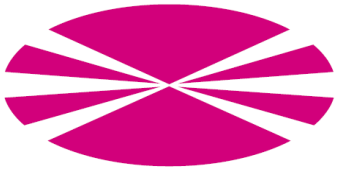


UNIVERSITY OF A CORUÑA

FACULTY OF INFORMATICS

Department of Computer Science

Ph.D. Thesis

New scalable machine learning methods: Beyond classification and regression

Author: Carlos Eiras-Franco
Advisors: Amparo Alonso Betanzos
Bertha Guijarro Berdiñas
Antonio Bahamonde

A Coruña, October 2019

6 de noviembre de 2019
UNIVERSITY OF A CORUÑA

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

"The strength of the team is each individual member. The strength of each member is the team." – Phil Jackson

Acknowledgements

Working on this thesis has really made apparent for me how much of a collaborative effort research is. Consider this page to be a non-exhaustive list of people that helped me on this effort.

Five years ago, after spending several years working as a software developer, I realized that what I most enjoyed was searching for solutions to difficult problems and started to consider pursuing a career in research. My first contacts with the university were somewhat discouraging, given the scarcity of funding for scientific research in Spain. However, when I knocked on Amparo's door it flung wide open and I received nothing but encouragement and help. The LIDIA research group has been incredibly welcoming and is the most nurturing place I could hope for my research. My sincerest gratitude must be, consequently, first shown to the LIDIA group and Amparo as its leader.

Possibly the most important factor in completing this work successfully is the amazing job done by my supervisors Amparo, Bertha and Antonio. I was lucky enough to have not one but three extremely accomplished and experienced supervisors that steered my efforts in the right direction, so they deserve a great deal of the credit for this work. They always found time in their very busy lives to work on this thesis and did so while making me feel welcome and appreciated, for which I am immensely grateful.

In this five years I have had the good fortune of working alongside very talented colleagues that ended up being co-authors of some of the papers that originated from this thesis. In no particular order, Verónica, David, Sabela, Jorge, Juan, Leslie and César offered their expertise and work which helped make this thesis much better. I thank all of them for their effort. I have encountered many other talented colleagues in this time that have enriched my understanding of the field with their ideas. In this category, I would like to thank my fellow LIDIA members who not only shared their knowledge but also made the non-research hours much more entertaining.

This work also greatly benefited from my two research stays abroad. I would like to thank Professor John Shawe-Taylor for being so kind to invite me to the University College of London for a month early in this endeavour. David and Leslie were kind enough to get me up to speed with part of their research and they were also very welcoming, for which I thank them. Later on, Professor Paulo Novais was also very generous for offering me to stay at the ISLab of the Universidade do Minho for two months. I thank him and Tiago and Marco for going out of their way to make my stay more pleasant and particularly Marco for the effort put in the research that we conducted at the moment which, although it unfortunately did not yield any publishable results, helped me better understand the problem at hand.

On the technical side, the nature of the research presented in this thesis required many computational resources. I have to thank both the Centro de Supercomputación de Galicia (CESGA) and Pluton Cluster of the Computer Architecture Group of the Universidade da Coruña and their respective staffs for the assistance they offered.

I would also like to thank my family. My parents and my brother instilled in me a love for knowledge and a work ethic that were instrumental in this endeavor and their support has been unwavering. Finally, my lovely wife has always shared my love for science and has been by my side every step of the way in this task, as in life. The love, support and joy that she and my son bring to my life help make the most difficult problems much more solvable and I can't thank them enough.

Carlos Eiras Franco

Abstract

The recent surge in data available has spawned a new and promising age of machine learning. Success cases of machine learning are arriving at an increasing rate as some algorithms are able to leverage immense amounts of data to produce great complicated predictions. Still, many algorithms in the toolbox of the machine learning practitioner have been render useless in this new scenario due to the complications associated with large-scale learning. Handling large datasets entails logistical problems, limits the computational and spatial complexity of the used algorithms, favours methods with few or no hyperparameters to be configured and exhibits specific characteristics that complicate learning. This thesis is centered on the scalability of machine learning algorithms, that is, their capacity to maintain their effectivity as the scale of the data grows, and how it can be improved. We focus on problems for which the existing solutions struggle when the scale grows. Therefore, we skip classification and regression problems and focus on feature selection, anomaly detection, graph construction and explainable machine learning. We analyze four different strategies to obtain scalable algorithms. First, we explore distributed computation, which is used in all of the presented algorithms. Besides this technique, we also examine the use of approximate models to speed up computations, the design of new models that take advantage of a characteristic of the input data to simplify training and the enhancement of simple models to enable them to manage large-scale learning. We have implemented four new algorithms and six versions of existing ones that tackle the mentioned problems and for each one we report experimental results that show both their validity in comparison with competing methods and their capacity to scale to large datasets. All the presented algorithms have been made available for download and are being published in journals to enable practitioners and researchers to use them.

Resumen

El reciente aumento de la cantidad de datos disponibles ha dado lugar a una nueva y prometedora era del aprendizaje máquina. Los éxitos en este campo se están sucediendo a un ritmo cada vez mayor gracias a la capacidad de algunos algoritmos de aprovechar inmensas cantidades de datos para producir predicciones difíciles y muy certeras. Sin embargo, muchos de los algoritmos hasta ahora disponibles para los científicos de datos han perdido su efectividad en este nuevo escenario debido a las complicaciones asociadas al aprendizaje a gran escala. Trabajar con grandes conjuntos de datos conlleva problemas logísticos, limita la complejidad computacional y espacial de los algoritmos utilizados, favorece los métodos con pocos o ningún hiperparámetro a configurar y muestra complicaciones específicas que dificultan el aprendizaje. Esta tesis se centra en la escalabilidad de los algoritmos de aprendizaje máquina, es decir, en su capacidad de mantener su efectividad a medida que la escala del conjunto de datos aumenta. Ponemos el foco en problemas cuyas soluciones actuales tienen problemas al aumentar la escala. Por tanto, obviando la clasificación y la regresión, nos centramos en la selección de características, detección de anomalías, construcción de grafos y en el aprendizaje máquina explicable. Analizamos cuatro estrategias diferentes para obtener algoritmos escalables. En primer lugar, exploramos la computación distribuida, que es utilizada en todos los algoritmos presentados. Además de esta técnica, también examinamos el uso de modelos aproximados para acelerar los cálculos, el diseño de modelos que aprovechan una particularidad de los datos de entrada para simplificar el entrenamiento y la potenciación de modelos simples para adecuarlos al aprendizaje a gran escala. Hemos implementado cuatro nuevos algoritmos y seis versiones de algoritmos existentes que tratan los problemas mencionados y para cada uno de ellos detallamos resultados experimentales que muestran tanto su validez en comparación con los métodos previamente disponibles como su capacidad para escalar a grandes conjuntos de datos. Todos los algoritmos presentados han sido puestos a disposición del lector para su descarga y se han difundido mediante publicaciones en revistas científicas para facilitar que tanto investigadores como científicos de datos puedan conocerlos y utilizarlos.

Resumo

O recente aumento na cantidade de datos dispoñibles deu lugar a unha nova e prometedora era no aprendizaxe máquina. Os éxitos neste eido estanse a suceder a un ritmo cada vez maior gracias a capacidade dalgúns algoritmos de aproveitar inmensas cantidades de datos para producir predicciónss difíciles e moi acertadas. Non obstante, moitos dos algoritmos ata agora dispoñibles para os científicos de datos perderon a súa efectividade neste novo escenario por mor das complicacións asociadas ao aprendizaxe a grande escala. Traballar con grandes conxuntos de datos leva consigo problemas loxísticos, limita a complexidade computacional e espacial dos algoritmos empregados, favorece os métodos con poucos ou ningún hiperparámetro a configurar e ten complicacións específicas que dificultan o aprendizaxe. Esta tese céntrase na escalabilidade dos algoritmos de aprendizaxe máquina, é dicir, na súa capacidade de manter a súa efectividade a medida que a escala do conxunto de datos aumenta. Tratamos problemas para os que as solucións dispoñibles teñen problemas cando crece a escala. Polo tanto, deixando no canto a clasificación e a regresión, centrámonos na selección de características, detección de anomalías, construcción de grafos e no aprendizaxe máquina explicable. Analizamos catro estratexias diferentes para obter algoritmos escalables. En primeiro lugar, exploramos a computación distribuída, que empregamos en tódolos algoritmos presentados. Ademáis desta técnica, tamén examinamos o uso de modelos aproximados para acelerar os cálculos, o deseño de modelos que aproveitan unha particularidade dos datos de entrada para simplificar o adestramento e a potenciación de modelos sinxelos para axeitalos ao aprendizaxe a gran escala. Implementamos catro novos algoritmos e seis versións de algoritmos existentes que tratan os problemas mencionados e para cada un deles expoñemos resultados experimentais que mostran tanto a súa validez en comparación cos métodos previamente dispoñibles como a súa capacidade para escalar a grandes conxuntos de datos. Tódolos algoritmos presentados foron postos a disposición do lector para a súa descarga e difundíronse mediante publicacións en revistas científicas para facilitar que tanto investigadores como científicos de datos poidan coñecelos e empregalos.

Contents

1. Introduction **1**

2. Parallel computation: Multithreaded and Spark parallelization of feature selection filters **7**

2.1. Introduction 7
2.2. Feature selection 9
2.2.1. Feature selection methods 10
2.3. Distributed computing approaches 12
2.3.1. Multithreaded processing 12
2.3.2. Parallelization with Apache Spark 12
2.4. Implemented algorithms 14
2.4.1. ReliefF algorithm 14
2.4.2. InfoGain algorithm 16
2.4.3. CFS algorithm 18
2.4.4. SVM-RFE algorithm 22
2.5. Experimental results 24
2.5.1. On the preprocessing of the datasets: Parallelization of a discretization algorithm 26
2.5.2. Analysis of the ReliefF implementations 28
2.5.3. Analysis of the InfoGain implementations 30
2.5.4. Analysis of the CFS implementations 32
2.5.5. Analysis of the SVM-RFE implementations 32
2.6. Conclusions 33

3. Approximate models: Scalable kNN Graph construction with Locality Sensitive Hashing **35**

3.1. Introduction 35
3.2. Related work 36
3.2.1. kNNG using Locality-Sensitive Hashing 37
3.3. Implementing the Algorithm 39
3.3.1. Hyperparameter tuning 42
3.3.1.1. Resolution 42

3.3.1.2. Hyperparameters for Euclidean distance as a similarity measure . . . 43
3.3.1.3. C_{MAX} and *desiredSize* . . . 44
3.4. Experimental design and Results . . . 46
3.4.1. Handling C_{MAX} and *desiredSize* . . . 47
3.4.2. Performance of the method . . . 50
3.4.3. Scalability of the method . . . 53
3.5. Conclusions . . . 55

4. Approximate FS: Scalable feature selection using ReliefF aided by Locality Sensitive Hashing 57
4.1. Introduction . . . 57
4.2. Related work . . . 58
4.3. Proposed algorithm . . . 59
4.4. Experimental settings . . . 62
4.4.1. Equipment and datasets . . . 63
4.4.2. Methodology . . . 64
4.5. Experimental results . . . 66
4.5.1. Regression and binary classification . . . 66
4.5.2. Multiclass datasets . . . 67
4.5.3. Scalability . . . 70
4.6. Conclusions . . . 70

5. Ad-hoc models: Large Scale Anomaly Detection in Mixed Numerical and Categorical Input Spaces 73
5.1. Introduction . . . 73
5.2. Related work . . . 75
5.3. Basic formulation . . . 77
5.3.1. Numerical part . . . 78
5.3.2. Categorical part . . . 79
5.4. Maximum likelihood parameter estimation . . . 79
5.5. Experimental settings and results . . . 80
5.5.1. Methodology . . . 81
5.5.1.1. Real datasets . . . 83
5.5.1.2. Synthetic dataset generator . . . 84
5.5.2. Results and discussion . . . 86
5.6. Conclusions . . . 91

6. Speculative computation: Explaining large-scale dyadic data 95

6.1. Introduction 95
6.2. Related work 97
6.3. Definitions 99
6.4. Proposed algorithm 101
6.5. Experimental setup 102
6.5.1. Dataset transformation 105
6.6. Results 110
6.6.1. Effect of the λ hyperparameter 110
6.6.2. Suitability of the method 110
6.6.3. Analysis of the explanations 112
6.6.4. Scalability of the method 113
6.7. Conclusions 115

7. Conclusions and future work **117**

I. Best hyperparameters for anomaly detection methods **121**

II. Additional results of the scalable dyadic data explanator **127**

III. Publications supporting this thesis **131**

Bibliography **133**

CHAPTER 1

Introduction

Recent years have seen the production of data increase at breakneck pace. Fueled by the low price of storage and sensors, the deployment of the Internet of Things [9], the ubiquity of smart devices and the sensorization of many industrial activities all contribute to the production of data in volumes previously unseen. The size of the datasets being generated, stored and analyzed has been steadily growing. Taking the datasets posted in the popular LibSVM Database [37] as a reference, their size has increased five hundredfold.

This phenomenon has sparked the interest in machine learning. The goal of this field is the development of methods and algorithms to transform raw data into useful insights by identifying patterns and using inference [19]. Machine learning algorithms rely exclusively in data to learn to perform a task and require no knowledge to be explicitly coded. Historically, the capacity of these methods to learn complex tasks was limited by the scarcity of data. Thanks to the mentioned increase in the data available, the bottleneck has shifted to the capacity of algorithms, that is, the complexity of the learned tasks is now limited by the ability of the machine learning method to extract the relevant patterns [25]. The capacity of algorithms to gracefully handle a growing amount of work is called scalability [24]. Increasing it allows the learning of more complex tasks.

Several success cases show that this abundance of data is an opportunity of which machine learning can take great advantage. An spectacular example is the case of neural networks, particularly of learning deep artificial neural network models or Deep Learning (DL) for short. These models had been studied for decades but only gained great popularity in the last decade, when the increase in the size of datasets allowed them to achieve state-of-the-art and even superhuman performance in a wide range of problems. It has been estimated that a DL model requires training on 5,000 labeled samples per category to achieve acceptable performance and that it can only reach human-level performance when the dataset reaches 10 million examples [98]. Having

abundant data to train on has allowed the once minority field of DL to achieve remarkable success in computer vision, machine translation, speech recognition and other important problems.

However, the opportunity that large datasets bring comes associated with a host of problems that complicate the use of machine learning algorithms. As a result, many popular techniques are rendered useless in the context of large-scale learning. Several factors can contribute to this effect:

1. Algorithms with a **high computational complexity** can require impractical execution times to process a large dataset. In the case of datasets with a very large number n of examples, the computational complexity of an algorithm should not exceed $\mathcal{O}(n)$. Conversely, if the data has a large number d of variables, then complexity should be $\mathcal{O}(d)$ or below. Algorithms with a higher complexity require using mitigation techniques that are not feasible in every case.

2. Similarly, a **high spatial complexity** can render an algorithm useless in the context of large-scale learning. The same rule of thumb as in the case of computational complexity applies to find the maximum spatial complexity that is manageable in such contexts. Again, algorithms that have a higher complexity demand memory at such a rate that special measures need to be applied, although they are not always available.

3. Moreover, a well-known complication with datasets that have many variables is the ***curse of dimensionality***. This term was coined by Bellman [16] to refer to the difficulty to optimize in high-dimensional spaces due to the impossibility of exhaustive enumeration, the mentioned computational and spatial demands and structural problems like the concentration of distances [157].

4. Simply handling large datasets poses **logistical problems** that need to be addressed in order to apply a machine learning method. Storing a large amount of data in a manner that allows quick access constitutes a challenge.

