not selected
- if $\chi_i^2 > mean + \lambda \cdot std$ then the feature i is selected
- otherwise, the feature i maintains the same state as in the previous iteration (note that the initial set of features is the full set of features)

When λ is 0, the features selected in each iteration fluctuate significantly. On the other hand, when λ tends to infinity, there is no feature selection process. A further discussion about the impact of different values of λ can be found in section 2.4.2. Figure 2.2 shows an example of the use of λ in the filter. In the current iteration, features with a χ^2 value over $mean + \lambda std$ are selected (features 1, 2 and 9) while features with a χ^2 value under $mean - \lambda std$ are not selected (features 3, 4, 5, 7, 8 and 10). Those features with a χ^2 value between $mean \pm \lambda std$ maintains the same state as in the previous iteration. In this case, assuming that feature 6 was selected in the previous iteration then it will be also selected now.

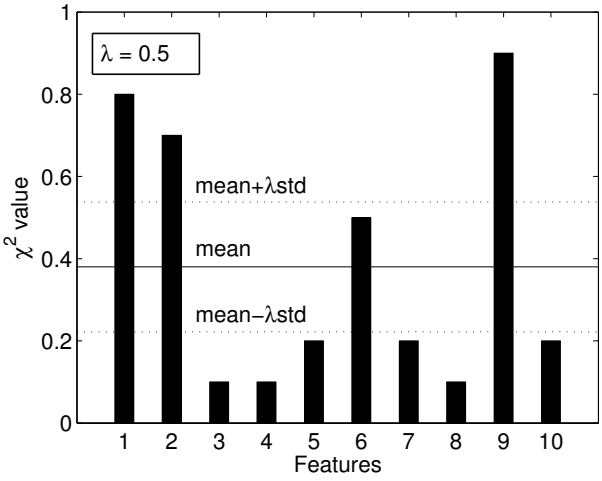


Figure 2.2: Example of the use of λ in the feature selection process.

2.2.3 Classifier

For the classification step of our online pipeline, a one-layer artificial neural network was chosen. Notice that in online applications, real-time response is often demanded. Thus, a light-weight machine learning algorithm is an appropriate election. Moreover, the algorithm must be incremental in both input and sample space, which is not a characteristic supported by most of the available classification algorithms.

In a previous work, an incremental training algorithm for one-layer ANNs [39], which reaches a minimum error in a few epochs of training and exhibits a higher speed when compared to other popular methods was presented. Besides these characteristics, the structure of this algorithm makes it suitable for a dynamic input space, as it is the case in this research. A new implementation is proposed herein, so as to be able to continuously adapt the input layer to the features selected in each iteration. Our proposal is a one-layer neural network which is fast and has the capability of adapting its number of inputs to the number of features that are selected at a given step, adding or removing neurons as needed.

In a one-layer ANN (see Figure 2.3), the set of equations relating inputs and outputs is given by

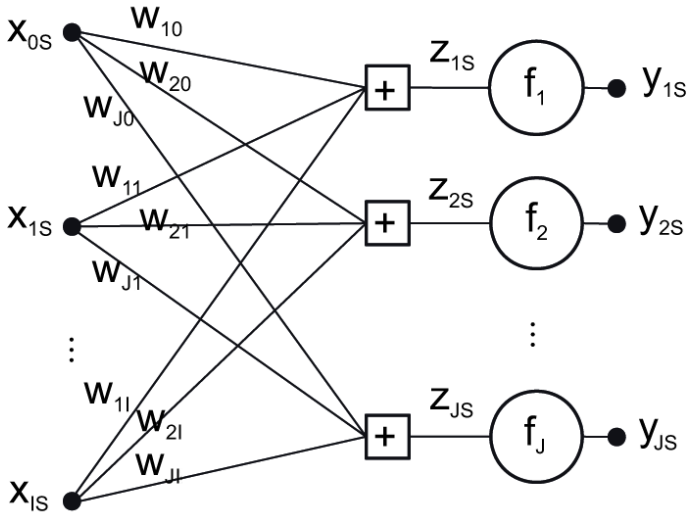


Figure 2.3: Architecture of a one-layer ANN.

$$y_{js} = f_j\left(\sum_{i=0}^{I} w_{ji}x_{is}\right); j = 1,2,\ldots,J; s = 1,2,\ldots,S \tag{2.3}$$

where I, J, S are the number of inputs, outputs and training samples, respectively, $x_{0s} = 1$, w_{ji} is the weight of the connection between the i-th input and the j-th output neuron, and f_j is the nonlinear activation function of j-th output neuron. The system described by Eq. 2.3 has $J \times S$ equations in $J \times (I+1)$ unknowns. However, since the number of samples of data is often large $(S >> I+1)$, in practice, this set of equations and w_{ji} is overdetermined and has no solution. Thus, as the errors ε_{js} between the real (y_{js}) and the desired output (d_{js}) of the network are defined by

$$\varepsilon_{js} = d_{js} - y_{js} = d_{js} - f_j\left(\sum_{i=0}^{I} w_{ji}x_{is}\right) \tag{2.4}$$

d_{js} being the desired output for neuron j, and usually the sum of squared errors is minimized to learn the weights w_{ji}.

$$P = \sum_{s=1}^{S}\sum_{j=1}^{J}\varepsilon_{js}^2 = \sum_{s=1}^{S}\sum_{j=1}^{J}\left(d_{js} - f_j\left(\sum_{i=0}^{I} w_{ji}x_{is}\right)\right)^2 \tag{2.5}$$

However, if it is assumed that the nonlinear activation functions f_j are invertible (as it is the case for the most commonly used functions), alternatively, the system of equations in Eq. 2.4 can be rewritten in the following way [39]:

$$\bar{\varepsilon}_{js} = \bar{d}_{js} - z_{js} = f_j^{-1}(d_{js}) - \sum_{i=0}^{I} w_{ji}x_{is} \tag{2.6}$$

where $\bar{d}_{js} = f_j^{-1}(d_{js})$ and $z_{js} = \sum_i w_{ji}x_{is}$. Eq. 2.6 measures the errors in terms of the inputs (x_{is}). It is important to note that in Eq. 2.6 the unknowns (weights of the network) are not affected by the nonlinear activation function f_j, i.e. the error is linear with respect to weights. Then, an alternative objective function is obtained to be minimized [39]:

$$Q = \sum_{s=1}^{S}\bar{\varepsilon}_{js}^2 = \sum_{s=1}^{S}\left(f_j^{-1}(d_{js}) - \sum_{i=0}^{I} w_{ji}x_{is}\right)^2 \tag{2.7}$$

whose global minimum can be computed by solving the system of equations obtained by equalizing its derivative to zero:

$$\sum_{i=0}^{I} A_{pi}w_{ji} = b_{pj} \tag{2.8}$$

where

$$A_{pi} = \sum_{s=1}^{S} x_{is} x_{ps} f_j'^2(\bar{d}_{js})$$
$$b_{pj} = \sum_{s=1}^{S} \bar{d}_{js} x_{ps} f_j'^2(\bar{d}_{js}) \tag{2.9}$$

For every output j, Eq. 2.9 has $I+1$ linear equations and unknowns and, thereby, there exists only one real solution which corresponds to the global optimum of the objective function. Several computationally efficient methods can be used to solve this kind of systems with a complexity of $O(J \times (I+1)^2)$, where I and J are the number of inputs and outputs of the ANN, respectively [22, 11]. Furthermore, this training algorithm is able to learn incrementally since the coefficients A_{pi} and b_{pj} are calculated as a sum of terms (see Eq. 2.9). Due to the commutative and associative properties of the sum, the same solution is obtained independently of the order of occurrence of the samples.

The structure of the matrices A and b is also suitable for a dynamic space of input features. On the one hand, removing a feature only comprises removing the row and column corresponding with that feature. On the other hand, adding a new feature only comprises adding a row and a column of zeros. Note that a row and a column of zeros in the matrices A and b corresponds with learning from the scratch that feature. Thus, this method is able to adapt the architecture of the one-layer ANN and deal with changing environments in which the relevant features may be different at one time or another.

2.3 Experimental evaluation: material and methods

The experiments presented in this section are focused on the evaluation of the online method proposed in this work. The three methods (discretizer, filter and classifier) will be evaluated both independently and also integrated in the unified pipeline.

2.3.1 Materials

Five different classification datasets were used during this research. Corral, LED, and Friedman are synthetic while Connect and Forest were selected from the UCI Machine Learning

Repository [2]. Table 4.8 depicts the characteristics of these datasets: number of features, number of instances, number of classes, and percentage of the majority class.

Dataset	No. features	No. instances	No. classes	% maj. class
Corral-100	100	10 000	2	56.02%
LED-100	100	10 000	10	10.00%
Friedman-100	100	10 000	2	54.57%
Connect4	42	67 557	3	65.83%
Forest	54	101 241	7	48.69%

Table 2.1: Characteristics of the datasets.

Corral-100

In this research, a modified version of the CorrAL dataset [57] will be used. The original dataset has four binary relevant features $f_1, \ldots, f_4$, one irrelevant feature f_5 and one feature f_6 correlated with the output. Its class value is $(f1 \wedge f2) \vee (f3 \wedge f4)$. This research is not focused on detecting correlation, but on discarding irrelevant features. The correct behavior for a given feature selection method is to select the four relevant features and to discard the irrelevant ones. A new dataset called Corral-100 will be employed, consisting of the 4 relevant features plus 96 irrelevant binary features randomly generated.

The LED-100 problem

The LED problem [18] is a simple classification task that consists of identifying the digit that the display is representing. Given the active leds described by seven binary attributes $f_1, \ldots, f_7$ (seven segments display), the task to be solved is its classification in ten possible classes available $C = \{0,1,2,3,4,5,6,7,8,9\}$. A 1 in an attribute indicates that the led is active, and a 0 indicates that it is not active. The LED-100 problem was constructed by adding 93 irrelevant features.

Friedman-100

This synthetic dataset uses a function suggested by Friedman [41]. It is defined by the equation

$$f(x_1, \ldots, x_5) = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \sigma(0,1) \tag{2.10}$$

where $\sigma(0,1)$ is zero mean unit variance Gaussian noise and the inputs $x_1,\ldots,x_5$ are sampled independently from a uniform distribution in the interval $[0,1]$. This dataset is a regression task. We transformed it into a classification task where the goal was to predict class 1 for the examples of output under 15 (prevalence around 55%) and class 2 for the other examples (prevalence around 45%). The Friedman-100 dataset was constructed by adding 95 irrelevant features to the previous Friedman dataset. The data for the added features were generated randomly from a uniform distribution $[0,1]$.

Connect4

This database contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced. The outcome class is the game theoretical value for the first player (win,loss,draw). The number of features is 42. All features are categorical with 3 possible values.

Forest

This dataset is a classification task with 7 classes (representing forest cover types). Predicting forest cover type from cartographic variables only (no remotely sensed data). The actual forest cover type for a given observation (30 x 30 meter cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types). The number of features is 54.

2.3.2 Performance measures

Discretization is concerned with the process of translating continuous values of features into discrete values. As a result, some error is committed during the process. The discrepancy between the exact value and some approximation to it is called approximation error. The *absolute approximation error* is defined as the magnitude of the difference between the exact value and the approximation. Given some value v and its approximation $\hat{v}$, the absolute error is computed as follows

$$\varepsilon = |v - \hat{v}| \tag{2.11}$$

The *relative approximation error* is the absolute error divided by the magnitude of the exact value, and it is computed as follows

$$\varepsilon = \frac{|v - \hat{v}|}{|v|} = \left|1 - \frac{\hat{v}}{v}\right| \tag{2.12}$$

Respect to the filter method, its efficiency was evaluated in terms of *precision*, *recall*, *percentage of selected features*, and *stability*. Precision is the fraction of the selected features that are relevant and it is computed as follows

$$precision = \frac{|\{\text{relevant features}\} \cap \{\text{selected features}\}|}{|\{\text{selected features}\}|} \tag{2.13}$$

Recall is the fraction of the relevant features that are selected and it is computed as follows

$$recall = \frac{|\{\text{relevant features}\} \cap \{\text{selected features}\}|}{|\{\text{relevant features}\}|} \tag{2.14}$$

Note that the relevant features of the dataset have to be known to compute these measures. The percentage of selected features is simply computed as the fraction of the features selected and the total number of features. To evaluate the stability of the filter the Jaccard index will be used. The Jaccard similarity coefficient [98] is a measure used for comparing the similarity of two sets of features A and B. It is defined as the fraction of the cardinality of the intersection and the cardinality of the union of the features

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.15}$$

In an online environment, the stability of the filter will be related with the set of features selected in the current step in comparison with the set of features selected in the previous step.

Finally, the performance of the classifier is computed in terms of *standard classification error* [139] which is defined as the fraction of samples incorrectly classified over the data.

2.4 Experimental evaluation: results

The experimental results obtained for each of the three methods and the proposed pipeline are presented in this section.

2.4.1 Discretizer

For this experiment, we chose the first feature from the Friedman dataset. Note that every feature in this set is sampled independently from a uniform distribution in the interval $[0, 1]$.

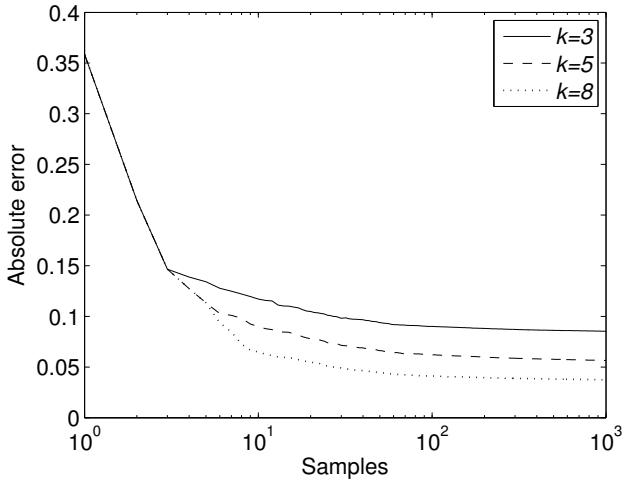
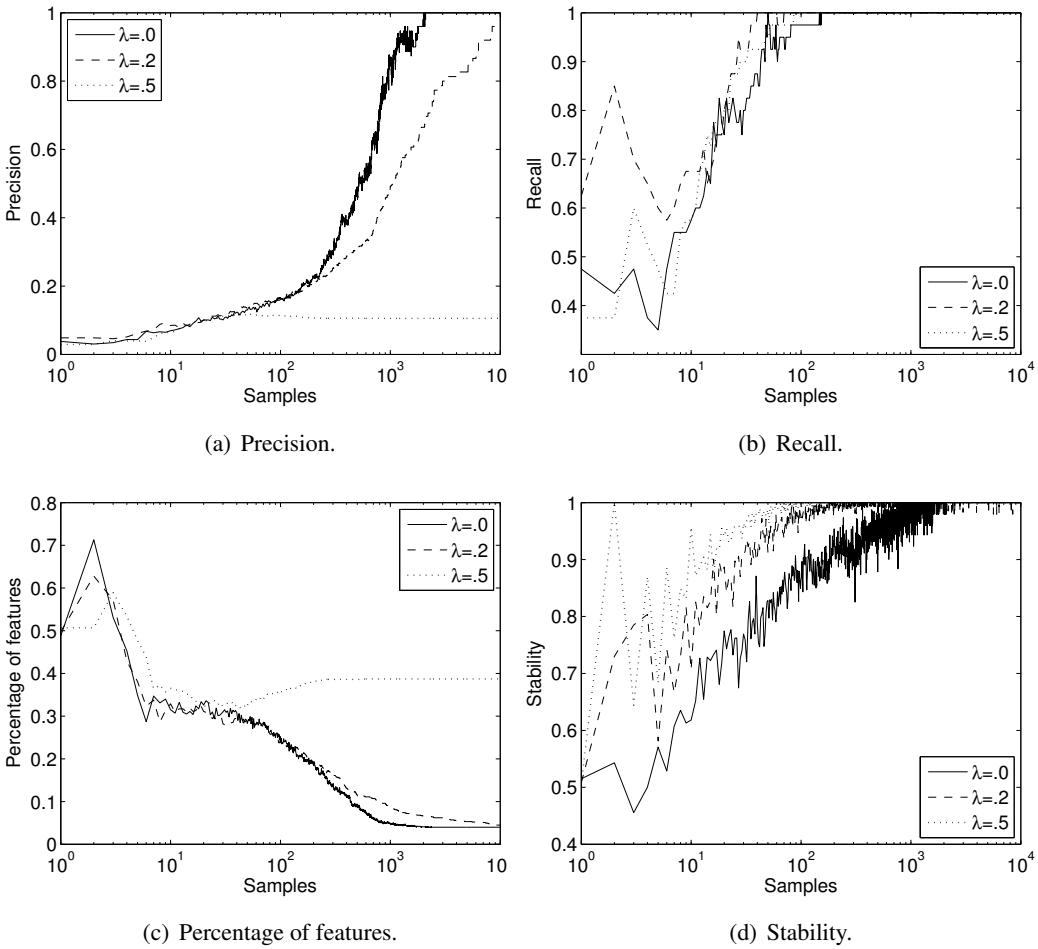


Figure 2.4: Absolute approximation error of the online discretizer in a feature sampled independently from a uniform distribution in the interval $[0, 1]$.

Figure 2.4 shows the absolute approximation error of the discretizer. The variable k indicates the number of clusters used in the discretizer. Notice that the values of the feature appear in random order along the interval $[0, 1]$. If the values of the features lay approximately along all their possible values the discretizer shows a very fast adjustment to the data, as seen in the curve. As expected, the larger the number of clusters, the better the adjustment. Note however that a very large number of clusters increases computations at the expense of a decrement in terms of approximation error that should be evaluated. For example, using $k = 8$ instead of $k = 5$ increases computation by 60% but only decreases the approximation error by 20% (from 0.06 to 0.04, see Figure 2.4). Note also that a larger number of k implies larger matrices in the feature selection filter.

2.4.2 Filter

To check the efficiency of the feature selection method proposed, two synthetic datasets (Corral-100 and LED-100) were employed. We have chosen to use artificially generated data because the desired output is known, therefore a feature selection algorithm can be evaluated with independence of the classifier. The main advantage of artificial datasets is the knowledge of the set of optimal features that must be selected; thus, the degree of closeness to any of these solutions can be assessed in a confident way.



(a) Precision.

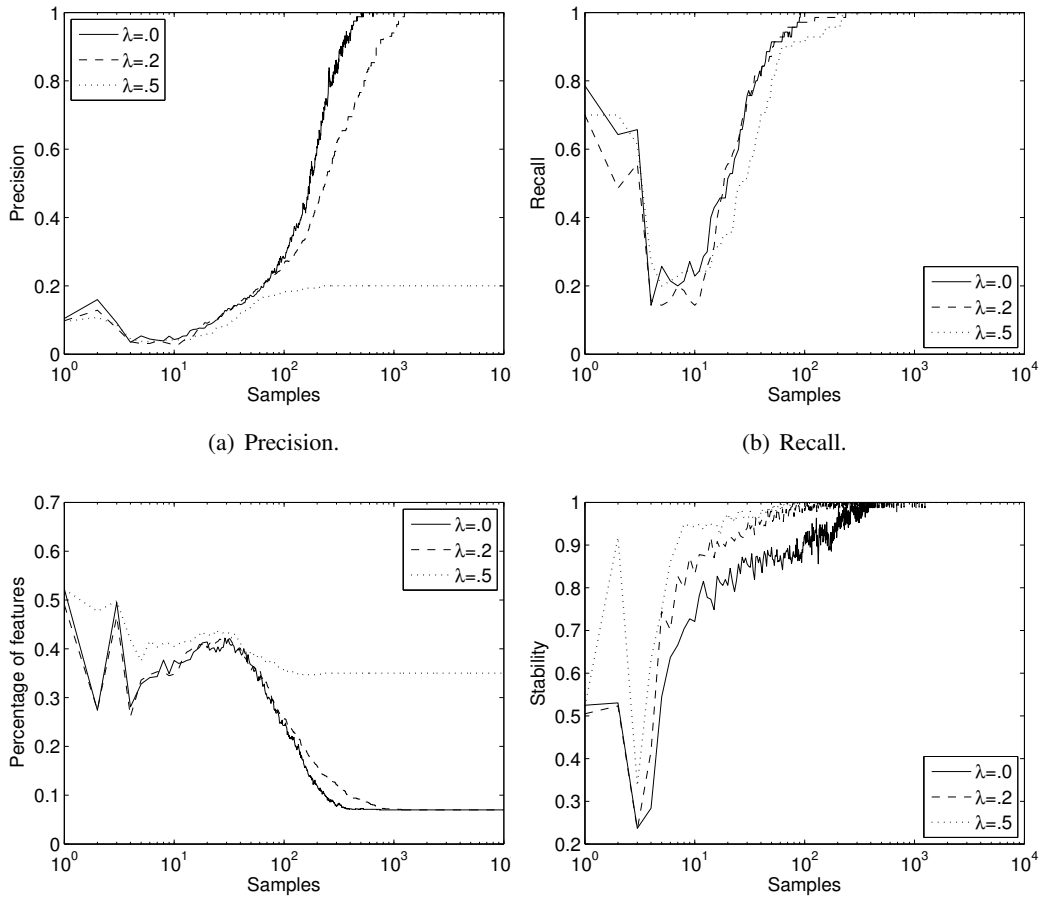
(b) Recall.

(c) Percentage of features.

(d) Stability.

Figure 2.5: Performance measures of the online feature selection filter in the Corral-100 dataset.

Figures 2.5 and 2.6 show the performance of the feature selection filter for the Corral-100 and LED-100 datasets, respectively, in terms of precision, recall, percentage of selected features, and stability for different values of the parameter λ (threshold for the filter, see Section



(a) Precision.

(b) Recall.

(c) Percentage of features.

(d) Stability.

Figure 2.6: Performance measures of the online feature selection filter in the LED-100 dataset.

2.2.2 for further details).

As can be seen in both Figures 2.5 and 2.6, the lower the value of the parameter λ the faster the curve of precision converges. Moreover, the percentage of selected features is lower but at the expense of a more unstable behavior. On the other hand, the higher the value of λ the slower the curve of precision converges. However, the filter shows a more stable behavior, because the number of selected features is larger in this case. Note that the classifier needs to learn from scratch each new feature. If the subset of features selected by the filter changes frequently then the classifier will lose partial knowledge many times during the learning process. Thus an appropriate selection of this parameter plays a crucial role in the different measures, for example in this case $\lambda = 0.2$ appears to be the most sensible solution. Notice that a large value of the parameter λ will not find the optimum subset of features (the curve of precision does not converge for $\lambda = 0.5$).

2.4.3 Classifier

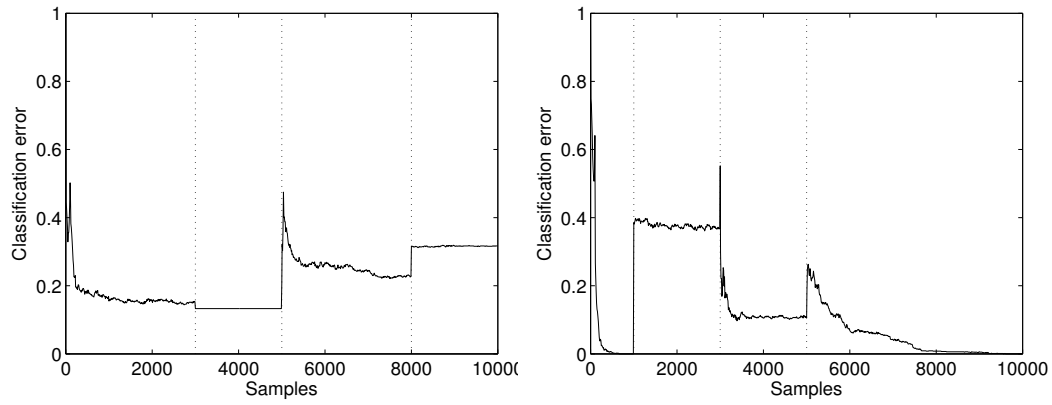
The efficiency of the classifier is also shown on the two synthetic datasets Corral-100 and LED-100. The objective is to train the classifier when the input space is changeable by adding or removing features as happens in online feature selection. Synthetic datasets are useful here to check the impact in terms of classification error when adding or removing relevant features.

Data were divided 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. In this research, the 10% of data were used for testing while the 90% were used for training. In other words, after a new training sample arrives and the model is updated, its performance is tested on the 10% data left as test set.

Regarding the set of features selected in each step, three different situations were considered:

- The set of selected features contains all relevant features and none irrelevant.
- The set of selected features contains all relevant features and some irrelevant.
- The set of selected features contains some relevant features and some irrelevant.

Figure 2.7 shows the classification error in the Corral-100 and LED-100 datasets when the



(a) Corral-100 dataset. (b) LED-100 dataset.

Figure 2.7: Test error of the classifier.

input space changes. Figure 2.7(a) shows the performance of the classifier on the Corral-100 dataset in the following situations,

- From 1 to 3000 samples, the set of selected features contains all (4) relevant features and 96 irrelevant
- From 3001 to 5000 samples, the set of selected features contains all relevant features and none irrelevant
- From 5001 to 8000 samples, the set of selected features contains 2 relevant features and some irrelevant
- From 8001 to 10000 samples, the set of selected features contains 1 relevant feature and some irrelevant

As can be seen in Figure 2.7(a), the classifier reaches its minimum classification error in few epochs. At this point, if the set of selected features is reduced to contain only relevant features (from 3001 to 5000 samples), the classifier maintains its performance. However, if the set of selected features only contains some of the relevant features (from 5001 to 10000 samples), the classification error of the classifier increases because it is only able to converge a global suboptimum, due to the lack of relevant information. As expected, the lesser the relevant features selected the worse the performance of the classifier.

It is also interesting to check the performance of the classifier when adding relevant features. In a similar manner to above, Figure 2.7(b) shows the performance of the classifier on

the LED-100 dataset in the following situations,

- From 1 to 1000 samples, the set of selected features contains all (7) relevant features and 94 irrelevant
- From 1001 to 3000, the set of selected features contains 3 relevant features and 44 irrelevant
- From 3001 to 5000 samples, the set of selected features contains 5 relevant features and 94 irrelevant
- From 5001 to 10000 samples, the set of selected features contains all relevant features and 44 irrelevant

As can be seen in Figure 2.7(b), the classifier improves its performance according as the set of selected features contains more relevant features. Notice that if only some (or none) relevant features are selected as the set of current features, the classifier is only able to reach a global suboptimum. Finally, as can be inferred from both Figures 2.7(a) and 2.7(b), the online classifier proposed in this research is able to efficiently adapt its structure to changes in the input feature space.

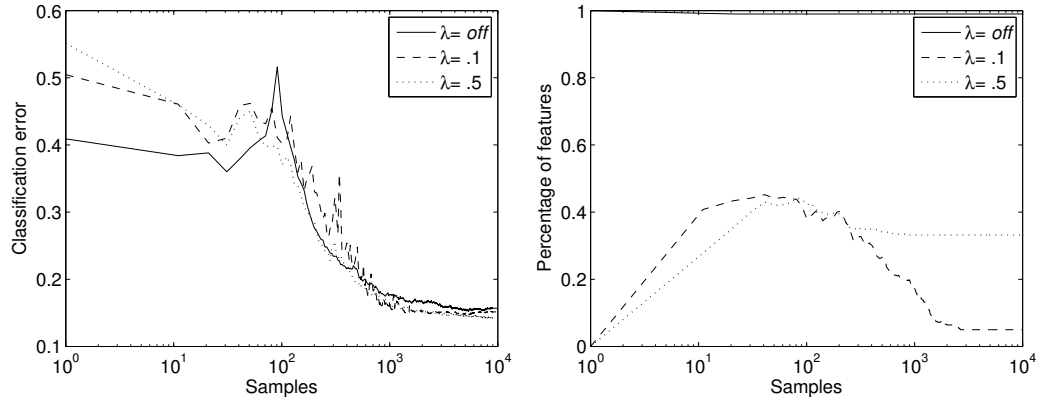
2.4.4 Pipeline

For testing the unified pipeline (discretizer plus filter plus classifier), the classification error will be used along with the number of features selected. Note that a slight degradation in the performance of the classifier may be acceptable if the number of features is significantly reduced. The following procedure was followed for each new sample:

1. Discretize the sample using the *k*-means discretizer.
2. Select the most relevant features using the χ^2 filter.
3. Update the one-layer ANN using those features.
4. Compute the test classification error and the percentage of selected features.

The goal here is to compare the performance of the system with and without feature selection. The experimental setup was the same as in the previous section. Data were divided using holdout validation, 10% of data were used for testing while the 90% were used for training. After a new training sample arrives, the model is updated and its accuracy is measured on the 10% data left as test set. This type of validation is appropriate because the size of the datasets is large. Moreover, every experiment was repeated 10 times in order to ensure unbiased results. The number of clusters k in the discretization stage was set to 10.

Finally, a Kruskal-Wallis test [140] was applied to check if there are significant differences among the medians of the methods for a level of significance $\alpha = 0.05$. If there are differences among the medians, we then applied a multiple comparison procedure [53] to find the method whose performance is not significantly different from the method with the best mean performance. In this work, a Tukey's honestly significant criterion [53] was used as multiple comparison test.



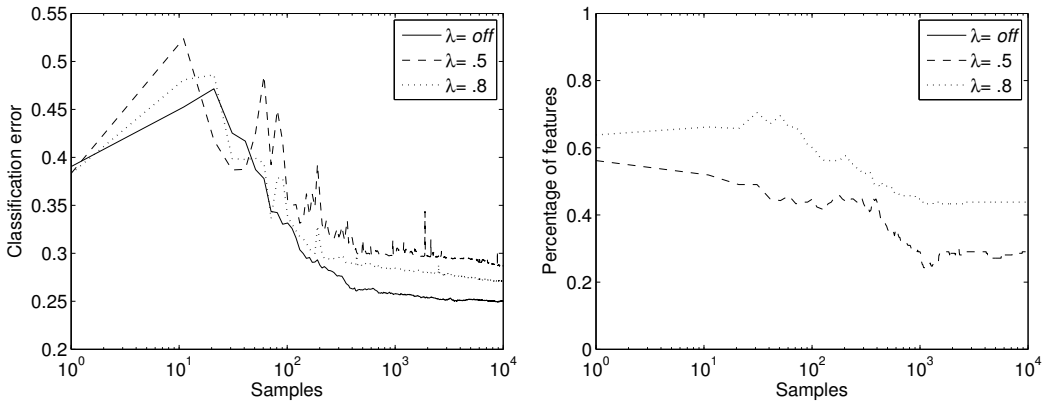
(a) Classification error.