5. Finally, algorithms that have **many hyperparameters** to be tuned require several cross-validated training steps to obtain the best combination of hyperparameters. In the context of large-scale learning, performing each of those training steps can be costly, forcing the practitioner to choose between an expensive well-tuned accurate result and a more inaccurate inexpensive one.

The machine learning literature contains many efforts to address these problems individually, although there is no silver bullet that solves all of them. In particular, numerous techniques can be used when dealing with high-dimensional datasets, which can have a large number of examples, many attributes describing each element or both characteristics. In the specific case of datasets with a large number of variables it is advisable to apply dimensionality reduction techniques to improve the performance of learning methods. The options available for data scientists facing this situation include feature extraction (FE), that transforms the input set of variables in a new, smaller set in which each attribute is the result of applying a function to various input variables, or feature selection (FS), that consists in obtaining a subset of features that describes the problem properly by discarding irrelevant or redundant variables from the input variable set [66]. Having a reduced set of variables facilitates the comprehension of the dataset and can also improve the effectiveness of learning methods applied to it. However, it is important to note that, paradoxically, many of the most popular FE and FS techniques suffer from some of the problems mentioned above and can not be applied to large-scale datasets.

The increase in the size of datasets has, therefore, created a environment in which the algorithms that are less affected by the associated problems and more able to take advantage of data abundance have thrived, while other popular and effective algorithms have been relegated to handling small datasets. In particular, models with parameters that can be optimized using Gradient Descent (such as DL models) generally show good scalability, and increasing the complexity of these models manages to learn a more complex task given enough data and provided that convergence is obtained in a reasonable time [26]. Besides those, tree based models are also popular in large-scale learning since they can be processed in parallel, increasing the speed of computation. Many other approaches that were popular and successful in small-scale learning have struggled to gracefully scale to the size of current datasets.

The goal of this thesis is to explore solutions to the mentioned problems associated with large-scale datasets that increase the scalability of machine learning algorithms which in their original form can not handle those large datasets. To achieve that, we explore several complementary approaches that alleviate one or various of the mentioned problems. For each of these approaches we have either studied a specific algorithm and obtained a more scalable version or proposed a novel algorithm that leverages the described solution. Namely, we have explored:

1. **Parallel execution** of operations that are independent. This is arguably the most widespread and successful technique for handling large datasets. It consists in finding operations in the learning process that have no dependencies and executing them in parallel in several computational units to speed up the computation. A particular case of this approach is the use of GPUs to perform several matrix operations in a single step, which is essential to the scalability of DL methods. However, this is restricted to methods that rely heavily in matrix operations. In this thesis we explore the more general idea of using computer clusters to distribute the computational load across multiple computers. This has the added benefit that, in many cases, the storage demands can also be shared among the involved machines. Chapter 2 introduces the distributed computation framework Apache Spark and explores its use to provide scalability to popular feature selection algorithms, comparing it to using a single machine with several computing threads for the same task. All of the methods presented in this thesis are designed so that their independent operations can be executed in parallel using distributed computation to increase their scalability.

2. The use of **approximate models** to alleviate the computational load that some algorithms require when the data is very numerous. This approach originates from the observation that obtaining an approximate solution often achieves results comparable to the exact solution, while requiring significantly less effort. In Chapter 3 we detail this line of action and we present a method that uses *Locality Sensitive Hashing* to obtain an approximate *k nearest neighbors graph*, a data structure used in many machine learning problems. In Chapter 4 we propose an adaptation of the popular ReliefF feature selection algorithm that uses that approximate k nearest neighbors graph instead of the exact one and we report experiments which highlight that using an approximation does not significantly affect accuracy while greatly improving scalability.

3. Designing ***ad-hoc* models** that take advantage of a characteristic of the input data to simplify learning, which can lead to efficient algorithms for that specific type of data. In Chapter 5 we present an anomaly detection method that works on input data that has both numeric and categorical variables. The proposed method leverages this characteristic to compose two simple models into a more complex one that can be learned much more easily.

4. The **enhancement of simple models** with speculative computation enabled by the parallel execution provided with distributed computation. This idea is explored in Chapter 6 were we present a method to obtain an explanation of

the relationships encoded in a dyadic dataset by building a modified decision tree. The accuracy of the obtained model is increased by speculatively computing many versions of the decision tree in parallel using distributed computation, effectively exploring a larger fraction of the solution space while requiring the same computational time.

In addition to exploring several lines of action to mitigate the problems derived from large datasets, we tried to obtain algorithms that tackle problems that have few solutions in the context of large-scale learning. Specifically, we present adapted versions of several feature selection algorithms to address the shortage of implementations of feature selection methods in large-scale computation platforms. Moreover, we introduce a novel anomaly detection algorithm in mixed numerical-categorical input spaces to provide an scalable algorithm for a problem that has few solutions that work with large datasets. Finally, we explore the nascent field of Explainable Artificial Intelligence (XAI) which aims to solve one of the main problems with the complex models needed to learn from large datasets: the difficulty that human supervisors encounter when trying to understand the rationale for the outputs and the information encoded in such models.

CHAPTER 2

Parallel computation: Multithreaded and Spark parallelization of feature selection filters

2.1. Introduction

The ability to collaborate and organize large groups of individuals to accomplish a task that would be unachievable individually is one of the trademarks of the human species. Perhaps because of that, when we are faced with a large endeavor one of our first instincts is to try and split it into simpler tasks that can be shared among several participants. This gives rises to all sorts of organization and synchronization problems, which are greatly simplified when the simple tasks originated can be performed independently; it is in those cases that this approach is more suited. Hence, it is no surprise that, when dealing with large-scale machine learning, data scientists quickly resorted to this strategy.

Parallel computing is arguably the single most effective line of action for handling machine learning at large scale. Computer hardware has long been capable of performing several calculations simultaneously, with CPUs packing up to 32 computing cores that can operate in parallel. Also, methods that rely on the optimization of parameters involved in matrix operations are very well suited to computation on special hardware called GPUs. These computational units were initially designed to perform the matrix operations needed to generate high quality computer graphics, but were later repurposed to deal with the matrix operations necessary in machine learning model optimization. Their effectiveness is behind every success case of deep learning, as well as many other machine learning methods. However, in this thesis we will focus on another approach for obtaining parallel computation: distributed computing. It can be used in conjunction with GPU learning and is also behind every major success in machine learning in recent years. Distributed computing consists of dividing the work at hand across multiple computational units, which can be processor cores on a single

machine or the processors of several machines in a computer cluster. Distributing the computation accelerates the response process and spreads the storage load. This approach gained significant traction with the introduction of the Map Reduce paradigm [46], an abstraction presented by Google in 2008 that facilitates the distribution of computations as long as they conform to two very general types of processing, namely, Map and Reduce operations. An open-source implementation of this idea was soon launched under the name Apache Hadoop [68]. This software platform enables the repurposing of various units of consumer hardware into computer clusters that can process vast amounts of data. Later on, more specialised frameworks were developed, among which Apache Spark [153] is probably the most popular. It was developed with the objective of maintaining reusable data in memory for as long as possible and providing a flexible programmer API. The success of these frameworks and its suitability for data science led to the creation of powerful libraries such as Mahout [111] for Hadoop and MLLib [114] for Spark, that contain distributed implementations of machine learning algorithms.

However, although these libraries contain a wide variety of machine learning algorithms, they are still lacking in certain aspects. The gradual increase in the dimensions of datasets has spawned a number of techniques designed to deal with such data. This dimensionality can refer to samples, features or both. In the case in which we confront with datasets containing numerous features, feature selection techniques are mandatory. Feature selection consists in the process of determining the relevant features and trying to remove as much irrelevant and redundant information as possible, without leading to a degradation in classification performance.

The go-to solution for many data scientists when performing feature selection is the Weka (Waikato Environment for Knowledge Analysis) suite [69], which has been downloaded over six million times. It can be used as a stand-alone application or imported as a library from the user's code. Feature selection is among its functionalities with several algorithms available to the user. This ample range of algorithms included in Weka makes its use widespread among data scientists for data analysis and for the development and testing of new algorithms. In addition, the fact that Weka runs on Java and is designed with single-machine setups in mind, makes it very suitable for the average user. Nevertheless, some of the implementations in Weka struggle when processing large datasets, requiring very long execution times, effectively limiting the size of the datasets that can be analyzed with it. An improvement in the time efficiency of these algorithms will enable its many users to process large datasets that up to now were out of reach for these implementations.

As mentioned above, Spark is designed for distributed computing and can achieve great performance processing large amounts of data, but few implementations of feature selection algorithms are available. Moreover, to be able to use the Mahout or MLlib libraries, the user needs to have a Hadoop or Spark installation and, although they can run on single-machine environments, a cluster of computers would be needed to fully exploit these libraries, which is not always available for regular users. A more viable solution for these users is the use single-machine software such as Weka.

In this work, which was published in the Journal of Computational Science [54], we will present new implementations of four popular feature selection algorithms and a discretization algorithm that are able to tackle sizable problems in different environments. We will also compare two alternatives for parallel execution and find out the suitability of these implementations to different amounts of computing resources[1]. To this end, multithreaded implementations for Weka and distributed versions in Spark will be proposed. This will allow users to analyze larger datasets in shorter times and choose the most adequate implementation for the resources available to them.

This chapter is organized as follows: Sections 2.2 and 2.3 are an overview of feature selection and parallelization approaches respectively. Section 2.4 describes the algorithms that are the object of this chapter. The results of our tests are presented in Section 2.5 and in Section 2.6 we discuss our conclusions.

2.2. Feature selection

Feature selection is the name given to the process that analyzes a dataset, detects relevant features and discards those that are redundant or irrelevant. The goal of this technique is to obtain a subset of features that has minimum degradation of performance when used by a classifier while describing the given problem properly. It simplifies the dataset both in size and in complexity of understanding [23], which leads to simpler and faster classification algorithms, better problem comprehension and reduced storage requirements.

[1]The implementations can be downloaded from http://www.lidiagroup.org/index.php/en/materials-en.html

2.2.1. Feature selection methods

Feature selection methods can be classified into two categories: individual evaluators or subset evaluators. Individual evaluators are also called rankers and they assign a weight to each attribute that represents its relevance. Subset evaluators, on the contrary, employ a search strategy to determine a candidate subset of features and have the advantage of removing redundant attributes at the cost of being more complex. According to the relationship with the learning method used, feature selection methods can also be divided as follows [65]:

- **Filters** are methods that are applied independently of the induction process. They are, in general, computationally inexpensive.
- **Wrappers** use the induction algorithm as a black box to evaluate the fitness of each candidate subset. This results in algorithms that are computationally demanding but more accurate.
- **Embedded methods** perform feature selection in the process of training and are typically specific to given learning algorithms.

In this work, three of the most commonly used filter methods (InfoGain, ReliefF and CFS) and an embedded method (SVM-RFE) were selected for reimplementation using a parallel approach. The first two filters are rankers that return an order for the features to be discarded below a threshold of the user's choice and they are included in the Weka suite:

- **Information Gain (InfoGain)** [125] is a filter that computes the mutual information of the different features with respect to the class and provides an ordered ranking of all the features according to this value.
- **ReliefF** [89] is a heuristic estimator built upon the Relief algorithm [88] that deals efficiently with noisy and incomplete datasets and with multiclass problems. It works by locating the nearest neighbors for each instance from the same and opposite class and updating the weights of each feature accordingly.

The remaining two algorithms are subset evaluators. To perform feature selection

Table 2.1: Theoretical computational complexity of the four feature selection methods focus of this work (where n is the number of examples and d is the number of attributes)

Method	Complexity
InfoGain	dn
ReliefF	dn^2
CFS	d^2n
SVM-RFE	$max(d,n)n^2$

they search through the space of all possible attribute combinations for the set that offers a better score according to a heuristic method that depends on the algorithm.

- **CFS** [70] is a subset evaluator independent from the induction process that tries to identify correlations between attributes and the class.

- **SVM-RFE** [67], which stands for Support Vector Machine Recursive Feature Elimination is an embedded method that filters the attributes iteratively using a SVM at each stage to rank them.

The choice of these algorithms was made to obtain a set of tools that are well suited to a wide range of datasets. CFS and InfoGain perform well when the data has a large number of attributes when compared to the number of instances and are very fast, but they do not perform as well when there is noise in the inputs. ReliefF is very good at eliminating redundant and correlated features, even when there is noise in the inputs and attributes are non-linear, but it is much slower and does not perform well when few examples are available. Lastly, SVM-RFE detects correlation and redundancy even with few examples, but it performs poorly when there is noise in the inputs and is very time consuming [23].

Table 2.1 shows the theoretical computational complexity of the four methods described above.

2.3. Distributed computing approaches

The main purpose of this work is to parallelize the standard implementations of ReliefF, InfoGain, CFS and SVM-RFE. In order to empower Weka users, multithreaded implementations are proposed. Furthermore, to enable users that can access computational clusters, we developed and tested Spark versions of the algorithms.

2.3.1. Multithreaded processing

Multithreading allows users to take advantage of multicore systems without imposing the overhead of creating multiple processes and providing direct access to a common address space. However the creation and management of threads introduces a computational overhead that makes the use of threads suboptimal when the tasks parallelized have low complexity.

Java provides parallel programming support in the core of the language. This feature enables programmers to write code that exploits multithreading without the need to use any external libraries. Since Weka is written in Java, we use this support to implement our multithreaded parallel version. We divide the feature selection algorithms in tasks that can be performed in parallel, which allows us to exploit the computational power of multicore machines.

2.3.2. Parallelization with Apache Spark

To alleviate the difficulties of developing distributed programs, a team of Google engineers developed the MapReduce framework [46] that handles the common aspects of distributed programs, providing the programmer with a tool to run parallel programs and handle large files without having to worry about anything but the implementation of the algorithm.

The programming paradigm introduced by MapReduce requires the tasks to be divided in two separate steps: the Map phase, that applies a function given by the user to every element; and the Reduce phase, that combines the resulting values. Oftentimes elements consist of key-value pairs and the Reduce phase merges results that have the

same key, although this is not mandatory. The abstraction resulting of decomposing a job in simple Map and Reduce functions allows the framework to divide both data and code across the computing nodes, a task performed by a master node. Typically, the framework splits the data in as many chunks as nodes are available and distributes it among them so that each node can apply the Map function to the assigned elements. The results are then rearranged by the master node, using a key partitioning scheme, and distributed again back to the nodes so that they perform the Reduce phase.

MapReduce was implemented in the open-source framework Hadoop [68] and rapidly achieved great popularity for its reliability and scalability. Still, this direct implementation left room for an important improvement that was later implemented by the Spark [153] framework: the transition between the Map and Reduce phase requires data to be shuffled by the master node and redistributed to the nodes, in a time-consuming process that is unnecessary when several Map transformations need to be applied before the Reduce phase or in iterative algorithms. By avoiding unneeded data movement and introducing other optimizations Spark performs several times faster than Hadoop for certain applications [153].

Spark allows the programmer to manage work distribution by means of using Resilient Distributed Datasets (RDDs), an abstraction that represents a read-only set of objects that is distributed across multiple machines. RDDs can be transformed, performing an operation on each element, which can be done in parallel in each node, and they can be reduced, combining elements to obtain a result. Only this step requires that the whole dataset is shuffled and redistributed to the nodes in a time-consuming process. Additionally, data can be sent to the nodes to work with by using broadcast variables, and the worker nodes can write increments to special variables named accumulators.

We decomposed the feature selection algorithms in independent tasks to obtain a Spark implementation that will allow the user to take advantage of a computer cluster to process large datasets in reduced time.