(b) Percentage of features.

Figure 2.8: Performance measures of the proposed method in Friedman dataset in which $\lambda =$ *off* means that no feature selection was applied.

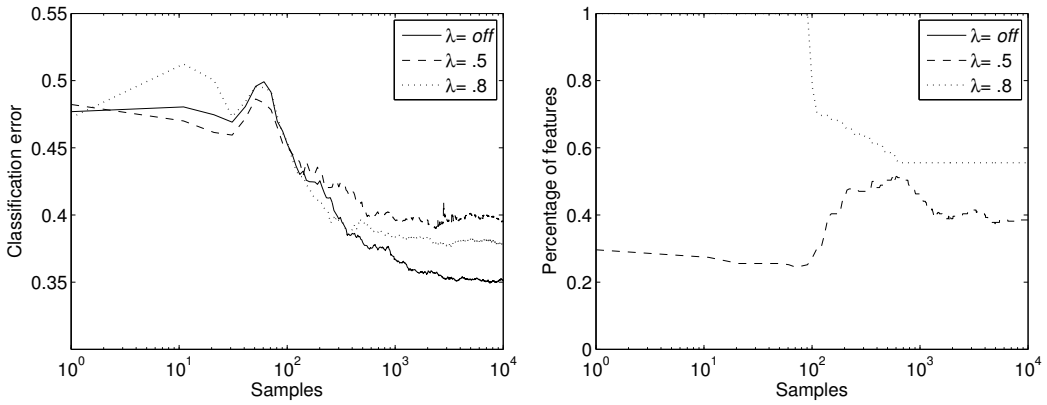
Figures 2.8, 2.9 and 2.10 show the classification error of the online classifier and the percentage of features selected by the online filter in Friedman, Connect4 and Forest datasets, respectively. Note that the percentage of selected features when no features selection is applied ($\lambda = off$) is always 1. For purposes of simplicity, only the first 10000 samples are shown. The classifier and the filter maintain their behavior for the remainder samples.

In any case, the statistical tests indicate that the classification error obtained when no feature selection is applied is similar to that obtained when the less aggressive λ is used ($\lambda = 0.8$), with the additional advantage, in this last case, that only 37%, 44% and 56% of the relevant features



(a) Classification error. (b) Percentage of features.

Figure 2.9: Performance measures of the proposed method in Connect dataset in which $\lambda = off$ means that no feature selection was applied.



(a) Classification error. (b) Percentage of features.

Figure 2.10: Performance measures of the proposed method in Forest dataset in which $\lambda = off$ means that no feature selection was applied.

are used for Friedman, Connect4 and Forest, respectively. If a more aggressive feature selection is performed, the percentage of features used for learning is significantly reduced but at the cost of a significant higher classification error.

Finally, the issue of selecting the parameter λ in real datasets is an open question. It will depend, on the first term, on the tradeoff between accuracy and speed (lower or higher number of features) and, on the last term, it will need to be determined by trial and error. As a rule of thumb, we suggest not using very low lambda values (next to 0.0) because it will tend to select smaller sets of features, consequently the filter will have a more unstable behavior.

2.5 Experimental evaluation: a case study on data order

In this section we introduce an exhaustive analysis of how the order of appearance of the samples affects the performance of the machine learning algorithms chosen in this work, an important aspect for on-line algorithms. In the previous experiments, samples appeared in a random order. However, notice that in an online environment data may arrive in a certain order, i.e. following biased distributions of data. It is possible that either all the data keep the same distribution, or that a so-called *concept drift* appears, leading to data following different distributions at different times of the process.

2.5.1 Discretization

In Section 2.4.1 we have shown discretization experiments when the values of the feature (the first feature of Friedman dataset) appeared in a random order along the interval $[0,1]$. However, as the discretizer is not order independent, it is also important to check its efficiency when data arrives in a certain order. The curve of error depicted in Figure 2.11 assumes a certain order of appearance of samples. In particular, the values of the feature are as follows

- From 1 to 250 samples its values lay in $[0, 0.33]$
- From 250 to 500 samples its values lay in $[0.34, 0.66]$
- From 500 to 750 samples its values lay in $[0.67, 1]$
- From 750 to 1000 samples its values lay in $[0, 1]$

As can be seen, the discretizer is also able to tackle with biased distributions of the values of a feature by storing past knowledge. At the end, the centroids of the clusters are located in averaged positions along the samples. Thus, the discretizer is considered to be quite robust to biased distributions.

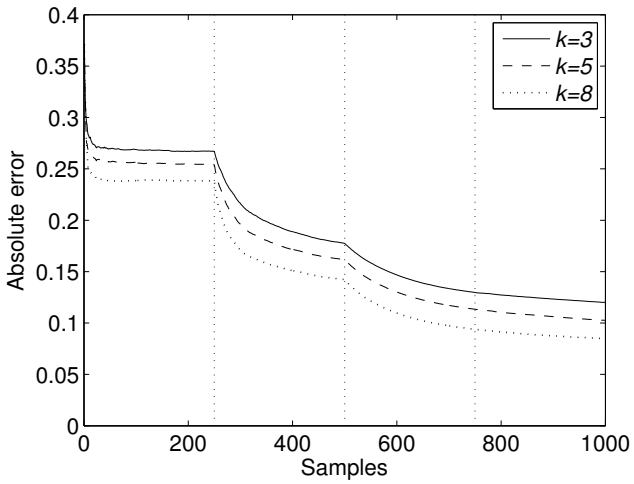


Figure 2.11: Absolute approximation error of the online discretizer in a feature, the first one of Friedman dataset with biased order of appearance of samples.

In addition, Figure 2.12 displays the curve of error of each execution when the feature values appear in random order. In this experiment, 1000 samples extracted from a uniform distribution in the interval $[0, 1]$ were employed. For each value of k (number of clusters), 50 executions were carried out. As expected, the larger the number of clusters, the larger the number of samples needed to reach the minimum error. It is interesting to note that this experiment proves that the order of appearance is not critical given a certain number of samples, since the error of each execution converges to similar values.

2.5.2 Feature selection

In an online scenario, data order can also affect the feature selection process, so a experiment was carried out on Friedman dataset. For this sake, 10 features are considered, where the first five are relevant and the remaining ones are irrelevant (see Section 2.3.1). The dataset used consists of 2000 samples where the order of appearance was randomly generated in each one of the 100 executions. For the discretization stage, different values of k (number of clusters) were employed, from 1 to 100. After seeing the whole dataset, the χ^2 value of each feature is computed. Figure 2.13 shows the distance between the minimum and maximum χ^2 values of

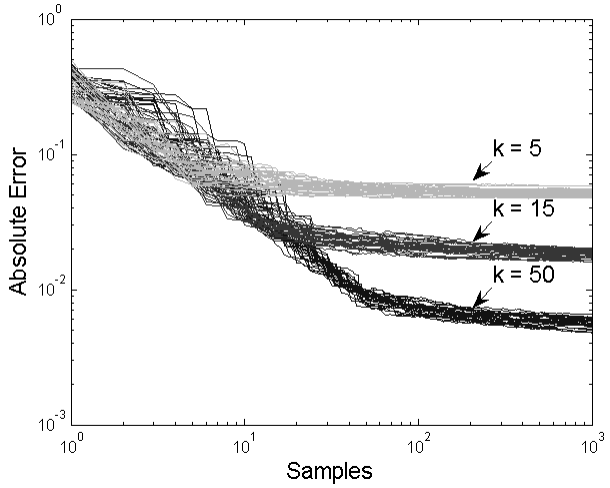


Figure 2.12: Absolute approximation error of the online discretizer of each execution when the feature values appear in random order (1^{st} feature of Friedman dataset)

the relevant and irrelevant features, respectively –i.e. the minimum margin between relevant and irrelevant features. The pale lines represent each one of the 100 executions, whilst the mean $\pm$ standard deviation are also displayed in bold lines. The margin increases until around $k = 20$ and from this point on, it stabilizes. In this case, data order seems to have a low impact on the performance of the filter because the maximum standard deviation is around 10%.

2.5.3 Classification

Finally, we test the influence of data order when the last step of classification is included in the learning process on Forest dataset. So far, it has been shown that the data order has little influence on the discretizer (see section 2.4.1) and consequently, on the filter (see section 2.4.2). Figure 2.14 displays the effect of the data order on the proposed pipeline. The parameter λ was established to 0.8, since it has been proved to obtain a good performance with this dataset. The training dataset used consists of 5000 balanced samples where the order of appearance was randomly generated in each execution. For assessing the classification error, a set of 1000 independent samples was used, and a total of 100 executions were accomplished.

On the one hand, Figures 2.14(a), 2.14(c) and 2.14(e) (first column) depict the classification error trace of each execution for different values of k (number of clusters for the discretization stage). On the other hand, Figures 2.14(b), 2.14(d) and 2.14(f) (second column) display the percentage of times that a feature is selected in average at the end of all the executions, again

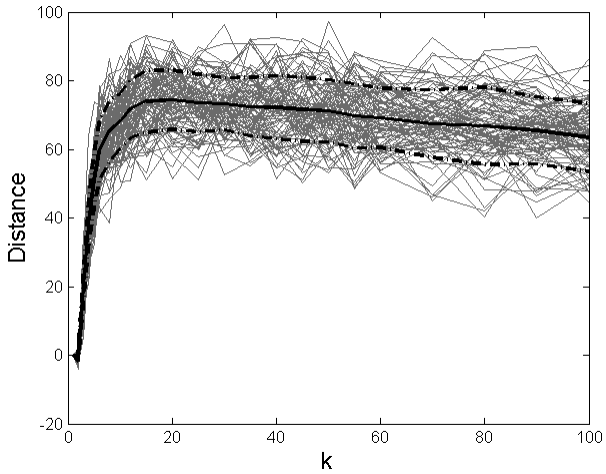
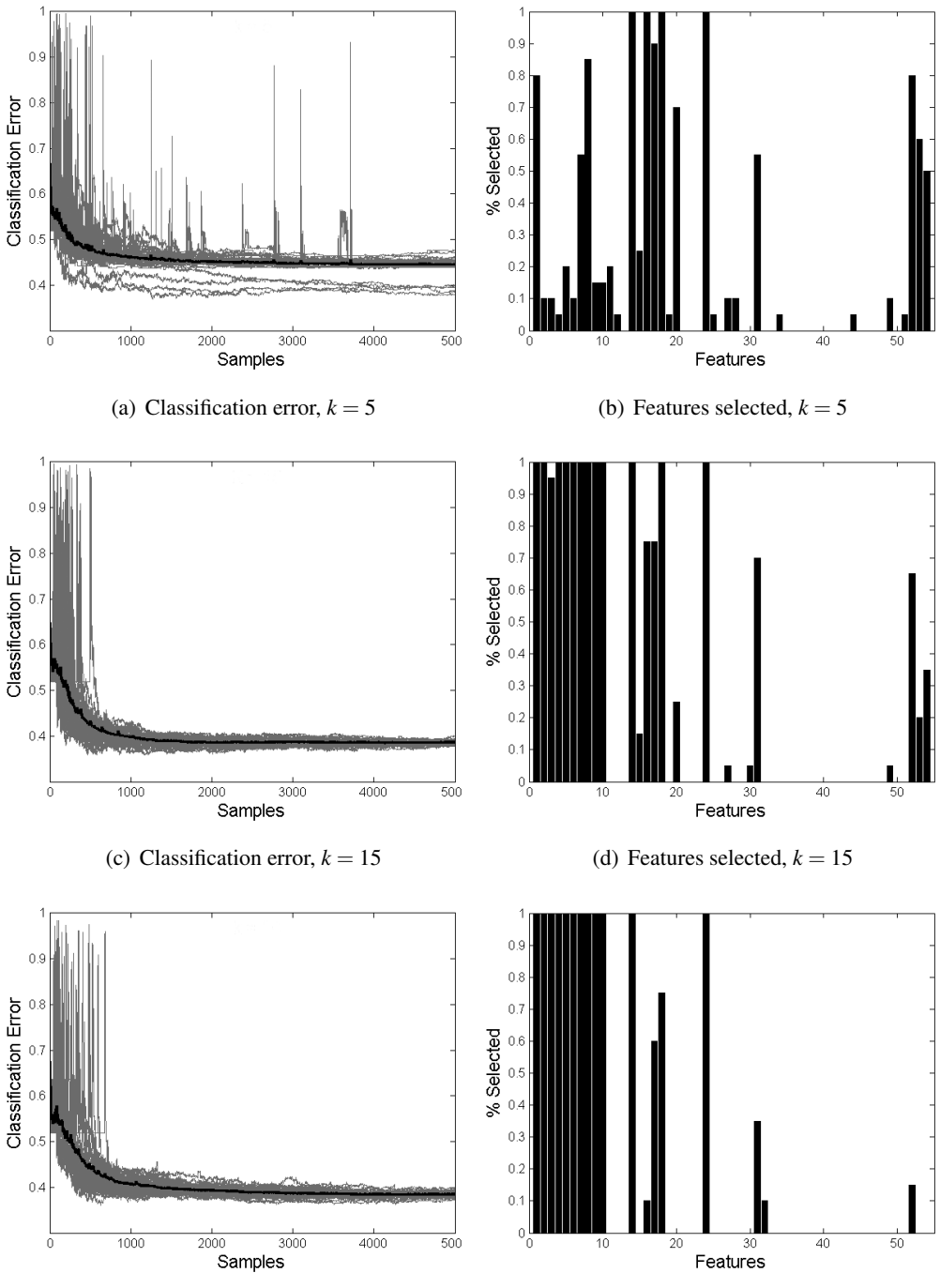


Figure 2.13: Minimum distance between relevant and irrelevant features according to χ^2 values on Friedman dataset

for different values of k. Remind that the smaller the k, the more information is missed in the discretization step. For this reason, when $k = 5$ it can be seen that there are more differences among the different executions. This fact may be caused by the information lost in the discretization stage, which also affects to the features selected by the filter, resulting in an irregular classification, where the data order plays an important role. However, when k increases, the behavior of the pipeline is more stable and independent of the order, as can be seen in Figures 2.14(c) and 2.14(e). The classification error decreases and the stability of the selected features raises. Finally, Figure 2.15 visualizes the average classification error for the three values of k tested. As can be seen, the classification error improves notably from $k = 5$ to higher values, whilst no significant differences are found between $k = 15$ and $k = 50$. In light of the above, it seems reasonable to select the lowest value for k without compromising the classification error. In this case, $k = 15$ might be the optimal, since it converges to the same error than $k = 50$ and decreases computations, as mentioned in Section 2.4.1.

2.6 Discussion

In this chapter, a complete pipeline (covering discretization, feature selection and classification) which is capable of continuously updating its model to learn from online data is presented. One of the strengths of the proposed method is that it consists of three independent stages that can be used alone or in a pipeline. Up to the authors' knowledge, there is no other work in the literature



(a) Classification error, $k = 5$

(b) Features selected, $k = 5$

(c) Classification error, $k = 15$

(d) Features selected, $k = 15$

(e) Classification error, $k = 50$

(f) Features selected, $k = 50$

Figure 2.14: Data order effect on the overall pipeline in Forest dataset.

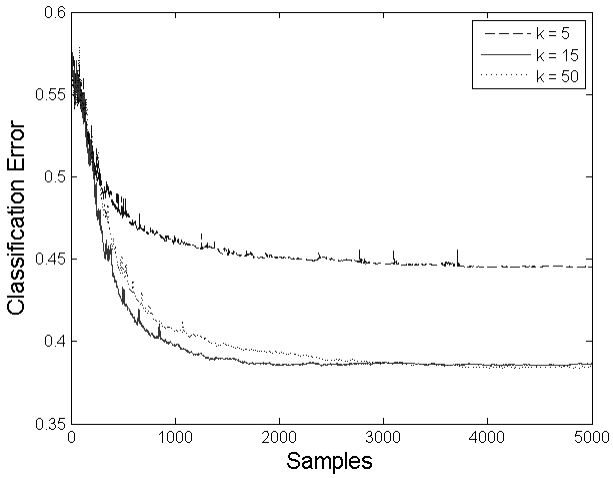


Figure 2.15: Average classification error of the data order experiment in Forest.

that covers efficiently these three stages, since some of the existing approaches apply feature selection in an off line fashion (because the online classifiers cannot deal with changing subsets of features) or they apply online feature selection but then this is not connected with an online classification stage. Therefore, the main advantage of our proposed method is that it allows researchers to perform both online feature selection and classification. The key contributions of this research are the following ones:

- Adaptation of the χ^2 filter using a new threshold λ to perform online feature selection.
- Since the χ^2 filter requires data to be discrete, adapting the k-means discretizer to be used in an online fashion.
- The adaptation of a learning algorithm (one layer neural network) to be incremental not only in the instance space, but also in the feature space, allowing for feature subsets that change, increasing or reducing in number during the learning process.
- Since an important aspect of on-line algorithms is the impact of data order on the performance of the methods, this issue is specially assessed, showing the robustness of the method. This is crucial in some real life environments in which concept-drift situations might appear.

However, although to the best of our knowledge a complete pipeline as this one has not been presented elsewhere, this proposal has some limitations. Most of all, it is important to

notice that not all the machine learning methods available in the literature can be adapted to deal with online data. Therefore, although more sophisticated learning methods exist, they cannot be adapted to learn in an online manner, and we had to choose simpler models such as the χ^2 filter and an ANN. Although simple, the chosen methods demonstrated to be adequate for this type of learning, exhibiting promising results, both separately and when combined in a pipeline. Experimental results showed that the classification error is decreasing over the time, adapting to the appearance of new data. Plus, the number of features is reduced while maintaining the classification performance. Another restriction, which is not specific of the pipeline, but general for machine learning methods (including preprocessing methods, such as discretization and feature selection), is the need for parameter estimation, that should be adapted to each problem under consideration. At this respect, some general recommendations are given through the specific subsections of the chapter, although this issue is still an open problem, in which researchers are still working.

CHAPTER 3

A SVD-based Autoencoder for Large-Scale One-Class Classification

In the previous chapter, the problem of online feature selection and classification was confronted. As mentioned, online feature selection is a research area that has not been treated frequently in the literature. Another area, in which scientific contributions are also relatively scarce is one-class classification, specifically regarding scalable methods. In the present chapter, we will focus in one-class classification algorithms when dealing with large amounts of data.

In a classical classification problem each unknown example is classified as belonging to one of all available categories. However, there are different scenarios where the classification task consists in deciding if a particular example fits a class or not. In order to handle appropriately this type of situations a one-class classification paradigm, in which one class (normal data or positive class) has to be distinguished from other classes (abnormal data), would be more appropriate. In this approach the positive class is well sampled in the training set, while the other classes are severely under-sampled or even nonexistent. The scarcity of abnormal examples can be due to several reasons, such as high costs to gather them or the low frequency at which this kind of events occur. This problem has received different denominations over the years as single classification [90], one-class classification [94] and others arisen from different problems to which this paradigm has been applied such as Outlier Detection [28] or Novelty Detection [8]. According to Chandola [28] outlier detection refers to the task of finding patterns that do not show the expected behavior (outlier/ anomaly observations) whereas novelty detection identifies unknown patterns that usually are incorporated as normal after their finding. This is a typical scenario in a wide variety of real environments and consequently this discipline has gained a lot of attention through the years involving large datasets obtained from critical systems. These include, but not limited to, the detection of medical diagnostic problems, fault detection in industrial machinery and robotics, intrusions in electronic security systems, video surveillance and document classification.

Among others, different neural networks approaches for one-class classification have arisen in previous years [85, 94]. One of these approaches is autoencoder which has been efficiently applied in a wide variety of application fields. However, the classical approach [56, 72] can not be directly applied to handle large-scale problems due to the excessive computational resources required to construct the model in those scenarios. In this chapter we present a new fast learning method for autoencoders that allows handling large-scale datasets. This research is based on previous results that propose the use of a cost function based on a square loss function that measures the error before the output neural function and scales it by the slope of the nonlinear activation functions at each point. The formula employed to obtain the optimal weights is derived from a system of linear equations that its further transformed by means of the Singular Value Decomposition (SVD).

This chapter is organized as follows. Section 3.1 contains a brief review of the main methods for one-class classification providing a general overview of this research field. Section 3.2 describes two previous techniques that will be used in this work. Section 3.3 describes our proposed learning method for autoencoders that is evaluated in Section 3.4 thanks to a comparative study. Finally, Section 3.5 sums up the contents of the chapter.

3.1 Background

Over the years a considerable amount of methods have been proposed to solve the one-class classification problem. In the literature can be found several reviews of one-class classification methods that analyze their suitability in different application fields such as mobile-masquerader detection [88], speaker verification problem [19], biometrics [6] or developing credit-scoring system [64].

Khan and Madden [65, 66] proposed a taxonomy for the study of one-class classification methods based on three broad categories: availability of training data, methodology used and application domain. The first category refers to learning with positive data only, learning with positive examples and with a limited amount of negative samples (or artificially generated outliers) or learning with positive and unlabeled data. Regarding the second category Khan and Madden indicate that the most important one-class classification algorithms can be categorized as either based on one-class support vector machines (OSVM) [127, 113] or based on other methods such as neural networks [8, 10], decision trees [52] or nearest neighbors [84]. Finally, for the third category authors distinguish between one-class classification applied in the field of document classification or in other domains. Other works should be mentioned such as the Mazhelis's research [88] who proposed a taxonomy of one-class classification techniques taking into account three main considerations: the internal model used by classifier, the type of data and the ability of classifiers to handle temporal relations among features.

Moreover, Tax [126] established a categorization for one-class classification methods distinguishing three main approaches: the density estimation methods, the boundary methods and the reconstruction methods. Density methods use a probabilistic approach to estimate the density of the positive class and assume that low density areas in the training set have a low probability of containing positive data [125]. As examples of density methods we can mention Gaussian model [9], mixture of Gaussians [33] and Parzen density estimators [97]. Boundary methods optimize a closed boundary around the target set. Some of well-known boundary methods are k-centers [145], one-class support vector machine [115] and support vector data description [128]. Finally, reconstruction methods involve training a regression model between inputs and outputs using only the positive data, and when a negative sample is mapped using this model the reconstruction error will be higher than the one obtained with a positive one. Some of the most important methods of this type are the k-means clustering [9], learning vector quantization [23], self-organizing maps [69], principal component analysis [9], diabolo networks [3, 51] and autoencoder networks [56]. The main differences among them are in the definition of subspaces, the reconstruction error and the optimization routine.

In this research we focus our attention on reconstruction methods, specifically in autoencoders. An autoassociative encoder [56, 72], or simply autoencoder or diabolo network [117], is a neural network approach to learn a representation of the data. It is a feedforward network which learn to map a model from its inputs to output nodes, through a narrow hidden layer, which attempts to reconstruct the input. As the network has a narrow hidden layer, it is forced to compress redundancies in the input while retaining and differentiating non-redundant information. In this way, the network is able to reduce noise in data by mapping inputs into the space of the correlation model and then the residuals of this mapping can be employed to detect novelties in future data points. It has been shown that autoencoders generate an input/middle layer mapping equivalent to principal component analysis (PCA) when linear activation functions are employed [3]. However, if the activation functions are nonlinear, the mapping is not equivalent to PCA and has different characteristics [55].

In the research presented by Sakurada and Yairi [109] a comparative study of autoencoders with linear PCA and kernel PCA on artificial data and real data was accomplished. Autoencoders have been widely applied in the novelty and anomaly detection problem. Hwang and Cho [54] analyzed the output characteristics of trained autoassociative multilayer perceptron and showed that it is a reliable solution for novelty detection. They also prove why nonlinearity in the hidden layer is necessary for novelty detection. The research by Thompson et al. [129] proposed a method for novelty detection through the application of an autoencoder. Sanz et al. [110] propose a new unsupervised method for monitoring the condition of rotating machinery from vibration analyses using wavelet transform and autoassociative neural networks. Miranda et al. [91] present a diagnosis system to incipient fault diagnosis in power transformers based on a set of autoassociative neural networks and the results of dissolved gas analysis.

The vast majority of previous research in this domain was oriented to develop learning algorithms for solving problems with small to medium-size datasets, for which the computational requirements were not an aspect to be taken into account. However, nowadays with the advent of new scenarios, such as the Internet of Things, the datasets in the field of anomaly detection have grown enormously in size. In this scenario, the classical methods, based on the available learning algorithms for autoencoders, present important limitations in its applicability and they can not be directly used, due to the computational load required to analyze the data. This aspect is accentuated in those applications that require a real-time analysis of the data. Therefore, this chapter presents a new fast learning method for autoencoders which allows its practical application in one-class classification domains where large-scale datasets exist.

3.2 Preliminaries

In this section, for the sake of comprehension, techniques that will serve as a starting point for this work are discussed. Firstly we will describe the Singular Value Decomposition (SVD) method, as it will be an important component of the proposed model and, afterwards, the LANN-SVD algorithm [40] that will be used for the learning process of the autoencoder.

3.2.1 Singular value decomposition

Low-rank matrix approximation is a minimization problem that tries to approximate a given matrix of data by another one (the optimization variable) subject to the constraint that the approximating matrix has reduced rank. According to the Eckart-Young-Mirsky theorem [34] the low-rank approximation with the h largest singular values is the solution with the least reconstruction error induced by the Frobenius norm. Therefore, based on this result, the partial singular value decomposition (SVD) provides the best approximation to the original matrix among all low-rank matrices. Given a real-valued matrix $\mathbf{P} \in \mathbb{R}^{m \times n}$, the full SVD of $\mathbf{P}$ is defined as the following:

$$\mathbf{P} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{3.1}$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\mathbf{V} \in \mathbb{R}^{n \times n}$ and $\mathbf{S} \in \mathbb{R}^{m \times n}$. Matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and $\mathbf{S}$ is a diagonal matrix with r non-negative values on the diagonal, in descending order, known as the singular values. Every matrix is guaranteed to have a SVD.

The SVD is frequently computed in an economy-size manner, where $\mathbf{U} \in \mathbb{R}^{m \times r}$, $\mathbf{V} \in \mathbb{R}^{n \times r}$ and $\mathbf{S} \in \mathbb{R}^{r \times r}$ and $r \leq min(m,n)$ is the maximum possible rank of matrix $\mathbf{X}$. This lightweight SVD is equivalent to the standard SVD but can be calculated much more efficiently, especially if $m \gg n$ or $n \gg m$.

3.2.2 LANN-SVD algorithm

LANN-SVD is a supervised learning method for training one-layer neural networks presented by Fontenla-Romero et al. [40]. It employs a new convex objective function based on the mean-squared error (MSE) but measured before the activation function. Consider a matrix of training samples $\mathbf{X} \in \mathbb{R}^{m \times n}$ and a vector $\mathbf{d} \in \mathbb{R}^{n \times 1}$ including the desired outputs. The output vector $\mathbf{y} \in \mathbb{R}^{n \times 1}$ of a one-layer feedforward neural network can be obtained using the following

equation:

$$\mathbf{y} = f(\mathbf{z}) = f(\mathbf{X}^T\mathbf{w})$$

where $\mathbf{w} \in \mathbb{R}^{m\times 1}$ is its weight vector and $f : \mathbb{R} \to \mathbb{R}$ the nonlinear function of the output neuron. To obtain the optimal weights it is common to derive the objective function in terms of the MSE on the training set. However, if f is nonlinear then local minima may appear in the objective function [120, 20], which may avoid achieving the global minima. To overcome this issue, the error is estimated before the nonlinear function [39]. In order to do this, the desired output needs to be backpropagated, i.e. $\bar{\mathbf{d}} = f^{-1}(\mathbf{d})$. Afterwards, the weights can be obtained by minimizing the MSE between $\mathbf{z} = \mathbf{X}^T\mathbf{w}$ and $\bar{\mathbf{d}}$ which leads to the following system of linear equations:

$$\mathbf{A}\mathbf{w} = \mathbf{b} \tag{3.2}$$

where $\mathbf{A}$ and $\mathbf{b}$ are defined as:

$$\mathbf{A} = \mathbf{X}\mathbf{F}\mathbf{F}\mathbf{X}^T \;\; ; \;\; \mathbf{b} = \mathbf{X}\mathbf{F}\mathbf{F}\bar{\mathbf{d}} \tag{3.3}$$

and $\mathbf{F} = diag(f'(\bar{d}_1), f'(\bar{d}_2), \ldots, f'(\bar{d}_n))$ being the diagonal matrix formed by the derivative of the f function at the components of the $\bar{\mathbf{d}}$ vector.

It has been proved that the global optimum obtained with this new objective function is approximately equivalent to the one of the regular MSE [39]. This approach resulted to be very efficient only when $n \gg m$, as the size of the system of linear equations in (3.2) and (3.3) relies on m. To make it efficient in the other case, that is when $m \gg n$, a transformation of the previous system of linear equations by means of SVD was proposed in [40].

Considering equations (3.2) and (3.3), the system of equations used as a starting point can be written as:

$$\mathbf{X}\mathbf{F}\mathbf{F}\mathbf{X}^T\mathbf{w} = \mathbf{X}\mathbf{F}\mathbf{F}\bar{\mathbf{d}} \tag{3.4}$$

By replacing $\mathbf{P} = \mathbf{X}\mathbf{F}$, this system can be rewritten as:

$$\mathbf{P}\mathbf{P}^T\mathbf{w} = \mathbf{P}\mathbf{F}\bar{\mathbf{d}} \tag{3.5}$$

Using SVD, factorization of matrix $\mathbf{P}$ can be done ($\mathbf{P} = \mathbf{U}\mathbf{S}\mathbf{V}^T$). By replacing $\mathbf{P}$ in equation (3.5) we obtain:

$$\mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}^T\mathbf{U}^T\mathbf{w} = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{F}\bar{\mathbf{d}} \tag{3.6}$$

Multiplying on the left by $\mathbf{U}^{-1}$ both sides of the equation, the following system is obtained:

$$\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}^T\mathbf{U}^T\mathbf{w} = \mathbf{S}\mathbf{V}^T\mathbf{F}\bar{\mathbf{d}} \tag{3.7}$$

As matrix $\mathbf{V}$ is orthogonal then it holds that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, and hence the system in (3.7) can be simplified to get:

$$\mathbf{S}\mathbf{S}^T\mathbf{U}^T\mathbf{w} = \mathbf{S}\mathbf{V}^T\mathbf{F}\bar{\mathbf{d}} \tag{3.8}$$

Although $\mathbf{S}\mathbf{S}^T$ is a square matrix ($m \times m$), its inverse can not be calculated in general, as when $m > n$ it is rank deficient (rank at most n as $\mathbf{S} \in \mathbb{R}^{m \times n}$). Therefore, the use of the Moore-Penrose pseudoinverse is proposed. This is a general way to find the solution to a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$. Moore and Penrose showed that there is a general solution to these equations of the form $\mathbf{x} = \mathbf{A}^\dagger\mathbf{b}$. This solution has the following properties:

- If $m = n$, then $\mathbf{A}^\dagger = \mathbf{A}^{-1}$ when $\mathbf{A}$ is full rank. The case when $\mathbf{A}$ is not full rank will be considered below.
- If $m > n$, the solution is the one that minimizes $||\mathbf{A}\mathbf{x} - \mathbf{b}||$. Then, the pseudoinverse provides the most closest solution in a least-squared sense.
- If $m < n$, the solution minimizes the 2-norm of $\mathbf{x}$, but this case will never happen since $\mathbf{S}\mathbf{S}^T$ is a $m \times m$ matrix.

Therefore, applying the Moore-Penrose pseudoinverse to equation (3.8) the following expression is obtained:

$$\mathbf{U}^T\mathbf{w} \approx (\mathbf{S}\mathbf{S}^T)^\dagger\mathbf{S}\mathbf{V}^T\mathbf{F}\bar{\mathbf{d}} \tag{3.9}$$

If both sides are multiplied by $\mathbf{U}$, as it is also an orthogonal square matrix and therefore $\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$, we obtain:

$$\mathbf{w} \approx \mathbf{U}(\mathbf{S}\mathbf{S}^T)^\dagger\mathbf{S}\mathbf{V}^T\mathbf{F}\bar{\mathbf{d}} \tag{3.10}$$

Considering that $\mathbf{H} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, and consequently $\mathbf{U}^T\mathbf{H} = \mathbf{U}^T\mathbf{U}\mathbf{S}\mathbf{V}^T$, then $\mathbf{U}^T\mathbf{H} = \mathbf{S}\mathbf{V}^T$. Using this result in equation (3.10), and the fact that $\mathbf{H} = \mathbf{X}\mathbf{F}$, the final solution is obtained:

$$\mathbf{w} \approx \mathbf{U}(\mathbf{S}\mathbf{S}^T)^\dagger\mathbf{U}^T\mathbf{X}\mathbf{F}\mathbf{F}\bar{\mathbf{d}} \tag{3.11}$$

that allows to approximate the global optimum of the objective function proposed in [39].

If a economy-sized SVD is employed, then $\mathbf{S} \in \mathbb{R}^{r \times r}$ is a square diagonal matrix and it holds that $\mathbf{S} = \mathbf{S}^T$. With this approach, in equation (3.11) it is only necessary to compute the inverse, or pseudoinverse if the matrix is ill-conditioned, of $(\mathbf{S}\mathbf{S})$, a $r \times r$ diagonal matrix where $r = min(m,n)$. Thus, the size of the matrix depends on the smaller value between the number of samples and the number of features. All the steps to implement the learning method are showed in Algorithm 1, where $.*$ notation represents the Hadamard product and $pinv(\cdot)$ computes the pseudoinverse of a matrix.

Algorithm 1 : LANN-SVD

Inputs: $\mathbf{X} \in \mathbb{R}^{m \times n}$ ▷ Dataset
$\mathbf{d} \in \mathbb{R}^{n \times 1}$ ▷ Desired outputs
f ▷ Nonlinear activation function
Output: $\mathbf{w} \in \mathbb{R}^{m \times 1}$ ▷ Optimal weights

function LANN-SVD($\mathbf{X}$,$\mathbf{d}$,f)
  $\mathbf{X} = [ones(1,n); \mathbf{X}]$; ▷ The bias is added (first row)
  $\bar{\mathbf{d}} = f^{-1}(\mathbf{d})$; ▷ Inverse of the neural function
  $\mathbf{f}_d = f'(\bar{\mathbf{d}})$; ▷ Derivate of the neural function
  $\mathbf{P} = \mathbf{X} * diag(\mathbf{f}_d)$;
  $[\mathbf{U}, \mathbf{S}, \sim] = svd(\mathbf{P})$; ▷ Economy size SVD
  $\mathbf{b} = \mathbf{X} * (\mathbf{f}_d . * \mathbf{f}_d . * \bar{\mathbf{d}})$;
  $\mathbf{w} = \mathbf{U} * pinv(\mathbf{S} * \mathbf{S}) * (\mathbf{U}^T * \mathbf{b})$;
end function

3.3 Proposed method

Consider the autoencoder neural network with one hidden layer depicted in Figure 3.1, being f_1 and f_2 nonlinear activation functions for the hidden and output neurons and $\mathbf{W}_1 \in \mathbb{R}^{m \times h}$ and $\mathbf{W}_2 \in \mathbb{R}^{h \times m}$ the weight matrices for the first and the second layer, respectively. In this network the output layer has the same number of nodes (m) as the input layer, since the goal is to reconstruct the input data, and the number of hidden neurons (h) is always strictly lower than the number of inputs. Given a training set represented by an input matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, where m is the number of input variables (including possibly a bias) and n the number of data points, the outputs of the first layer $\mathbf{H} \in \mathbb{R}^{h \times n}$ can be calculated by the following equation

$$\mathbf{H} = f_1(\mathbf{W}_1^T \mathbf{X}) \tag{3.12}$$

and the output of the network ($\hat{\mathbf{X}} \in \mathbb{R}^{m \times n}$) is given by:

$$\hat{\mathbf{X}} = f_2(\mathbf{W}_2^T \mathbf{H}) \tag{3.13}$$

Autoencoders are trained to minimize the reconstruction error between the input and the output defined by the following equation:

$$E(\mathbf{X}, \hat{\mathbf{X}}) = \|\mathbf{X} - \hat{\mathbf{X}}\|_F^2 \tag{3.14}$$

where $\|\cdot\|_F$ is the Frobenius norm.

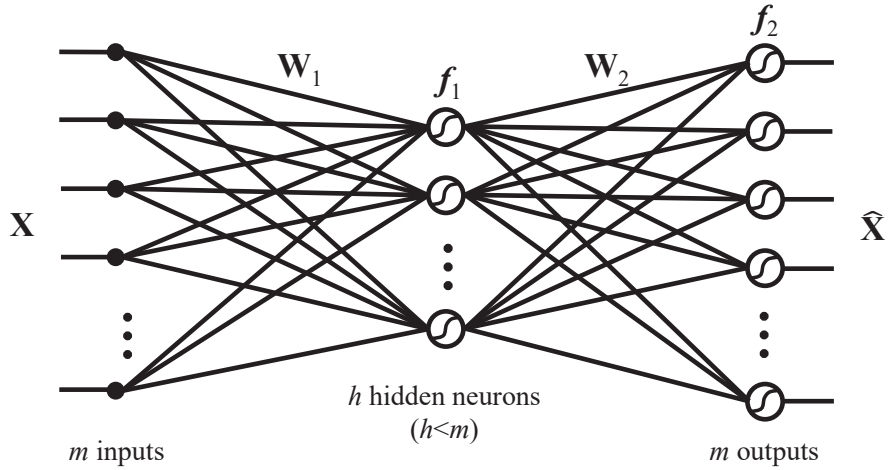


Figure 3.1: Autoencoder neural network.

The usual way to train the autoencoder is by classical first-order iterative learning algorithms such as backpropagation or by fast second-order iterative methods such as scaled conjugate gradient descent. However, these methods can present computational difficulties, due to time or space complexity, in applications with large datasets where a large number of iterations is required to achieve the convergence of the method. In this research, a new non-iterative learning method to train an autoencoder which allows a substantial improvement in the training time required to obtain the optimal weights is presented.

In the proposed approach the learning process is carried out using a two-step procedure, one step for each of the layers of the network. In the first layer, the method learns a vector space embedding for the input data extracting meaningful low-dimensional representation by means of Singular Value Decomposition. The SVD of the input matrix $\mathbf{X}$ is a factorization of the form

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{3.15}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$ and $\mathbf{S} \in \mathbb{R}^{m\times n}$ is a diagonal matrix with the singular values on the diagonal. Hence, the optimal weights for the first layer are obtained using the first h columns of the $\mathbf{U}$ matrix of the optimal rank-h SVD for the input data ($\mathbf{X}$). Then, applying equation 3.12, we get the outputs of the first layer $\mathbf{H} \in \mathbb{R}^{h\times n}$.

For the second layer, the goal is to reconstruct the input from the low-dimensional representation provided by the output of the hidden layer. This is a minimization problem based on a sum-of-squares error function. If the activation function of the output neurons is linear the solution for this least-squares problem can be found exactly in a simple closed form using

the pseudo-inverse of a matrix [10]. However, in the most general case (nonlinear activation functions) this solution is no longer possible and the use of iterative algorithms, based mainly on gradient descent, is required. Although this kind of methods are effective they can need a large number of steps to converge to the optimal value, which is not a very desirable behavior in problems with large datasets since they may cause enormous training times. Therefore, to obtain the optimal weights for the second layer the LANN-SVD algorithm, described in Section 3.2.2, is employed in this work. In the proposed method the inputs of the LANN-SVD algorithm are the outputs of the hidden layer $\mathbf{H} \in \mathbb{R}^{h \times n}$ (see Figure 3.1). As LANN-SVD is a supervised learning method, the desired outputs have to be provided. Then, since the goal of the autoencoder is to reconstruct the input data, the desired outputs are actually the same inputs ($\mathbf{X}$). The main advantages of this non-iterative learning algorithm are that it can be used with nonlinear activation functions and its complexity depends on the minimum value of n and h thus implying savings in computational requirements for large datasets, especially in the usual case in which the number of hidden neurons (h) is much smaller than the number of data points. For the output neurons of the network the weights of the second layer are independent of each other, hence the minimization problem can be split in several sub-problems (one for each output neuron) that can be solved separately.

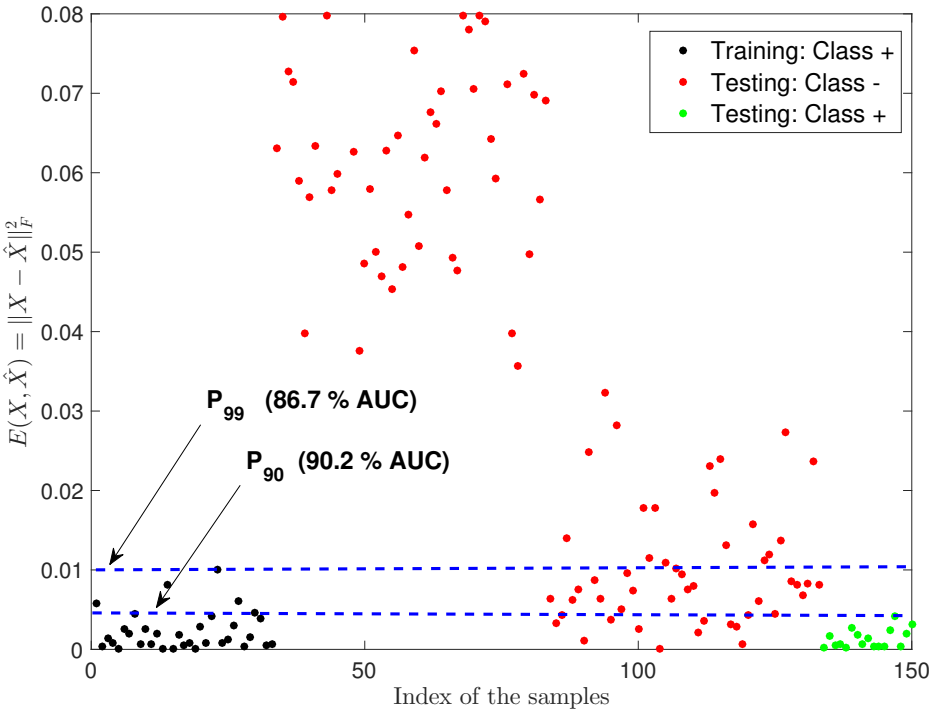


Figure 3.2: Decision threshold.

This fast learning SVD-Autoencoder allows its practical application in one-class classification domains where large-scale datasets exist. In order to do so, a regression model is trained using only positive data, and when a negative sample is mapped using this model the reconstruction error will be higher than the one obtained with a positive training sample. This is true when positive and negative samples are separable. However, that is not the usual case in a real world scenario. Besides, an outlier in the training set can have a high reconstruction error that could lead to bad classification of new negative samples. In order to prevent this and adjust our classifier, a threshold th based on the value of the j^{th} percentile of the reconstruction error achieved with the training data is employed (see equation 3.14). P_j, the j^{th} percentile, leaves below the $j\%$ observations of the training set reconstruction error. Figure 3.2 depicts an example of this proposed threshold.

As can be seen, if a high error, like the one of the P_{99}, in the training data (black dots) is used as threshold, several negative samples (red dots) fall below it and are misclassified as positive samples. However, if a smaller percentile like the P_{90} is used, the overall classification improves significantly, from 86.7% (P_{99}) to 90.2% of AUC (Area Under the Roc Curve). Thus, using this threshold the classifier can be adjusted and overfitting avoided. A new parameter $p \in \mathbb{N}$ that takes values from 1 to 99 is added to the proposed method to select the percentile used as threshold.

Algorithm 2 details all the steps of the proposed method. As can be seen, for obtaining the outputs of the first layer only matrix $\mathbf{U}$ of the SVD is required, and therefore the computation of matrices $\mathbf{V}$ and $\mathbf{S}$ can be avoided. Furthermore, each LANN-SVD sub-problem is independent of the others. This fact leads us to another interesting characteristic of the proposed method, its parallelizable nature. Each one of the m LANN-SVD sub-problems can be solved in a different processor at the very same time, making the proposed algorithm highly scalable and efficient when the number of variables m is high.

Algorithm 2 : SVD-autoencoder

Training Stage

Inputs: $\mathbf{X} \in \mathbb{R}^{m \times n}$ ▷ Input data (n samples)
f_1, f_2 ▷ Activation functions for the 1st and 2nd layers, respectively
h ▷ Number of hidden neurons ($h < m$)
p ▷ Decision threshold parameter ($0 < p < 100$)

Outputs: $\mathbf{W}_1 \in \mathbb{R}^{m \times h}, \mathbf{W}_2 \in \mathbb{R}^{h \times m}$ ▷ Optimal weights for the 1st and 2nd layers
$th_p \in \mathbb{R}$ ▷ Decision threshold for classification

function TRAIN-SVD-AUTOENCODER($\mathbf{X}$,f_1,f_2,h)
  $[\mathbf{U}, \sim, \sim] = svd(\mathbf{X})$ ▷ Economy size SVD
  $\mathbf{W}_1 = \mathbf{U}(:, 1:h)$ ▷ Rank-h matrix
  $\mathbf{H} = f_1(\mathbf{W}_1^T \mathbf{X})$ ▷ Outputs of the hidden layer
  for i = 1 to m ▷ One sub-problem for each output neuron
    $\mathbf{W}_2(:, i) = LANN\_SVD(\mathbf{H}, \mathbf{X}(i, :), f_2)$
  end
  $\hat{\mathbf{X}} = f_2(\mathbf{W}_2^T \mathbf{H})$ ▷ Outputs of the network
  for i = 1 to n
    $\mathbf{e}_{Tr}(i) = \|\mathbf{X}(:, i) - \hat{\mathbf{X}}(:, \mathbf{i})\|^2$ ▷ Reconstruction error for training data
  end
  $th_p = percentile(\mathbf{e}_{Tr}, p)$ ▷ The p^{th} percentile of $\mathbf{e}_{Tr}$ is the decision threshold
end function

Classification Stage

Inputs: $\mathbf{x} \in \mathbb{R}^m$ ▷ New test sample
$\mathbf{W}_1, \mathbf{W}_2$ ▷ Optimal weights for the 1st and 2nd layers, respectively
th ▷ Decision threshold

Outputs: $Result \in \{Positive, Negative\}$ ▷ Classification decision

function TEST-SVD-AUTOENCODER($\mathbf{X}$,f_1,f_2,h)
  $Result = Positive$
  $\mathbf{H} = f_1(\mathbf{W}_1^T \mathbf{x})$ ▷ Outputs of the hidden layer
  $\hat{\mathbf{x}} = f_2(\mathbf{W}_2^T \mathbf{H})$ ▷ Outputs of the network
  $e_{Tst} = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$ ▷ Reconstruction error for the new data
  if ($e_{Tst} > th$) then
    $Result = Negative$
  end
end function

3.4 Experimental study

In this section several experiments are presented and the results discussed in order to assess the performance and the characteristics of the SVD-autoencoder.

3.4.1 Comparative study of performance

This first study tries to demonstrate how the proposed method behaves in comparison with some other well known one-class methods available in the literature. Our aim is to prove that the SVD-autoencoder exhibits a good classification accuracy when dealing with classical benchmark datasets. With this goal, three one-class classification methods were selected for the study:

- The regular autoencoder neural network [72] using the scaled conjugate gradient descent algorithm [93] to obtain the optimal weights. This is the original method in which SVD-autoencoder is based on.
- The Approximate Polytope Ensemble algorithm (APE) proposed by Casale et al. [26]. This boundary method has been selected because it showed a very good performance when compared against several standard classification algorithms. In this work we projected data to 2 dimensional spaces (APE-2), because it provides better results than using 1D projections. A brief description of this method can be seen in the next chapter of this thesis, where a distributed learning method based on this APE algorithm is presented.
- The state-of-the-art One-class ν-Support Vector Machine [115] with nonlinear kernel (radial basis function, RBF).

Eleven UCI machine learning repository datasets [76], described in Table 4.8, where used to test these methods. For each dataset, one-class problems were obtained considering the data of one of the classes as the positive examples and the rest of the data as the negative examples.

All the experiments were carried out on an Intel Core i7-4790 processor with 3.60GHz clock speed and 16GB RAM. Besides, in order to obtain significant results a 10-fold cross-validation was employed and every experiment was repeated 30 times. All datasets have been previously normalized in the interval [0, 1]. In the case of ν-SVM and APE-2 the parameters where tuned to achieve comparable results to those reported in the literature.

Dataset	# classes	# instances	# features
Balance	3	625	4
Breast	2	683	10
Car	4	1728	6
Glass	3	214	10
Haberman	2	306	3
Ionosphere	2	351	34
Iris	3	150	4
Pima	2	768	8
Sonar	2	208	60
TicTacToe	2	958	9
Wine	3	178	13

Table 3.1: Characteristics of the datasets employed in the comparative study.

Table 3.2 shows the mean test results of the proposed method against the classical Autoencoder. In the second column, the class considered as target and the number of samples of the target and negative classes are displayed, respectively. Remaining columns of Table 3.2 show the optimal parameters for each method, the mean Area Under the Roc Curve (AUC) and the standard deviation (SD) in percentage for the 30 simulations. The proportional training time comparison against SVD-autoencoder is also showed in parentheses. Both methods have the same parameters, except for the maximum number of training epochs in the Autoencoder, that was set by default to 1000. The rest of the values were obtained empirically. The activation function of the second layer f_2 can be logistic or linear (LOG and LIN in Table 3.2). The value of f_1 is not displayed because it is a nonlinear function in every problem (LOG). As it was said in Section 3.3, the number of hidden neurons h should be strictly lesser than the number of features ($1 \leq h \leq m$). Finally, P_j, represents the j^{th} percentile used to establish the classification threshold, leaving below the $j\%$ observations of the training set reconstruction error (parameter p in the Algorithm 2). A pairwise t-test [31] was applied to evaluate the statistical difference at a 95% significance level. As can be seen, fourteen times against six, SVD-autoencoder achieves better significant results than the classical one. Besides, it can be observed that, on average, the new proposed method is hundreds of times faster than the original Autoencoder.

The results of the comparative of the method proposed against APE-2 and ν-SVM algorithms are presented in Table 3.3. In the case of ν-SVM, the RBF kernel was used and the ν parameter, that represents the estimation of spurious data, was set to 0.01. In APE-2, the number of projections was set by default to 100. The rest of the parameters have been obtained empirically and their optimal values are showed in Table 3.3. As in the previous table,

		SVD-autoencoder		Autoencoder	
Dataset	Class	Parameters	AUC±SD	Parameters	AUC±SD
Balance	1 (288,337)	$LOG, h:3, P_{85}$	**83.69±3.3***	$LIN, h:3, P_{80}$	81.53±3.7 (281)
	2 (49,576)	$LOG, h:3, P_{55}$	**66.29±10.8***	$LIN, h:3, P_{75}$	59.33±10.6 (439)
	3 (288,337)	$LOG, h:3, P_{85}$	**83.81±3.0***	$LIN, h:3, P_{80}$	81.26±3.8 (277)
Breast	1 (444,239)	$LOG, h:1, P_{95}$	**96.69±1.5**	$LOG, h:1, P_{95}$	96.62±1.4 (76)
	2 (239,444)	$LOG, h:5, P_{95}$	**94.93±2.9***	$LIN, h:1, P_{50}$	68.85±6.0 (100)
Car	1 (1210,518)	$LOG, h:5, P_{75}$	**71.87±2.1***	$LIN, h:4, P_{75}$	71.07±2.7 (230)
	2 (384,1344)	$LIN, h:5, P_{95}$	89.91±1.9	$LIN, h:5, P_{95}$	**92.75±2.1*** (253)
	3 (69,1659)	$LIN, h:3, P_{99}$	**97.57±2.9***	$LIN, h:3, P_{95}$	95.77±4.0 (164)
	4 (65,1663)	$LIN, h:3, P_{99}$	**97.30±3.3**	$LIN, h:2, P_{99}$	97.11±3.7 (195)
Glass	1 (70,144)	$LIN, h:3, P_{99}$	**97.64±3.0***	$LIN, h:5, P_{95}$	94.23±6.6 (259)
	2 (76,138)	$LIN, h:8, P_{99}$	**98.22±3.6***	$LIN, h:9, P_{85}$	87.57±7.7 (325)
	3 (68,146)	$LOG, h:9, P_{95}$	**91.47±6.3***	$LOG, h:9, P_{95}$	83.43±7.0 (191)
Haberman	1 (225,81)	$LIN, h:2, P_{70}$	55.23±5.2	$LIN, h:2, P_{65}$	**56.53±4.1*** (234)
	2 (81,225)	$LIN, h:2, P_{55}$	**55.37±8.6***	$LIN, h:2, P_{75}$	52.93±9.0 (261)
Ionosphere	1 (225,126)	$LIN, h:5, P_{95}$	**94.82±2.6***	$LIN, h:5, P_{95}$	92.01±3.3 (126)
	2 (126,225)	$LIN, h:32, P_{55}$	50.00±7.1	$LIN, h:1, P_{99}$	50.00±2.0 (16)
Iris	1 (50,100)	$LIN, h:2, P_{99}$	100±0.0	$LIN, h:2, P_{99}$	100±0.0 (153)
	2 (50,100)	$LIN, h:3, P_{99}$	**93.41±9.2**	$LIN, h:3, P_{95}$	93.20±5.4 (154)
	3 (50,100)	$LOG, h:3, P_{95}$	**89.33±6.8***	$LIN, h:3, P_{95}$	84.98±7.2 (185)
Pima	1 (500,268)	$LIN, h:2, P_{70}$	68.54±3.1	$LIN, h:1, P_{70}$	**69.13±3.2*** (98)
	2 (268,500)	$LOG, h:7, P_{60}$	57.20±7.3	$LIN, h:5, P_{70}$	**57.87±4.6** (226)
Sonar	1 (97,111)	$LIN, h:5, P_{95}$	59.57±5.7	$LIN, h:25, P_{95}$	**66.86±7.9*** (274)
	2 (111,97)	$LOG, h:24, P_{95}$	65.92±7.7	$LIN, h:12, P_{85}$	**72.44±7.6*** (81)
TicTacToe	1 (626,332)	$LOG, h:6, P_{70}$	66.05±2.9	$LOG, h:7, P_{90}$	**72.47±2.6*** (144)
	2 (332,626)	$LOG, h:1, P_{95}$	**72.78±3.9***	$LOG, h:1, P_{95}$	71.76±4.2 (168)
Wine	1 (59,119)	$LIN, h:1, P_{95}$	**96.14±5.5***	$LIN, h:4, P_{99}$	95.23±6.3 (163)
	2 (71,107)	$LIN, h:5, P_{90}$	**83.82±8.8**	$LIN, h:2, P_{95}$	83.47±7.6 (164)
	3 (48,130)	$LIN, h:9, P_{99}$	**95.89±5.3**	$LIN, h:3, P_{99}$	95.40±6.3 (149)

Table 3.2: AUC ± SD in percentage. In parentheses is the proportional training time comparison against SVD-autoencoder. Absolute best results are boldfaced. Methods that are statistically different are marked with an asterisk.

		SVD-autoenc	APE-2		ν-SVM	
Dataset	Class	AUC±SD	α	AUC±SD	σ	AUC±SD
Balance	1	83.69±3.3	0.02	**90.84±3.4**† (37)	0.35	87.12±3.9 (79)
	2	66.29±10.8	0.05	**84.62±10.2**† (36)	0.4	71.95±12.6 (79)
	3	83.81±3.0	0.03	**90.90±3.0**† (38)	0.45	87.17±3.8 (74)
Breast	1	**96.69±1.5**†	0.2	95.14±1.9 (16)	0.3	94.11±2.7 (49)
	2	**94.93±2.9**†	0.15	84.53±4.9 (24)	9	92.88±3.8 (49)
Car	1	71.87±2.1	$5e^{-4}$	71.83±1.7 (31)	0.4	**79.03±2.1**† (72)
	2	89.91±1.9	0.15	**95.07±1.5**† (74)	0.6	87.63±2.8 (88)
	3	97.57±2.9	0.7	**98.48±1.5**† (66)	9	86.61±6.5 (94)
	4	97.30±3.3	0.6	**99.20±1.5**† (62)	16	91.48±7.6 (92)
Glass	1	**97.64±3.0**†	1.7	95.35±4.8 (12)	8.5	92.22±6.9 (46)
	2	**98.22±3.6**†	0.55	93.06±6.1 (8)	0.4	87.68±7.4 (27)
	3	**91.47±6.3**†	0.4	87.58±8.0 (8)	0.8	88.87±6.6 (29)
Haberman	1	55.23±5.2	$1e^{-4}$	50.51±3.1 (26)	0.15	**56.37±4.5**† (98)
	2	**55.37±8.6**	$1e^{-4}$	53.04±7.3 (31)	0.15	54.99±7.9 (88)
Ionosphere	1	**94.82±2.6**†	0.4	90.09±3.0 (5)	0.9	91.15±4.3 (14)
	2	50.00±7.1	-0.99	50.00±4.0 (3)	0.3	**51.99±2.8**† (6)
Iris	1	100±0.0	1.5	100±0.0 (20)	0.8	100±0.0 (81)
	2	**93.41±9.2**†	1.1	91.31±5.6 (21)	0.5	89.50±8.8 (82)
	3	89.33±6.8	1.1	**91.64±6.8**† (20)	0.3	89.06±8.7 (81)
Pima	1	**68.54±3.1**†	0	62.03±3.2 (17)	0.2	65.24±3.3 (60)
	2	**57.20±7.3**	$1e^{-3}$	56.87±5.0 (24)	0.25	57.11±5.0 (58)
Sonar	1	59.57±5.7	0.3	**60.43±7.4** (2)	1	59.14±8.3 (7)
	2	65.92±7.7	0.2	**66.53±7.0** (1)	0.9	66.29±7.1 (4)
TicTacToe	1	**66.05±2.9**†	0.07	62.63±2.3 (17)	0.85	54.94±2.6 (49)
	2	72.78±3.9	0.05	63.07±4.3 (41)	4	**83.57±3.6**† (74)
Wine	1	**96.14±5.5**†	0.9	95.36±4.8 (12)	4	92.94±7.4 (44)
	2	**83.82±8.8**†	0.55	79.61±8.1 (8)	0.8	78.66±7.6 (30)
	3	**95.89±5.3**†	0.9	93.89±6.7 (7)	15	**9**2.87±9.0 (30)

Table 3.3: AUC $\pm$ SD in percentage. In parentheses is the proportional training time comparison against SVD-autoencoder. Absolute best results are boldfaced. Methods which differences are not statistically significant are marked with a † symbol.

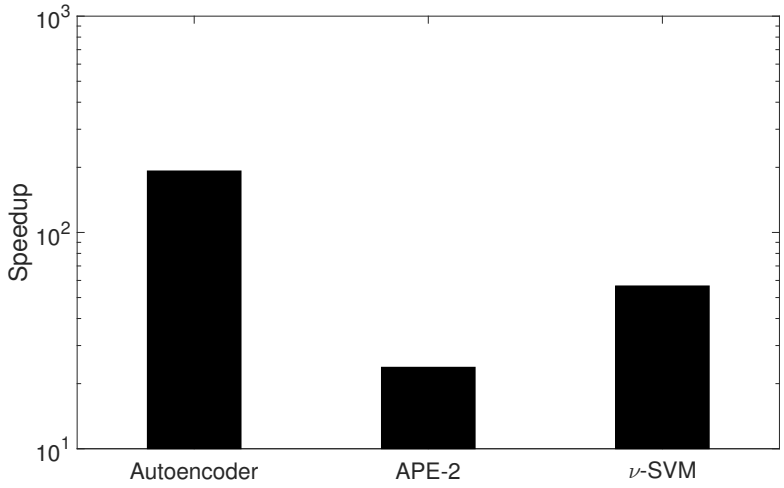


Figure 3.3: Average speedup factor relative to SVD-autoencoder times.

the second column of Table 3.3 shows the class considered as target. In this case, the results show that on a bigger number of problems, SVD-autoencoder is significantly better at 95% significance level than the other two methods.

Figure 3.3 shows the average speedup of the proposed SVD-autoencoder against the other three methods in the 28 one-class problems studied. As can be seen, the SVD-autoencoder is several times faster than the other approaches, specially in the cases of the classical Autoencoder and the state-of-the-art ν-SVM. Thus, it can be concluded that, with respect to classification performance SVD-autoencoder is a competitive option and, in terms of computational time, this approach considerably improves the results obtained by the other methods.

3.4.2 Performance of the SVD-autoencoder with large-scale datasets

This second study intends to demonstrate that SVD-autoencoder shows a good balance between classification accuracy and training time when dealing with large-scale datasets, where the high size comes from the number of samples. Besides, the proposed SVD-autoencoder is compared in terms of performance and training times against the classical Autoencoder [72].