Table 2.2: Summary of algorithms in this work

Algorithm	Multithreaded Weka	Spark implementation
ReliefF	New implementation	New implementation
InfoGain	New implementation	Available in Spark packages
CFS	Included in Weka	New implementation
SVM-RFE	New implementation	New implementation

2.4. Implemented algorithms

Four algorithms (listed in Table 2.2) were the object of this work. Of the 8 possible implementations (a Weka multithreaded and a Spark version for each algorithm), 2 were already available and 6 were developed as part of this work.

2.4.1. ReliefF algorithm

The original ReliefF algorithm [89] loops through a set of instances $\mathcal{D}$ finding for each instance I its k nearest neighbors from the same class, called nearest hits H, and the k nearest neighbors from each different class, which are denoted as nearest misses M_c. When all neighbors are found, the weight for each attribute $W[a]$ is updated by subtracting the weighted average distance (computed with the DIFF function, that returns the Manhattan distance between two instances) of each hit H and adding, for each class c other than that of I, the weighted average of the distance to each miss M_c. When computing averages, distances are weighted by the probability P of the class and divided by the total number of instances n.

Regarding the multithreaded implementation, the job is divided into as many tasks as threads we want to use, then a thread is created for each task. This approach avoids the need for a thread pooler to manage the execution of threads. This process is detailed in Algorithm 1.

The process of finding the nearest neighbors for each instance (by means of the loop

described between Lines 2 and 7 of Algorithm 1) is very time consuming since it requires comparing it with all other instances. This search can be executed independently for each instance and therefore it can be performed in parallel with no synchronization issues.

Algorithm 1: Pseudo-code for multithreaded ReliefF

Input: $\mathcal{D}$ ←Set of instances with attributes $\mathcal{A}$ classified in classes $\mathcal{C}$

Output: W ← vector storing the weight of each attribute

1 set all weights **W[A]** ← 0

2 **for** $i \leftarrow$ *1* ***to*** *THREADS_AVAILABLE* **do in parallel**

3 $D_i \leftarrow$ disjoint subset of instances

4 **foreach** I ***in*** D_i **do**

5 $I.H \leftarrow$ findKNearestHitsIn(D_i)

6 **foreach** $c \in \mathcal{C} \ / \ c \neq I.class$ **do**

7 $I.M_c \leftarrow$ findKNearestMissesOfClassIn(c, D_i)

end

end

end

8 **foreach** a ***in*** $\mathcal{A}$ **do**

9 **foreach** I ***in*** $\mathcal{D}$ **do**

10 $\mathbf{W[a]} \leftarrow \mathbf{W[a]} - \sum_{j=1}^{k} \frac{\text{DIFF}(a,I,I.H(j))}{m*k} + \sum_{c \neq I.class} \left[\frac{P(c)}{1-P(I.class)} \sum_{j=1}^{k} \frac{\text{DIFF}(a,I,I.M_c(j))}{m*k} \right]$

end

end

In our Spark implementation the work is split in the same way: each node computes the nearest neighbors to a subset of the examples. Every possible pairing of example indices is generated and stored in a Spark RDD, which is then distributed to the nodes. The whole dataset is sent to the nodes as a broadcast variable, so that they use it as a lookup table. This approach obtains a considerable speed gain, but effectively limits the size of the dataset to the maximum size a Spark broadcast variable can handle.

2.4.2. InfoGain algorithm

The InfoGain algorithm assigns the weight (W) of each attribute (a) by contrasting its information gain with respect to the class. To calculate this value, the entropy (H) of each class given the attribute in question is subtracted from the entropy of that class:

$$\begin{aligned} InfoGain(Class, Attribute) = \\ H(Class) - H(Class|Attribute) \end{aligned} \tag{2.1}$$

Entropy of a variable is defined as $-\sum_i p(i) * log(p(i))$, where i loops through every possible value of the variable. The observed probability of a variable taking a value is represented by $p(i)$, and it is calculated as the ratio of cases where the variable takes that value divided by the total number of appearances of the given variable. If the variables are not discrete, the dataset needs to be preprocessed as described in further detail in Section 2.5.1.

Weka implements this calculation by looping through the entire dataset counting the number of appearances of every possible value for each attribute, storing the counts in an array. Then this array is used to compute the information gain of each attribute. This process has linear complexity.

In our proposed multithreaded solution, detailed in Algorithm 2, the counting of every possible value is performed in parallel for a subset of the samples (Line 4). This requires an additional step, described in Line 7, that combines the counts of each thread into a global count. Since this division is performed on the number of instances, it will be more effective when the dataset has numerous instances. For small datasets, the additional accumulative step can take more time than is gained from counting in parallel, but for large datasets the time required to add up the partial counts is negligible when compared to the counting process.

Lastly, the process of obtaining the information gain values from the counts can also be performed independently for each attribute, therefore it can be computed in parallel (Line 11). The functions Entropy and ConditionalEntropy shown in Line 14 represent the calculation of $H(Class)$ and $H(Class|Attribute)$ respectively.

Again, the use of a thread pooler was avoided by creating as many tasks as threads

Algorithm 2: Pseudo-code for multithreaded InfoGain

Input: $\mathcal{D}$ ←Set of instances with attributes $\mathcal{A}$ classified in classes $\mathcal{C}$

Output: W ← vector storing the weight of each attribute

```
1  set all counts ← 0
2  for t ← 1 to THREADS_AVAILABLE do in parallel
3      D_t ← disjoint subset of instances
4      foreach I in D_t do
5          foreach a in A do
6              partial_counts_{t,a,I.a,I.class} ← partial_counts_{t,a,I.a,I.class} + I.weight
           end
       end
   end
7  for t ← 1 to THREADS_AVAILABLE do
8      foreach a in A do
9          foreach v in values_a do
10             foreach c in C do
                   counts_{a,v,c} += partial_counts_{t,a,v,c}
               end
           end
       end
   end
11 for t ← 1 to THREADS_AVAILABLE do in parallel
12     A_t ← disjoint subset of attributes
13     foreach a in A_t do
14         W[a] ← ENTROPY(counts_a) − CONDITIONALENTROPY(counts_a)
       end
   end
```

are available.

The InfoGain algorithm is already included in the Spark Infotheoretic Feature Selection package [6] that implements several algorithms that share a common structure by the use of a framework [31]. This was the version tested in this work.

2.4.3. CFS algorithm

CFS is a subset evaluator that uses the correlation between attributes to obtain a score for a group of attributes. The computational cost for this algorithm is greatly influenced by the need to obtain the matrix that contains the Pearson product-moment correlation coefficients between every possible pair of attributes. The time complexity of this process grows quadratically with the number of attributes and linearly with the amount of samples, and results in most of time of the CFS algorithm being spent in this process. Once the correlation matrix and the standard deviations of each attribute have been computed, CFS searches the space containing every possible attribute subset looking for one that obtains the highest score in its evaluation method.

The search algorithms used can vary in their complexity, from simple greedy algorithms as the one described in Algorithm 3 that simply adds to the set the best candidate at each step, to more complex backtracking ones like BestFirst, listed in Algorithm 4. This search method keeps a list with every candidate set that it encounters ordered by their score in the evaluating function. For each candidate, it explores every possible addition to the set, adding the resulting new set to the candidate list if its score is high enough. This process goes on until the examination of candidate sets renders no new candidates for a given number of iterations (named MAX_STALE in Line 4 of Algorithm 4).

The evaluation function used by CFS is described in Algorithm 5. It increases when the attributes are highly correlated with the class and it decreases when any attribute is highly correlated with other attributes that are already in the set.

In the existing Weka implementation, which is included by default in the Weka suite, the computation of the correlation matrix is performed in parallel by several threads, although this only occurs when the user chooses to precompute the correlation matrix. Otherwise the matrix is computed in an on-demand basis, which offers better

Algorithm 3: Greedy stepwise search used in CFS

Input: $\mathcal{A}$ ←Set of all possible attributes

Input: S ←Previously selected attributes

Input: *previous_merit* ←Merit of S

Output: *S_out* ← Selected attributes

```
1  best_merit ← previous_merit
2  best_set ← S
3  for a ← A do in parallel
       set_merit ← computeMerit(S, a)
4      if set_merit > best_merit then
5          best_merit ← set_merit
6          best_set ← (S, a)
       end
   end
7  if best_set ! = S then
8      return greedyStepwise(A, best_set, best_merit)
   end
   else
9      return best_set
   end
```

Algorithm 4: Best-first search used in CFS

Input: $\mathcal{A}$ ←Set of all possible attributes

Output: S ← Selected attributes

```
1  candidates ← new OrderedList((∅, 0))
2  merit_cache ← ∅, best_set ← ∅
3  stale ← 0, best_merit ← 0
4  while candidates.hasElements() and stale < MAX_STALE do
5      S, S_Merit ← candidates.popFirst()
6      added ← false
7      for a ← A do in parallel
8          if (S, a) in merit_cache then
               set_merit ← merit_cache.getMerit(S, a)
           end
           else
               set_merit ← computeMerit(S,a)
               merit_cache.storeMerit((S, a), set_merit)
           end
9          if set_merit > S_merit then
10             candidates.push((S, a), set_merit)
11             if set_merit > best_merit then
12                 added ← true, stale ← 0
13                 best_merit ← set_merit
14                 best_set ← (S, a)
               end
           end
       end
15     if not added then
16         stale ← stale + 1
       end
   end
```

Algorithm 5: Subset evaluation in CFS

Input: $\mathcal{A} \leftarrow$ Subset of attributes

Input: C $\leftarrow$ Matrix containing the correlation between the ith and jth attributes in $C[i][j]$

Input: SDev $\leftarrow$Array containing the standard deviation for each attribute

Output: M $\leftarrow$ Merit of subset

1 $numerator \leftarrow 0$

2 $denominator \leftarrow 0$

3 **for** $a \leftarrow \mathcal{A}$ **do**

4 $\quad numerator \leftarrow numerator + C[a][class] * SDev[a]$

5 $\quad denominator \leftarrow denominator + SDev[a]^2$

6 $\quad$ **for** $b \leftarrow A$ *where* $b < a$ **do**

7 $\quad\quad denominator \leftarrow denominator + 2 * SDev[a] * 2 * SDev[b] * C[a][b]$

$\quad$ **end**

end

8 $M \leftarrow \frac{numerator}{\sqrt{denominator}}$

performance.

Our proposed Spark implementation first performs the correlation matrix computation in parallel and then the search process (either BestFirst or GreedyStepwise) is performed, evaluating the different candidate subsets also in parallel.

2.4.4. SVM-RFE algorithm

To perform feature selection, the SVM Recursive Feature Elimination (SVM-RFE) algorithm makes use of support vector machine classifiers to assign a weight to each attribute. Starting with the whole set of attributes, an SVM is trained to classify binary datasets. The weights assigned to the features by the SVM are then examined and those with the lowest absolute value are removed from the set and added to the ranking in the lowest positions, as shown in Line 14 of Algorithm 6 (the number of elements added at each iteration can be configured with the $STEP$ variable). Then the process is repeated for the remaining attributes until the ranking is complete (Line 10).

In order to work with multiclass datasets, a different ranking is obtained for each class (Line 3) using a one-vs-all approach, that is, assuming that those elements pertaining to a class other than the one being analyzed are negative examples. Then those rankings are combined by looping through them and adding to the final ranking the best of each list, then the second best and so on, in a loop described in Line 4. The process for obtaining the ranking for each class can be done in parallel, and this is the approach taken in our multithreaded Weka implementation. This allows the new version to take much less time when processing multiclass datasets, while not hindering the performance when used with binary datasets.

In the Spark implementation, by contrast, it is the process of training the SVMs that is done in parallel, allowing to save time both on multiclass and binary datasets. This can be done by using the existing SVM with stochastic gradient descent (SGD) implementation in Spark's MLlib library. SGD is an incremental algorithm that is well suited for parallelization. Weka employs Sequential Minimal Optimization (SMO [123]), an analytical method that is generally faster, but much harder to parallelize. This change in the nature of the SVM training algorithm results in a selected set of features that can be different from that obtained with Weka.

Algorithm 6: Pseudo-code for multithr. SVM-RFE

Input: $\mathcal{D}$ ←Set of instances with attributes $\mathcal{A}$ classified in classes $\mathcal{C}$

Output: *ordered* ← Ordered attributes

```
1  attributeScoresByClass ← ∅, ordered ← ∅
2  for c ← C do in parallel
3  |  attributeScoresByClass[c] ← RankBySVM(c, D)
   end
4  for a ← A do
5  |  foreach c in C do
6  |  |  if not ordered.contains(attributeScoresByClass[c][a]) then
7  |  |  |  ordered.add(attributeScoresByClass[c][a])
   |  |  end
   |  end
   end
8  return ordered
   function rankBySVM(c, D)
9  |  numAttrs ← empty stack  // Number of attributes ranking
10 |  while numAttrs > 0 do
11 |  |  weights ← new SVMClassifier(D,c).weights
12 |  |  foreach w in weights do
13 |  |  |  weights[w] = weights[w]^2
   |  |  end
14 |  |  for i ← 0 to STEP do
15 |  |  |  worstAttr ← findWorst(weights)
16 |  |  |  D.removeAttr(worstAttr), ranking.add(worstAttr)
   |  |  end
   |  end
17 |  return ranking
   end
```

2.5. Experimental results

The goal of this work is to take advantage of multithreaded and distributed processing to speed up feature selection. Hence, the features selected and the weights assigned by the new versions of the algorithms are the same as those obtained with the original versions, excluding any differences that may arise due to rounding or numeric processing (except in the case of SVM-RFE that obtains different results in Spark due to the change of the nature of the underlying SVM). Consequently, these new versions do not modify the classification accuracy, but aim at being able to perform feature selection in a reasonable, shorter time.

It is worth mentioning that the time complexity of the studied algorithms is very variable. Furthermore, their impact on the total time needed for the whole feature selection process can also be significantly different. Since one of the goals of this work is to provide a reference guide to help users select one implementation, we have decided to list the total execution time instead of just the time invested in the part of the algorithm that actually performs feature selection because this will give users a more accurate idea of what to expect from a certain implementation. There may be some use cases where the algorithm is used in a different context (for instance, loading a dataset once and then performing several iterations of a feature selection algorithm), that take more advantage from the gain associated with the parallel implementation. Nonetheless, the most common use case is performing feature selection on a dataset contained in a file.

In order to provide a variety of scenarios to test the proposed Weka and Spark implementations, seven high dimensional datasets were chosen (see their characteristics in Table 2.3). We used the *Higgs* dataset, which consists of 11,000,000 instances with 28 numerical attributes that represent kinematic properties of particles detected in an accelerator [79]. The second dataset used, from here on called *Epsilon*, was artificially created in 2008 for the Pascal Large Scale Learning Challenge [139]. A preprocessed version available on the LibSVM dataset repository [100] was used. This dataset consists of 500,000 instances that have 2,000 numerical features each. Since both datasets mentioned above are binary datasets, one additional dataset with several classes was selected, *KDD99* [78]. It contains close to 5 million samples of 41 computer network connection parameters each that are categorized in 23 different classes. Also, SVMs require that datasets have numeric attributes only, so any non-numeric attribute needs to be transformed. Therefore, three multiclass datasets with numeric features were

Table 2.3: Dataset description

Dataset	Features	Instances	Classes
Higgs	28	11,000,000	2
Epsilon	2,000	500,000	2
KDD99	41	4,898,430	23
Isolet	617	7,900	27
USPS	256	7,291	10
Poker	10	1,025,010	10
KDDB	29,890,095	19,264,097	2

chosen: *Isolet* [101] consists of almost 8,000 instances with 617 attributes each, divided in 27 classes. *USPS* [82] is a dataset containing over 7,000 examples of elements with 256 attributes, representing handwritten characters, with 10 different labels. Lastly, the *Poker* dataset contains over a million elements with 10 features each, classified in 10 different classes, representing possible hands in the poker card game. An additional larger dataset named *KDDB* consisting of 19 million samples with 30 million attributes was included as an example of very high dimensionality [142].