Three large datasets were employed. Table 4.3 shows their main characteristics. The first one is a subset of the KDD Cup 99 dataset [1]. This dataset was used for the KDD (Knowledge Discovery and Data Mining Tools Conference) Cup 99 Competition. The complete version has

Dataset	# classes	# instances	# features
KDD Cup	2 (97278, 396743)	494021	6
MiniBooNE	2 (93565, 36499)	130064	50
Covertype	2 (568772, 12240)	581012	54

Table 3.4: Characteristics of the datasets.

almost 5 million input patterns and each record represents a TCP/IP connection that is composed of 41 features. KDD version employed in this work contains 494021 samples and only 6 features, as in [14] those were found to be the most representative and lead to better results than the competition winner and other algorithms. The second dataset is the MiniBooNE dataset [76], used to distinguish electron neutrinos (signal) from muon neutrinos (background). It consist of 50 features and 130064 samples divided in two classes (signal and background).Finally, the last one is the Forest Covertype dataset [76], used to classify forest cover type from cartographic variables. It has 581012 input patterns, each one of them composed of 54 features. In the previous chapter a smaller version of this dataset was used in the context of supervised classification. The original dataset distinguish between seven types of forest cover, but as five of them are different species of pine trees, we reduced the dataset to two classes by considering all the five pine species as one class (568772 samples) and the other two species as the other (12240 samples). Each problem was evaluated using 10-fold stratified cross-validation on 10 different permutations of the data and all datasets have been previously normalized in the interval [0, 1]. As previously, all the experiments were carried out on an Intel Core i7-4790 processor with 3.60GHz clock speed and 15.9Gb RAM.

	Target Class 0		Target Class 1	
	Autoencoder	SVD-autoencoder	Autoencoder	SVD-autoencoder
Hidden Neurons	3	2	4	3
A.F. 1^{st} layer	LOG	LOG	LOG	LOG
A.F. 2^{nd} layer	LOG	LIN	LOG	LOG
Threshold	P_{80}	P_{90}	P_{95}	P_{95}
AUC $\pm$ SD	89.87$\pm$0.31	**91.78$\pm$0.17**	94.01$\pm$1.21	**97.77$\pm$0.17**
Training Time	10.520	**0.029**	256.452	**0.230**

Table 3.5: Parameters used for the proposed method and the Autoencoder in KDD Cup dataset, and results obtained. Absolute best results are boldfaced.

Table 3.5 presents the optimum parameters employed and the results obtained in the experiments with the KDD Cup 99 dataset. In the second column of the table, class 0 was used to train the model. This means that 87551 samples were used to train and 406470 samples were used to test in each execution of the experiments. In the third column class 1 was the target class,

then 357069 training samples and 136952 testing samples were employed in the experiments. As can be seen, in the two problems derived from the KDD Cup dataset, SVD-autoencoder achieves better significant results than the classical approach. In the case of the Autoencoder, as in the previous experiments, the maximum number of training iterations was set to 1000. As can be seen, the differences in training time among this classical iterative method and the non-iterative one proposed are enormous.

	Target Class 0		Target Class 1	
	Autoencoder	SVD-autoencoder	Autoencoder	SVD-autoencoder
Hidden Neurons	31	19	6	6
A.F. 1^{st} layer	LOG	LOG	LOG	LOG
A.F. 2^{nd} layer	LIN	LIN	LIN	LIN
Threshold	P_{65}	P_{70}	P_{70}	P_{85}
AUC $\pm$ SD	65.12$\pm$0.86	**71.35$\pm$0.39**	70.88$\pm$1.93	**71.67$\pm$0.20**
Training Time	176.465	**0.504**	245.088	**0.532**

Table 3.6: Parameters used for the proposed method and the Autoencoder in MiniBooNE dataset, and results obtained. Absolute best results are boldfaced.

Results and parameters of each MiniBooNE experiment are displayed on Table 3.6. When class 0 was selected to train the model, 32849 training samples and 97215 testing samples were used. In the case that class 1 was the target class, 84209 training samples and 45855 testing samples were employed in the experiments. As can be seen in Table 3.6, SVD-autoencoder achieves better significant results than the classical approach in both MiniBooNE problems. Once again, the savings in training time using the proposed SVD-autoencoder are very big. In spite of using less training samples than in the previous experiments (see Table 3.5), training times are longer in both methods. This is due to the number of inputs/outputs and hidden neurons, that is higher in this case.

	Target Class 0		Target Class 1	
	Autoencoder	SVD-autoencoder	Autoencoder	SVD-autoencoder
Hidden Neurons	20	9	38	39
A.F. 1^{st} layer	LOG	LOG	LOG	LOG
A.F. 2^{nd} layer	LIN	LOG	LIN	LOG
Threshold	P_{65}	P_{70}	P_{80}	P_{80}
AUC $\pm$ SD	64.55$\pm$0.38	**69.77$\pm$0.37**	76.72$\pm$1.13	**77.18$\pm$0.92**
Training Time	1956.312	**5.131**	60.07	**0.247**

Table 3.7: Parameters used for the proposed method and the Autoencoder in Covertype dataset, and results obtained. Absolute best results are boldfaced.

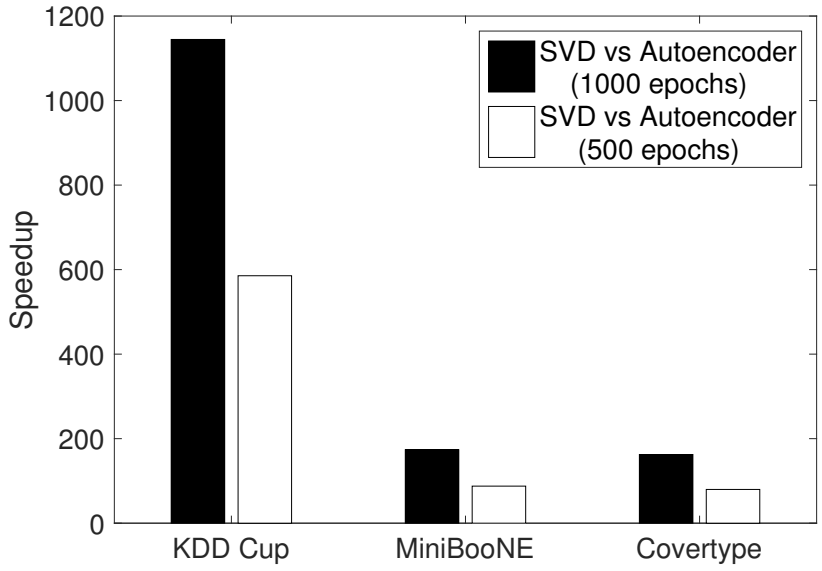


Figure 3.4: Speedup factor for SVD-autoencoder vs Autoencoder.

Analogously, Table 3.7 shows the optimum parameters used and the results achieved in the experiments made using the Covertype dataset. When class 0 was selected to train the model, 511895 training samples and 69117 testing samples were used. In the case that class 1 was the target class, 11016 training samples and 569996 testing samples were employed in the experiments. In this case, SVD-autoencoder achieves better significant results than the classical method only when the target class is 0. However, the big savings in training time obtained using the proposed SVD-autoencoder, thanks to its non-iterative nature, make it an adequate choice for both Covertype problems.

A final experiment was made to measure the training time improvements using the proposed SVD-Autoencoder instead of the classical Autoencoder for every large dataset studied. In order to do a fair comparison, 90000 patterns of the class with the higher number of samples were used for each dataset. A non-linear activation function for the output neurons and the maximum number of neurons in the hidden layer were also employed. As it was said in Section 2, the number of neurons in the hidden layer h has to be strictly lesser than the number of inputs. Thus, 5, 49 and 53 neurons were selected for the KDD, MiniBooNE and Covertype experiments, respectively. Figure 3.4 shows the average speedup factor, measuring the times SVD-autoencoder is faster than the classical Autoencoder (with 500 and 1000 maximum number of epochs). The differences between both methods are remarkable, specially in the case of KDD Cup dataset, where the small number of hidden and output neurons emphasize these

differences.

3.4.3 Parallel performance of the SVD-autoencoder

In this last experimental section we introduce an analysis of how the parallelization of the SVD-autoencoder algorithm reduces the time needed to train a classifier. As it was said in Section 3.3, the weights of the second layer for each output neuron are independent of the others, hence their computation can be split in m (number of outputs) sub-problems that can be solved in different processors at the same time.

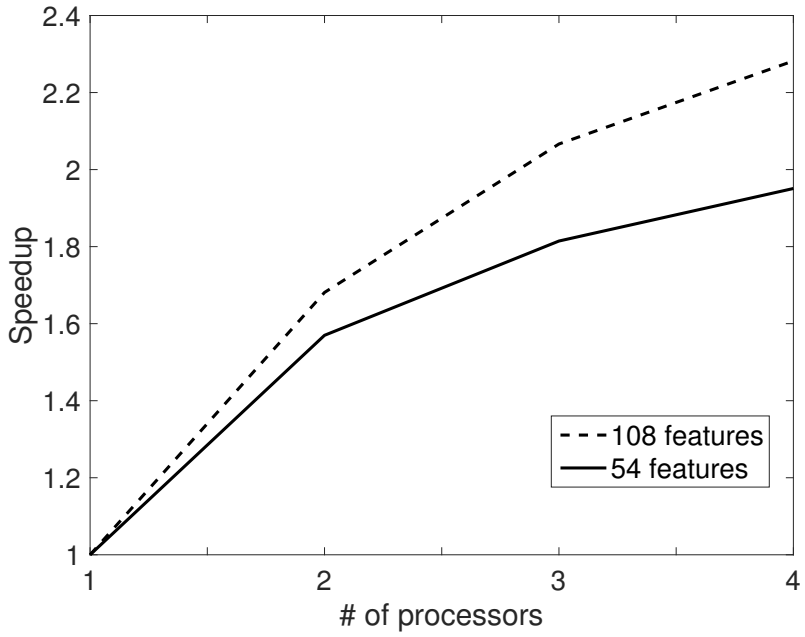


Figure 3.5: Speedup factor for serial SVD-autoencoder vs parallel SVD-Autoencoder.

The Covertype dataset was used in this section (see Table 3.4). In order to get more representative computational times 500000 samples of class 0 with 54 and 108 features (same 54 features repeated two times) were used to train the model. Besides, a non-linear activation function for the second layer and the maximum number of neurons in the hidden layer (53 and 107) were also selected. The experiments were carried out on the same Intel Core i7-4790 with 4 cores used previously. Thus, training times with 1, 2, 3 and 4 processors working in parallel were measured. Figure 3.5 shows the speedup factor. As can be seen, when the number of features increases the parallelizable nature of the algorithm is better exploited. Due to parallel overhead and the time required to transfer data from the master processor to the workers and

back, the speedup factor is smaller than the ideal.

3.5 Summary

New machine learning techniques and tools are continuously appearing in order to be able to deal with the increasing amount of available data. In this chapter, we have presented SVD-autoencoder, a new fast learning neural network for one-class classification that allows handling large-scale datasets. This method proposes the use of a previously presented cost function based on the mean-squared error that measures the error before the output neural function. The formula used to achieve the optimal weights is derived from a system of linear equations that is further transformed by means of the Singular Value Decomposition (SVD). Furthermore, the proposed method allows dimensionality reduction in a very efficient way. In the first layer, the method learns a vector space embedding for the input data extracting meaningful low-dimensional representation by means, again, of Singular Value Decomposition. This reduction is controlled by the number of hidden neurons selected h that has to be strictly lesser than the number of inputs. The main advantages of this proposed algorithm are that it can be used with nonlinear activation functions and that it provides savings in computational requirements for large datasets, especially in the usual case in which the number of hidden neurons is much smaller than the number of data samples. Besides, the proposed algorithm did not run into any convergence issues thanks to its non-iterative optimization.

Another interesting characteristic of the SVD-autoencoder is its parallelizable nature. The minimization problem to obtain the optimal weights of the second layer can be split in several sub-problems (one for each output neuron) that can be solved independently in a parallel manner. Thus, this characteristic makes the SVD-autoencoder highly scalable and efficient also if the number of features is high.

Several experiments have been performed over the domain of one-class classification and it has been demonstrated that this is a fast non-iterative method and, in comparison to some other state of the art methods, this gain in efficacy does not have as a consequence an accuracy loss.

CHAPTER 4

One-class Convex Hull-Based Algorithm for Classification in Distributed Environments

This chapter, in line with the previous one, is dedicated to the development of a new scalable one-class classification algorithm, but in this proposal distributed scenarios will be the use case to confront. Distributed Learning is interesting not only as a way of coping with big data by distributing them among several nodes, but also on itself when data is already distributed in origin.

As a reminder of what has been already stated in the previous chapter, a particular problem in pattern recognition is the task of classifying new data that may differ in some respect from the data that is available during training. This is an unsupervised classification task, known as one-class classification, for which only information of one class (target class) is available for learning. This means that the classifier does not use any assumption on the outlier data to estimate the decision boundary. One-class classification is also called novelty (or outlier) detection and concept learning [66]. These methods are typically applied to datasets in which a very large number of examples of the target class (also known as normal class and positive examples) is available and where there are insufficient data to describe the outlier class (also known as negative examples). The scarcity of negative samples can be due to high measurement costs, or the low frequency at which outliers occur. For example, in a machine monitoring system [101], measurements of the machine during its normal operational state are easy to obtain. However, measurements of failure are very expensive to collect as they would require a crash in the machine [87]. Besides, one-class classification was proven to be an important tool for fraud and intrusion detection [44, 75], text/document classification [82], disease detection [43], etc.

One-class classification techniques are grouped according to three general categories: density estimation, reconstruction based, and boundary based techniques [66, 126]. The first approach uses probabilistic methods that involve a density estimation of the target class, like mixture models and kernel density estimators [37]. Reconstruction based methods involve training a re-

gression model using the target class. They can autonomously model the underlying data, and when test data are presented to the system, the reconstruction error, defined to be the distance between the test vector and the output of the system, can be related to the novelty score. The SVD-autoencoder presented in the previous chapter is an example of this kind. Finally, boundary methods try to model the boundary of the target class without focusing on the description of the underlying distribution. Some well known methods that follow this latter approach are nearest neighbour-based methods, cluster approaches, one-class support vector machines and convex hull approaches [101]. A recent experimental evaluation of one-class classification methods can be found in the work of Ding et al. [32]. Besides, recent reviews about one-class classification (novelty detection) and its applications can be found in Khan and Madden [65, 66], Chandola et al. [28] and Pimentel et al. [101].

Nowadays, the advances in the ICT (Information and Communications Technology) field have contributed to the proliferation of big databases, usually distributed in several machines and in different locations. Performing predictive modeling, such as one-class classification, in this big data scenario is a difficult task. The majority of current one-class classification algorithms are unable to handle this new situation. The classical approximation is based on gathering the several partitions of data into one location to build up a monolithic set of data on which to apply the algorithm. But, in many occasions, data from multiple locations cannot be aggregated or exchanged due to the significant communication, the high cost of storing all data in a unique place and privacy reasons, so the classification task can only be done in local data sets. Therefore, it is important to ensure that the algorithms which worked well on classical scenarios can scale out over distributed architectures [50]. It is common in fields like multiagent algorithms, control systems and automatic monitoring systems to deal with distributed architectures [80, 96, 122], however only a few recent works propose solutions to specific one-class classification problems in distributed environments.

In order to face this issue, a new field of research has arisen that takes care of learning in distributed environments. Zhou et al. [150] presented a framework for detecting anomalous behavior from terabytes of flight record data from distributed data sources that cannot be directly merged. In Branch et al. [17] the problem of unsupervised outlier detection in wireless sensor networks is addressed. They developed a flexible outlier detection method that computes the result in-network to reduce both bandwidth and energy consumption and uses only single-hop communication, thus permitting very simple node failure detection and message reliability assurance mechanisms. Finally, in a most recent work by Castillo et al. [27] a distributed version of the state of the art ν-SVM algorithm was proposed, where several models are considered, each one determined using a given local data partition on a processor, and the goal is to find a global model.

In this chapter:

1. A new distributed version of a one-class classification algorithm previously presented by Casale et. al [26] is proposed. In it, the geometrical structure of the convex hull (CH) is used to define the class boundary in one-class classification problems. This technique has been widely used in multiclass classification problems [68, 79, 147] and it was first proposed to tackle one-class problems by Casale et al. [25]. The use of conventional implementations of the CH in high dimensions, due to its high computational complexity, is not feasible. New implementations have been proposed to deal with this problem [107, 124], but the computation of high dimensional CHs continues to be an issue. Therefore, the proposed method, called SCH algorithm, approximates the D-dimensional CH decision by means of random projections and an ensemble of CH models in very low dimensions ($d \ll D$) which makes it suitable for larger dimensions in an acceptable execution time. Besides, the SCH algorithm not only expands the capabilities of the original one to manage distributed scenarios, but also proposes other changes that might be used both in monolithic and distributed approaches, allowing for the use of different centers, that avoid the appearance of non-convex situations (a drawback of the original method).

2. An extension of the SCH algorithm to improve its efficiency when dealing with big data problems is presented. The appropriate number of random projections, needed to approximate the decision of the original CH into an ensemble of projected decisions on very low dimensions, is difficult to establish and it must be high to ensure an optimal approximation. Due to that, the ensemble model can contain several redundant or non-relevant projections. With this new approach we try to get rid of the less relevant projections that could lead to bad classification models in the low dimensional space and increase the computational complexity of the algorithm.

3. A proposal for an online learning version of the SCH algorithm is also described in this chapter. Online learning, along with distributed learning, have become trending areas in the last years since they allow to deal with extremely large datasets. Some promising experiments that encourage us to keep working on this approach are also showed.

This chapter is structured as follows: Section 4.1 outlines the main characteristics of the original one-class algorithm in which is based this research, Section 4.2 includes a detailed description of the proposed SCH algorithm and its distributed version, along with the experimental support. Section 4.3 describes the extension for improving the efficiency of the SCH algorithm removing non-relevant projections and shows its performance on several datasets.

The main characteristics of the new online learning approach are outlined in Section 4.4. Finally, Section 4.5 sums up the contents of the chapter.

4.1 Preliminaries: APE algorithm

In this section, for the sake of comprehension, the main characteristics of the original one-class algorithm proposed by Casale et al. [25] are discussed. The major contributions of their work in the field of one-class classification are:

1. The use of the geometric structure of the convex hull (CH) to model the boundary of the target class defining the one-class problem.
2. The use of reduced and enlarged versions of the original convex hull, called *extended convex polytope* (ECP) in order to avoid over-fitting and to find the optimal operating point of the classifier.
3. As the computation of the ECP to decide whether a point belongs to the target class is unfeasible in high dimensional spaces, an approximation of the D-dimensional ECP model was made by using an ensemble of decisions in very low-dimensional spaces $d \ll D$ (usually $d = 1$ or $d = 2$). This was called *approximate convex polytope decision ensemble* (APE).

The convex hull of a finite set of points $S \subseteq \mathbb{R}^D$ is the (unique) minimal convex set containing S, and is defined as the convex combination of points $\mathbf{x}_i$ in S where all the coefficients θ_i are non-negative and add up to one [103]:

$$CH(S) = \left\{ \sum_{i=1}^{|S|} \theta_i \mathbf{x}_i \,\middle|\, (\forall i : \theta_i \geq 0, \mathbf{x}_i \in S) \wedge \sum_{i=1}^{|S|} \theta_i = 1 \right\}.$$

The CH provides a tight approximation among several convex forms to the class region of a set of points $S \subseteq \mathbb{R}^D$. However, this approximation is prone to over-fitting. An outlier in the training set can lead to shapes that do not represent the target class accurately. To avoid this, reduced/enlarged versions of the original CH were used. Vertices of this ECP are defined with respect to the center point $\mathbf{c} = \frac{1}{|S|}\sum_i \mathbf{x}_i, \forall \mathbf{x}_i \in S$ and the expansion parameter $\alpha \in \mathbb{R}$ as in

$$V^{\alpha} : \left\{ \mathbf{v} + \alpha \frac{(\mathbf{v} - \mathbf{c})}{\|\mathbf{v} - \mathbf{c}\|} \,\middle|\, \mathbf{v} \in CH(S) \right\} \tag{4.1}$$

Fig. 4.1 shows a reduced (inner dashed polygon) and enlarged (outer dashed polygon) ECP.

Unfortunately, calculating the ECP in high dimensional spaces is computationally hard. To overcome this limitation, the APE algorithm was proposed. It consists in approximating the

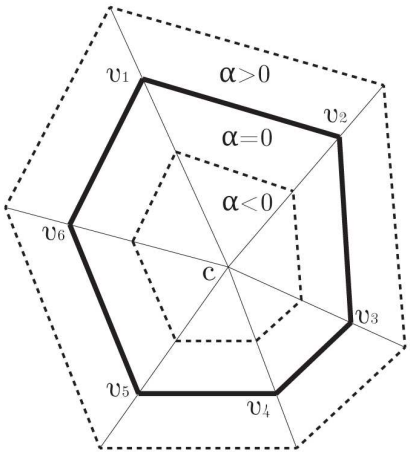


Figure 4.1: Reduced/enlarged extended convex polytope.

decision made by the ECP in the original D-dimensional space by means of a set of $\tau \in \mathbb{N}$ randomly projected decisions made on low-dimensional spaces. Random projections have recently emerged as a powerful method for dimensionality reduction that preserves distances quite well. This data structure preservation has been demonstrated by Johnson and Linderstrauss [58] and it allows to create simple and powerful techniques. In this scenario, the **decision rule** is the following: *a point does not belong to the modeled class if and only if there exists at least one projection in which the point lies outside the projected convex polytope*.

Fig. 4.2 shows a graphical description of the method. In this example a 3-dimensional convex figure (cylinder) is approximated by three random projections in 2 dimensions. As can be seen, a point that lies outside the original cylinder might appear inside one or more projections.

Expansion factor in the low dimensional space

The decision that a point belongs to the target class is made by considering the extended convex polytope in a low-dimensional space. Since the projection matrix is randomly generated, the norm of the original space is not preserved in the resulting one. Thus, a constant value of α in the original space corresponds to a set of values γ_i in the projected one. So, the set of vertices that define the low-dimensional approximation of the expanded polytope are:

$$\overline{V}^{\alpha} : \left\{ \overline{\mathbf{v}_i} + \gamma_i \frac{(\overline{\mathbf{v}_i} - \overline{\mathbf{c}})}{\|\overline{\mathbf{v}_i} - \overline{\mathbf{c}}\|} \right\}, i = 1 \ldots n \tag{4.2}$$

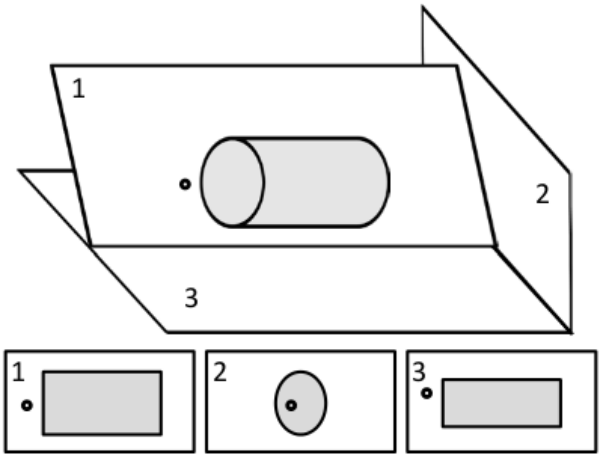


Figure 4.2: APE strategy on 2D.

where $\overline{\mathbf{c}} = \mathbf{Pc}$ represents the projection of the center c given a random projection matrix $\mathbf{P}$, $\overline{\mathbf{v}_i}$ is the ith convex hull vertex of the projected data and γ_i is defined as

$$\gamma_i = \frac{(\mathbf{v}_i - \mathbf{c})^T \mathbf{P}^T \mathbf{P} (\mathbf{v}_i - \mathbf{c})}{\|\mathbf{v}_i - \mathbf{c}\|} \alpha \tag{4.3}$$

where $\mathbf{v}_i$ is the ith vertex of the CH in the original space.

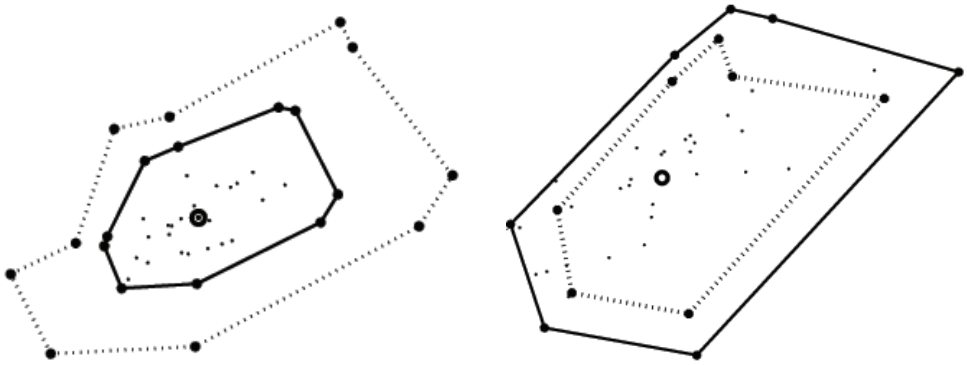
4.2 Proposed method

In this research a new version of the one-class APE algorithm, called Scaled Convex Hull (SCH) algorithm, is proposed. Besides, an extension that expands the capabilities of the SCH algorithm to manage distributed scenarios is also presented.

4.2.1 Scaled Convex Hull (SCH) algorithm

In this work some modifications related to the way the extended convex polytope (ECP) is obtained in the previous APE algorithm (see Section 4.1) are proposed. The main purpose is to avoid an awkward behavior detected when the expanded/reduced version of the polytope is obtained and provide, at the same time, a more intuitive expansion parameter and a more flexible method.

As can be seen in Section 4.1, the expansion factor used in the APE algorithm takes into account the distances between the vertices and the center of the CH in the original space to calculate the ECP. Due to this, each vertex $\overline{\mathbf{v}_i}$ in the projected space has its own expansion parameter γ_i. These expansion factors can lead to obtain non-convex polytopes, which is a non desirable behavior of the algorithm, that aims at checking whether a point lies inside of a convex polytope. Figures 4.3(a) and 4.3(b) depict two examples of this awkward behavior. Data of class 1 (iris setosa) of the Iris dataset [76] and two different random projections into 2 dimensions were used to get these results.



(a) $\alpha = 0.4$ (b) α = -0.1

Figure 4.3: Examples of non-convex polytopes as a result of the (a) expansion or (b) reduction of the projected CH. Dashed polygons represent the expanded/reduced polytope for each example.

In this work, to calculate the expanded polytope in the low dimensional space, the formula presented by Liu et al. [79] is proposed. Vertices of this scaled convex hull are defined with respect to the center point $\mathbf{c} = \frac{1}{|S|}\sum_i \mathbf{x}_i, \forall \mathbf{x}_i \in S$ and the expansion parameter $\lambda \in [0, +\infty)$ as in

$$V^{\lambda} : \{\lambda \mathbf{v} + (1-\lambda)\mathbf{c} | \, \mathbf{v} \in CH(S)\} \tag{4.4}$$

The parameter λ specifies a constant contraction ($0 \leq \lambda < 1$) or extension ($\lambda > 1$) of the convex hull with respect to $\mathbf{c}$.

As in the original algorithm, the decision whether a point belongs to the target class is made by considering the scaled convex hull projected in a low-dimensional space. Thus, the set of vertices that define the low-dimensional approximation of the scaled polytope are:

$$\overline{V}^{\lambda} : \{\lambda \overline{\mathbf{v}_i} + (1-\lambda)\overline{\mathbf{c}}\}, i = 1 \ldots n \tag{4.5}$$

where $\overline{\mathbf{c}} = \mathbf{Pc}$ represents the projection of the center $\mathbf{c}$ given a random projection matrix $\mathbf{P}$, $\overline{\mathbf{v}_i}$ is the ith vertex of the convex hull obtained with the projected data and λ is the expansion factor.

The use of Equation 4.5 to calculate the expanded convex hull brings some benefits in relation with the one proposed in Casale et al. [26]. Remember that they proposed an expansion factor that takes into account the distances between the vertices and the center of the CH in the original space to calculate the expanded polytope. However, as we have showed, this expansion factor can lead to obtain non-convex polytopes, which is a non desirable behavior of the algorithm. The formula described in Equation 4.5 is more intuitive as it only involves information in the projected space and due to that fact all the vertices of the projected CH are expanded by the same factor λ, avoiding the previous problem. Note that when $\lambda = 0$ all the CH vertices are equal to $\overline{\mathbf{c}}$ and when $\lambda = 1$ no expansion/reduction is obtained.

Furthermore, using this alternative, it is possible to employ different definitions of CH "center". In the original method, the average of all points in the D-dimensional space was the unique possible choice as a center (see Section 4.1). However, in the SCH algorithm three different definitions of center are proposed:

- Average of all the training points in the projected space:

$$\overline{\mathbf{c}} = \frac{1}{\left|\overline{S}\right|} \sum_i \overline{\mathbf{x}_i}, \forall \overline{\mathbf{x}_i} \in \overline{S} \tag{4.6}$$

  where $\overline{S} \subseteq \mathbb{R}^d$ represents the projection into a low-dimensional space of the original set of points $S \subseteq \mathbb{R}^D$.

- Average of the convex hull vertices in the projected space:

$$\overline{\mathbf{c}} = \frac{1}{\left|CH(\overline{S})\right|} \sum_i \overline{\mathbf{v}_i}, \forall \overline{\mathbf{v}_i} \in CH(\overline{S}) \tag{4.7}$$

- Average position of all the points in the projected polytope (Centroid) [4].

As can be seen in Figure 4.4, each type of center leads to different decision regions ($\lambda = 0.8$ has been used in this example), giving more flexibility to this method. For instance, when the majority of the training samples fall in the same region and some other are underrepresented, but important ones, fall in other areas of the space, the scaled convex hull obtained using the average of all the training points as a center will be biased towards the preponderant samples. In such cases, the selection of one of the other types of center proposed to scale the CH could improve the final decision region.

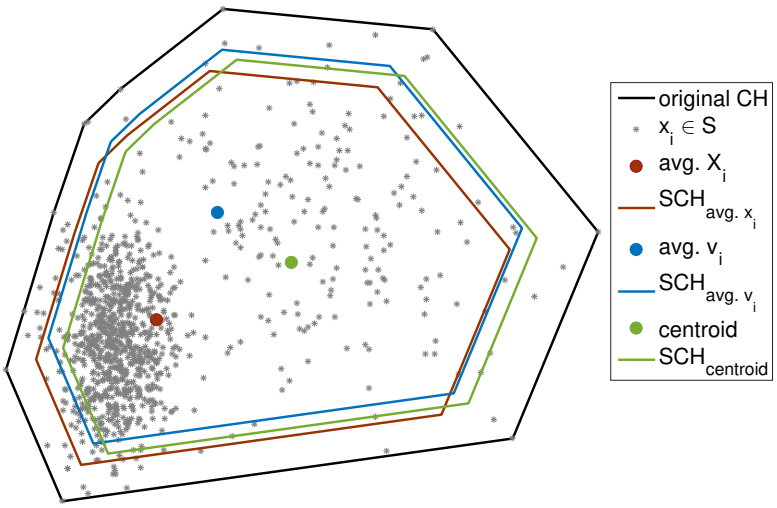


Figure 4.4: Decision regions for each type of center.