The experiments were run on up to 8 nodes of a computer cluster. Each node has the specifications described in Table 2.4. The Weka version used was 3.7.12 running on OpenJDK 1.7.0_55. The OS installed in this machine was Rocks 6.1, based on CentOS 6.x. Spark applications were run using the MapReduce Evaluator (MREv) tool, that unifies the configuration of various distributed computing environments [146].

To measure the performance of the new versions of the algorithms comparatively to the original implementations we used the speed-up measure, defined as the ratio between the original sequential time and the parallel one.

Table 2.4: Computer cluster description

16 nodes consisting of:	
Processor:	2 × Intel Xeon E5-2660 Sandy Bridge-EP at 2.20Ghz
Cores:	8 per processor (16 per node)
Threads:	2 per core (total of 32 threads per node)
Hard drive:	1 × SSD 480GB SATA3
RAM:	64 GB DDR3 1600 MHz
Network:	InfiniBand FDR & Gigabit Ethernet

2.5.1. On the preprocessing of the datasets: Parallelization of a discretization algorithm

Some feature selection algorithms, such as InfoGain, require the attributes of the dataset to be discrete. This specification often forces the user to preprocess the dataset in order to obtain a modified version with discrete features. Weka provides an implementation of the Fayyad-Irani Minimum Descriptive Length (MDL) algorithm [59] that fulfills that purpose, although this process can be very time consuming. The goal of this algorithm is to transform real-valued attributes to discrete ones while maintaining as much information as possible. To achieve this, real values need to be assigned to different bins that cover the whole range of values of the attribute. The size, number, and distribution of the bins is decided by the algorithm in a long process that is performed independently for each attribute. This allows us to obtain better performance by using separate threads to compute different attributes, as described in Algorithm 7. A similar parallelization with Spark has not been addressed in this section as it was already available in Spark packages [140]. Table 2.5 shows the execution times for the sequential implementation compared to the multithreaded one when run on a 16 core machine using the three more general datasets (with and without numerical features, as explained at the beginning of this section).

Although the computing process is independent for each thread, a separate copy of the dataset needs to be allocated for each task, since its first step is to order it by the attribute being examined. The overhead created by copying the dataset can be

Algorithm 7: Fayyad-Irani discretization

Input: $\mathcal{A} \leftarrow$ List of attributes

Input: $\mathcal{D} \leftarrow$ dataset

1 **for** $a \leftarrow \mathcal{A}$ **do in parallel**

2 $\quad orderedD \leftarrow \mathcal{D}.\text{ORDERBY}(a)$

3 $\quad bins[a] \leftarrow \text{COMPUTECUTPOINTS}(orderedD, a)$

$\quad$ `// computeCutPoints uses mutual information to obtain the bins in which to discretize the values for attribute` a.

end

4 **for** $i \leftarrow$ *1* ***to*** *THREADS_AVAILABLE* **do in parallel**

5 $\quad S_i \leftarrow$ disjoint subset of instances

6 $\quad$ **foreach** I ***in*** S_i **do**

7 $\quad\quad$ **for** $a \leftarrow \mathcal{A}$ **do**

8 $\quad\quad\quad S_i[a] \leftarrow bins[a].\text{TRANSFORM}(S_i[a])$

$\quad\quad$ **end**

$\quad$ **end**

end

Table 2.5: Execution times of the discretization algorithm implementations

	Runtime (s)		
	1 core	16 cores	Speed-up
Higgs	1585	1709	0.93
KDD99	316	196	1.61
Epsilon	1976	881	2.24

quite large if the dataset is sizable, but in most cases it is not as large as the gain obtained by computing in parallel. In our experiments all datasets but one obtained a favorable speed-up, independently of their size. The new version performed worse than the sequential one for the *Higgs* dataset, due to its large size and few attributes, which amounts to costly copies of the dataset and less parallelism.

2.5.2. Analysis of the ReliefF implementations

The good adaptability of ReliefF to a parallel environment (which is often referred to as being "embarrassingly parallel") translates into significant decreases in terms of execution time. Despite this improvement, ReliefF's complexity grows quadratically with the number of samples and linearly with the number of features and this still makes it yield long times when the number of instances of the dataset is very high. However, our multithreaded implementation can take advantage of machines with a large number of cores, decreasing computational times.

In order to be able to make a comparison with the sequential version, we have used reduced versions of the largest general datasets (with numerical and non-numerical features) when analyzing the ReliefF implementation. For the *Epsilon* and the *KDD99* datasets the top 10% of the instances were used, amounting to a total of 50,000 and almost 500,000 instances, respectively. The *Higgs* dataset had to be further trimmed, using the top 4%, consisting of 440,000 instances.

We performed tests with different number of threads processing the same datasets in order to illustrate the relation between the execution time and the number of threads employed. The results of these experiments are shown in Figure 2.1. The node used

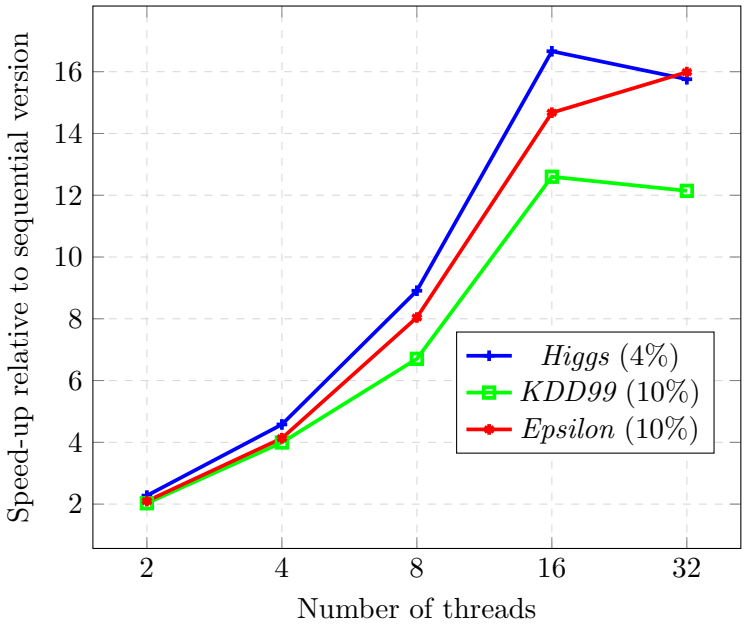


Figure 2.1: Speed-up vs number of threads for ReliefF

to run the benchmarks offered 16 cores, each one capable of running two threads using HyperThreading. When 16 threads are used, they are mapped to different cores with exclusive use of resources, obtaining maximum performance. On the contrary, when we request the use of 32 threads, they are placed two on each core, competing for the core resources [128]. This results in a degradation of performance that, in our best case, barely improves on the use of 16 threads. Therefore, all subsequent experiments were made using just the 16 cores.

The left part of Table 2.6 lists the execution times of sequential and multithreaded Weka implementations. The multithreaded version was executed using the 16 cores available. A significant performance increase exists for all datasets. When the dataset being analyzed is large, the time taken to manage threads becomes irrelevant in comparison to the time gained by making computations in parallel. The multithreaded version of the algorithm was able to process the large datasets between 12.6 and 16.7 times faster than the sequential one. The good adaptability of this algorithm to a parallel paradigm reflects in the superlinearity of the speed-up obtained for the Higgs dataset.

For comparison purposes, Table 2.6 also shows the Spark execution times for different amount of cores. The *Epsilon* dataset was chosen for this comparison since its

Table 2.6: Execution times of ReliefF implementations[2]

Runtime (s)									
	Weka			Spark					
# cores	1	16	Speed-up	16	32	64	128	Speed-up	⇑
Higgs (4%)	105443	6328	16.7	-	-	-	-	-	-
KDD99 (10%)	154305	10517	14.7	-	-	-	-	-	-
Epsilon (10%)	84149	6678	12.6	5382	2840	1076	608	8.85	10.98

execution time was high on Weka and its size was suitable for the Spark implementation. As discussed in Section 2.4.1, the Spark implementation of ReliefF requires that the entire dataset is broadcast to all nodes. Good scalability is observed when more nodes are added and, even with one node (16 cores), the Spark implementation is more efficient than the Weka one. To assess the advantage of using Spark and a computer cluster vs Weka on a single machine, the speed-up shown as ⇑ is the best Spark result vs the multithreaded Weka result.

2.5.3. Analysis of the InfoGain implementations

The Weka implementation of the InfoGain feature selection algorithm requires the attributes to be discrete, so it performs a discretization process when needed before the feature selection is started. This discretization is independent from the InfoGain algorithm so, to eliminate its impact in the execution time and obtain a more accurate comparison of the two versions of the algorithm, all datasets used to test the InfoGain feature selector were discretized beforehand using the same algorithm employed by Weka [59]. This resulted in datasets that, in some cases, had several attributes with constant value. Additionally, to speed up this process for users, a multithreaded implementation of this algorithm is provided, as described in Section 2.5.1.

The left part of Table 7 shows the comparison of the Weka execution times between both versions of the algorithm (sequential and multithreaded using 16 cores) when run on the different datasets that have been previously discretized.

Table 2.7: Execution times of InfoGain implementations[2]

Runtime (s)									
	Weka			Spark					
# cores	1	16	Speed-up	16	32	64	128	Speed-up	⇑
Higgs	204	192	1.06	578	375	353	173	3.34	1.11
Epsilon	458	424	1.08	1067	642	448	335	3.19	1.27
KDD99	145	140	1.04	-	-	-	-	-	-
KDDB	200	192	1.04	631	500	384	407	1.55	0.47

When put in relation with the whole execution time, the speed improvement is negligible. Nevertheless, a deeper analysis of the implementation reveals that most of the time needed to perform InfoGain feature selection in Weka is spent getting the dataset ready, first reading it from disk and then checking that the attributes are fit for the algorithm. The feature selection process itself takes a short time when compared to the total execution time, so even a dramatic improvement in the time efficiency of the algorithm would lead to modest speed-ups for datasets that take a long time to process. Nevertheless, as discussed earlier, some use cases may take advantage of the speed-up obtained when just comparing the time devoted to the algorithm which, in the Weka implementation we are presenting, is close to the number of cores employed, around 16 in this case.

The Spark implementation tested was the one included in the InfoTheoretic Feature Selection Spark package [6]. Results can be seen in right part of Table 2.7. Instead of the *KDD99* dataset, *KDDB* was used to illustrate how this method is capable of handling very high dimensional datasets.

Although performance increases when adding more cores, for the same number of cores the existing Spark implementation performs much worse than Weka. This results in the need of more nodes to achieve the same times than in Weka, being highly inefficient in terms of resources. For this particular algorithm and datasets it would be more advisable to use Weka on a single machine rather than the existing Spark implementation.

[2]Speed-ups listed are 16 cores vs 1 core for Weka and 128 cores vs 16 cores for Spark. ⇑ indicates the speed-up for 128 cores using Spark vs 16 cores using Weka, that is, the gain of the parallel approach.

Table 2.8: Execution times of CFS implementations[2]

Runtime (s)									
	Weka			Spark					
# cores	1	16	Speed-up	16	32	64	128	Speed-up	⇑
Higgs	1350	1173	1.15	110	98	95	91	1.21	12.89
Epsilon	7183	8642	0.83	579	438	356	324	1.79	26.67

2.5.4. Analysis of the CFS implementations

The existing multithreaded implementation of the CFS algorithm included in Weka does not offer a significant improvement over the sequential one, being even slower in some cases. This is a result of the parallelization approach used, that requires that the entire correlation matrix is precomputed beforehand, in contrast with the sequential version, that only calculates each value when needed. Since the search method does not try every possible combination of attributes, oftentimes only a small fraction of the correlation matrix needs to be computed. Avoiding to compute these unnecessary values saves significant time that, in some cases, results in smaller computation times than the ones obtained by precomputing the entire correlation matrix with several cores. Our Spark implementation computes the entire correlation matrix every time, but it is still much more time-efficient than the Weka one, as shown in Table 2.8. The computation time decreases as more nodes are added which, when combined with the much better performance than the Weka algorithm obtained for the same number of cores, results in high speed-ups.

2.5.5. Analysis of the SVM-RFE implementations

Since the parallelization approach taken for the multithreaded Weka implementation divides the work along classes, multiclass datasets were needed for this experiment. Execution times are shown in Table 2.9 (please note that in this case times marked with − are executions that take more than three days). The different SVM training algorithm used in Weka and Spark makes a real difference regarding the kind of dataset that can be tackled with each implementation. The Weka version (and thus our multithreaded

Table 2.9: Execution times of SVM-RFE implementations[2]

Runtime (s)									
	Weka			Spark					
# cores	1	16	Speed-up	16	32	64	128	Speed-up	⇑
Isolet	86730	15415	5.63	-	-	-	-	-	-
USPS	10098	2508	4.03	-	-	-	-	-	-
Poker	-	-	-	1229	1536	1220	1447	0.85	-
Poker (20 %)	28621	10280	2.78	530	520	472	465	1.14	22.11

version), which uses SMO (see Section 2.4.4), performs really well when there is a large number of attributes and, therefore, the SVM training process has to be repeated a large number of times. In this case the approach used by the Spark version takes much longer, as for every new training process the data needs to be shuffled. This, in some cases, makes its use unfeasible (for instance, for *Isolet* and *USPS* datasets). On the contrary, when datasets have fewer attributes (such as *Poker*), the SVM training process is repeated fewer times and SGD can be leveraged to train the model with a large number of examples in a much smaller time than SMO. This clearly differentiates both implementations in terms of the datasets that they handle efficiently. Table 2.9 shows how SGD is suitable for datasets with a large number of attributes and few instances (*Isolet* and *USPS*), whereas SMO performs better when there is a large number of instances and fewer attributes (such as *Poker*).

2.6. Conclusions

This chapter has explored new implementations of four popular feature selection algorithms. We have proposed new versions for their use in Weka that take advantage of multithreaded processing to speed up the computation, and also distributed versions that use Apache Spark, enabling users to tackle bigger datasets in a reasonable time. For those implementations that already existed (see Table 2.2), tests were performed to assess their suitability for different kinds of datasets. In doing so, we show the usefulness of distributed computing to increase the ability of machine learning algorithms to tackle

large-scale datasets.

The experimental results obtained show a significant improvement in execution time for the ReliefF algorithm, achieving even superlinear speed-ups for large real-world datasets on a 16 core node, and scaling well in number of nodes for Spark. A considerable improvement was also obtained for a new distributed CFS implementation in Apache Spark that largely outperforms the existing multithreaded version included in Weka, and scales well when more cores are added. A new multithreaded InfoGain implementation was developed and compared to the existing Spark one, finding that its short execution times make the time gain obtained using a cluster less relevant, therefore advising the use of our proposed implementation on a single computer. Lastly, a new SVM-RFE multithreaded implementation enables users to process multiclass datasets up to four times faster than the sequential counterpart included in Weka, and a new Spark version allows the analysis of datasets that because of their dimensions could not be processed by Weka.

As future work, it would be interesting to explore different sampling techniques and their effects on the features selected for a variety of datasets, since this approach may offer a way to use algorithms that are computationally demanding on reduced versions of large datasets. Also, approximate methods could be used to alleviate the computational cost of the most expensive algorithms. Chapter 4 explores this possibility and describes a ReliefF implementation with Spark that can handle larger datasets than the ones at reach for the implementation presented in this chapter.

CHAPTER 3

Approximate models: Scalable kNN Graph construction with Locality Sensitive Hashing

3.1. Introduction

Some machine learning models demand a computational effort that exceeds the available resources even with the use of techniques like distributed computing. This situation often leads to the dismissal of the model in favour of a simpler one, although in some cases an approximation of the model that can be computed at a much inferior cost will offer better performance. That is the situation that we will explore in this chapter.

We will focus on a popular data structure that is widely used in machine learning but missing from distributed computing libraries. The k nearest neighbors graph (kNNG) is a representation of all elements of a dataset $\mathcal{D}$ as a directed graph in which for n data points, $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n\}$, edges $(\mathbf{x}_i, \mathbf{x}_j)$ indicate that $\mathbf{x}_j$ is amongst the k most similar elements to the point $\mathbf{x}_i$ under a specified similarity measure $\sigma(\mathbf{x}_i, \mathbf{x}_j)$. This data structure allows one to easily navigate elements that are similar to each other. This is useful in areas such as data mining [45], computer graphics [129] and machine learning, specifically outlier detection [83], feature selection [89] and classification [41]. Despite being a conceptually simply idea, the computational cost necessary to obtain the kNNG by brute force is high, since it requires performing $n(n-1)/2$ pairwise comparisons, which amounts to $\mathcal{O}(n^2)$ time complexity. As a result, there have been attempts to obtain algorithms that compute this graph at a lower cost.

In this chapter we present a novel approach to compute an approximate version of the kNNG that is based on Locality Sensitive Hashing (LSH) [8] schemes. A preliminary version of this work was presented in the 25th European Symposium on Artificial Neu-

ral Networks, Computational Intelligence and Machine Learning. LSH is a technique designed to speed up the retrieval of points in a dataset that are similar to a query point by pre-building a data structure. The main idea behind LSH is that if two points are close in the original space, they will continue to be so after a projection, which is used to group points that are similar. Therefore, we developed an algorithm that leverages the locality sensitive property to compute an approximate kNNG. Additionally, taking advantage of the structure of the algorithm, we provide a distributed implementation in Apache Spark which can use a computer cluster to apply this algorithm to very large datasets. Our experimental results show that the proposed algorithm outperforms the current state-of-the-art alternative algorithms, both in terms of kNN graph accuracy and computational cost.

In Section 3.2 we discuss the state of the art in the field, in Section 3.3 we describe the presented algorithm, while in Section 3.4 we report the experiments performed to assess the validity of our proposal. Finally in Section 3.5 we summarize our conclusions and we reflect on which future developments of the algorithm could be made.

3.2. Related work

The great number of applications of the kNNG and the complexity of its calculation has motivated researchers to obtain efficient variations of the kNN algorithm. The literature reflects solutions that are computationally effective under certain conditions. When the dimensionality of the input space is small, the use of multidimensional binary search trees named k-d trees has been proven fast [17], but this solution rapidly becomes inefficient as the dimension of the input space grows (curse of dimensionality). An effective approach has also been proposed for when the similarity metric used is the cosine similarity [7], which first computes an approximation of the graph and then refines it by using the theoretic properties of this particular similarity measure.

However, so far the only way to cope with general metrics and high dimensional datasets at a reasonable computational cost is to build an approximate version of the kNNG, introducing a tradeoff between the computational effort invested and the accuracy of the obtained graph with respect to its exact counterpart. Different approaches have been proposed using a number of techniques to reduce computational complexity. The divide-and-conquer based approaches include the use of recursive inexpensive bi-

section steps [38, 147] that still amount to a high, although reduced with respect to the original, computational complexity. Local search approaches that take advantage of the fact that the neighbor of a neighbor is likely to also be a neighbor, such as NN-Descent [50] are a good option, with a reported complexity of $\mathcal{O}(n^{1.14})$ for k=20, but yet again their results suffer when the dataset has high intrinsic dimensionality. Moreover, the complexity increases greatly for larger values of k. Several modifications of NN-Descent have been proposed to address these shortcomings [29], but so far none of them has given a universal solution. Finally, the use of LSH enables a generic strategy for approximating the kNNG under any similarity measure [155]. Since this approach is the base of our proposal, we will analyze both its theoretical foundations and the existing methods that use it in the next subsection.

3.2.1. kNNG using Locality-Sensitive Hashing

The use of LSH for the construction of the kNN graph is based on its ability to group elements that are similar. In particular, the main idea underlying LSH is that if two points are similar, they will continue to be so after a projection. This idea is used to reduce the search space for a given query point, that is, given a data point $\mathbf{x}$, when trying to retrieve its k nearest neighbors the use of LSH allows to search only points that are likely to be similar to $\mathbf{x}$ instead of the whole of $\mathcal{D}$. This is accomplished with a Locality-Sensitive Hash function, that is, a function that maps elements from a high-dimensional space, which is generally sparse, to a lower-dimensional more dense space and does so in a manner such that elements that are close in the input space are mapped to the same point of the image space with a high probability. A family of hash functions $\mathcal{H}$ is called $(r; cr; P_1; P_2)$-sensitive with respect to a given similarity measure σ if for any two points $\mathbf{p}, \mathbf{q} \in \Re^d$:

$$\sigma(\mathbf{p}, \mathbf{q}) \leq r \longrightarrow Pr(h(\mathbf{p}) = h(\mathbf{q})) \geq P_1 \tag{3.1}$$

$$\sigma(\mathbf{p}, \mathbf{q}) > cr \longrightarrow Pr(h(\mathbf{p}) = h(\mathbf{q})) \leq P_2 \tag{3.2}$$

with $h \in \mathcal{H}$. Specifically, given $\mathbf{p}, \mathbf{q} \in \Re^d$ if $\sigma(\mathbf{p}, \mathbf{q}) \leq r$ they will be considered similar and the random hash function h will produce a collision, that is, assign them the same value, with a probability at least P_1. Conversely, if $\sigma(\mathbf{p}, \mathbf{q}) > cr$ $\mathbf{p}$ and $\mathbf{q}$ will not be considered similar and the probability of h assigning them the same value will be lower than P_2. If $P_1 >> P_2$ then those points that are given the same hash value will be very likely to be similar in the input space. Moreover, if a point is given a hash value $h(\mathbf{x})$ then most elements similar to $\mathbf{x}$ will be given the same hash

value. These two characteristics make LSH very useful for reducing the search space to elements that are similar to the query. In some cases P_1 is just slightly larger than P_2; a common approach to increase this difference consists in concatenating several hash function values [8]. Additionally, in order to increase the number of collisions it is also a common practice to generate several hash keys for each point using various hash functions from $\mathcal{H}$.

As mentioned above, this technique was originally used to perform similarity queries in sublinear time [8, 40] by constructing a data structure that organizes the input data according to the values assigned by the LSH function. Specifically, a group of hash functions is computed and the hash values of existing points in the dataset are stored. For a given query point, only those elements of the dataset that share the same hash value (i.e. very likely to be similar) are compared to it, greatly reducing the number of pairwise comparisons and, therefore, reducing the computational complexity. Despite this being the usual approach to leveraging LSH, there is still some degree of uncertainty given its dependence on probabilities P_1 and P_2. The data structures and the query methods used in LSH are an active area of research [40]. As a result, the optimal way of exploiting LSH remains an open problem.

The described scheme is used to tackle two problems closely related to kNNG construction. The first one is named *nearest neighbor search* [147], which consists in retrieving the k nearest neighbors in a dataset $\mathcal{D}$ to a query point $\mathbf{p}$ not present in the dataset, has been successfully used in fields such as search engines [73], computational linguistics [126] and computational biology [32]. The second one is *spherical range reporting* [122, 2], that requires retrieving all points $\mathbf{x} \in \mathcal{D}$ such that $\sigma(\mathbf{x}, \mathbf{p}) < r$ for a given query point $\mathbf{p}$ not in $\mathcal{D}$ and a threshold value r. Still, computing the kNNG entails a different set of restrictions from the aforementioned problems. Mainly, the focus for approximate kNNG algorithms is obtaining a graph as accurate as possible in the least possible amount of time so that additional processing can be done using it as a starting point. The data structure built in the process is discarded, which is a contrast to *nearest neighbor search* and *spherical range reporting*, in which besides accuracy, both the size of the resulting data structure and the speed of each query answer (i.e. the effectiveness of the data structure) need to be taken into account, but the time invested in computing the structure is not crucial. Therefore, it may be advisable in such problems to invest some more time in computing a finely tuned data structure. These differences make the adaptation of algorithms that solve *nearest neighbor search* or *spherical range reporting* to tackle kNNG construction non-trivial. Up to the authors knowledge, so far only one work, by Zhang *et al.* [155], has used

LSH to compute the kNNG. This algorithm first splits data into groups of similar elements using LSH, then computes the pairwise similarities of the elements in each of these groups, which are used to build a partial graph for each group. These partial graphs are finally merged, producing the final approximate kNNG. Still, this algorithm has many dataset-dependent hyperparameters that need to be tuned which complicates the obtention of good results, which is a common theme to LSH methods [51]. Additionally, datasets that have very uneven density of elements in different regions can cause poor performance.

Our approach is based on the algorithm proposed by Zhang *et al.*, but it addresses the mentioned shortcomings. We take advantage of the structure of the algorithm, which can be implemented following the MapReduce paradigm to leverage parallel computation. Also, we provide an implementation in the distributed computation framework Apache Spark, which can use a cluster of computers to perform the computation, amounting to a great scalability of the method.

3.3. Implementing the Algorithm

We present Variable Resolution LSH (VRLSH)[2], an algorithm that uses LSH repeatedly to explore groups of similar points that increase size at each step. Additionally, the points that have been sufficiently explored are removed from the dataset at each step. This iterative approach is a major difference with the existing LSH based algorithm [155], and it enables the proposed algorithm to adapt to datasets with uneven densities without affecting the computational cost.

VRLSH works as described in Algorithm 8. First, every element $\mathbf{x}$ of the dataset $\mathcal{D}$ is given a hash value $h(\mathbf{x})$ using a LS hash function that will produce collisions for elements with a similarity value larger than a given resolution. After that, elements with the same value of $h(\mathbf{x})$ are grouped, forming buckets of points with a high probability of being similar. A kNN subgraph is computed for each of these buckets by computing all possible pairwise distances and linking each element in the bucket to its k nearest neighbors. Afterwards, overlapping subgraphs are merged, forming an approximate kNNG. At this step, all points that have already been involved in a fixed number of pair-wise computations (C_{MAX} in Algorithm 8) are removed from the dataset. The line

[2]Spark implementation available for download at `https://github.com/eirasf/KNiNe`

of reasoning for this step is that, since all points that have been compared to a given one, $\mathbf{x}$, are very likely to be similar to it, thanks to the LSH filtering performed, once a point has been involved in a large number of such comparisons, it will be very probable that all of its k nearest neighbors will have already been compared to it. Moreover, all points for which $\mathbf{x}$ is one of their k nearest neighbors will be very likely to have also been involved in those pairwise comparisons and one can, therefore, remove said point $\mathbf{x}$ from the dataset. Finally, the resolution is decreased in the following iteration, which lowers the similarity threshold. In consequence, points that were not considered to be similar in the current step because they are too different may be considered to be similar with a lower resolution. The process is then repeated on until the simplified dataset $\mathcal{D}'$ is empty or has very few elements or all of its elements end in the same bucket. After that, the elements that ended up in the graph with less than k neighbors are returned to the dataset. This can occur when an element is removed from the dataset for having been involved in more than C_{MAX} comparisons; C_{MAX} is always selected to be larger than k, but since the elements involved in the comparisons can not be recorded, some comparisons can be repeated, and, in rare cases, this can amount to a number of relevant comparisons lesser than k. A final step is performed in the algorithm if needed, for the rare cases when very few points are left in the simplified dataset. In this case, instead of continuing the hashing process which would, presumably, yield few meaningful collisions, it is preferable to compare these points to the neighbors of its neighbors in the partial approximated graph, that is, perform a local search using neighbor descent, step that is described from line 10 on.

Managing the resolution of the similarity function as described allows the algorithm to process mostly small buckets of elements that are very likely to be close, avoiding performing numerous unnecessary pairwise comparisons. The mentioned dataset simplification step manages to keep the number of elements in each bucket small when the resolution is decreased. Using these two innovations, VRLSH manages to compare each point $\mathbf{x}$ to points that are very likely to be near neighbors, which works towards the accuracy of the approximated kNNG, while maintaining the number of pairwise comparisons low, which leads to low computational cost.

The resulting algorithm is a good fit for parallel computation, which transforms it in a very scalable solution. The hashing step can be performed in parallel across several computing units, then the data can be distributed so that each computating node calculates the subgraph for a subset of the resulting buckets. This parallel processing speeds up the computation substantially. The addition of a registry that records the pairwise distances that have already been computed would allow the avoidance of dupli-

Algorithm 8: Pseudo-code for VRLSH algorithm.

Input: $\mathcal{D}$,k ←Set of points, Number of neighbors to be obtained

Input: R_0 ← Initial resolution

Input: C_{MAX} ←Max number of comparisons per element

Output: G ← Graph containing the k nearest neighbors for each point

```
1  G ← ∅ , D' ← D, R ← R_0
2  while |D'| > k and |buckets| > 1 do
3      hashElems ← LSHash(D', R)
4      buckets ← hashElems.groupByHash()
5      foreach b in buckets do
6          if (b.size > 1) then G ← G ∪ exactKNN(b.elems, k) end
       end
7      D' ← D' − G.getNodesWithAtLeastComparisons(C_MAX)
8      decrease R
   end
9  D' ← D' ∪ G.getNodesWithFewerNeighborsThan(k)
10 if |D'| > 1 then
11     foreach p in D' do
12         if |p.neighbors| = 0 then
13             p.neighbors ← randomSample(D, k)
           end
           else
14             p.neighbors ← topK(k, p.neighbors ∪ neighborDescent(p, G))
           end
       end
   end
```

cate calculations, but it would also impact the memory usage and, more importantly, it would diminish the suitability for distributed computation, so we opted not to include it.

3.3.1. Hyperparameter tuning

As mentioned in Section 3.2, state-of-the-art LSH methods are hindered by the number of hyperparameters that need to be tuned for the LSH scheme to be efficient. The optimal value for these hyperparameters varies with the dataset, which further complicates its obtention. This constitutes a problem for all LSH algorithms. Although there has been some work aimed at tuning the hyperparameters in the particular case of *nearest neighbor search* problems [15, 51], the current research in the field offers no general solution for this problem. The process of hyperparameter tuning is even more important in the case of kNNG construction since, as stated in Section 3.2, it is a one-shot algorithm that attempts to speed up a computationally costly process and any time devoted to hyperparameter tuning decreases the temporal efficiency of the method, making the algorithm less valuable. This is a contrast to LSH algorithms tackling *nearest neighbor search* for which the main goal is speed and accuracy at query time and, consequently, those algorithms can spend more time in hyperparameter tuning. We present a fast hyperparameter tuning process that performs a guided search of the hyperparameter space until finding a suitable set of values. In the next subsections we describe how each hyperparameter is managed and we detail the complete process.

3.3.1.1. Resolution

Although in many cases setting an initial resolution of 0.1 is a valid value that will trigger the creation of aptly-sized buckets [56], this may not be the case for some datasets, which may end up creating buckets with too many (or too few) elements, which would amount to a great number of unnecessary pairwise comparisons (or unnecessary iterations of the algorithm), resulting in extra computational cost. To address this problem, we added a quick estimation procedure that, given a desired initial bucket size, obtains a suitable R_0 value. First, with R_0 set to 0.1, the whole dataset is hashed and the size of the resulting buckets is checked. If they contain too few or too many elements, the resolution is halved or doubled, respectively, and the process is repeated.