Learning and testing algorithms

Learning and testing procedures of the SCH method are described in Algorithms 3 and 4, respectively. In the learning algorithm, the number of projections τ, the expansion parameter λ, and the type of center used to calculate the extended polytope have to be defined. An ensemble model E containing τ projection matrices and their respective expanded CH vertices is obtained at the end of the training procedure. In the original method [26] vertices of the ECP are obtained in the testing algorithm. However, changing this operation from the testing to the learning algorithm, as we propose, reduces computational time as calculations are done only once.

Algorithm 4 takes as inputs: the model E and a test point $\mathbf{x} \in \mathbb{R}^D$. At each iteration t, the test point is projected into the low dimensional space spanned by the t-th projection matrix. Then, given the set of expanded vertices, it is possible to check whether the point lies inside the projected polytope or not. Remember that if the point lies outside one of the projected convex polytopes then it does not belong to the model and no more checking is needed.

Algorithm 3 : SCH learning algorithm

Inputs: $S \subset \mathbb{R}^D$ ▷ Training set
τ ▷ Number of projections
$\lambda \in [0, +\infty)$ ▷ Expansion parameter
tc ▷ Type of center
Outputs: E ▷ Ensemble model

1: $E = \phi$; ▷ Initialize the Ensemble; empty at the beginning
2: **for** t = 1 to τ
3: $\mathbf{P}_t \sim N(0,1)$ ▷ Create a random projection matrix $[d \times D]$
4: $\overline{S_t} : \{\mathbf{P}_t\mathbf{x} | \mathbf{x} \in S\}$ ▷ Project original data
5: $V_t = \mathrm{CH}(\overline{S_t})$ ▷ Return the vertices of the CH
6: $\overline{\mathbf{c}} = getCenter(tc, \overline{S_t})$ ▷ Return the selected center in the d dimensional space
7: $V_t^\lambda : \{\lambda \mathbf{v}_i + (1-\lambda)\overline{\mathbf{c}} | \, \mathbf{v}_i \in V_t\}$ ▷ ECP in the d dimensional space
8: $E = E \cup (\mathbf{P}_t, V_t^\lambda)$ ▷ Store the vertices of the expanded CH and the projection matrix
9: **end for**

Algorithm 4 : SCH testing algorithm

Inputs: $\mathbf{x} \in \mathbb{R}^D$ ▷ Test point
E ▷ Trained ensemble model
Outputs: $Result \in \{INSIDE, OUTSIDE\}$

1: $Result = INSIDE$
2: **for** t = 1 to τ
3: $\overline{\mathbf{x}_t} : \{\mathbf{P}_t\mathbf{x}\}$ ▷ Project test point
4: **if** ($\overline{\mathbf{x}_t} \notin \mathrm{CH}(V_t^\lambda)$) **then**
5: $Result = OUTSIDE$
6: Break
7: **end if**
8: **end for**

4.2.2 Distributed Scaled Convex Hull (DSCH) algorithm

In this section a novel distributed one-class classification approach based on an extension of the previous SCH algorithm is presented. Nowadays, due the proliferation of large distributed databases, it is important to ensure that the algorithms which worked well on classical small scenarios can scale out over distributed architectures.

In order to do so, the new distributed algorithm proposed has to fulfill some requirements:

1. Avoid the cost of storing all the distributed data in a unique node;
2. Prevent the high computational cost of training in a unique node;
3. Reduce the cost of dispatching a big amount of data between different nodes;
4. Avoid the exchange of sensitive or private data through the net.

Thus the general architecture of the distributed algorithm proposed consist of:

- A fully connected network of N **independent nodes**. As the idea is to deal with distributed databases, these nodes are supposed to be computers (with their own databases) located in different parts of the world. They are independent because they do not need to know anything about the data managed in the others;
- 1 **head node**. It is the responsible of communicating with all the other nodes to start the training process, sharing the information needed. It can be any of the N nodes;
- 1 **decision node**. It is the responsible of communicating with all the other nodes to share the information needed to test a new data point and to take its final classification decision. In this case, as it will be seen, any of the N nodes might be the decision node at some point.

In order to fulfill the previous requirements, i.e. reducing the exchange of data between nodes to a minimum and avoiding the dispatch of sensitive or private data through the net, communications in this approach are reduced to:

- The set of random projections matrices $\{\mathbf{P}_t\}$, $t = 1 \dots \tau$ has to be sent to each training node in order to create homogeneous models. Figure 4.5 shows this step. *Node N* was

selected as the head node in this example. Thus, the set of τ random projections matrices P is created in this node and delivered to the other nodes.

- The projections of each new test point are sent to each testing node. After the training phase, if a new point $\mathbf{x}$ appears in a *Node i* ($i = 1 \ldots N$), it needs to be classified. *Node i* will be the decision node during the testing process of point $\mathbf{x}$. Figure 4.6 depicts this step. *Node N* was the decision node in this example, but the graph will be exactly the same changing *Node N* for any other. It can be seen that instead of sending the original point $\mathbf{x}$ directly to the testing nodes, the decision node sends it encrypted by means of the low dimensional projections $\overline{\mathbf{x}_t} = \mathbf{P}_t\mathbf{x}, t = 1 \ldots \tau$, avoiding the exchange of private data through the net.

- The local decision about point $\mathbf{x}$ obtained in each node needs to be delivered to the decision node in order to obtain the final result. Figure 4.7 reflects this step.

Training and testing procedures, as well as the global decision rule are described in detail in the following subsections.

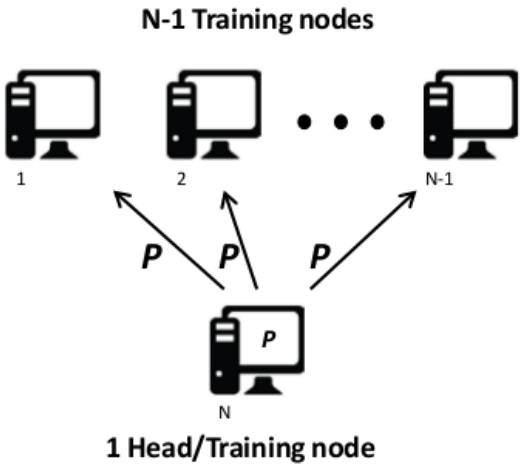


Figure 4.5: The set of τ projections matrices P is sent from the head node to every other training node.

4.2.2.1 Distributed learning procedure

1. The first thing to do is to select the **Head node**. It can be any of the N available nodes. This node is responsible for starting the training process.

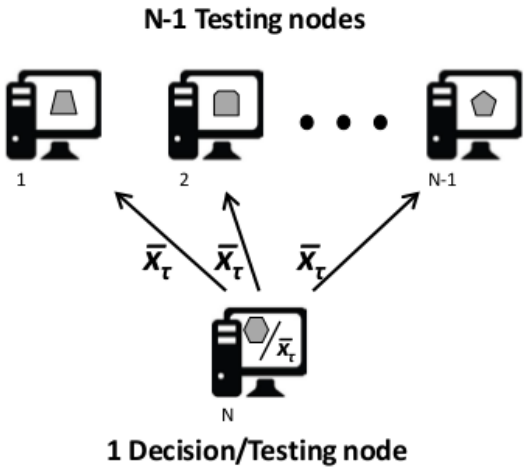


Figure 4.6: Projections of a new test point **x** are sent from the decision node to each testing node.

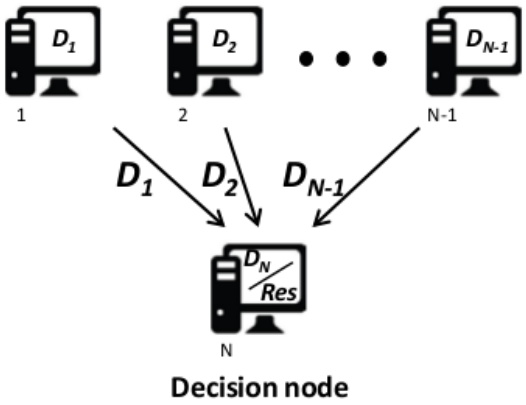


Figure 4.7: Local decisions ($D_1 ... D_N$) from each testing node are gathered into the decision node to obtain the final result (*Res*).

2. The set of τ random projections P is created in the **Head node** and delivered to all the **training nodes**.

3. Each **training node** creates its own local model $E_i, i = 1 \ldots N$, composed of τ projection matrices and their respective expanded convex hull vertices, using its own data. This process is almost the same as the one described in Algorithm 3, but in this case is applied locally in each one of the N distributed nodes. The only difference is that now the set of random projection matrices P is provided as an input and, therefore, line 3 in Algorithm 3 is removed.

After this process, each training node will become a testing node with its own model. Therefore, N different local models are available to classify each new test point.

4.2.2.2 Distributed decision algorithm

As it was said before, if a new test point arrives to *Node i* ($i = 1 \ldots N$) to be classified, then *Node i* will be the decision node during the testing process of that point, and the process is the following:

1. **Decision node**: the new test point is projected into the τ low dimensional spaces (using P) and delivered to all the **testing nodes**. Remember that the set of random projections P was sent to every node in the network at the beginning of the training phase. Therefore, each node has a local copy of P.

2. **Testing nodes**: each projection of the test point is checked in the appropriate expanded convex hull (SCH), following basically the same procedure already showed in the original SCH algorithm (see Algorithm 4). But now the τ projections of the test point are provided as an input, instead of the original test point, therefore, line 3 in Algorithm 4 is deleted.

3. **Testing nodes**: **local decision** D_i (i-th node) is sent to the **decision node**.

4. **Decision node:** the decisions obtained in each **testing node** are combined to obtain the final classification using a **global decision rule**, described in the next section.

4.2.2.3 Global decision rule

This distributed decision algorithm, as the original one, has to decide whether or not a new point belongs to the target class. But in this case, this target class is represented by means of N different local models. So to make a final decision the results obtained in each individual classifier need to be combined in a specific way. Such interpretation allows forming an ensemble through algebraic combination rules [102], like majority voting or sum/product. In this work two different decision rules are proposed:

- **Sum (OR)**. The final decision is the sum of the local decisions D_i (i-th node). This means that if the test point is classified as normal (belonging to the target class) in at least one of the testing nodes, then it will be classified as normal in the final decision.

$$Result = (D_1 \; OR \; D_2 \; ... \; OR \; D_n) \tag{4.8}$$

- **Majority voting (MV)**. The final decision is obtained by majority vote of the local decisions D_i (i-th node). For any given test point, the decision chosen by most classifiers is the final decision. An extensive analysis of the majority voting approach can be found in [73].

The performance of this method and the effects of these two different decision rules are showed in the experimental section. Furthermore, a comparative against the distributed ν-SVM [26] is presented. This method has some similarities with our proposal but a different philosophy. The main difference is that the method proposed is privacy preserving and the distributed ν-SVM is not (because some data points are shared through the net).

4.2.3 Experimental study

This section will present the results of the experiments to assess the performance of the one-class classification algorithm (SCH) and its distributed extension (DSCH) presented in this chapter. In the first section a study on the performance of the proposed one-class approach when compared with other state of the art methods is presented. The second section describes the performance of the distributed approach with very large datasets and compares it with the results obtained by another distributed one-class method.

4.2.3.1 Comparative study of performance

In order to assess the adequacy of the performance obtained by the proposed SCH algorithm, a comparison against two one-class pattern recognition algorithms is made. The first one is the APE (Approximate Polytope Ensemble) algorithm proposed by Casale et al. [25]. This algorithm was selected because is the base of the SCH algorithm proposed and it showed a very good performance when compared against several standard classification algorithms. The second one is the state-of-the-art ν-SVM algorithm [114]. These methods were evaluated over the same 28 one-class problems derived from 11 UCI machine learning repository datasets [76] used in the previous chapter (see Table 4.1). For each dataset, one-class problems were obtained considering one of the classes as target and the rest as outliers. Each problem was evaluated using 10-fold stratified cross-validation on 10 different permutations of the data. This experimental framework is employed as it is the same that has been used in a previous work [25] to show the superiority of the original APE algorithm against several state-of-the-art one-class pattern recognition methods. All datasets have been previously normalized between [0, 1]. As it was shown in Casale et al. [25] that using 2D projections provides better results than using 1D projections, in this work we projected data to 2D spaces, and arbitrarily set the number of projections used to 100. Furthermore, in the case of SCH, three different SCH approaches were tested, varying the type of center employed (see Section 4.1). All the experiments were carried out on an Intel Core i7-4790 processor with 3.60GHz clock speed and 15.9Gb RAM.

Dataset	No. features	# instances	# classes
Balance	4	625	3
Breast	10	683	2
Car	6	1728	4
Glass	10	214	3
Haberman	3	306	2
Ionosphere	34	351	2
Iris	4	150	3
Pima	8	768	2
Sonar	60	208	2
TicTacToe	9	958	2
Wine	13	178	3

Table 4.1: Characteristics of the UCI datasets.

Results obtained are reported in Table 4.2. In the second column of Table 4.2, the class considered as target and the number of samples of the target and outlier classes are displayed in brackets, respectively. Remaining columns of the table show the % of the mean Area Under the Roc Curve (AUC) and the standard deviation (SD) obtained. In order to compute the AUC,

each curve was evaluated on 100 points varying the α parameter in APE-2, the λ parameter in SCH approaches and the parameter σ that controls the width of the distribution in ν-SVM .

Dataset	Class	$APE-2$	$SCH_{avg.x_i}$	$SCH_{avg.v_i}$	$SCH_{centroid}$	ν-SVM
Balance	1 (288,337)	90.84±3.4	90.75±3.4	**90.96±3.1**†	90.85±3.0	87.12±3.9
	2 (49,576)	84.62±10.2	**87.49±6.0***†	87.29±6.0	87.10±6.0	71.95±12.6
	3 (288,337)	90.90±3.0	90.91±3.3	**91.00±3.1**†	90.87±3.1	87.17±3.8
Breast	1 (444,239)	95.14±1.9	95.21±1.7	95.19±1.8	**95.26±1.7**†	94.11±2.7
	2 (239,444)	84.53±4.9	85.66±4.4*	85.30±4.3	85.61±4.3	**92.88±3.8**†
Car	1 (1210,518)	71.83±1.7*	71.15±1.8	71.27±1.7	71.29±1.7	**79.03±2.1**†
	2 (384,1344)	95.08±1.3	**95.69±1.2***†	95.51±1.3	95.51±1.2	87.62±2.8
	3 (69,1659)	98.48±1.5	98.66±1.8	**99.16±0.6***†	98.79±1.5	86.61±6.8
	4 (65,1663)	99.20±1.5	**99.74±0.3***†	99.41±1.3	99.55±1.0	91.48±7.6
Glass	1 (70,144)	95.36±4.8	96.79±3.7	**97.00±3.6***†	96.73±3.9	92.22±4.7
	2 (76,138)	93.06±6.1	93.32±6.0	93.04±6.3	**93.33±5.6**†	87.68±7.4
	3 (68,146)	87.58±8.0	88.33±8.0	88.26±8.1	88.44±7.9*	**88.87±6.5**
Haberman	1 (225,81)	50.51±3.1	51.11±1.3	51.17±1.3*	51.13±1.3	**56.37±4.5**†
	2 (81,225)	57.80±7.8	**60.06±8.2***†	59.73±9.1	59.96±8.4	54.99±7.9
Ionosphere	1 (225,126)	90.09±3.0	90.06±3.2	90.19±3.2	90.17±3.4	**91.15±4.3**†
	2 (126,225)	50.00±0.0	50.00±0.0	50.00±0.0	50.00±0.0	**51.99±2.8**†
Iris	1 (50,100)	100±0.0	100±0.0	100±0.0	100±0.0	100±0.0
	2 (50,100)	91.31±5.6	93.02±2.9	93.07±2.6	**93.10±2.5***†	84.50±8.8
	3 (50,100)	91.64±6.8	92.03±6.5	92.23±6.9	**92.35±6.6***†	89.06±8.7
Pima	1 (500,268)	62.03±3.2	62.12±2.9	62.24±3.0	62.17±2.9	**65.24±3.3**†
	2 (268,500)	56.87±5.0	**57.20±5.1**	57.11±4.8	57.06±4.9	57.11±5.0
Sonar	1 (97,111)	60.43±7.4	**61.59±7.6***†	61.57±8.0	60.99±7.7	59.14±8.3
	2 (111,97)	66.53±7.0	**67.96±7.6***†	67.18±7.2	67.26±7.3	66.29±7.1
TicTacToe	1 (626,332)	62.63±2.3	62.64±2.0	62.50±2.1	**62.80±2.0**†	54.94±2.6
	2 (332,626)	63.07±4.3	63.09±4.4	63.32±4.5	63.48±4.2	**83.57±3.6**†
Wine	1 (59,119)	95.36±4.8	96.26±5.1	**96.68±4.6***†	96.48±4.7	92.94±7.5
	2 (71,107)	79.61±8.1	81.47±7.7	81.18±7.7	**81.49±8.1***†	78.66±7.6
	3 (48,130)	93.89±6.7	96.17±3.0	**96.80±2.0***†	96.45±2.6	92.87±9.0

Table 4.2: AUC and SD. In bold font, the best result for each problem. Methods that are statistically different are marked with an * **(SCH vs APE-2)** or with a † **(SCH vs ν-SVM).**

Two pairwise t-test [31] were applied to evaluate the statistical difference at 95% significance level. In the first one, the best result of the SCH approach varying the type of center (i.e. avg. of every training sample x_i, avg. of every CH vertex v_i and centroid of the polytope) and the result obtained by the APE were compared and those methods statistically different were marked with an * in Table 4.2. In the second one, the best result of the SCH was again compared against the result obtained by the ν-SVM and the results statistically different were

marked with a †. SCH versions, varying the "center" used to compute the extended CH, generally achieve better significant results than APE-2. Sixteen times against one, SCH is statistically better than APE-2. As can be seen in the results table, none of the SCH approaches is consistently better than the others. So, depending on the problem at hand (the nature of the data), different "centers" may be used, obtaining dissimilar decision regions that can better fit the data. The last column of Table 4.2 shows the results achieved by the ν-SVM in the same problems. It can be seen that SCH generally get better significant results, eighteen times against seven for the kernel method.

4.2.3.2 Performance of the DSCH with very large datasets

In order to asses the performance of the proposed distributed algorithm the same three large datasets seen in the previous chapter were employed. Table 3.4 presents their main characteristics. The first one is a reduced version of the KDD Cup 99 dataset [1]. In this study, the dataset has over 800000 samples divided in two subsets, one for training and the other for testing. This partition of the data was also employed in the original KDD Cup competition. Besides, only the 6 features that were found to be the most representative in the work of Bolon-Canedo et al. [14] were used. The second one is the MiniBooNE dataset [76], intended to distinguish electron neutrinos (signal) from muon neutrinos (background). It has 130064 samples, each one of then composed of 50 features. The last one is the Forest Covertype dataset [76], used to classify forest cover type from cartographic variables. It consist of 54 features and 581012 input patterns. The dataset was reduced from seven to two classes by considering all the five pine species as one class and the other two species as the other. For the three datasets, one class problems were obtained considering the class with the higher number of samples as target and the other one as outlier, that is the typical situation in a real one-class scenario. These datasets were selected such as to be able to obtain results for a monolithic approach, so as to compare performance in accuracy and time.

Dataset	No. features	# instances	# classes
KDD Training set	6	494021	2 (97278, 396743)
KDD Test set	6	311029	2 (60593, 250436)
MiniBooNE	50	130064	2 (93565, 36499)
Covertype	54	581012	2 (568772, 12240)

Table 4.3: Characteristics of the datasets.

Five different configurations of 1, 2, 5, 10, and 20 nodes were considered to test the proposed distributed algorithm, where every node receives a disjoint random subset of the training

data with an equal amount of samples (total number of training samples divided by the number of nodes). In the case of just one computational node (monolithic approach), the DSCH learning and testing algorithms are the same as the original SCH algorithms (see Section 4.2.1). Therefore, a comparative study in terms of performance (AUC) and training time between the SCH algorithm and its distributed version (DSCH) was made. Each problem was evaluated 30 times, thus varying the training data and the way it is distributed between the nodes. Besides, an experiment to calculate the optimal number of projections for each dataset was made. Finally, the proposed DSCH algorithm has been compared in terms of performance and training times against the distributed ν-SVM (Dν-SVM) presented by Castillo et al. [27]. This algorithm has been selected because, to our knowledge, it was the unique one-class algorithm specifically adapted to work with distributed data available in the literature. Besides, it is a distributed version of the well-known ν-SVM used in the previous section. This method employs the idea of projecting the training samples to a higher dimensional feature space and then separates most of the samples from the origin using a maximum margin hyperplane. In the distributed approach several bands are considered, each one obtained using a given local data partition on a processor, and the goal is to find a global classifier. In this respect it is similar to our proposal but the philosophy is different. In the Dν-SVM there is a master node that controls the training phase from the beginning to the end (until convergence is attained), allowing the exchange of data points between nodes in order to have similar data points in the same node [27]. Therefore, in this method the exchange of sensitive data through the net is not avoided and the initial distribution of the data is keystone to get rapid convergence. That does not occur with the method proposed in this work, where the training phase is independent in each node.

Table 4.4 shows the parameters employed and the results obtained in the experiments with the KDD Cup 99 dataset. To learn the model 300000 random samples of the target class extracted from the KDD Training set (see Table 4.3) were used. In a distributed case with n nodes, each node will use 300000/n random samples for training. To test the model, all the samples of the KDD Test set (see Table 4.3) were utilized. As can be seen, the number of test samples per node is the same independently of the number of nodes employed. That is because each new test point needs to be tested in each node's classifier, remember that the decision node sends the projections of each new test point to every training/testing node (see Section 4.2.2.2). Table 4.4 also shows the true positives, true negatives and AUC obtained for each architecture and for the two decision rules proposed, sum-OR and Majority Voting-MV (see Section 4.2.2.3). As can be seen, the performance of the method is not affected by the distribution of the data, the best overall results being obtained with 20 nodes and majority voting (MV) as the decision rule. Regarding the computational time, as each node receives the same amount of training data, time spent to train is virtually reduced by a factor equal to the number of nodes. Here we assume that all the nodes (independent machines or processors)

work in parallel. Therefore, for this dataset the distributed version of the SCH algorithm with 20 nodes achieves the best performance and reduces the training time from almost 1 minute (monolithic approach) to just 3.68 seconds.

	SCH	Distributed SCH			
Training nodes	1	2	5	10	20
Training samples	300000	150000	60000	30000	15000
Testing samples	311029	311029			
Center type	avg. v_i	avg. v_i			
λ	1.01	1.1			
# of projections	3000	3000			
Decision rule	-	OR — MV	OR — MV	OR — MV	OR — MV
True positives (%)	92.54	92.54 — 92.27	92.55 — 90.84	92.54 — 90.69	92.55 — 90.46
True negatives (%)	90.57	90.60 — 93.42	90.60 — 96.65	93.26 — 97.03	94.06 — 97.31
AUC (%)	91.56	91.57 — 93.72	91.57 — 93.75	92.90 — 93.86	93.31 — **93.89**
Training Time	58.32 s	32.54 s	12.51 s	7.09 s	3.68 s

Table 4.4: Parameters used for the DSCH in KDD Cup dataset, and results obtained for 1, 2, 5, 10 and 20 training nodes using two different decision rules: OR (sum) and MV (Majority Voting).

.

Analogously, Table 4.5 presents the parameters used and the results achieved in the experiments made using the MiniBooNE dataset. The training set consist of 60000 random samples of the target class (the one with the higher number of samples) and the rest were used to test. As it is shown in Table 4.5, performance of the method is not affected by the distribution of the data and the best overall results were obtained with 5 nodes and the sum (OR) (see equation 4.8) as the decision rule. Computational time is also reduced by a factor equal to the number of nodes. Thus, for this dataset the distributed version of the SCH algorithm with 5 nodes achieves the best performance and reduces the training time from 27.4 seconds to just 6.45 seconds.

Results and parameters of each Covertype experiment are displayed on Table 4.6. Training set consist of 550000 random samples of the target class and the rest were used to test. In the same way that the previous experiments, performance is not influenced drastically by the distribution of the training data. In this case, the best overall results were obtained with 20 nodes and MV as the decision rule. Training time was reduced from 50.54 seconds (1 node) to 2.12 seconds (20 nodes).

In addition, Figures 4.8, 4.9 and 4.10 show how the number of projections affects the overall performance of the DSCH algorithm. 30 repetitions of the experiment permutating the data were made for each number of projections and the average AUC and the standard

	SCH	Distributed SCH			
Training nodes	1	2	5	10	20
Training samples	60000	30000	12000	6000	3000
Testing samples	70064	70074			
Center type	Centroid	Centroid			
λ	0.74	0.76			
# of projections	5000	5000			
Decision rule	-	OR — MV	OR — MV	OR — MV	OR — MV
True positives (%)	67.31	63.53 — 56.10	58.10 — 50.89	53.54 — 42.47	48.78 — 34.91
True negatives (%)	75.76	80.21 — 86.28	85.83 — 90.87	89.35 — 94.78	92.03 — 96.92
AUC (%)	71.54	71.87 — 71.19	**71.97** — 70.88	71.45 — 68.63	71.41 — 65.97
Training Time	27.40 s	17.77 s	6.45 s	4.12 s	2.67 s

Table 4.5: Parameters used for the DSCH in MiniBooNE dataset, and results obtained for 1, 2, 5, 10 and 20 training nodes using two different decision rules: OR (sum) and MV (Majority Voting).

	SCH	Distributed SCH			
Training nodes	1	2	5	10	20
Training samples	550000	275000	110000	55000	27500
Testing samples	31012	31012			
Center type	avg. v_i	avg. v_i			
λ	0.87	0.87			
# of projections	1000	1000			
Decision rule	-	OR — MV	OR — MV	OR — MV	OR — MV
True positives (%)	88.79	88.69 — 87.25	88.32 — 85.38	87.98 — 83.31	87.80 — 82.80
True negatives (%)	33.30	33.44 — 34.91	34.62 — 37.00	34.41 — 39.16	34.67 — 39.77
AUC (%)	61.05	61.06 — 61.11	61.15 — 61.19	61.21 — 61.26	61.25 — **61.30**
Training Time	50.54 s	25.75 s	9.46 s	4.61 s	2.12 s

Table 4.6: Parameters used for the DSCH in Covertype dataset, and results obtained for 1, 2, 5, 10 and 20 training nodes using two different decision rules: OR (sum) and MV (Majority Voting).

deviation are showed. In Figure 4.8, results for the KDD Cup dataset were obtained with the same parameters showed in Table 4.4 and varying the number of projections. The arrow points out the optimal number of projections for the KDD dataset (τ=3000). Figure 4.9 shows how the number of projections modifies the performance of the DSCH algorithm for the MiniBooNE dataset. The same procedure described for the KDD Cup dataset was used. In this case any value of τ between 4500 and 5000 is a good choice, because it offers a good compromise between performance and computational complexity (the higher the number of projections, the higher the computational complexity). Finally, Figure 4.10 presents the number of projections against the classification performance of the DSCH algorithm for the Covertype dataset. In this case 1000 was chosen as the optimal number of projections for this dataset.

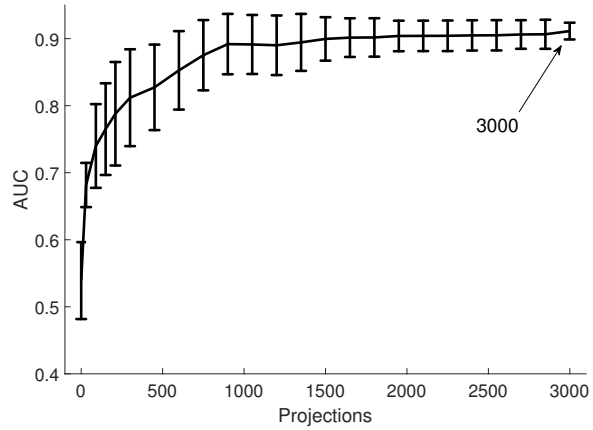


Figure 4.8: AUC vs Projections (KDD Cup dataset).

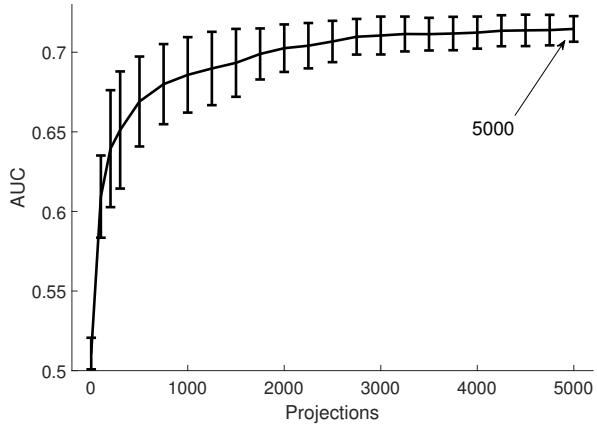


Figure 4.9: AUC vs Projections (MiniBooNE dataset).

Figure 4.11 shows a summary of the results obtained by the DSCH in AUC (%) and training time for the three datasets employed. It can be easily seen that the training time of the

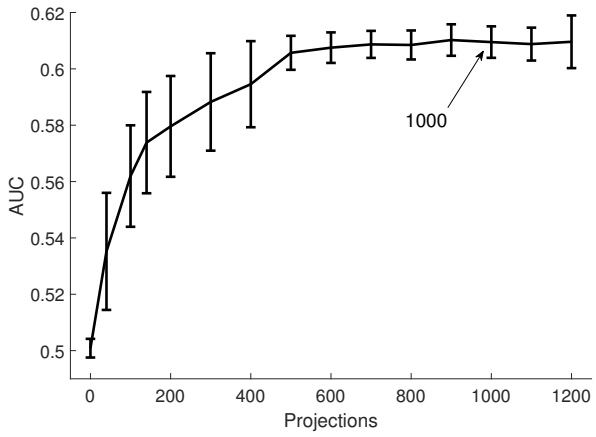


Figure 4.10: AUC vs Projections (Covertype dataset).

monolithic architecture (1 node) is almost divided by the number of nodes used in the experiments. This characteristic is fulfilled for all the datasets used. The graph also shows how the performance of the distributed algorithm maintains or slightly improves (in the case of KDD) the performance of the original SCH algorithm. In KDD and Covertype datasets the AUC showed was achieved with the majority voting (MV) decision rule. However, in Covertype the sum (OR) decision rule provided better results. In any case, as the results in performance are very similar, a reasonable approach could be the use of 20 nodes for all cases, as time reduction is more pronounced.

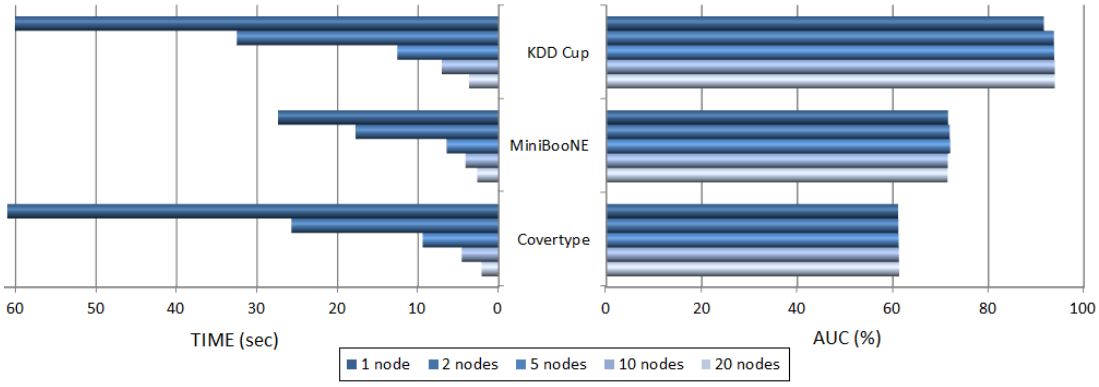


Figure 4.11: Training time vs AUC for the three datasets used in the experiments for DSCH.

Table 4.7 shows the comparative results of the proposed DSCH algorithm against the distributed version of the ν-SVM. In order to be able to compare both methods, the three datasets described in Table 3.4 and the same experimental setup previously used for the DSCH algorithm have been employed to obtain the results of the Dν-SVM. Each cell of the table shows

the % of the mean Area Under the Roc Curve (AUC) and the standard deviation (SD) obtained for both methods (DSCH and Dν-SVM), as long as the training time. In order to compute the AUC, each curve was evaluated on 100 points varying the λ parameter in DSCH and the σ parameter in Dν-SVM. Besides, the ν-SVM parameter that represents the estimation of spurious data in the training set, ν, was set to 0.01. A pairwise t-test [31] between the results obtained by the DSCH algorithm and the Dν-SVM for each dataset and each configuration of nodes was applied to evaluate the statistical difference at 95% significance level. Methods that are statistically different are marked with an asterisk in Table 4.7. As can be seen, the performance of both methods is not affected by the distribution of the data. The method proposed beats the distributed ν-SVM in terms of AUC seven times against one in the fifteen problems presented (seven ties). Regarding the computational time, DSCH algorithm exhibits consistently a better behavior than Dν-SVM. Figure 4.12 shows the average speedup factor, measuring how much the training time is faster using the DSCH approach rather than the Dν-SVM for every problem studied. The differences are specially remarkable in the case of the Covertype dataset, where our proposal is hundred of times faster than the SVM approach. This is because the DSCH algorithm complexity mainly relies on the number of projections (1000 in this example), being the number of samples and features of the training set less important. However, the complexity of the SVM approach is severely affected by the dimensionality of the dataset, both in samples and features [114] (550000 samples and 54 features in this case).

	KDD Cup		MiniBooNE		Covertype	
Nodes	DSCH	Dν-SVM	DSCH	Dν-SVM	DSCH	Dν-SVM
	91.56±0.21*	82.94±0.04	**71.54±0.64***	69.11±0.07	61.05±0.14	**61.91±0.58**
1	58.32 s	957.40 s	27.40 s	889.22 s	50.54 s	22678.10 s
	93.72±0.31*	82.45±0.04	**71.87±0.51***	69.94±0.17	61.11±0.13	**62.04±0.65**
2	32.54 s	362.79 s	17.77 s	367.26 s	25.75 s	10482.90 s
	93.75±0.30*	83.02±0.06	**71.97±0.53**	71.04±0.25	61.18±0.15	**61.24±0.63**
5	12.51 s	81.73 s	6.45 s	132.65 s	9.46 s	3102.35 s
	93.86±0.27*	83.03±0.07	71.45±0.62	**71.95±0.18**	**61.26±0.17**	61.23±0.76
10	7.09 s	31.49 s	4.12 s	63.06 s	4.61 s	1216.10 s
	93.89±0.32*	83.07±0.06	71.41±0.70	**73.01±0.17***	**61.30±0.19**	61.02±0.47
20	3.68 s	14.91 s	2.67 s	39.66 s	2.12 s	476.92 s

Table 4.7: AUC, SD and Training Time. In bold font, the best result for each problem. Methods that are statistically different are marked with an *.

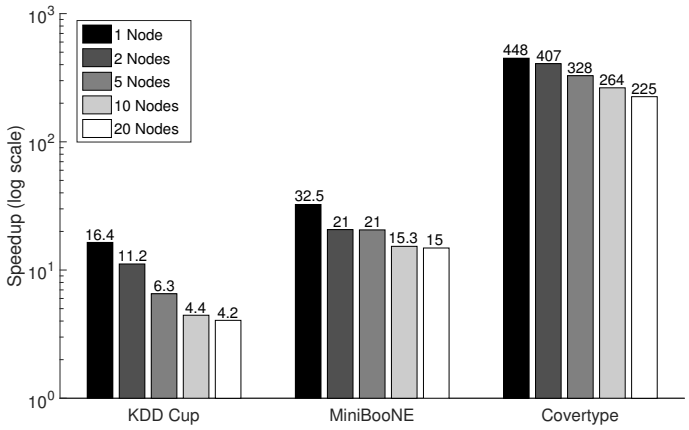


Figure 4.12: Speedup factor for DSCH versus Dν-SVM.

4.3 Mutual Information for improving the efficiency of the SCH algorithm

In this section an extension for improving the efficiency of the SCH algorithm is presented. As it has been seen in the previous experimental section, the number of random projections τ needed to ensure a good performance of the algorithm in the three large datasets is very high. This fact can lead to non-relevant and redundant projected models that increase the computational complexity of the algorithm in the testing phase (see Algorithm 4). Thus, we propose to add a new phase to the SCH algorithm with the aim of reducing the number of random projections used to classify new data removing the less relevant ones.

4.3.1 Projection pruning phase

Right after the SCH training phase (see Algorithm 3) and before starting with the classification of new samples, a new procedure, called *projection pruning phase*, is added to the proposed SCH algorithm. In it, the mutual information criterion is used to evaluate the set of τ projections and sort them in order of relevance. For two random variables X and Y the mutual information [30] is defined as:

$$I(X;Y) = H(X) - H(X|Y) = \sum_{x,y} p(x,y) log \frac{p(x,y)}{p(x)p(y)} \tag{4.9}$$

where $H(X)$ is the entropy of X and $H(X|Y)$ is the conditional entropy of X knowing Y. This measure is symmetric in X and Y, nonnegative and is equal to zero if and only if the variables are independent. The mutual information measures arbitrary dependencies between random variables and it is suitable for estimating their "information content" in complex classification tasks.

Algorithm 5 shows the steps proposed for this phase. Firstly, τ random variables containing information about each one of the projections are needed. After the *training phase*(see Alg. 3) an ensemble E of τ classification models in 2-dimensions is obtained. A set of n non-seen points, called *validation set* $Val \subset \mathbb{R}^D$, is evaluated against each one of the projected models $\mathbf{P}_i$ and the result is the matrix $\mathbf{M}$ (see Figure 4.13(a)). Each cell shows the result of checking whether the point $\mathbf{x}_i \in Val$, $i = 1 \ldots n$ is inside (1) or outside (0) the projected CH created by the projection $\mathbf{P}_i$, $i = 1 \ldots \tau$. Thereafter, we considered each column i of the matrix $\mathbf{M}$ as a random variable containing information about the projection $\mathbf{P}_i$. Thus, the mutual information against each pair of variables can be calculated and the upper triangular matrix $\mathbf{MI}$ is the result.

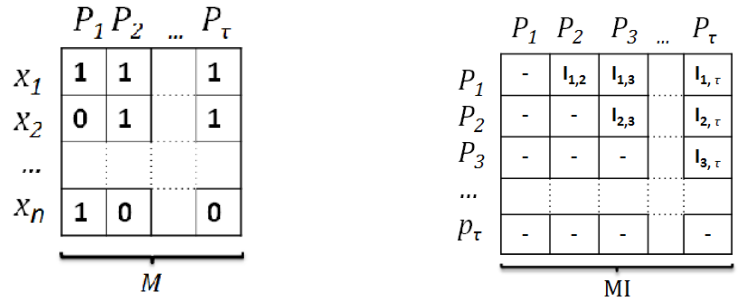


Figure 4.13: (a) Validation results matrix $[n \times \tau]$ (b) Mutual information matrix $[\tau \times \tau]$

Notice that the class of each validation sample is not needed, just the result of the classification. As can be seen in Figure 4.13 (b), the diagonal and the lower triangular part were not calculated because the mutual information values of one projection against itself are not useful here and the lower triangular part is not necessary because of the mutual information symmetry property. Afterwards, the maximum value of mutual information in each row of $\mathbf{MI}$ is stored in a vector $\mathbf{l}_{max}$, where the first value contains the maximum mutual information for $\mathbf{P}_1$, the second one the maximum for $\mathbf{P}_2$ and so on.

Finally, the vector $\mathbf{l}_{max}$ is sorted in ascending order and the index containing the new positions of the projections is stored in $\mathbf{l}$. This index is a rank of the relevant projections, from less to more relevant. Following $\mathbf{l}$, the less relevant projections can be removed from the model making the ensemble model E_τ lightweight and reducing the computational time of Algorithm

Algorithm 5 : Projection pruning phase

Inputs: E ▷ Trained ensemble model (output of Alg. 3)

$Val \subset \mathbb{R}^D$ ▷ Validation set

Outputs: $\mathbf{l}$ ▷ Index of relevant projections

$\mathbf{l}_{max} = \phi$;

$\mathbf{M} = \phi$; ▷ Classification results matrix $[n \times \tau]$

for t = 1 to τ

  $\overline{Val}_t : \{\mathbf{P}_t\mathbf{x} | \mathbf{x} \in Val\}$ ▷ Project original validation data

  for t = 1 to n

    $\mathbf{M}(i,t) = 0$;

    if ($\overline{\mathbf{x}_i} \in \mathrm{CH}(V_t^{\lambda})$) **then**

      $\mathbf{M}(i,t) = 1$;

    end if

  end for

end for

$\mathbf{MI} = \phi$; ▷ Mutual Information matrix $[\tau \times \tau]$

for $i = 1..\tau$

  for $j = i+1..\tau$

    $\mathbf{MI}(i,j) = I(\mathbf{M}(:,i);\mathbf{M}(:,j))$; ▷ Mutual Information between projections i and j

  end for

end for

for $i = 1..\tau$

  $\mathbf{l}_{max}(i) = max(\mathbf{MI}(i,:))$; ▷ Vector containing the maximum value of each row of **MI**

end for

$[\mathbf{l}_{max}, \mathbf{l}] = SORT(\mathbf{l}_{max}, \text{“}ascend\text{”})$; ▷ Sorted list $\mathbf{l}_{max}$ and index $\mathbf{l}$ of the new positions

4. In the next section some experimental results demonstrate the effectiveness and the efficiency of this approach.

4.3.2 Experimental study

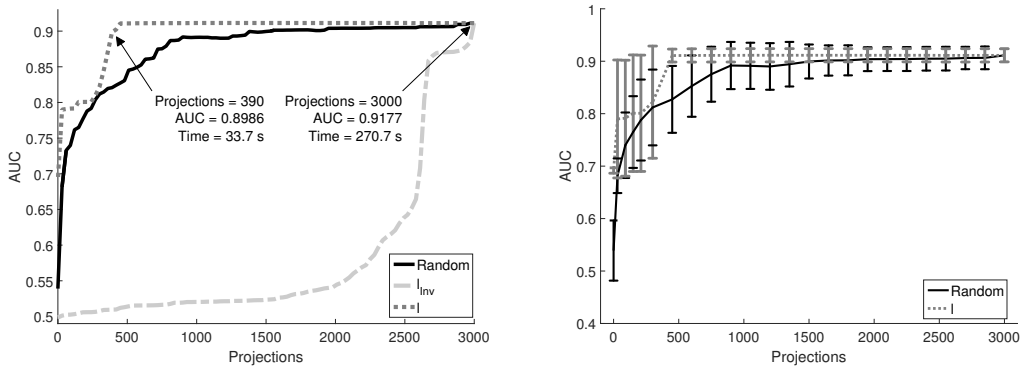
The same three large datasets seen in the previous experimental section (see 4.2.3.2) were employed here in order to assess the savings in computational complexity and storage space obtained with the proposed method. Once again, for the three datasets, the class with the higher number of samples was considered as target and the other one as outlier.

Dataset	Train set	Validation set	Test set	λ	τ	Center type
KDD	250000	2000	309029	1.01	3000	Avg. v_i
MiniBooNE	60000	2000	68064	0.74	5000	Centroid
Forest Covertype	500000	2000	79013	0.88	1000	Avg. v_i

Table 4.8: Number of samples and parameters used for each dataset.

The number of samples and the optimal parameters of the algorithm for each problem are listed in Table 4.8. These parameters were previously calculated using a 10-fold cross validation on 10 different permutations of the data. Each experiment was repeated 30 times varying the training, validation and testing sets. After the training and pruning phases we have an ensemble model E_τ and a ranking $\mathbf{l}$ of the relevant projections. The idea is to remove the less relevant projections from E_τ maintaining the accuracy of the original method. To assess the adequacy of the pruning phase, we have calculated the testing time and the Area Under the ROC Curve (AUC) for all the different ensemble models obtained removing projections in the order exhibited by $\mathbf{l}$; starting with the original model (τ projections) and finishing with a model of just one projection (the last one on $\mathbf{l}$). Besides, we compared these results with the ones obtained removing projections randomly and following the ranking $\mathbf{l}$, but in the opposite way ($\mathbf{l}_{inv}$), from more to less relevant. A pairwise t-test between the result obtained by the original model E_τ and the result obtained by each lightweight model $E_{\tau-i}$ was made to evaluate the statistical difference at 95% significance level.

Figure 4.14(a) displays the results of the KDD dataset. As can be seen, removing projections in the order indicated by $\mathbf{l}$ produces the best results, making it possible to obtain a model with 390 projections with no evidence that it is statistically significantly different than the original. This lightweight model remarkably reduces the time employed for testing, from 270 to 33 seconds. It can also be seen that eliminating projections in the opposite way ($\mathbf{l}_{inv}$) produces the worst result. In Figure 4.14(b) the mean AUC and the standard deviation (SD) for

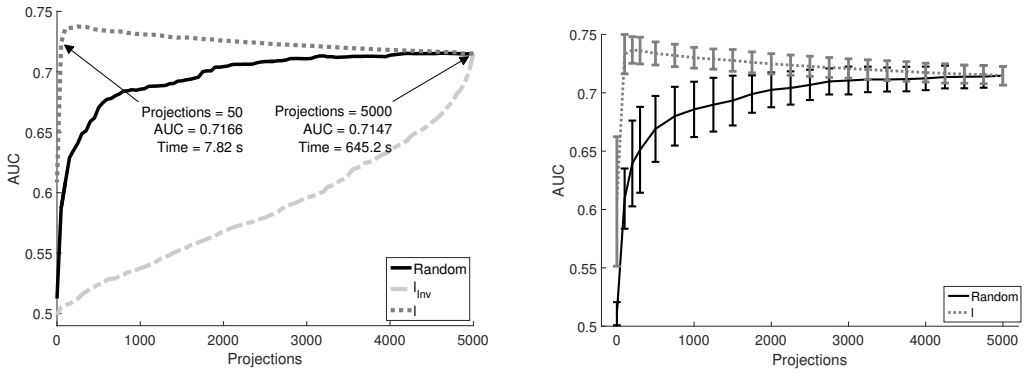


(a) AUC vs Projections. (b) Avg. AUC and SD.

Figure 4.14: KDD results removing projections randomly and using the proposed index **l**.

the lightweight models obtained by removing projections randomly and following the ranking **l** is showed. As can be seen, the results during the 30 repetitions of the experiment are much more stable when we get rid of the projections using **l**.

Figures 4.15(a) and 4.15(b) display the results of the MiniBooNE dataset. Again, removing projections in the order indicated by **l** produces the best results. In this case an even better model than the original with just 50 projections can be obtained, reducing drastically the testing time, from 645 to 7.8 seconds.

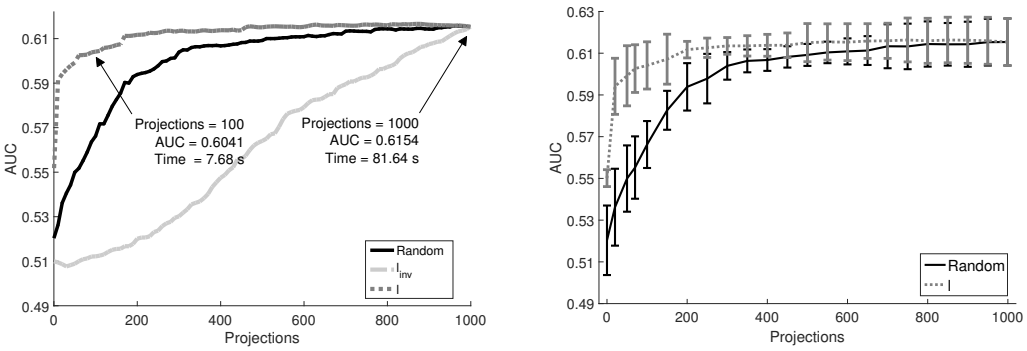


(a) AUC vs Projections. (b) Avg. AUC and SD.

Figure 4.15: MiniBooNE results removing projections randomly and using the proposed index **l**.

Experimental results of the Covertype dataset are showed in Figures 4.15(a) and 4.15(b). In this case, a model with around 100 projections, with no evidence that it is statistically different than the original in terms of performance, can be obtained. Again, especially when the number

of projections is around 200, the results are much better and more stable when we use the proposed method **l**.



(a) AUC vs Projections.

(b) Avg. AUC and SD.

Figure 4.16: Covertype results removing projections randomly and using the proposed index **l**.

4.4 An online SCH algorithm adaptable to changing environments

In this section, a proposal for an online version of the SCH algorithm is presented. Online learning has become a trending area in the last years since it allows to deal with extremely large datasets. In the context of one class classification, an interesting application for online learning methods is stream anomaly detection. The goal of this task is to detect temporal or permanent deviations in a continuous stream of data [86]. This kind of task has been successfully applied in areas such as intrusion detection, medical anomaly detection [28] and machinery fault detection [38, 87, 133]. Its main characteristics are:

- Anomalous events are scarce and do not have a known signature, so this is an unsupervised task.
- Non severe deviations in the data should not generate false alarms.
- The method should be able to adapt to changing decision boundaries in order to accurately capture the normal support of complex streams of data.

Some previous works have treated the problem of stream change detection. Camci et Chinnam [21] presented an adaptation of One-class SVM to deal with non-stationary data. This method obtains good representations but suffers from a computational complexity burden when

facing large scale datasets. Yamanishi et al. [144] proposed a fully probabilistic model for stream anomaly detection that obtains adequate results assuming a predefined probability distribution. Some recent works based on classification trees [123] and clusters [35] have been proposed. These are adaptations of classical bath techniques and need to keep batches of data in order to update the model and detect deviations, which can lead to storage burdens and detection delays. Finally, Martinez-Rego et al. [86] presented a new algorithm for stream change detection based on the combination of an online kernel one-class classification method with a Bernoulli CUSUM chart.

In this work, a proposal for an online one-class classification method based on the previous SCH algorithm is proposed. This method is capable of continuously adapting its classification model in a one-pass manner. In the following sections the algorithm and its response in stationary and dynamical environments are presented.

4.4.1 Online SCH algorithm

Before exposing the changes introduced to train the algorithm in an online manner, a new formulation to calculate the scaled convex polytope is introduced. Instead of a unique expansion parameter λ controlling the growth/compression of the original convex hull (see Section 4.2.1), in this case, each one of the vertices conforming the CH will have its own dynamic expansion parameter. The rationale of the strategy is to penalize old and unrepresentative vertices and strengthen the new ones in order to obtain a decision boundary adaptable to changing environments. Besides, this will allow us to mitigate the effects of outlier noisy patterns in the training set.

Once again, to calculate the expanded polytope in the low dimensional space, the formula presented by Liu et al. [79] is used. However, in this case, each CH vertex is expanded/reduced by its own λ parameter as follows:

$$\overline{V}^{\lambda} : \{\lambda_i \mathbf{v}_i + (1-\lambda_i)\overline{\mathbf{c}}\}, i = 1 \ldots n \tag{4.10}$$

where n is the number of vertices, $\overline{\mathbf{c}} = \mathbf{P}\mathbf{c}$ represents the projection of the center $\mathbf{c}$ given a random projection matrix $\mathbf{P}$, $\mathbf{v}_i$ is the ith vertex of the convex hull obtained with the projected data and λ_i is its expansion factor. Each parameter specifies a contraction ($0 \leq \lambda < 1$) or extension ($\lambda > 1$) of the appropriate vertex with respect to $\overline{\mathbf{c}}$. As in the previous SCH algorithm, with this formulation, the CH "center" can be obtained by three different ways (see Section 4.2.1): 1) the average of every training sample, 2) the average of the CH vertices and 3) the centroid of the convex polytope.

The proposed algorithm only takes into account the latest sample for training while maintaining the information of previous data through the ensemble model containing the vertices of the previous scaled convex hull (SCH) and the set of expansion parameters Λ related to them. Another difference with respect to the original SCH algorithm (see Section 4.2.1) is that the new proposal includes two new parameters for controlling the influence in the scaled convex hull of prior knowledge and new information at each step. The first parameter, $\lambda_{new} \in [0, +\infty)$, indicates the initial value of the expansion factor λ for each pattern that is added to the set of CH vertices for the first time. This occurs every time that a new sample falls outside the prior SCH. The second parameter, $\gamma \in (0, 1]$, penalizes old vertices that are located far from where new patterns are appearing and strengthen those that are close to new data. If γ is equal to 1, no penalization/reinforcement is made and the expansion parameter for each vertex of the CH will be equal to the initial value λ_{new}.

Algorithm 6 describes all the steps necessary to learn a model in an online fashion. It is a variation of the SCH learning algorithm (see Algorithm 3) but adapted to online environments. The number of projections τ, the type of center tc, a initial set of samples $I \in \mathbb{R}^D$ and the values of the two new parameters (γ and λ_{new}) have to be defined. An ensemble model containing τ projection matrices, the original values of the patterns that belong to the SCH vertices and their respective expansion parameters are obtained at the end of each step. Lines 1 to 6 describe the creation of the initial ensemble model. After that, steps 7 to 25 are repeated for each new training pattern $\mathbf{x}$. At each iteration t, the new point is projected to the low dimensional space spanned by the t-th projection matrix. Then the scaled convex hull (SCH) is obtained (using formula 4.10) and a new convex hull is calculated with the SCH vertices and the new projected pattern $\overline{\mathbf{x}_t}$. Function CH_{online} returns the new set of SCH vertices and updates the sets of expansion parameters Λ_t and original patterns V_t if some of the previous vertices are not part of the new SCH.

The remaining of the algorithm describes the updating procedure for the expansion parameters. On the one hand, if the new pattern $\overline{\mathbf{x}_t}$ is one of the new vertices of the SCH, then $\overline{\mathbf{x}_t}$ is added to the set of original vertices V_t and a new expansion factor with value λ_{new} is added to Λ. Besides, previous expansion factors are multiplied by γ to reduce its value (remember that $\gamma \in (0, 1]$). On the other hand, when $\overline{\mathbf{x}_t}$ does not belong to the new set of SCH vertices (i.e. $\overline{\mathbf{x}_t}$ falls inside the decision boundary), the expansion parameter of the closest vertex to $\overline{\mathbf{x}_t}$ is reinforced by dividing its previous value by γ^3 (i.e. the new value will be higher than the previous one) and the rest of the vertices are penalized. After the new pattern is processed, an updated ensemble model is available for classifying new patterns using the same testing procedure previously seen in algorithm 4.

Algorithm 6 : Online SCH learning algorithm

Inputs: τ ▷ Number of projections

tc ▷ Type of center (1-Avg. V_t; 2-Centroid)

$I \subset \mathbb{R}^D$ ▷ Initial set with the first 3 training samples

$\gamma \in (0,1]$ ▷ Penalty factor for old CH vertices

$\lambda_{new} \in [0,+\infty)$ ▷ Initial expansion factor value λ for each new CH vertex

Outputs: $(\mathbf{P}_t, V_t, \Lambda_t), t = 1..\tau$ ▷ Projection matrix, CH vertices and λ for each vertex

for t = 1 to τ ▷ Initialization

  $\mathbf{P}_t \sim N(0,1)$ ▷ Create a random projection matrix $[d \times D]$

  $\overline{I_t} : \{\mathbf{P}_t\mathbf{x} | \mathbf{x} \in I\}$ ▷ Project the initial set of data

  $V_t = \mathrm{CH}(\overline{I_t})$ ▷ Return the vertices of the CH

  $\Lambda_t = \{\lambda_{new}, \lambda_{new}, \lambda_{new}\}$ ▷ Expansion factor for each of the three initial CH vertices

end for

for each new training pattern $\mathbf{x} \in \mathbb{R}^D$

  for t = 1 .. τ

    $\overline{\mathbf{x}_t} : \{\mathbf{P}_t\mathbf{x}\}$ ▷ Project training point

    $\overline{\mathbf{c}} = getCenter(tc, V_t, \overline{\mathbf{x}_t})$

    $V_t^{\lambda} : \{\lambda_i \mathbf{v}_i + (1-\lambda_i)\overline{\mathbf{c}} | \lambda_i \in \Lambda_t, \mathbf{v}_i \in V_t\}$ ▷ Calculate SCH vertices

    $[V_t^{\lambda}, V_t, \Lambda_t] = CH_{online}(V_t^{\lambda} \cup \overline{\mathbf{x}_t})$ ▷ Update the new SCH vertices; return former λs and original vertices that remain in the SCH

    if $(\overline{\mathbf{x}_t} \in V_t^{\lambda})$ **then**

      $V_t = V_t \cup \overline{\mathbf{x}}_t$ ▷ Add the new point to the set of original vertices

      $\Lambda_t : \{\gamma\lambda_i | \lambda_i \in \Lambda_t\} \cup \lambda_{new}$ ▷ Update the set Λ_t and add a λ_{new} for vertex $\overline{\mathbf{x}_t}$

    else

      for i = 1 .. card(V_t)

        $z_i = \|\overline{x}_t - v_i\|^2, v_i \in V_t$ ▷ Distances from the new point to each vertex

      end for

      $k = pos\_min(\mathbf{z})$ ▷ Obtain the position of the nearest vertex

      $\lambda_i = \gamma\lambda_i, \forall \lambda_i \in \Lambda_t \wedge i \neq k$ ▷ Update expansion parameters

      $\lambda_k = \lambda_k / \gamma^3, \lambda_k \in \Lambda_t$ ▷ Rise expansion parameter of the nearest vertex

    end if

  end for

end for

4.4.2 Preliminary results

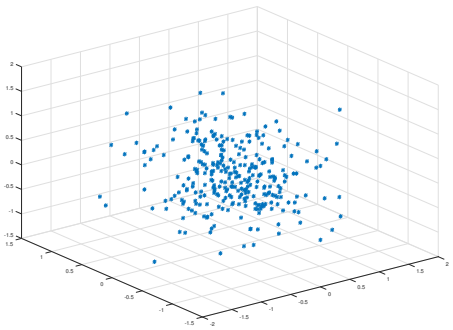
The experiments proposed in this section are intended to test the response of the proposed online one-class classification algorithm in stationary and dynamical environments in order to show its main properties. The characteristics of the three artificial datasets employed are listed in Table 4.9. For depiction purposes, three 3-dimensional datasets were generated and tracked by the proposed method. While datasets #1 and #2 are a static Gaussian, dataset #3 changes the center of a Gaussian continuously along an arc. Similar datasets have been used in previous works [21, 86] in order to asses the performance of one-class classifiers in dynamical environments.

Dataset	#1	#2	#3
Dimensionality	3	3	3
Size	300	300	600
Stationarity	YES	YES - Contains outliers	NO - Smooth change
λ_{new}	1.02	1.05	1.1
γ	0.999	0.996	0.995
Number of projections τ	3	3	3
Center type	Avg. x_i	Centroid	Avg. v_i

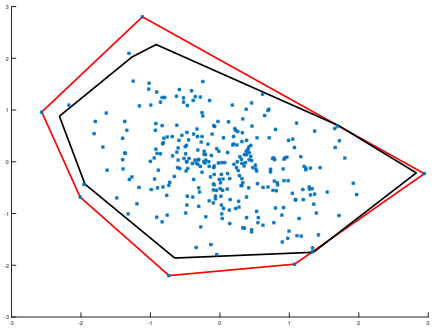
Table 4.9: Characteristics of the artificial datasets and parameters of the online method.

Table 4.9 also lists the parameter values employed for building the classifier in each dataset. This parameters were manually tuned in order to show the main characteristics of the online SCH method. Figure 4.17(a) depicts the 3D original dataset #1. Figures 4.17(b), 4.17(c) and 4.17(d) show the final decision boundary obtained by the proposed method (black polytope) and the global convex hull obtained with all the data (red polytope) for each 2D projection, respectively. It can be seen that the model approximates very well the high density region of the distribution, leaving outside only a few far away points. In this case both parameters, the penalty factor γ and the initial expansion parameter λ_{new}, were set to values near 1 in order to obtain a SCH model very similar to the global CH.

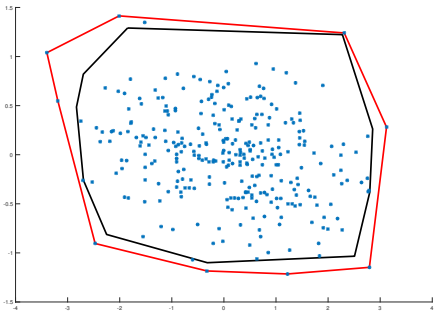
In dataset #2, 300 points belonging to a static Gaussian, similar to the one of the previous dataset, were employed. However, in this case, some outliers were randomly generated and mixed with the first 250 samples of the dataset. Figure 4.18(a) displays the original dataset #2 in 3 dimensions. Again, the final scaled convex hulls and the global ones obtained with the projected data are depicted in Figures 4.18(b), 4.18(c) and 4.18(d). In this experiment, the differences between the global convex hull and the scaled one, obtained with the online proposal, are clearly seen. The resulting scaled convex hull approximates better the underlying distribution than the global CH, leaving outside the undesired samples. Every outlier falls



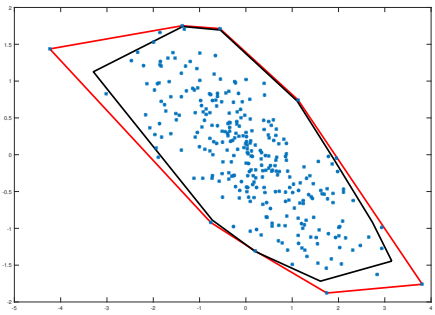
(a) Original 3D stationary dataset.