If two R_0 values are found to be one too small and the other too large then a binary search is performed. This process is stopped as soon as a suitable R_0 value is found. Although this procedure may require a sizeable number of hashing and grouping steps, it can be performed rapidly since these operations are carried out in parallel across the computing nodes. The resulting execution time of this procedure is very small, compared to the total execution time of the kNNG computation, and the impact of using a R_0 of the correct size in the total time of the algorithm can be considerable. Therefore the use of this tuning procedure is very advisable.

3.3.1.2. Hyperparameters for Euclidean distance as a similarity measure

Also, in the particular case of using the Euclidean distance as a similarity measure, the family of locality sensitive hash functions that is normally used is based on performing random projections of the datapoints. In this case, hash keys are vectors calculated using Equation 3.3. For a given sample $\mathbf{x} \in \Re^d$ each component c of the key is calculated as the integer part of the dot product $\mathbf{x} \cdot \mathbf{w_c} + b_c$ where $\mathbf{w_c}$ is a vector with d components randomly sampled from a $N(0,1)$ and b_c is a scalar bias sampled in the same way. For ease of notation, the corresponding α $\mathbf{w}$ vectors and α biases that determine a hash are joined into a matrix $\underset{(d+1)\times\alpha}{\mathrm{M}}|m_{i,j} \sim \mathcal{N}(0,1)$. Equation 3.3 can be interpreted as projecting the samples onto a random hyperplane and segmenting the projected vectors according to their length.

$$\mathbf{hash_i}(\mathbf{x}) = floor\left((\mathbf{x}, 1) \cdot M_i \cdot R\right) \tag{3.3}$$

A fixed number β of such hashes are calculated for each element, as described in Equation 3.4, to ensure that there are enough meaningful collisions.

$$Keys(\mathbf{x}) = \{\mathbf{hash_0}(\mathbf{x}), \mathbf{hash_1}(\mathbf{x}), \ldots, \mathbf{hash}_\beta(\mathbf{x})\} \tag{3.4}$$

This formulation introduces two additional hyperparameters: α (or key length), representing the length of the hashes, and β (or number of tables) which accounts for the number of hashes generated per element. The effect of these hyperparameters in the performance of the algorithm can be characterized as follows: α affects the size of the

buckets since it dictates how many projections determine a hash. A large α will produce hashes that are very specific and, therefore, generate fewer collisions than a small α, although the elements assigned to the same bucket will have a higher probability of being similar for larger values of α. We would, then, prefer to use the largest value of α that produces a suitable number of collisions. The effect of β is increasing the number of collisions by assigning several hashes to each element.

In order to tune these hyperparameters with the initial resolution, we use a procedure described in Algorithm 9, that extends the above-mentioned. We empirically discovered that setting the β hyperparameter to a fixed constant value and then tuning α and R was the more suitable choice. When providing a value for β we should take into account the fact that high dimensional spaces are more sparse than low dimensional spaces and, consequently, a higher β is needed in order to produce enough collisions as the dimension of the space grows. Therefore, we opted for a simple logarithmic formula depending on the input dimension d and set $\beta = (\log_2 d)^2$. Additionally, a suitable value for the α hyperparameter is estimated as $\alpha_0 = ceil\left(\log_2\left(\frac{n}{d}\right)\right) + 1$, formula inspired by the work of Zhang *et al.* [155]. Then a binary search for a suitable α is performed in the range $[\alpha_0/2, \alpha_0 * 1.5]$, selected to tolerate some variation in the found α while maintaining it close to α_0. This search corresponds to the loop on line 4. To conduct this search, R is set to $R = 0.1$ and the hashing and counting procedures described for the resolution hyperparameter tuning are performed. If any α value in that range produces buckets of the desired size, then all three hyperparameters have been set. Otherwise, a suitable R is searched using the procedure described at the beginning of this section (which is represented in the pseudocode by the function *findResolution*) and once that value is set, the search for α in the aforementioned range is repeated. This procedure yields a combination of the three hyperparameters that configures LSH to produce buckets of the desired size, and does so without having much impact in the execution time of the method, since the operations involved are much less costly than the numerous pairwise distance measurements involved in the iterations of the main algorithm.

3.3.1.3. C_{MAX} and $desiredSize$

Finally, the algorithm has another hyperparameter named C_{MAX} that represents the number of comparisons in which an element of the dataset should be involved for it to be removed from the dataset in a simplification step. This ensures that every

Algorithm 9: Pseudo-code for the hyperparameter tuning procedure.

Input: $\mathcal{D}$,k ←Set of points, Number of neighbors to be obtained

Input: $desiredSize$ ←Desired bucket size

Output: R_0 ← Initial resolution

Output: α, β ←Euclidean distance LSH hyperparameters.

$R \leftarrow 0.1$, $\beta \leftarrow (\log_2(\mathcal{D}.dimension))^2$

$minS \leftarrow desiredSize * 0.5$, $maxS \leftarrow desiredSize * 1.5$

$\alpha_0 \leftarrow ceil(log_2(|D|/D.dim)) + 1$, $left\alpha \leftarrow \alpha_0 * 0.5$, $right\alpha \leftarrow \alpha_0 * 1.5$

while *True* **do**

  $current\alpha \leftarrow (left\alpha + right\alpha)/2$

  $hashElems \leftarrow$ EucLSHash$(\mathcal{D}, R, \alpha, \beta)$

  $sizes \leftarrow hashElems.$countByHash()

  if $sizes.max \in [minS, maxS]$ **then**

    return $R, current\alpha, \beta$

  end

  else

    if $sizes.max < minS$ **then**

      $right\alpha \leftarrow current\alpha$

    end

    else

      $left\alpha \leftarrow current\alpha$

    end

    if $left\alpha >= right\alpha$ **then**

      $R \leftarrow$ findResolution$(current\alpha, \beta, minS, maxS)$

      $left\alpha \leftarrow \alpha_0 * 0.5$, $right\alpha \leftarrow \alpha_0 * 1.5$

    end

  end

end

Dataset	Features	Instances
Audio	192	54387
Shape	544	28775
Corel	14	662317

Table 3.1: Datasets used in the study.

element in the final graph will be compared to, at least, C_{MAX} other elements. The closely related *desiredSize* hyperparameter indicates how large the buckets generated by the LSH procedure should be. In Section 3.4 we detail the experiments performed in order to determine how to handle these two hyperparameters.

3.4. Experimental design and Results

In order to verify the validity of our approach we performed various sets of experiments on three real-world datasets, listed in Table 3.1. These datasets, representing audio signals, 3D shapes and images, respectively, were selected because they were employed by other authors in previous works to benchmark the approximate kNNG-building algorithm NN-Descent [50] and an LSH approach to the *nearest neighbor seach* problem [51].

We used three performance measures in our experiments. The first one is related to the *accuracy* of the computed graph for which we employed the *recall* measure, defined as the ratio of common edges between the approximate and the exact graphs with respect to the total number of edges. This metric is the most usual when assessing the quality of the retrieved k nearest neighbors [10]. Secondly, we gauged the performance of the algorithm by counting the *number of pairwise computations* performed and dividing that number by the number of pairwise computations that the naïve algorithm would use, which is $n(n-1)/2$ where n is the number of elements of the dataset. This metric, known as *scan rate*, is also very commonly used in the literature. Finally, to measure more precisely the quality of the approximate graphs by making a difference between graphs containing the same number of mistakes, we added an additional measure that quantifies those mistakes: the mean error (ME) in the distance of the retrieved

neighbors, defined as

$$ME = \frac{\sum_{i=0}^{n} \sum_{j=0}^{k} \sigma(p_i, n(p_i)_j) - \sigma(p_i, n^*(p_i)_j)}{n \cdot k} \tag{3.5}$$

where $n(p)_k$ represents the k-th neighbor of p in the approximate graph and $n^*(p)_k$ represents the k-th neighbor of p in the exact graph.

3.4.1. Handling C_{MAX} and *desiredSize*

As mentioned in Subsection 3.3.1, C_{MAX} establishes a threshold to the number of comparisons per element. Once an element is compared to candidate neighbors more than C_{MAX} times, it will be removed from the dataset, working on the assumption that it has been compared to enough elements as to have a high probability of having encountered its k nearest neighbors. To observe the effect of C_{MAX} in the obtained recall and scan rate we ran the algorithm using different values of C_{MAX} for the *Audio* and *Shape* datasets. The results of these experiments are showed in Figure 3.1.

The recall of the obtained graphs has a positive dependence on C_{MAX}. As C_{MAX} grows, the recall grows linearly in both datasets. This is consistent with the expected effect: the larger C_{MAX} is, the more accurate the resulting graph will be, since there are more possibilities of finding the k nearest neighbors in a larger set of elements, but also the costlier the computation will be, since a larger number of pairwise comparisons will be performed. This can be appreciated in the plots that represent the Scan rate vs C_{MAX}. Moreover, this dependency is superlinear, that is, the scan rate grows at an increasing and faster rate than C_{MAX}. It can be seen, in consequence, that this parameter manages the balance between accuracy and computational cost that is intrinsic to this problem. It is important to note that since C_{MAX} has a stronger effect on the scan rate than on the recall of the graph, it is not advisable to use large values for C_{MAX} since the computational cost would become too large. We decided to allow the user to modify this hyperparameter to manage the balance between precision and speed of computation, but we have, nonetheless, provided a default value $C_{MAX} = 10 \cdot k$ (truncated to a max of 250 except for $k > 225$ in which case it is $C_{MAX} = 1.1 \cdot k$) which we empirically found to offer a suitable balance.

On the other hand, the *desiredSize* hyperparameter, which is highly related to C_{MAX}, indicates how large the buckets created in the LSH steps should be. Its relation with C_{MAX} determines how many hashing steps will most elements in the dataset

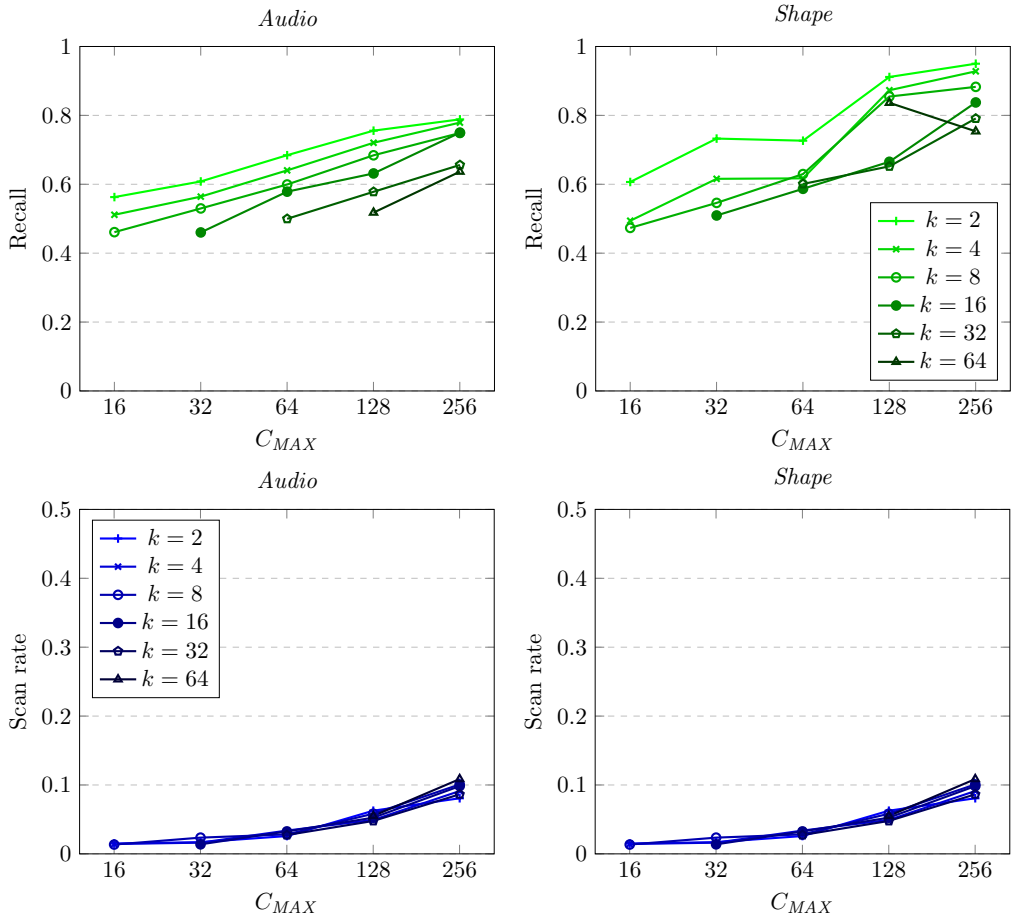


Figure 3.1: Recall vs C_{MAX} for *Audio* and *Shape* datasets and Scan rate vs C_{MAX} for those same datasets, using $desiredSize = 4 \cdot C_{MAX}$ in both cases.

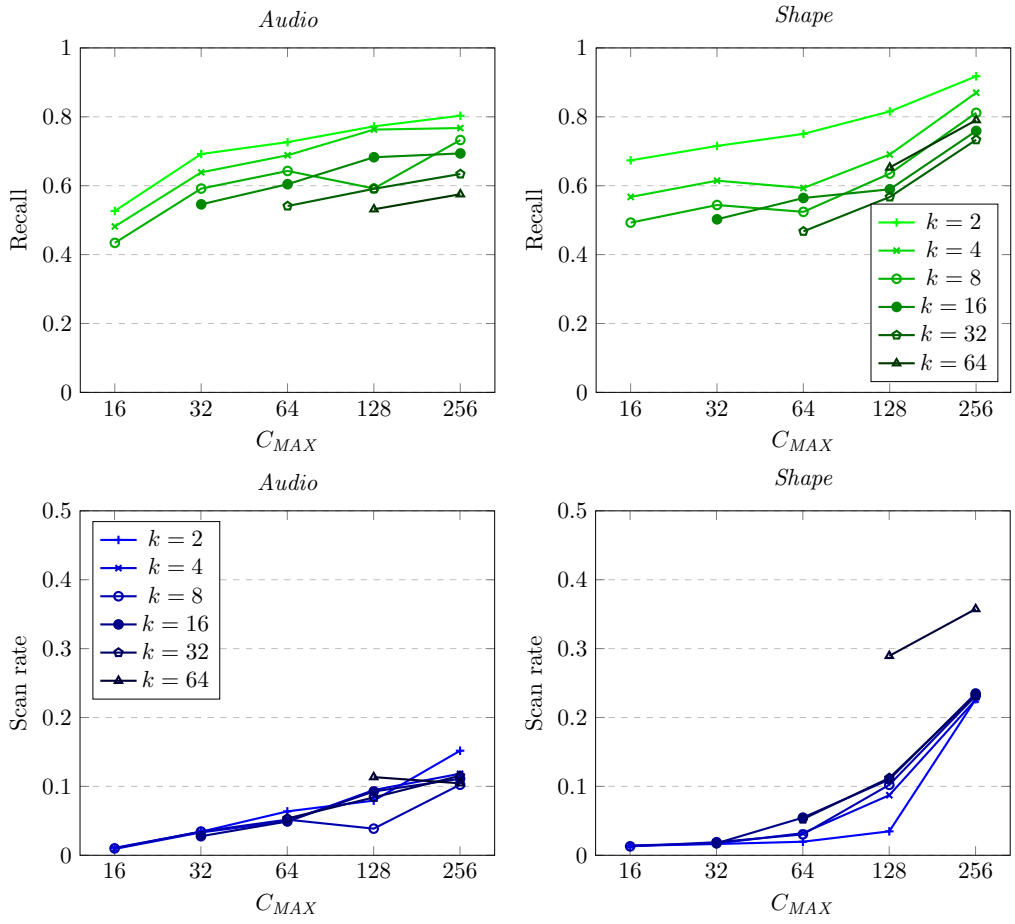


Figure 3.2: Recall vs C_{MAX} for *Audio* and *Shape* datasets and Scan rate vs C_{MAX} for those same datasets, using $desiredSize = 0.8 \cdot C_{MAX}$ in both cases.

endure. If *desiredSize* is much smaller than C_{MAX}, elements will need to be hashed several times until they reach the necessary number of comparisons. Conversely, if *desiredSize* is larger than C_{MAX}, many elements will undergo a single hashing and grouping step. To analyze this behaviour we ran the mentioned experiment with two values for *desiredSize*, representing two different configurations: $desiredSize = 4 \cdot C_{MAX}$, which should force many elements to be discarded for having enough comparisons after a single hashing step, depicted in Figure 3.1 and $desiredSize = 0.8 \cdot C_{MAX}$, shown in Figure 3.2, which should keep elements for a longer number of iterations of the hashing step before removing them in a simplifying step.