(b) Outcome of the proposed online SCH method for projection $\mathbf{P}_1$.



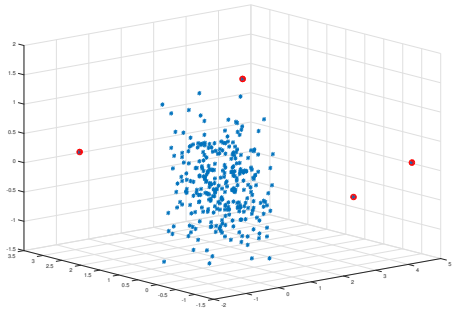
(c) Online SCH outcome for projection $\mathbf{P}_2$.



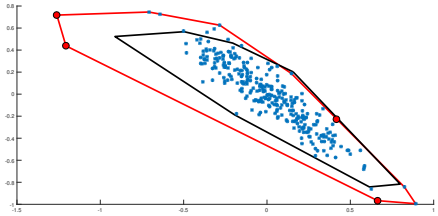
(d) Online SCH outcome for projection $\mathbf{P}_3$.

Figure 4.17: Results for artificial dataset #1. Global CH is depicted in red and the proposed SCH in black.

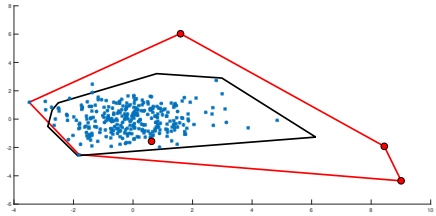
outside the SCH model in at least one of the projections. Then, the ensemble model combining the 2 dimensional SCH decisions will correctly alleviate the adverse effect that outliers usually cause during training. In this case, parameters were set to more aggressive values in order to forget faster those vertices that are not relevant to the model. In Figure 4.18(d) it can be clearly seen that the SCH model is bigger than the global CH in one particular area, that is due the λ_{new} parameter being higher than 1 and, thus, new vertices are expanded with respect to the center point in order to give them more relevance when they appear.



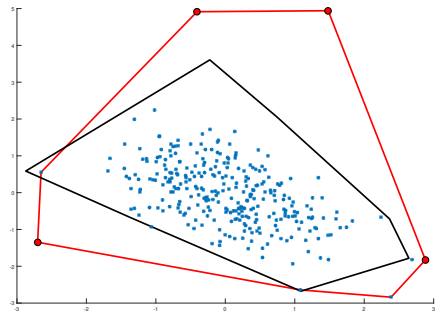
(a) Original 3D stationary dataset with outliers in red.



(b) Online SCH outcome for projection $\mathbf{P}_1$.



(c) Online SCH outcome for projection $\mathbf{P}_2$.



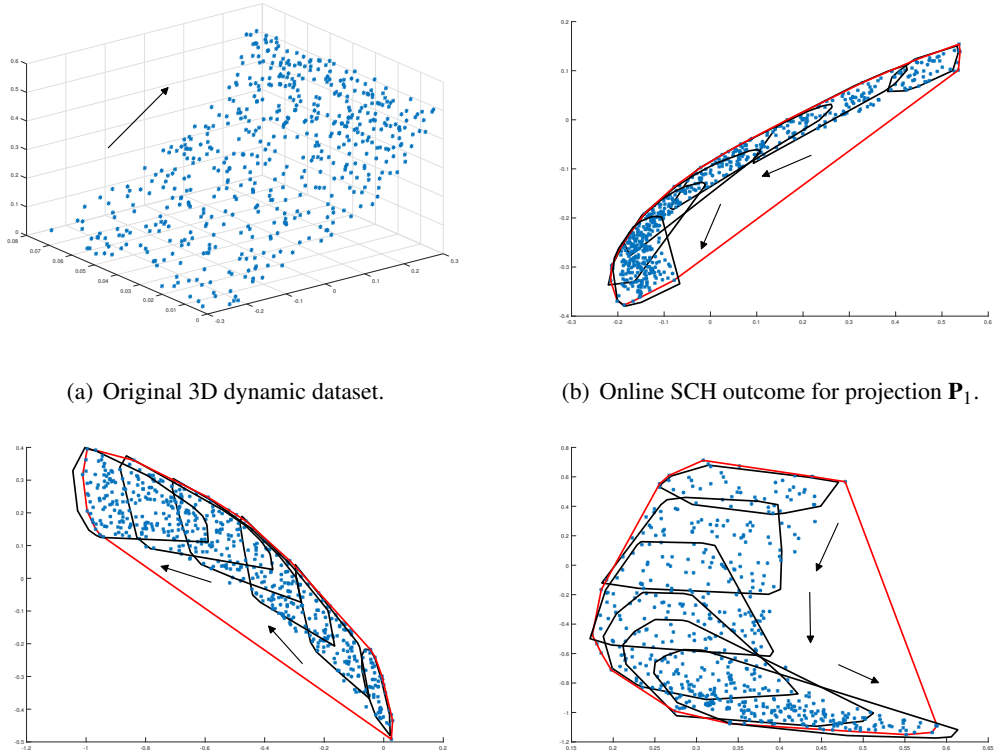
(d) Online SCH outcome for projection $\mathbf{P}_3$.

Figure 4.18: Results for artificial dataset #2. Red dots represent the four outliers. Global CH is depicted in red and the proposed SCH in black.

Finally, Figure 4.19(a) depicts the dynamic dataset #3, where the class moves continuously following a curve in the arrow direction. A new scaled convex hull is depicted after every 100 points in Figures 4.19(b), 4.19(c) and 4.19(d). As it can be observed, the decision boundary evolves in the direction of the dynamic dataset approximating appropriately the shape of the newest data when complex variations appear. It can also be seen that the global CH (red polygon) does not approximate adequately non-convex shapes like these.

The previous experiments demonstrated that in evolving environments or datasets where

the presence of outliers is very likely, a learning method able to adapt is decision model in an online manner, like the one proposed in this work, is of the most interest. This promising results encourage us to keep working on this proposal in order to obtain a complete stream change detection method able to point out significant deviations in continuous stream of data.



(a) Original 3D dynamic dataset.

(b) Online SCH outcome for projection $\mathbf{P}_1$.

(c) Online SCH outcome for projection $\mathbf{P}_2$.

(d) Online SCH outcome for projection $\mathbf{P}_3$.

Figure 4.19: Results for artificial dataset #3.

4.5 Discussion

In this chapter, a new distributed one-class classification algorithm (DSCH) is presented. It is based on the SCH algorithm, that in turn is a new and improved version of APE algorithm, designed to avoid the possible appearance of non convex situations and proposes three different centers that can be used. The one-class algorithm proposed is a distributed version of the SCH, aiming at managing those situations, scarcely contemplated in the specialized literature, in which the data resides in several different locations. Besides, it is possible to mine distributed data without revealing information that compromises the privacy of the individual

sources, unlike previous works as [27]. Many companies and public institutions are concerned about sharing their data. Examples include personal health records and bank-client confidential information. In this case, the information shared between nodes is encrypted because of the dimensionality reduction.

Our proposal is based on a modified version of the approximate polytope ensemble algorithm, that presents a different formula to calculate the expanded polytope in order to avoid an undesirable behavior detected in the original algorithm, the appearance of non-convex polytopes. Besides, this modification allowed the use of a new parameter (the center of the polytope), that provides more flexibility to our algorithm.

On the one hand, a comparative study in terms of Area Under the ROC Curve (AUC) between the proposed SCH algorithm, the APE algorithm and the state-of-the-art ν-SVM was made. To do so, 28 one-class problems derived from 11 UCI datasets were employed. Experimental results demonstrated that the proposed algorithm improves the performance of the other two algorithms. Furthermore, results showed that the choice of centers may affect considerably classification results for a given problem. On the other hand, three large-scale datasets that have a very large number of samples were used in order to assess the performance of the distributed one-class classification algorithm. Experimental results validated the distributed proposal, not only in terms of performance but also in terms of computational time savings. Performance of the distributed algorithm matches and even improves the performance achieved by the SCH algorithm. Besides, computational time employed to train the classifier is also reduced by a factor proportional to the number of nodes. Finally, a comparison of the proposed method against the distributed version of the state-of-the-art ν-SVM algorithm presented by Castillo et al. [27] has been made. Experimental results showed that our algorithm improves the results of Dν-SVM both in performance and computational time savings.

Furthermore, a new phase, called projection pruning phase, is added to the SCH one-class classification algorithm in order to improve its efficiency when testing a big bunch of data is required. During this stage, a rank of the relevant projections is obtained by means of mutual information. Besides, the validation dataset used in this phase does not need to be labeled, which is important in one-class problems where outlier data can be difficult to obtain. Experimental results demonstrated that the proposed approach maintains the performance of the original method with a minimum number of projection models, leading to savings in computational time that make this method more suitable for big data problems. Besides, this new phase could be straightforwardly applied to each computational node of the distributed SCH algorithm, creating ensemble models with different sets of projections according to the local data available in each node. However, in spite of the good performance achieved with this

proposal in the experiments, at this time, we lack of a proper theoretical explanation that would determine the properties of this algorithm from an information theory perspective.

Finally, the description of a new proposal for an online version of the SCH algorithm along with some promising initial results are also presented in this chapter. This is a work in progress that will also be part of our future research in this area.

The chapter, thus, contains different approaches for the scalability of one-class algorithms, including distributed and online versions, that are undoubtedly trending topics in the machine learning field as they allow for analyzing extremely large datasets.

CHAPTER 5

Conclusions

Nowadays, it is a fact that traditional computational learning algorithms cannot deal with very large datasets and plausibly obtain a good performance with reasonable requirements of computation, memory and time. This dissertation is devoted to the development of novel algorithms, focused in two aspects of machine learning: dimensionality reduction and classification, with the common aim of confronting large-scale datasets. There exists, basically, three different ways of ensuring algorithms scalability as datasets continues to grow in size and complexity, that are: 1) online learning, 2) non-iterative learning, and 3) distributed learning. Each part of this thesis covers one of this topics, presenting the problem and providing new solutions.

In light of the above the main contributions of this thesis are the following:

- Most of state-of-the art machine learning algorithms deal with the case where all data is available beforehand and we train a model in a batch manner. However, when data becomes available sequentially or when it is necessary for the algorithm to continuously adapt to new scenarios, learning one instance at a time in an online manner is needed. For these reasons, advances in this field have recently appeared, both for online feature selection and online classification tasks. In spite of that recent progress, the combination of both tasks in a single method is still an open issue. In this thesis, a new method which is capable of continuously updating its model to learn from online data is presented. The proposed method consist of three independent stages that can be used alone or in a pipeline, that are:
  - Discretization. This stage consist of an adaptation of the k-means discretizer to be used in an online manner.
  - Feature selection. The second stage is based on the χ^2 filter adapted by means of a new threshold to perform online feature selection.
  - Classification. This last stage consist of an adaptation of a one layer neural network to be incremental not only in the instance space, but also in the feature space, allowing for feature subsets that change during the learning process.

The main advantage of our proposed method is that it allows researchers to perform both online feature selection and classification. Up to the authors' knowledge, there is no other work in the literature that covers efficiently these two stages, since some of the existing approaches apply feature selection in an off line fashion or alternatively apply online feature selection but then there is not an online classification stage. Furthermore, since an important aspect of on-line algorithms is the impact of data order on their performance, this aspect is specifically addressed, demonstrating the robustness of the method. It is important to take into account that, although more sophisticated learning methods exist, they cannot be adapted to learn in an online manner, and we had to choose simpler models such as the χ^2 filter and an ANN. Experimental results demonstrate that the proposed pipeline appropriately solves the three tasks involved on both synthetic and real datasets.

- Traditionally, learning algorithms have been focused on training their models from monolithic datasets with all data stored in main memory. When the complexity of the algorithm surpasses the computational resources then the algorithm does not scale well. Training procedures such as backpropagation make this problem even worse, as they rely on an iterative process to adjust their parameters. One of those classical machine learning methods that suffers the consequences of iterative training procedures is the autoencoder. This is an autoassociative neural network for one-class classification widely and successfully used in the past. However, its iterative nature impedes its efficient application to large-scale datasets. Trying to solve this problem, in this thesis a new fast non-iterative learning neural network for one-class classification, called SVD-autoencoder, is presented. This method proposes the use of a previously presented cost function based on the mean-squared error that measures the error before the output neural function. The optimal weights are derived from a system of linear equations that is further transformed by means of the Singular Value Decomposition (SVD). Besides, the SVD-autoencoder allows dimensionality reduction in a very efficient manner. In the hidden layer, the method extracts a meaningful low dimensional representation of the input data by means, again, of Singular Value Decomposition. The main advantages of this proposed algorithm are that it can be used with nonlinear activation functions and that it does not run into any convergence issues thanks to its non-iterative optimization. Another particularity of this proposal is that it can be easily parallelized. The minimization problem of the second layer can be divided in several sub-problems (one for each output neuron) that can be solved in parallel. Thus, this characteristic makes the SVD-autoencoder highly scalable and efficient also when the number of features is high. Experimental results demonstrate that the proposed method successfully solves a wide range of one class classification problems and is able to efficiently deal with large-scale datasets.

- Nowadays, several sources in different locations produce data creating scenarios with big distributed databases. However, most existing learning algorithms cannot deal with this fact. For that cases and applications where data is distributed across different sites, traditional learning methods require gathering all the data in a single node for central processing, but this can not be done for several reasons -e.g. communication cost, storage cost, computational cost, privacy, etc-. There exists some previous works related with distributed machine learning, but very few in the context of one-class classification. Therefore, in the fourth chapter of the thesis a new one-class convex hull-based classification algorithm (DSCH) that can scale out over distributed architectures is proposed. This proposal is based on a modified version of the approximate polytope ensemble algorithm, that presents a different formula to calculate the expanded polytope in order to avoid an undesirable behavior detected in the original algorithm, the appearance of non-convex polytopes. This proposed method approximates the high dimensional convex hull decision by means of a dimensionality reduction technique (i.e. random projections) and an ensemble of convex hull decisions in very low dimensions. One of the most important characteristics of the DSCH algorithm is that it allows to mine distributed data without revealing information that compromises the privacy of the individual sources, unlike some previous works. Experimental results demonstrate the adequacy of the distributed proposal, not only in terms of performance but also in computational time savings. Furthermore, a new phase, called projection pruning phase, to eliminate the less relevant and redundant random projections in order to obtain a lightweight ensemble model, more scalable and efficient, is also proposed. It employs mutual information in order to obtain a rank of the most relevant projections.

The following lines of research are proposed as future work:

- Extend the pipeline presented in Chapter 2 for most sophisticated feature selection methods.
- Try the SVD-autoencoder proposed in Chapter 3 with high dimensional datasets (i.e. very large number of features).
- In spite that the projection pruning phase presented in Chapter 4 appropriately solves the problem of selecting the most relevant projections, it lacks of a proper theoretical explanation from an information theory perspective, that would allow to discard the less relevant projections at the beginning of the training phase.
- In the last part of Chapter 4, a brief description of a proposal for an online version of

the SCH algorithm is presented. This is a work in progress that will also be part of our future research.

APPENDIX

Author's key publications

As a result of the research carried out during the thesis, the following articles have been published.

JCR Journals

- Fernández-Francos, D., Martínez-Rego, D., Fontenla-Romero, O. and Alonso-Betanzos, A. *Automatic bearing fault diagnosis based on one-class ν-SVM*. Computers & Industrial Engineering (vol. 64, no. 1, pp. 357-365, 2013).

- Martínez-Rego, D., Fernández-Francos, D., Fontenla-Romero, O. and Alonso-Betanzos, A. *Stream change detection via passive-aggressive classification and Bernoulli CUSUM*. Information Sciences (vol. 305, pp. 130-145, 2015).

- Bolon-Canedo, V., Fernández-Francos, D., Peteiro-Barral, D., Alonso-Betanzos, A., Guijarro-Berdiñas, B. and Sánchez-Maroño, N. *A unified pipeline for online feature selection and classification*. Expert Systems with Applications (vol. 55, pp. 532-545, 2016)

JCR Journals (Under review process)

- Fernández-Francos, D., Fontenla-Romero, O. and Alonso-Betanzos, A. *One-class convex hull-based algorithm for classification in distributed environments*. IEEE Transactions on Systems, Man, and Cybernetics: Systems (2016).

Conferences

- Fernandez-Francos, D., Fontenla-Romero, O. and Alonso-Betanzos, A. *One-class classification algorithm based on convex hull*. In Proc. 24th European Symposium on Artificial Neural Networks (ESANN'16), pp. 477-482, Bruges, Belgium, April 2016.
- Fernandez-Francos, D., Fontenla-Romero, O. and Alonso-Betanzos, A. *Mutual information for improving the efficiency of the SCH algorithm*. In Proc. 25th European Symposium on Artificial Neural Networks (ESANN'17), Bruges, Belgium, April 2016.

Book chapters

- Alonso-Betanzos, A., Bolon-Canedo, V., Fernandez-Francos, D., Porto-Díaz, I. and Sánchez-Maroño, N. *Up-to-date feature selection methods for scalable and efficient machine learning in Efficiency and Scalability Methods for Computational Intellect*, B. Igelnik and J. M. Zurada, Eds: IGI Global, 2013, pp. 1-26.

APPENDIX II

Resumen en castellano

En 2011, según la *International Data Corporation* (IDC), el volumen general de datos en el mundo era de 1.800 exabytes. Desde entonces hasta 2020, esa cantidad se doblará cada dos años [29]. En general, este aumento explosivo del volumen de datos ha modificado las maneras de compartir información y ha puesto en evidencia la necesidad de desarrollar nuevos métodos eficientes para procesar y almacenar grandes cantidades de datos. Además, el análisis automático de estos datos se ha convertido en una gran oportunidad económica y científica. Las empresas que invierten y adquieren con éxito valor de sus propios datos poseen una ventaja obvia sobre sus competidores. La investigación científica también se ha visto revolucionada en campos como la astronomía, la bioinformática, la detección de intrusos en redes informáticas, la clasificación de textos o la ingeniería, en los que el tamaño y la cantidad de conjuntos de datos disponibles están creciendo de forma exponencial.

El aprendizaje computacional es un área de la inteligencia artificial dedicada al diseño, desarrollo y análisis de algoritmos de aprendizaje automático. En particular, dichos algoritmos pueden aprender a partir de los datos, hacer predicciones o crear representaciones exactas basadas en observaciones. En este contexto, en el cual el número de datos crece más rápido que la velocidad de los procesadores, la capacidad de los algoritmos de aprendizaje máquina se encuentra limitada por el tiempo de computación. Una base de datos se considera de gran tamaño cuando: el número de muestras es muy grande; el número de características es muy grande; o ambos son muy grandes. Los métodos de aprendizaje tradicionales tienen grandes dificultades cuando se aplican en bases de datos con alrededor de 10.000.000 de datos (siendo datos = muestras $\times$ características). En este trabajo, hablaremos de grandes conjuntos de datos para referirnos a aquellas bases de datos de alta dimensionalidad en las que el número de muestras es considerablemente mayor que el número de características.

En teoría, parece lógico concluir que a mayor cantidad de información mejores serán los resultados. Sin embargo, no siempre es así debido a la llamada maldición de la dimensionalidad (*curse of dimensionality*) [5]. Este fenómeno ocurre cuando la dimensionalidad de los datos crece y el tiempo requerido por el algoritmo de aprendizaje computacional para entre-

nar aumenta drásticamente. Además, al tratar con gran cantidad de datos, los algoritmos de aprendizaje pueden degenerar su rendimiento debido al sobreajuste (*overfitting*) y su eficiencia decae de acuerdo con el tamaño. Por lo tanto, la escalabilidad ha dejado de ser una característica deseable de los algoritmos de aprendizaje para convertirse en una propiedad crucial cuando se trabaja con conjuntos de datos muy grandes.

Dos de las tareas más comunes en el aprendizaje automático, en las cuales se centra esta tesis, son: 1) clasificación, donde el algoritmo asigna muestras no vistas a una serie de categorías; y 2) reducción de la dimensionalidad, donde las muestras son simplificadas mediante el mapeo a espacios de menor dimensión. De acuerdo con la naturaleza de los conjuntos de datos de aprendizaje disponibles, las tareas anteriores también pueden clasificarse en: a) aprendizaje supervisado, donde todos los datos están etiquetados y los algoritmos aprenden a predecir el resultado a partir de los datos de entrada; b) aprendizaje no supervisado, donde los datos no están etiquetados y los algoritmos averiguan la estructura inherente de los datos de entrada; y c) aprendizaje semi-supervisado, en el que algunos datos están etiquetados, pero la mayoría de ellos no lo están, permitiendo el uso de una variedad de técnicas supervisadas y no supervisadas.

A la hora de tratar con grandes bases de datos, un aspecto esencial es la preparación adecuada de los datos para ser procesados por los algoritmos de aprendizaje. Esto ha desencadenado la utilización de procedimientos de reducción de dimensionalidad como un paso previo al procesado, con el fin de reducir el tamaño de los datos y mejorar el rendimiento de los algoritmos de aprendizaje. Existen básicamente dos tipos de técnicas de reducción de dimensionalidad: 1) extracción de características [78], donde un nuevo conjunto de características es obtenido mediante un mapeo funcional de los datos originales a un espacio de menor dimensión; y 2) selección de características [149], cuyo objetivo es la determinación de las características más relevantes y la eliminación de las redundantes.

Clasificación, el otro problema en que se centra este trabajo, es el área del aprendizaje máquina centrada en la identificación de la categoría a la que pertenece una nueva observación. Los métodos tradicionales utilizan datos de todas las clases para aprender el modelo (i.e. clasificación supervisada) [71]. Sin embargo, existen muchos problemas reales en los que la gran cantidad de datos disponible impide el etiquetado manual de la clase de cada muestra, siendo necesaria la utilización de métodos de clasificación no supervisada o semi-supervisada. Un caso particular que ha experimentado un auge importante en los últimos años es la clasificación uniclase, en la que solo es posible obtener datos de una clase (clase normal o positiva) para entrenar; los datos de otras clases, las clases atípicas, son muy difíciles o imposibles de obtener. Los algoritmos de aprendizaje uniclase solamente utilizan datos de una clase para construir un

modelo y su objetivo es identificar los datos pertenecientes a esa clase y rechazar los de todas las demás. Esta técnica ha sido utilizada con éxito en una amplia variedad de problemas reales, como por ejemplo: la detección de fallos en maquinaria industrial, el diagnóstico médico, la detección de intrusos en sistemas de seguridad y la clasificación de documentos.

Una práctica común en el aprendizaje automático es la aplicación, como paso previo al procesamiento, de técnicas de selección o extracción de características a problemas de clasificación. Esta es una de las premisas que deben seguir los métodos desarrollados en esta tesis. La otra premisa es que los nuevos métodos propuestos deben ser escalables y mejorar el rendimiento de los algoritmos clásicos en este nuevo escenario formado por grandes conjuntos de datos. Los algoritmos de aprendizaje de máquina tradicionales se diseñaron para extraer la mayor cantidad de información del número limitado de datos disponible. Sin embargo, hoy en día, el nuevo factor limitante lo constituye la incapacidad de los algoritmos de aprendizaje para hacer frente a la ingente cantidad de datos disponible en un tiempo de computación razonable. Con el fin de manejar esta nueva situación, ha surgido un nuevo campo en el aprendizaje automático: el aprendizaje a gran escala (*large-scale learning*), cuyo objetivo es el desarrollo de algoritmos de aprendizaje escalables con respecto a los requerimientos de complejidad computacional, memoria y tiempo. En los últimos años, el aprendizaje a gran escala ha atraído mucha atención. Existen, básicamente, tres enfoques diferentes para asegurar la escalabilidad de los algoritmos a medida que los conjuntos de datos continúan creciendo en tamaño y complejidad [146]: 1) aprendizaje en tiempo real, 2) aprendizaje no iterativo y 3) aprendizaje distribuido. Estos tres temas se han cubierto de manera independiente en cada una de las tres partes principales en las que se divide esta tesis. En este resumen se proporciona una breve descripción de las intenciones, resultados y conclusiones del presente trabajo en cada campo.

II.1 Método de selección de características y clasificación para aprendizaje en tiempo real

Registros de eventos en la red, registros de llamadas telefónicas y secuencias de vídeo vigilancia constituyen ejemplos de datos que fluyen de forma constante. Los enfoques tradicionales de aprendizaje (*batch learning*) no pueden manejar este tipo de datos, ya que para crear sus modelos necesitan aprender sobre todo el conjunto de datos al mismo tiempo. Por lo tanto, para hacer frente a esta situación, son necesarios algoritmos capaces de aprender muestra a muestra, de manera secuencial. El campo de la inteligencia artificial dedicado al aprendizaje de este tipo de datos se denomina aprendizaje en tiempo real (*online learning*). Este campo

se ha convertido en una tendencia en los últimos años, ya que permite aprender un modelo de manera incremental. Ha sido utilizado principalmente para aquellas situaciones en las que los datos están disponibles de forma secuencial o cuando es computacionalmente inviable entrenar sobre todo el conjunto de datos [136]. También es una técnica útil en aquellas situaciones en las que es necesario que el algoritmo se adapte de manera dinámica a posibles cambios en la distribución subyacente de los datos (*concept drift*). Por estos motivos, en los últimos años han aparecido numerosos trabajos en este campo, tanto para la selección de características en tiempo real (*online feature selection*) como para las tareas de clasificación en tiempo real (*online classification*). A pesar de estos avances, la combinación de ambas tareas en un solo método sigue siendo una cuestión pendiente. En esta parte de la tesis, se presenta un nuevo método que es capaz de actualizar continuamente su modelo para aprender de los datos en tiempo real. El método propuesto consta de tres etapas independientes que se pueden utilizar solas o en conjunto, que son:

- Discretización. Esta etapa consiste en una adaptación del método de discretización k-means para ser utilizado de forma incremental.
- Selección de características. La segunda etapa se basa en el filtro χ^2 adaptado para realizar la selección de características en tiempo real mediante la utilización de un nuevo umbral automático.
- Clasificación. Esta última etapa consiste en una red neuronal de una capa modificada para ser incremental no sólo en el número de muestras, sino también en el número de características, permitiendo que los subconjuntos de características varíen de tamaño durante el proceso de aprendizaje.

La ventaja principal del método propuesto es que permite realizar tanto la selección de características como la clasificación en tiempo real. Hasta el momento, no existe ningún otro trabajo en la literatura que cubra eficientemente estas dos etapas. Además, dado que un aspecto importante de los algoritmos incrementales es el impacto que el orden de aparición de los datos pueda tener sobre su rendimiento, este aspecto se aborda de forma específica en el estudio experimental, demostrando la robustez del método. Es importante tener en cuenta que, aunque existen métodos de aprendizaje más sofisticados, éstos no pueden ser adaptados para aprender de manera incremental, y por ello se han elegido modelos más simples como el filtro χ^2 y la red de neuronas de una capa. Los resultados experimentales demuestran que el método propuesto resuelve apropiadamente las tres tareas involucradas, tanto en conjuntos de datos sintéticos como reales.

Una línea de investigación futura relacionada con este tema consiste en la extensión del método propuesto de forma que permita la utilización de algoritmos de selección de características y clasificación más potentes.

II.2 Método no iterativo de clasificación uniclase para grandes conjuntos de datos

Tradicionalmente, los algoritmos de aprendizaje operan con todos los datos almacenados en la memoria principal. De esta forma, cuando la complejidad del algoritmo sobrepasa los recursos computacionales éste no escala bien. Además, muchos de los algoritmos clásicos de aprendizaje automático, especialmente los basados en redes neuronales artificiales (RNAs), ajustan sus parámetros libres mediante un procedimiento de entrenamiento iterativo con el fin de reducir el error del modelo al mínimo. La mayoría de estos métodos iterativos se basan en *backpropagation* [108] y en alternativas de segundo orden [93, 116]. Hoy en día, los grandes conjuntos de datos disponibles provocan que muchos algoritmos de aprendizaje iterativos no sean aplicables debido a sus altos requerimientos computacionales y a su baja velocidad de convergencia. En esos casos, una forma de reducir la complejidad de los métodos iterativos clásicos consiste en el desarrollo de nuevos métodos de optimización no iterativos para el ajuste de los parámetros del modelo. Sin embargo, las propuestas no iterativas son escasas en la literatura, con sólo algunas contribuciones destacadas para las RNAs [105, 112, 135].

Uno de los métodos tradicionales de aprendizaje que sufre las consecuencias de tratar con conjuntos de datos muy grandes es el autoencoder [138]. Se trata de una red neuronal autoasociativa para clasificación uniclase utilizada de forma exitosa en el pasado en múltiples problemas. Sin embargo, su naturaleza iterativa impide su aplicación eficiente para grandes conjuntos de datos. Tratando de resolver este problema, en esta parte de la tesis se presenta una nueva red neuronal de aprendizaje no iterativo para clasificación uniclase, llamada SVD-autoencoder. Este método propone el uso de una función de coste basada en el error cuadrático medio, que mide el error antes de la función neural de salida. Los pesos óptimos se derivan de un sistema de ecuaciones lineales que son a su vez transformadas por medio de la Descomposición en Valores Singulares (SVD). Además, el SVD-autoencoder permite reducir la dimensionalidad de una manera muy eficiente. En la capa oculta, el método extrae una representación significativa de menor dimensión de los datos de entrada por medio, de nuevo, de la SVD. Las principales ventajas de este algoritmo son que se puede utilizar con funciones de activación no lineales y que no presenta ningún problema de convergencia gracias a su optimización no iterativa.

Otra particularidad de esta propuesta es que puede ser fácilmente paralelizable. El problema de minimización de la segunda capa, mediante el que se calculan los pesos, se puede dividir en varios subproblemas (uno para cada neurona de salida) que es posible resolver en paralelo. Por lo tanto, esta característica hace que el SVD-autoencoder sea altamente escalable y eficiente también cuando el número de características es alto. Los resultados experimentales demuestran que el método propuesto resuelve con éxito una amplia gama de problemas de clasificación de uniclase y es capaz de manejar de manera eficiente grandes conjuntos de datos.

Una posible línea de investigación propuesta como trabajo futuro dentro de este campo consistiría en el análisis del rendimiento y la escalabilidad del SVD-autoencoder con conjuntos de datos altamente dimensionales, es decir, conjuntos con gran número de muestras y de características.