These experiments show that using $desiredSize = 4 \cdot C_{MAX}$ offers predictable results in terms of scan rate, while the computational effort required for building the graph

becomes much more variable when $desiredSize = 0.8 \cdot C_{MAX}$. Moreover, the scan rates are slightly higher when $desiredSize = 0.8 \cdot C_{MAX}$, but contrary to the expected behaviour, this increase in computational effort does not revert in higher recall values; on the contrary, the recall values for the graphs obtained are slightly lower than those obtained when $desiredSize = 4 \cdot C_{MAX}$. These results can be explained because the increased number of iterations required for each element when $desiredSize = 0.8 \cdot C_{MAX}$ results in more comparisons $\sigma(\mathbf{p}, \mathbf{q})$ being repeated for the same values of $\mathbf{p}$ and $\mathbf{q}$, as described in Section 3.3, resulting in turn in more elements accumulating purposeless repeated computations that count towards the C_{MAX} threshold and amount to more elements being left without k neighbors after the LSH loop. These points need to be added for completion in the final steps of the algorithm, in the process described from line 9 on in Algorithm 8. Since these steps are more costly and do not benefit from the locality-sensitive reduced search space generated with the LSH steps, the scan rate increases without a significant improvement of the recall. Therefore, we decided to set $desiredSize = 4 \cdot C_{MAX}$, to ensure the predictability of the results and optimize the use of the LSH steps.

3.4.2. Performance of the method

In order to establish the fitness of the proposed method compared to the current state of the art for this problem, we performed another set of experiments in which the results obtained by VRLSH were compared to those obtained with NN-Descent [50][3], which was selected for being the best alternative for computing an approximate kNNG on high dimensional datasets using generic distance metrics, in particular using the Euclidean distance, as discussed in Section 3.2.

The measurements of the computational cost shown in Table 3.2 demonstrate that the scan rate used by VRLSH remains nearly constant regardless of the number of neighbors, in clear contrast with NN-Descent which requires a scan rate that grows very fast as the number of neighbors is incremented, making its use unadvisable for large values of k. This constitutes an important advantage for VRLSH when the value of k is large ($k > 16$). In terms of the graph accuracy, the results shown on Table 3.3, demonstrate that for small values of k, the accuracy of the graph computed by VRLSH is significantly higher than that obtained by NN-Descent. In particular, when $k = 2$ VRLSH obtains graphs with recall values between 0.82 and 0.89, compared to

[3]Implementation available at `https://code.google.com/archive/p/nndes/`

Data	Method	k					
		2	4	8	16	32	64
Audio	VRLSH	0.031	0.024	0.025	**0.025**	**0.025**	**0.028**
	NNDES	**0.001**	**0.007**	**0.022**	0.067	0.214	0.762
Shape	VRLSH	0.029	0.029	**0.026**	**0.028**	**0.046**	**0.030**
	NNDES	**0.002**	**0.014**	0.039	0.120	0.382	1.471
Corel	VRLSH	0.003	0.003	0.003	**0.002**	**0.003**	**0.003**
	NNDES	**0.000**	**0.001**	**0.002**	0.007	0.023	0.077

Table 3.2: Scan rate required by VRLSH and NNDES while calculating the kNNG with different values of k on the studied datasets. The best results for each configuration are highlighted in boldface.

the nearly 0 obtained by NN-Descent. As k grows, the quality of the computed graph decreases for VRLSH while it increases for NN-Descent. For instance, when $k = 64$ the recall of NN-Descent is almost perfect in all cases, while VRLSH obtains values ranging from 0.57 to 0.63. Nevertheless, as mentioned above, NN-Descent obtains these superior results at the expense of the scan rate, which in some cases exceeds 1, which means that the number of pairwise measurements performed is larger than what the naïve algorithm to obtain the exact kNNG would require, rendering the NN-Descent approach invalid in such cases. In contrast, the scan rate used by VRLSH remains in the range $[0.003 - 0.03]$. Moreover, if we analyze the mean error of the calculated kNNG, listed in Table 3.4, it becomes apparent that although the recall obtained with VRLSH descends as k grows, the mean error does not grow as fast, indicating that the retrieved neighbors are close to the exact kNN, constituting a good approximation.

For the cases when such approximation is not enough, we propose adding to the computation an additional stage of refinement of the graph consisting of a single neighbor descent step, that is, for each element in the graph, looking for nearest neighbors among the neighbors of its calculated nearest neighbors in the approximate kNNG. This can be done at an additional cost $\mathcal{O}(n \cdot k)$, which is a reasonable addition to the reduced scan rate required to compute the kNNG with VRLSH, especially for large datasets. We will refer to this method hereafter as VRLSH+. Figure 3.3 shows that the results obtained with this method are very satisfactory, since it greatly increases the recall obtained by VRLSH while keeping the scan rate well below 1. Namely, in

Data	Method	k					
		2	4	8	16	32	64
Audio	VRLSH	**0.825**	**0.756**	0.716	0.663	0.608	0.569
	NNDES	0.002	0.429	**0.892**	**0.982**	**0.998**	**1.000**
Shape	VRLSH	**0.891**	**0.836**	0.794	0.732	0.748	0.599
	NNDES	0.003	0.641	**0.958**	**0.994**	**0.998**	**0.999**
Corel	VRLSH	**0.879**	**0.898**	0.821	0.740	0.683	0.633
	NNDES	0.000	0.419	**0.950**	**0.996**	**0.999**	**0.999**

Table 3.3: Recall of the approximate kNNG calculated by VRLSH and NNDES with different values of k on the studied datasets. The best results for each configuration are highlighted in boldface.

Data	Method	k					
		2	4	8	16	32	64
Audio	VRLSH	**0.015**	**0.032**	0.041	0.055	0.075	0.093
	NNDES	0.852	0.109	**0.009**	**0.001**	**0.000**	**0.000**
Shape	VRLSH	**0.001**	**0.001**	0.002	0.003	0.004	0.008
	NNDES	0.102	0.007	**0.000**	**0.000**	**0.000**	**0.000**
Corel	VRLSH	**0.001**	**0.001**	0.003	0.006	0.010	0.016
	NNDES	0.892	0.022	**0.000**	**0.000**	**0.000**	**0.000**

Table 3.4: Mean distance error of the approximate kNNG calculated by VRLSH and NNDES with different values of k on the studied datasets. The best results for each configuration are highlighted in boldface.

the aforementioned case of $k = 64$ the recall grows from $[0.57 - 0.63]$ in VRLSH to $[0.93 - 0.99]$ in VRLSH+, with the scan rate growing only moderately, resulting in values ranging from 0.07 to 0.3, depending on the dataset, as opposed to 0.21 to 1.64 scored by NN-Descent. It is worth noting that the scan rate is a measure relative to the square of the number of elements in the dataset, so a percent point in scan rate represents an amount of computation that depends on the dataset size. This means that for large datasets, a difference of a single percent point can represent significant time, while for small datasets the scan rate can approach 1 even for approximate methods. This effect can be noticed on the scan rates measured, which are significantly larger in the case of the smallest dataset (*Shape*) compared to the largest dataset (*Corel*). Moreover, in small datasets the scan rate of an approximate method can become larger than 1, which implies that it would be advisable to use the naïve method to compute the exact graph. For such small datasets we recommend calculating the exact graph and we provided a multithreaded implementation of the naïve method in our code. Conversely, for the large datasets that this method is intended for, the advantage in terms of scan rate that our method offers becomes very significant since it represents a great amount of calculations.

3.4.3. Scalability of the method

Finally, to measure the scalability of the method and the provided distributed implementation in Apache Spark, we performed the same computation while varying the number of computing nodes. These experiments were run in a computer cluster formed by 8 machines with 12 computing cores each. The technical specifications of each node are listed on Table 3.5. The Spark version used was 2.4.0, on Hadoop 3.0.0-cdh6.1.0. The operating system of the machines was CentOS Linux release 7.4.1708.

To ascertain the suitability of the method for processing large datasets, we used a dataset with more examples for this experiment. In particular, we selected the *Higgs* dataset[4] [12], which describes measurements of particle collisions and consists of 11 million examples with 28 attributes. With such a large number of elements, calculating the exact kNNG is completely out of reach, and calculating an approximation is the only option. To measure the scalability of our algorithm, we calculated the approximate 4NNG for the *Higgs* dataset several times using a growing number of computing nodes and recorded the execution time invested in the calculation. For all these exper-

[4]Available for download at `https://archive.ics.uci.edu/ml/datasets/HIGGS`

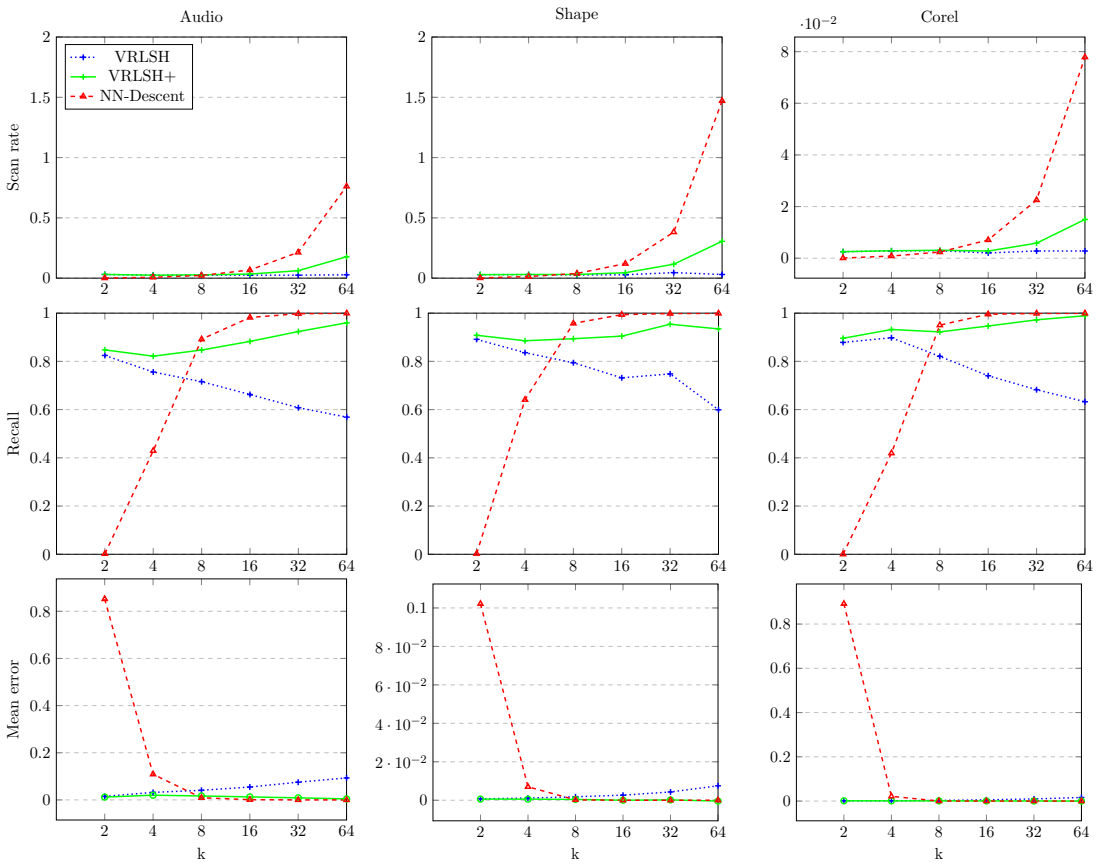


Figure 3.3: Scan rate / Recall / Mean error vs number of neighbors plots for *Audio*, *Shape* and *Corel* datasets.

Table 3.5: Computer cluster overview:

8 nodes with the following characteristics:	
Processor:	2 × Intel Xeon E5-2620 v3 at 2.40Ghz
Cores:	6 per processor (12 per node)
Threads:	2 per core (24 total per node)
Storage:	12 × 2TB NL SATA 6Gbps 3.5" G2HS
RAM:	64 GB
Network:	1x10Gbps + 2x1Gbps

VRLSH				
# units	Time (s)	Scan rate	Ops/s	Speed-up
1	6778	$1.95 \cdot 10^{-4}$	$1.74 \cdot 10^{6}$	1.00
2	3571	$1.91 \cdot 10^{-4}$	$3.23 \cdot 10^{6}$	1.86
4	1868	$2.06 \cdot 10^{-4}$	$6.68 \cdot 10^{6}$	3.83
VRLSH+				
# units	Time (s)	Scan rate	Ops/s	Speed-up
1	7390	$1.98 \cdot 10^{-4}$	$1.62 \cdot 10^{6}$	1.00
2	3972	$1.93 \cdot 10^{-4}$	$2.94 \cdot 10^{6}$	1.82
4	2045	$2.09 \cdot 10^{-4}$	$6.17 \cdot 10^{6}$	3.81

Table 3.6: Scalability vs number of computational units for the computation of the approximate 4NNG for the *Higgs* dataset. The speed-up listed is the ratio between the operations per minute obtained and the operations per minute performed with a single computational unit (12 cores).

iments we used $C_{MAX} = 32$. The results, listed on Table 3.6, show that the distributed implementation provided manages to harness the computational power of the available machines, obtaining almost linear speed-up, that is, accelerating the execution in proportion to the number of cores available. This feature enables the user to analyze very large datasets in a reasonable time as long as enough computational units are available.

3.5. Conclusions

In this chapter we explore the use of an approximate solution as a means to adapt complex models to large-scale scenarios. In particular, we tackle the problem of constructing a kNNG, for which obtaining an exact solution is extremely costly for very large datasets, due to the quadratic computational complexity of the available general algorithms. We present VRLSH (implementation in Apache Spark available for download at `https://github.com/eirasf/KNiNe`), a kNNG approximation algorithm which produces high recall graphs using a low scan rate irrespective of the number of neighbors selected. Taking advantage of the reduced scan rate, the obtained kNNG

can be further refined with a single step of neighbor descent improving the recall of the obtained graph while maintaining the scan rate at manageable values. Additionally, we provide a distributed implementation of this algorithm in Apache Spark, which exploits the structure of the algorithm to provide a distributed solution that can handle datasets with a large number of elements in manageable times by using several computational units. This enables practitioners to tackle large datasets that are out of reach of other state-of-the-art methods. We also present a method for estimating the hyperparameters of the algorithm and, additionally, those required by the Euclidean distance similarity measure, which is very commonly used. This solves the problem of hyperparameter tuning common to other LSH-based solutions. Our tests show that our method outperforms the currently preferred method for kNNG computation.