II.3 Método de clasificación uniclase para entornos de datos distribuidos

En general, la llegada del *Big Data* ha contribuido a la proliferación de grandes bases de datos, normalmente distribuidas, cuyo análisis automático es de gran interés. Sin embargo, la mayoría de los algoritmos de aprendizaje existentes no pueden manejar esta circunstancia. La gran mayoría de ellos requiere reunir las distintas particiones de datos en un único nodo para su procesamiento centralizado. Sin embargo, existen situaciones en las que esta aproximación es inviable o ineficaz [131], debido a:

- *Costes de almacenamiento*. No se puede disponer de la capacidad de almacenamiento necesaria. Por ejemplo, el almacenamiento central de los datos de todos los hospitales de un país (imágenes médicas, registros de pacientes, etc.) requeriría un enorme almacén de datos de gran coste económico.
- *Costes de comunicación.* El tiempo para transferir eficientemente a través de una red este gran volumen de datos es enorme.
- *Costes computacionales*. Los algoritmos de aprendizaje pueden ser incapaces de tratar con tales volúmenes de datos debido a los requisitos computacionales y de memoria.
- *Privacidad y confidencialidad de los datos*. La necesidad de preservar la privacidad de los datos hace imposible compartirlos entre lugares distintos. Una vez más, los registros

de un paciente son un ejemplo de datos que si se comparten a través de una red pondrían en riesgo la privacidad.

Con el fin de aportar soluciones a estos nuevos problemas, ha surgido un nuevo campo de investigación, el aprendizaje distribuido. Este campo está atrayendo mucha atención últimamente y se está convirtiendo en una de las líneas de investigación más prometedoras en el aprendizaje automático. El problema con la mayoría de los algoritmos distribuidos existentes es que prácticamente ninguno de ellos considera todas las restricciones y condiciones mencionadas anteriormente y que surgen al trabajar en este tipo de entornos. Esto es más cierto, si cabe, en el caso de los algoritmos de aprendizaje de una clase.

Por lo tanto, en la última parte de la tesis se propone un nuevo algoritmo de clasificación uniclase basado en el cierre convexo (*Distributed Scaled Convex Hull*, DSCH), especialmente diseñado para trabajar sobre arquitecturas distribuidas. Esta propuesta se basa en una versión modificada del algoritmo propuesto por Casale et al. [26] (APE), que introduce una fórmula diferente para calcular el politopo expandido/reducido con el fin de evitar un comportamiento indeseable detectado en el algoritmo original, la aparición de politopos no convexos. El método propuesto (SCH) aproxima la decisión del cierre convexo de alta dimensión por medio de una técnica de reducción de dimensionalidad (proyecciones aleatorias) y un conjunto (*ensemble*) de decisiones de cierre convexo en dimensiones muy bajas. Una de las características más importantes del algoritmo DSCH es que permite trabajar con datos distribuidos sin revelar información que comprometa la privacidad de las fuentes individuales, a diferencia de algunos trabajos propuestos con anterioridad. Los resultados experimentales demuestran la adecuación de la propuesta distribuida, no sólo en términos de rendimiento, sino también en el ahorro de tiempo computacional.

Además, se propone una nueva fase, denominada *projection pruning phase*, para eliminar las proyecciones aleatorias menos relevantes y redundantes con el fin de obtener un *ensemble* más escalable y eficiente. Se utiliza la información mutua para obtener un ranking de las proyecciones más relevantes. A pesar de que la *projection pruning phase* propuesta resuelve de forma adecuada el problema de seleccionar las proyecciones más relevantes, carece de una explicación teórica adecuada desde el punto de vista de la teoría de la información. La obtención de dicha explicación se propone como línea de trabajo futuro.

Finalmente, en la última parte del capítulo 4, se presenta una breve descripción de una propuesta para una versión incremental del algoritmo SCH. Se trata de un trabajo en progreso y constituye una de las líneas de investigación futura.

II.4 Estructura de la tesis

Esta tesis se organiza en cinco capítulos y dos anexos:

1. El capítulo 1 presenta la introducción, el dominio de la investigación y la estructura de la tesis.
2. El capítulo 2 describe un nuevo método combinado de selección de características y clasificación en tiempo real.
3. El capítulo 3 propone un nuevo método no iterativo de clasificación uniclase escalable y eficiente para tratar grandes conjuntos de datos.
4. El capítulo 4 presenta un nuevo método de clasificación uniclase para entornos de datos distribuidos basado en el cierre convexo.
5. El capítulo 5 resume las contribuciones y conclusiones obtenidas de este trabajo.
6. El Anexo I incluye una lista de las publicaciones obtenidas como fruto de la investigación realizada en esta tesis.
7. El Anexo II presenta un breve resumen en castellano del contenido de esta tesis.

Bibliography

[1] KDD Cup 99 dataset. *http://kdd.ics.uci.edu/databases/kddcup99* (Accesed 28.02.15).

[2] ASUNCION A. AND NEWMAN D. Uci machine learning repository (2007).

[3] BALDI P. AND HORNIK K. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks* **2**(1), 53 – 58 (1989).

[4] BASHEIN G. AND DETMER P. R. Centroid of a polygon. In "Graphics gems IV", pages 3–6. Academic Press Professional, Inc. (1994).

[5] BELLMAN R. Dynamic programming. *Science* **153**(3731), 34–37 (1966).

[6] BERGAMINI C., KOERICH A. L., AND SABOURIN R. Combining different biometric traits with one-class classification. *Signal Processing* **89**(11), 2117–2127 (2009).

[7] BINGHAM E. AND MANNILA H. Random projection in dimensionality reduction: applications to image and text data. In "Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining", pages 245–250. ACM (2001).

[8] BISHOP C. Novelty detection and neural network validation. *IEEE Proceedings on Vision, Image and Signal Processing* **141**, 217–222 (1994).

[9] BISHOP C. "Neural Networks for Pattern Recognition". Oxford University Press (1995).

[10] BISHOP C. M. "Pattern Recognition and Machine Learning (Information Science and Statistics)". Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006).

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

[12] BOLÓN-CANEDO V., SÁNCHEZ-MAROÑO N., AND ALONSO-BETANZOS A. Recent advances and emerging challenges of feature selection in the context of big data. *Knowledge-Based Systems* **86**, 33–45 (2015).

[13] BOLON-CANEDO V., FERNÁNDEZ-FRANCOS D., PETEIRO-BARRAL D., ALONSO-BETANZOS A., GUIJARRO-BERDIÑAS B., AND SÁNCHEZ-MAROÑO N. A unified pipeline for online feature selection and classification. *Expert Systems with Applications* **55**, 532–545 (2016).

[14] BOLON-CANEDO V., SANCHEZ-MARONO N., AND ALONSO-BETANZOS A. Feature selection and classification in multiple class datasets: An application to kdd cup 99 dataset. *Expert Systems with Applications* **38**(5), 5947–5957 (2011).

[15] BOLÓN-CANEDO V., SÁNCHEZ-MAROÑO N., AND ALONSO-BETANZOS A. A review of feature selection methods on synthetic data. *Knowledge and information systems* **34**(3), 483–519 (2013).

[16] BOTTOU L. Large-scale machine learning with stochastic gradient descent. In "Proceedings of COMPSTAT'2010", pages 177–186. Springer (2010).

[17] BRANCH J. W., GIANNELLA C., SZYMANSKI B., WOLFF R., AND KARGUPTA H. In-network outlier detection in wireless sensor networks. *Knowledge and information systems* **34**(1), 23–54 (2013).

[18] BREIMAN L. "Classification and regression trees". Chapman & Hall/CRC (1984).

[19] BREW A., GRIMALDI M., AND CUNNINGHAM P. An evaluation of one-class classification techniques for speaker verification. *Machine Learning* **27**(4), 295–307 (2007).

[20] BUDINICH M. AND MILOTTI E. Geometrical interpretation of the back-propagation algorithm for the perceptron. *Physica A: Statistical Mechanics and its Applications* **185**(1-4), 369–377 (1992).

[21] CAMCI F. AND CHINNAM R. B. General support vector representation machine for one-class classification of non-stationary classes. *Pattern Recognition* **41**(10), 3021–3034 (2008).

[22] CARAYANNIS G., KALOUPTSIDIS N., AND MANOLAKIS D. Fast recursive algorithms for a class of linear equations. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **30**, 227–239 (1982).

[23] CARPENTER G., GROSSBERG S., AND ROSEN D. Art 2-a: an adaptive resonance algorithm for rapid category learning and recognition. *Neural Networks* **4**(4), 493–504 (1991).

[24] CARVALHO V. R. AND COHEN W. W. Single-pass online learning: Performance, voting schemes and online feature selection. In "Proceedings of the 12th ACM SIGKDD

international conference on Knowledge discovery and data mining", pages 548–553. ACM (2006).

[25] CASALE P., PUJOL O., AND RADEVA P. Approximate convex hulls family for one-class classification. In "International Workshop on Multiple Classifier Systems", pages 106–115. Springer (2011).

[26] CASALE P., PUJOL O., AND RADEVA P. Approximate polytope ensemble for one-class classification. *Pattern Recognition* **47**(2), 854–864 (2014).

[27] CASTILLO E., PETEIRO-BARRAL D., BERDIÑAS B. G., AND FONTENLA-ROMERO O. Distributed one-class support vector machine. *International journal of neural systems* **25**(07), 1550029 (2015).

[28] CHANDOLA V., BANERJEE A., AND KUMAR V. Anomaly detection: A survey. *ACM computing surveys (CSUR)* **41**(3), 15 (2009).

[29] CHEN M., MAO S., AND LIU Y. Big data: A survey. *Mobile Networks and Applications* **19**(2), 171–209 (2014).

[30] COVER T. M. AND THOMAS J. A. "Elements of information theory". John Wiley & Sons (2012).

[31] DEMŠAR J. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* **7**, 1–30 (2006).

[32] DING X., LI Y., BELATRECHE A., AND MAGUIRE L. P. An experimental evaluation of novelty detection methods. *Neurocomputing* **135**, 313–327 (2014).

[33] DUDA R. AND HART P. "Pattern Classification and Scene Analysis". John Wiley & Sons, New York (1973).

[34] ECKART C. AND YOUNG G. The approximation of one matrix by another of lower rank. *Psychometrika* **1**(3), 211–218 (1936).

[35] ELAHI M., LI K., NISAR W., LV X., AND WANG H. Efficient clustering-based outlier detection algorithm for dynamic data stream. In "Fuzzy Systems and Knowledge Discovery, 2008. FSKD'08. Fifth International Conference on", volume 5, pages 298–304. IEEE (2008).

[36] ELWELL R. AND POLIKAR R. Incremental learning in nonstationary environments with controlled forgetting. In "Neural Networks, 2009. IJCNN 2009. International Joint Conference on", pages 771–778. IEEE (2009).

[37] ERDOGMUS D., JENSSEN R., RAO Y., AND PRINCIPE J. Multivariate density estimation with optimal marginal parzen density estimation and gaussianization. In "Machine Learning for Signal Processing, 2004. Proceedings of the 2004 14th IEEE Signal Processing Society Workshop", pages 73–82. IEEE (2004).

[38] FANG S. AND ZIJIE W. Rolling bearing fault diagnosis based on wavelet packet and rbf neural network. In "Control Conference, 2007. CCC 2007. Chinese", pages 451–455. IEEE (2007).

[39] FONTENLA-ROMERO O., GUIJARRO-BERDIÑAS B., PÉREZ-SÁNCHEZ B., AND ALONSO-BETANZOS A. A new convex objective function for the supervised learning of single-layer neural networks. *Pattern Recognition* **43**(5), 1984–1992 (2010).

[40] FONTENLA-ROMERO O., PÉREZ-SÁNCHEZ B., AND GUIJARRO-BERDIÑAS B. LANN-SVD: a non-iterative svd-based learning algorithm for one-layer neural networks. *Submitted to IEEE Transactions on Neural Networks and Learning Systems, 2016* (2016).

[41] FRIEDMAN J. Multivariate adaptive regression splines. *The annals of statistics* pages 1–67 (1991).

[42] GANTZ J. AND REINSEL D. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future* **2007**(2012), 1–16 (2012).

[43] GARDNER A. B., KRIEGER A. M., VACHTSEVANOS G., AND LITT B. One-class novelty detection for seizure analysis from intracranial eeg. *Journal of Machine Learning Research* **7**(Jun), 1025–1044 (2006).

[44] GIACINTO G., PERDISCI R., DEL RIO M., AND ROLI F. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Information Fusion* **9**(1), 69–82 (2008).

[45] GLOCER K., EADS D., AND THEILER J. Online feature selection for pixel classification. In "Proceedings of the 22nd international conference on Machine learning", pages 249–256. ACM (2005).

[46] GUYON I., GUNN S., NIKRAVESH M., AND ZADEH L. "Feature Extraction. Foundations and Applications". Springer (2006).

[47] GUYON I., WESTON J., BARNHILL S., AND VAPNIK V. Gene selection for cancer classification using support vector machines. *Machine learning* **46**(1), 389–422 (2002).

[48] HALL M., FRANK E., HOLMES G., PFAHRINGER B., REUTEMANN P., AND WITTEN I. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter* **11**(1), 10–18 (2009).

[49] HALL M. A. "Correlation-based feature selection for machine learning". PhD thesis, The University of Waikato (1999).

[50] HAYES M. A. AND CAPRETZ M. A. Contextual anomaly detection in big sensor data. In "2014 IEEE International Congress on Big Data", pages 64–71. IEEE (2014).

[51] HERTZ J., KROGH A., AND PALMER R. "Introduction to the theory of neural computation". Addison Wesley Publishing Company (1991).

[52] HODGE V. J. AND AUSTIN J. A survey of outlier detection methodologies. *Artificial intelligence review* **22**(2), 85–126 (2004).

[53] HSU J. "Multiple comparisons: theory and methods". Chapman & Hall/CRC (1996).

[54] HWANG B. AND CHO S. Characteristics of auto-associative MLP as a novelty detector. In "International Joint Conference on Neural Networks - volume 5", IJCNN'99, pages 3086–3091 (1999).

[55] JAPKOWICZ N., HANSON S. J., AND GLUCK M. A. Nonlinear autoassociation is not equivalent to PCA. *Neural Computation* **12**(3), 531–545 (2000).

[56] JAPKOWICZ N., MYERS C., AND GLUCK M. A novelty detection approach to classification. In "Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1", IJCAI'95, pages 518–523. Morgan Kaufmann Publishers Inc. (1995).

[57] JOHN G., KOHAVI R., AND PFLEGER K. Irrelevant features and the subset selection problem. In "Proceedings of the eleventh international conference on machine learning", volume 129, pages 121–129. San Francisco (1994).

[58] JOHNSON W. B. AND LINDENSTRAUSS J. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics* **26**(189-206), 1 (1984).

[59] JOVIĆ A., BRKIĆ K., AND BOGUNOVIĆ N. A review of feature selection methods with applications. In "Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on", pages 1200–1205. IEEE (2015).

[60] JUSZCZAK P., TAX D. M., PE E., DUIN R. P., ET AL.. Minimum spanning tree based one-class classifier. *Neurocomputing* **72**(7), 1859–1869 (2009).

[61] KALKAN H. AND ÇETISLI B. Online feature selection and classification. In "Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on", pages 2124–2127. IEEE (2011).

[62] KATAKIS I., TSOUMAKAS G., AND VLAHAVAS I. Dynamic feature space and incremental feature selection for the classification of textual data streams. *Knowledge Discovery from Data Streams* pages 107–116 (2006).

[63] KEERTHIKA G. AND PRIYA D. S. Feature subset evaluation and classification using naive bayes classifier. *Journal of Network Communications and Emerging Technologies (JNCET) www. jncet. org* **1**(1) (2015).

[64] KENNEDY K., NAMEE B. M., AND DELANY S. J. Credit scoring: Solving the low default portfolio problem using one-class classification. In "Irish Conference on Artificial Intelligence and Cognitive Science", AICS'09, pages 168–177 (2009).

[65] KHAN S. S. AND MADDEN M. G. A survey of recent trends in one class classification. In "Irish conference on Artificial Intelligence and Cognitive Science", pages 188–197. Springer (2009).

[66] KHAN S. S. AND MADDEN M. G. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review* **29**(03), 345–374 (2014).

[67] KHOSHGOFTAAR T. M., GOLAWALA M., AND HULSE J. V. An empirical study of learning from imbalanced data using random forest. In "Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on", volume 2, pages 310–317. IEEE (2007).

[68] KODELL R. L., ZHANG C., SIEGEL E. R., AND NAGARAJAN R. Selective voting in convex-hull ensembles improves classification accuracy. *Artificial Intelligence in Medicine* **54**(3), 171–179 (2012).

[69] KOHONEN T. "Self-organizing maps". Springer-Verlag, Heidelberg, Germany (1995).

[70] KONONENKO I. Estimating attributes: analysis and extensions of relief. In "European conference on machine learning", pages 171–182. Springer (1994).

[71] KOTSIANTIS S. B., ZAHARAKIS I., AND PINTELAS P. Supervised machine learning: A review of classification techniques (2007).

[72] KRAMER M. A. Autoassociative neural networks. *Computers & Chemical Engineering* **16**(4), 313–328 (1992).

[73] KUNCHEVA L. I. "Combining pattern classifiers: methods and algorithms". John Wiley & Sons (2004).

[74] LEVI D. AND ULLMAN S. Learning to classify by ongoing feature selection. *Image and Vision Computing* **28**(4), 715–723 (2010).

[75] LI C., ZHANG Y., AND LI X. Ocvfdt: one-class very fast decision tree for one-class classification of data streams. In "Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data", pages 79–86. ACM (2009).

[76] LICHMAN M. UCI machine learning repository. *http://archive. ics. uci. edu/ml* (2013).

[77] LIU H. AND SETIONO R. Chi2: Feature selection and discretization of numeric attributes. In "Tools with Artificial Intelligence, 1995. Proceedings., Seventh International Conference on", pages 388–391. IEEE (1995).

[78] LIU H. AND MOTODA H. "Feature extraction, construction and selection: A data mining perspective", volume 453. Springer Science & Business Media (1998).

[79] LIU Z., LIU J., PAN C., AND WANG G. A novel geometric approach to binary classification based on scaled convex hulls. *Neural Networks, IEEE Transactions on* **20**(7), 1215–1220 (2009).

[80] MA H., WANG Z., WANG D., LIU D., YAN P., AND WEI Q. Neural-network-based distributed adaptive robust control for a class of nonlinear multiagent systems with time delays and external noises. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **46**(6), 750–758 (2016).

[81] MACQUEEN J. ET AL.. Some methods for classification and analysis of multivariate observations. In "Proceedings of the fifth Berkeley symposium on mathematical statistics and probability", volume 1, page 14. California, USA (1967).

[82] MANEVITZ L. M. AND YOUSEF M. One-class svms for document classification. *Journal of Machine Learning Research* **2**(Dec), 139–154 (2001).

[83] MAO Z., YUAN J., WU Z., QU J., AND LI H. Real-time compressive tracking based on online feature selection. In "Proceedings of International Conference on Computer Science and Information Technology", pages 431–438. Springer (2014).

[84] MARKOU M. AND SINGH S. Novelty detection: a review—part 1: statistical approaches. *Signal processing* **83**(12), 2481–2497 (2003).

[85] MARKOU M. AND SINGH S. Novelty detection: a review—part 2:: neural network based approaches. *Signal processing* **83**(12), 2499–2521 (2003).

[86] MARTÍNEZ-REGO D., FERNÁNDEZ-FRANCOS D., FONTENLA-ROMERO O., AND ALONSO-BETANZOS A. Stream change detection via passive-aggressive classification and bernoulli cusum. *Information Sciences* **305**, 130–145 (2015).

[87] MARTÍNEZ-REGO D., FONTENLA-ROMERO O., AND ALONSO-BETANZOS A. Power wind mill fault detection via one-class ν-svm vibration signal analysis. In "Neural Networks (IJCNN), The 2011 International Joint Conference on", pages 511–518. IEEE (2011).

[88] MAZHELIS O. One-class classifiers: a review and analysis of suitability in the context of mobile masquerader detection. *South African Computer Journal* **36**, 29–48 (2006).

[89] MEJÍA-LAVALLE M., SUCAR E., AND ARROYO G. Feature selection with a perceptron neural net. In "Proceedings of the international workshop on feature selection for data mining", pages 131–135 (2006).

[90] MINTER T. C. Single-class classification. In "Symposium on Machine Processing of Remotely Sensed Data", pages 2A12–2A15 (1975).

[91] MIRANDA V., CASTRO A. R. G., AND LIMA S. Diagnosing faults in power transformers with autoassociative neural networks and mean shift. *IEEE Transactions on Power Delivery* **27**(3), 1350–1357 (2012).

[92] MOLINA L. C., BELANCHE L., AND NEBOT À. Feature selection algorithms: A survey and experimental evaluation. In "Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on", pages 306–313. IEEE (2002).

[93] MØLLER M. F. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks* **6**(4), 525–533 (1993).

[94] MOYA M. R., KOCH M. W., AND HOSTETLER L. D. One-class classifier networks for target recognition applications. In "World Congress on Neural Networks", pages 797–801 (1993).

[95] NGUYEN T. M., WU Q., AND MUKHERJEE D. An online unsupervised feature selection and its application for background suppression. In "Computer and Robot Vision (CRV), 2015 12th Conference on", pages 161–168. IEEE (2015).

[96] PANG C., YAN J., AND VYATKIN V. Time-complemented event-driven architecture for distributed automation systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **45**(8), 1165–1177 (2015).

[97] PARZEN E. On estimation of a probability density function and mode. *Annals of Mathenatical Statistics* **3**, 1095–1076 (1962).

[98] PAUL J. Étude comparative de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Société Vaudoise des Sciences Naturelles* (1901).

[99] PERKINS S., LACKER K., AND THEILER J. Grafting: Fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research* **3**, 1333–1356 (2003).

[100] PERKINS S. AND THEILER J. Online feature selection using grafting. In "MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-", volume 20, page 592 (2003).

[101] PIMENTEL M. A., CLIFTON D. A., CLIFTON L., AND TARASSENKO L. A review of novelty detection. *Signal Processing* **99**, 215–249 (2014).

[102] POLIKAR R. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine* **6**(3), 21–45 (2006).

[103] PREPARATA F. P. AND SHAMOS M. "Computational geometry: an introduction". Springer Science & Business Media (2012).

[104] QUINLAN J. R. Induction of decision trees. *Machine learning* **1**(1), 81–106 (1986).

[105] REZNIK A. M. Non-iterative learning for neural networks. In "Neural Networks, 1999. IJCNN'99. International Joint Conference on", volume 2, pages 1374–1379. IEEE (1999).

[106] ROY A. Automated online feature selection and learning from high-dimensional streaming data using an ensemble of kohonen neurons. In "Neural Networks (IJCNN), 2015 International Joint Conference on", pages 1–8. IEEE (2015).

[107] RUANO A., KHOSRAVANI H. R., AND FERREIRA P. M. A randomized approximation convex hull algorithm for high dimensions. *IFAC-PapersOnLine* **48**(10), 123–128 (2015).

[108] RUMELHART D. E., DURBIN R., GOLDEN R., AND CHAUVIN Y. Backpropagation: The basic theory. *Backpropagation: Theory, Architectures and Applications* pages 1–34 (1995).

[109] SAKURADA M. AND YAIRI T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In "Proceedings of the 2nd Workshop on Machine Learning for Sensory Data Analysis", MLSDA'14, pages 4–11 (2014).

[110] SANZ J., PERERA R., AND HUERTA C. Fault diagnosis of rotating machinery based on auto-associative neural networks and wavelet transforms. *Journal of Sound and Vibration* **302**(4-5), 981–999 (2007).

[111] SCHADT E. E., LINDERMAN M. D., SORENSON J., LEE L., AND NOLAN G. P. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics* **11**(9), 647–657 (2010).

[112] SCHMIDT W. F., KRAAIJVELD M. A., AND DUIN R. P. A non-iterative method for training feedforward networks. In "Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on", volume 2, pages 19–24. IEEE (1991).

[113] SCHÖLKOPF B., PLATT J. C., SHAWE-TAYLOR J. C., SMOLA A. J., AND WILLIAMSON R. C. Neural computation. *The Knowledge Engineering Review* **13**(7), 1443–1471 (2001).

[114] SCHÖLKOPF B., SMOLA A., WILLIAMSON R., AND BARTLETT P. New support vector algorithms. *Neural Computation* **12**(5), 1207–1245 (2000).

[115] SCHÖLKOPF B., WILLIAMSON R. C., SMOLA A. J., SHAWE-TAYLOR J., AND PLATT J. C. Support vector method for novelty detection. In "Advances in Neural Information Processing Systems 12", NIPS'00, pages 582–588 (2000).

[116] SCHRAUDOLPH N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation* **14**(7), 1723–1738 (2002).

[117] SCHWENK H. The diabolo classifier. *Neural Computation* **10**(8), 2175–2200 (1998).

[118] SONNENBURG S., FRANC V., YOM-TOV E., AND SEBAG M. Pascal large scale learning challenge. In "25th International Conference on Machine Learning (ICML2008) Workshop. http://largescale. first. fraunhofer. de. J. Mach. Learn. Res", volume 10, pages 1937–1953 (2008).

[119] SONNENBURG S., RÄTSCH G., AND RIECK K. Large scale learning with string kernels. *Large Scale Kernel Machines* pages 73–103 (2007).

[120] SONTAG E. D. AND SUSSMANN H. J. Backpropagation can give rise to spurious local minima even for networks without hidden layers. *Complex Systems* **3**(1), 91–106 (1989).

[121] SOPHIAN A., TIAN G. Y., TAYLOR D., AND RUDLIN J. A feature extraction technique based on principal component analysis for pulsed eddy current ndt. *NDT & e International* **36**(1), 37–41 (2003).

[122] STRASSER T., ZOITL A., AND GRUVER W. A. Guest editorial: Special issue on industrial applications of distributed intelligent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **44**(3), 261–262 (2014).

[123] TAN S. C., TING K. M., AND LIU T. F. Fast anomaly detection for streaming data. In "IJCAI Proceedings-International Joint Conference on Artificial Intelligence", volume 22, page 1511 (2011).

[124] TANG M., ZHAO J.-Y., TONG R.-F., AND MANOCHA D. GPU accelerated convex hull computation. *Computers & Graphics* **36**(5), 498–506 (2012).

[125] TARASSENKO L., HAYTON P., AND BRADY M. Novelty detection for the identification of masses in mammograms. In "IEEE Conference on Artificial Neural Networks", volume 409, pages 442–447 (1995).

[126] TAX D. M. J. "One-class Classification". PhD thesis, Delft University of Technology (2001).

[127] TAX D. M. J. AND DUIN R. P. W. Machine learning. *The Knowledge Engineering Review* **54**, 45–66 (2004).

[128] TAX D. M. J. AND DUIN R. P. W. Support vector data description. *Machine Learning* **54**, 45–66 (2004).

[129] THOMPSON B. B., MARKS R. J., CHOI J. J., EL-SHARKAWI M. A., HUANG M.-Y., AND BUNJE C. Implicit learning in autoencoder novelty assessment. In "International Joint Conference on Neural Networks - volume 3", IJCNN'02, pages 2878–2883 (2002).

[130] TOU J. AND GONZÁLEZ R. "Pattern recognition principles". Addison-Wesley (1977).

[131] TSOUMAKAS G. AND VLAHAVAS I. Distributed data mining. *Encyclopedia of Data Warehousing and Mining* (2009).

[132] TSYMBAL A. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin* (2004).

[133] VAKHARIA V., GUPTA V., AND KANKAR P. Ball bearing fault diagnosis using supervised and unsupervised machine learning methods. *EDITORIAL OFFICE* page 244 (2015).

[134] VENTURA D. AND MARTINEZ T. An empirical comparison of discretization methods. In "Proceedings of the Tenth International Symposium on Computer and Information Sciences", pages 443–450 (1995).

[135] VISHWAKARMA V. P. A non-iterative learning based artificial neural network classifier for face recognition under varying illuminations. In "International Conference on Contemporary Computing", pages 383–394. Springer (2012).

[136] WANG H., FAN W., YU P., AND HAN J. Mining concept-drifting data streams using ensemble classifiers. In "Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining", pages 226–235. ACM (2003).

[137] WANG J., ZHAO P., HOI S. C., AND JIN R. Online feature selection and its applications. *Knowledge and Data Engineering, IEEE Transactions on* **26**(3), 698–710 (2014).

[138] WANG Y., YAO H., AND ZHAO S. Auto-encoder based dimensionality reduction. *Neurocomputing* **184**, 232–242 (2016).

[139] WEISS S. AND KULIKOWSKI C. Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning and exp. (1990).

[140] WOLFE D. AND HOLLANDER M. Nonparametric statistical methods. *Nonparametric statistical methods* (1973).

[141] WU X., YU K., WANG H., AND DING W. Online streaming feature selection. In "Proceedings of the 27nd International Conference on Machine Learning" (2010).

[142] WU X., KUMAR V., QUINLAN J. R., GHOSH J., YANG Q., MOTODA H., MCLACHLAN G. J., NG A., LIU B., PHILIP S. Y., ET AL.. Top 10 algorithms in data mining. *Knowledge and Information Systems* **14**(1), 1–37 (2008).

[143] WU X., ZHU X., WU G.-Q., AND DING W. Data mining with big data. *ieee transactions on knowledge and data engineering* **26**(1), 97–107 (2014).

[144] YAMANISHI K., TAKEUCHI J.-I., WILLIAMS G., AND MILNE P. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In "Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining", pages 320–324. ACM (2000).

[145] YPMA A. AND DUIN R. Support objects for domain approximation. In "International Conference on Artificial Neural Networks", ICANN'98, pages 719–724 (1998).

[146] ZAKI M. J. AND HO C.-T. "Large-scale parallel data mining". Number 1759. Springer Science & Business Media (2000).

[147] ZENG M., YANG Y., ZHENG J., AND CHENG J. Maximum margin classification based on flexible convex hulls for fault diagnosis of roller bearings. *Mechanical Systems and Signal Processing* **66**, 533–545 (2016).

[148] ZHANG C., RUAN J., AND TAN Y. An incremental feature subset selection algorithm based on boolean matrix in decision system. *Convergence Information Technology* pages 16–23 (2011).

[149] ZHAO Z. A. AND LIU H. "Spectral feature selection for data mining". CRC Press (2011).

[150] ZHOU X., ZHONG Y., AND CAI L. Anomaly detection from distributed flight record data for aircraft health management. In "Computational and Information Sciences (ICCIS), 2010 International Conference on", pages 156–159. IEEE (2010).