In the future we will explore the possibility of using memory-efficient registers such as Bloom filters [21] to keep track of the pairwise computations that have been performed, thus helping avoid the repetition of computations. Adapting this algorithm to similar problems such as *nearest neighbor search* and *spherical range reporting* is also a research avenue of great interest. Also, advances in automated parameter tuning over a Pareto frontier by implementing multi objective genetic algorithms [150, 14] can be used to further optimize the parameters of the proposed algorithms to improve both the accuracy and the speed.

As mentioned in Section 3.1, the kNNG is used in many machine learning fields. Exploring the possibilities opened by the increased efficiency of this computation algorithm in each of those fields is a promising research line that may yield significant advances. As an example, in Chapter 4 we show how VRLSH can be used to further increase the scalability of ReliefF, the popular feature selection algorithm that was one of the subjects of Chapter 2.

CHAPTER 4

Approximate FS: Scalable feature selection using ReliefF aided by Locality Sensitive Hashing

4.1. Introduction

The previous chapter showed how helpful it is to obtain a fast approximate solution for kNNG computation. The obtained graph reproduces the exact solution with good accuracy with only a small fraction of the computational effort. As mentioned, the kNNG is used in a variety of machine learning problems that can benefit from the increased speed of the approximate calculation. Still, the effect that the small inaccuracies contained in the approximate kNNG can have in the solution of each of these problems needs to be studied.

In this work we focus on the use of the approximate kNNG for feature selection and we present an adaptation of the popular feature selection algorithm ReliefF that reduces its computational complexity. ReliefF relies heavily on the nearest neighbor graph which needs to be calculated in a process that accounts for most of its computational load. In order to reduce this computational complexity, unsuitable for Big Data, we substituted the graph building process with our VRLSH implementation. This approach, which achieved good preliminary results [55], is fully developed and thoroughly tested. The resulting algorithm, which now allows multiclass datasets and requires no manual hyperparameter tuning, also vastly reduces computational complexity. This solution is implemented in Apache Spark to take advantage of distributed computation, since many of its computations can be performed in parallel. Consequently, it allows the processing of datasets that are far out of reach for the original ReliefF. Also, our method can be applied to any dataset that the original ReliefF can process, in contrast with the existing alternatives.

In Section 4.2 of this chapter we describe ReliefF and list the alternatives that attempt to increase its scalability. Section 4.3 describes the adapted algorithm that we present. In Section 4.4 we detail the experiments performed while in Section 4.5 we list the obtained results. Finally, Section 4.6 presents the conclusions reached from this work.

4.2. Related work

Relief is a feature ranking method that factors in the relevance of each feature for classification [88]. It is a supervised method that returns a list of the features sorted according to their importance, which is then used to perform feature selection by setting a threshold so that features with an importance score less than that value are dismissed. The core idea is assigning a weight to each attribute according to its ability to differentiate elements that are very close together. To do this for classification datasets Relief searches, for each example, for the nearest element within the same class (nearest *hit*) and the closest element of a different class (nearest *miss*) and updates the weight of each attribute A proportionally to the difference of its value in the example and in the nearest hit and nearest miss, respectively. The final weight W of attribute A has, therefore, a probabilistic interpretation since, given an example $\boldsymbol{x}$, it is an approximation of the following difference of conditional probabilities:

$$\begin{aligned} W[A] = P(\boldsymbol{x}(A) <> \text{nearest\_miss(A)}|\text{nearest\_miss}) \\ -P(\boldsymbol{x}(A) <> \text{nearest\_hit(A)}|\text{nearest hit}) \end{aligned} \tag{4.1}$$

Its good performance led to the development of extensions that are able to deal with multiclass problems and examples affected by noise or incomplete [89]. One of these extensions was named ReliefF and ended up being more popular than the original algorithm, with numerous implementations of it available in libraries and machine learning software. Later on, specializations of the algorithm were presented [145], adapting it to regression problems [127], multilabel datasets [141, 137] or taking into account the cost of obtaining each attribute [22]. Work has also been done to optimize the algorithm to enable its use on large datasets. In addition to the use of distributed computing [54, 120], as described in Chapter 2, sampling [58] and random k-d-trees have been used to approximate the nearest neighbor graph [152], but the applicability

of these algorithms to large-scale datasets is still limited. Of these alternatives, the most efficient work is DiReliefF [120], which computes an approximation of the ReliefF attribute ranking. To do so, it uses a sampling scheme to greatly reduce the number of calculated neighbors. DiReliefF also provides a distributed implementation in Apache Spark that makes the handling of large datasets easier.

4.3. Proposed algorithm

The aim of this work is obtaining an algorithm that approximates the result of ReliefF with less computational effort and is able to replicate the most popular implementation of ReliefF [127], being capable to handle multiclass and regression datasets. Most of the computational effort in this algorithm is devoted to obtaining the nearest hits and misses for each element, which requires computing the kNN-graph. Therefore, we propose replacing the computation of the exact kNN-graph in the ReliefF algorithm with an approximation of the kNN-graph obtained by VRLSH [56], which greatly reduces the computational effort. The VRLSH algorithm can be used as it is by ReliefF for regression as, in this type of problems, it is not required to make a distinction between hits and misses, but it needs to be modified for the obtained graph to be used by ReliefF when computing the weights W of each attribute for classification datasets. The graph obtained by VRLSH contains a single list of neighbors for each element in the dataset, but for classification datasets ReliefF needs to make a distinction between hits and misses, so VRLSH needs to be modified to keep several lists of neighbors, one for each class, for every element in the dataset.

Also, the simplification of the original dataset $\mathcal{D}$ along the VRLSH process (see line 7 in Algorithm 8) needs to be revised to handle classification problems. For a given element $\boldsymbol{x_i}$, the count of pairwise comparisons, which we will call $count\_x_i$, determines when a point should be removed from the dataset (when $count\_x_i > C_{MAX}$). This scalar count needs to be expanded to a vector of separate $count\_x_i^j$ of pairwise comparisons with elements of each class c_j. With this change, an element should only be removed from the dataset when it has been involved in at least C_{MAX} comparisons for every possible class, i.e. $count\_x_i^j > C_{MAX} \forall c_j \in \mathcal{C}$ where $\mathcal{C}$ is the set of possible classes. Moreover, depending on the distribution of the classes in the input space, it can be possible that some points are distant from a given class c_k and for them to be involved in C_{MAX} pairwise comparisons requires maintaining them in the dataset for a large

number of iterations. This situation can lead to such points accumulating a number of pairwise comparisons much larger than C_{MAX} for some classes, i.e. $count\_x_i^j >> C_{MAX}$ for some $j \neq k$, and constitutes a challenge to the ability of the method to reduce the number of pairwise calculations. To alleviate this problem, we propose a variation of the bucketing step that avoids comparing points $\boldsymbol{x_i}$ with points of class c_j if $count\_x_i^j > C_{MAX}$. To ease notation we will say that point $\boldsymbol{x_i}$ *requests* class c_j if $count\_x_i^j < C_{MAX}$, i.e. it still needs to be compared to elements of class c_j before being removed. To achieve the mentioned behaviour an additional component h_r is added to every hash that effectively splits, according to the *requested* classes, the buckets of similar elements originated by the LSH function. Therefore, a given point $\boldsymbol{x_i}$ of class c_j that is given hash h by the LSH function, will have an updated set of hashes H' computed as described in Algorithm 10.

Algorithm 10: Hashing procedure modification for multiclass problems

Input: $h \leftarrow$ Hash given to $\boldsymbol{x_i}$ by the LSH function

Input: $c_j \leftarrow$ Class of $\boldsymbol{x_i}$

Input: $count\_x_i \leftarrow$ Pairwise comparisons of $\boldsymbol{x_i}$ to elements of each class.

Output: H' $\leftarrow$ Set of modified hashes for $\boldsymbol{x_i}$

```
1  H' ← (h, c_j)
2  if count_x_i^k < C_MAX ∀c_k ∈ C then
3  |  H'.APPEND((h, ∅))
   end
4  else
5  |  foreach c_k in C do
6  |  |  if count_x_i^k < C_MAX then
7  |  |  |  H'.APPEND((h, c_k))
   |  |  end
   |  end
   end
```

First, H' will always contain the hash (h, c_j), as described in Line 1, intended to make the $\boldsymbol{x_i}$ available to other elements that *request* class c_j. Then, there is an alternative: if $\boldsymbol{x_i}$ *requests* all possible classes Line 3 ensures that $\boldsymbol{x_i}$ is compared to any element in the same situation; otherwise Line 7 adds to H' a hash for every requested

class. This process originates two types of buckets that need to be processed according to their characteristics, rendering the bucketing step in VRLSH (corresponding to the loop in Line 5 of Algorithm 8) invalid and demanding an updated version, which is detailed in Algorithm 11.

Algorithm 11: Updated bucket processing procedure

Input: $G \leftarrow$ Graph containing the computed nearest neighbors for each point

Input: *buckets* $\leftarrow$ Buckets of points that received the same hash value

Output: $G \leftarrow$ Updated graph

```
1  foreach ((h, h_r), points) in buckets do
2      if h_r = ∅ then
3          G ← G ∪ EXACTKNN(points, k)
       end
       else
4          targets ← ∅ , requesters ← ∅
5          foreach p in points do
6              if (h_r = p.class) then targets.APPEND(p) end
7              if (h_r ≠ p.class or p.counts[p.class] < C_MAX) then
                   requesters.APPEND(p) end
           end
8          G ← G ∪ PAIRKNN(requesters, targets, k)
       end
   end
```

In this modified procedure, which is depicted in Figure 4.1, buckets of elements with the same hash are processed according to the modifier component h_r added to their hash. On the one hand, Line 3 processes buckets of elements that request all classes performing every possible pairwise comparison of the elements in the bucket, as occured in Algorithm 8. This ensures that such elements get compared to every point that the LSH function deems similar to them, rapidly finding neighbors of all classes nearby. On the other hand, the buckets containing elements that either *request* a class or belong to a *requested* class are processed differently. Lines 6 and 7 divide each bucket with hash (h, h_r) into two sets: the points that *request* h_r (*requesters*) and the points of class h_r (*targets*). Then, in Line 8 the function PAIRKNN compares

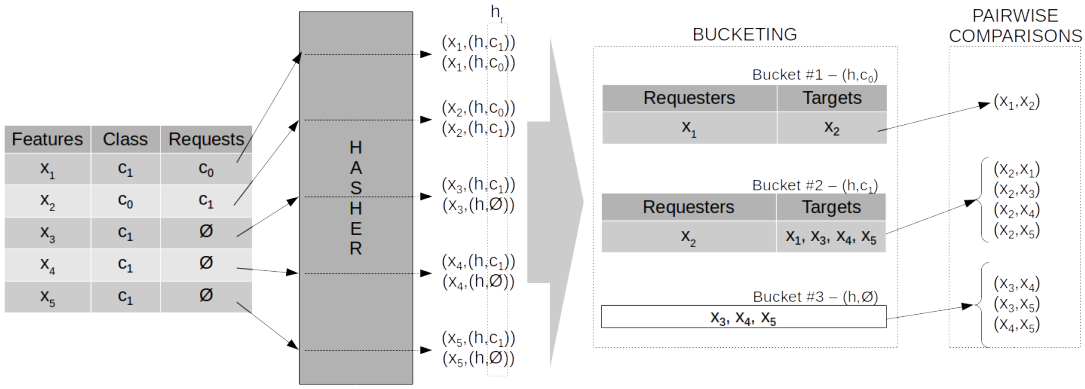


Figure 4.1: Modified hashing and bucketing scheme. x_i are assumed to be similar enough to receive the same hash value h. Also, the hasher only emits a single hash per element, which is later augmented with the h_r value.

each *requester* to every *target* and obtains the corresponding subgraph. This avoids performing *requester-requester* and *target-target* comparisons, which, added to the fact that appending h_r to the hash divides the bucket in more specific and smaller buckets, greatly reduces the number of operations needed.

For a given dataset of size $n \times d$, where n is the number of elements and d is the number of variables, if the hasher generates t hashes of length l for each element, the memory requirements of the algorithm are $\mathcal{O}(nlt)$. This impact on memory is overcome with the distribution of the computation. Moreover, since the dimension d of the dataset is not a direct factor of the memory complexity, the space required by the hashes and the dataset are on par for high dimensional data. This makes the memory overhead smaller for high dimensional datasets, which are our focus.

We provide an implementation of the resulting algorithm, named ReliefF-LSH, in the distributed computing framework Apache Spark that can be downloaded from `https://github.com/eirasf/ReliefF-LSH`.

4.4. Experimental settings

In order to assess the validity of the presented method, two sets of experiments were carried out. We compared the performance of our method with the exact Reli-

efF and also with the recent approximated method DiReliefF, which is the comparable method that offers better performance [120], as discussed in Section 4.2. First, the execution time needed to calculate the weights using ReliefF in real datasets was measured and compared to that needed by ReliefF-LSH and DiReliefF. Since ReliefF-LSH and DiReliefF are both approximate methods, the accuracy of the results obtained was determined by comparing the rankings calculated by each method to the ground truth. The second set of experiments aimed at studying the scalability of Relief-LSH. To that end, we measured the execution times of applying ReliefF-LSH to the same dataset using varying amounts of computational cores.

4.4.1. Equipment and datasets

All experiments were performed in a computer cluster formed by machines with 12 computing cores. The description of each cluster node is shown in Table 4.1. The Apache Spark version used for Relief-LSH was 2.4.0, on Hadoop 3.0.0-cdh6.1.0 while the Apache Spark version used for DiReliefF was 1.6.1, running on Hadoop 2.7.1.2.4.2.0-258, since the available implementation required it. The operating system used by these machines is CentOS Linux release 7.4.1708.

Table 4.1: Cluster description

32 nodes with the following specifications:	
Processor:	2 × Intel Xeon E5-2620 v3 a 2.40Ghz
Cores:	6 per processor (12 per node)
Threads:	2 per core (24 total per node)
Storage:	12 × 2TB NL SATA 6Gbps 3.5" G2HS
RAM:	64 GB
Network:	1x10Gbps + 2x1Gbps

Nine real-world high-dimensional datasets were selected to perform these experiments. These datasets and their characteristics are listed on Table 4.2. We selected these datasets to represent all problems that ReliefF is capable of handling, namely, regression ($Year_{small}$) and binary (*Higgs* [12], $Higgs_{small}$, $Epsilon_{small}$) and multiclass classification ($KDD99_{small}$, CT_{small}, *Cifar10* [95], *SVHN* [117], *Sensorless* [52]). They

Table 4.2: Dataset description

Dataset	Features	Instances	Classes
$Year_{small}$	90	46.371	-
Higgs	28	11.000.000	2
$Higgs_{small}$	28	55.000	2
$Epsilon_{small}$	2.000	50.000	2
$KDD99_{small}$	41	48.984	23
CT_{small}	54	58.101	7
Cifar10	3.072	50.000	10
SVHN	3.072	73.257	10
Sensorless	84	58.509	11

cover various problems such as intrusion detection or computer vision[5]. In some cases, the high number of elements in some datasets places them out of reach for the ReliefF original version, since their processing would take several weeks even using 12 computing cores. As a result, we used reduced versions of the large datasets for our first experiment which compares the execution times of the original ReliefF with ReliefF-LSH and DiReliefF. We reduced these datasets by taking only their top N elements. In particular, $Year_{small}$ is the top 10% of the YearPredictionMSD [52] dataset, $Higgs_{small}$ contains the top 0.5% of the *Higgs* dataset (55.000 samples), $Epsilon_{small}$ consists of the top 10% (50.000 elements) of Epsilon [139], $KDD99_{small}$ contains the top 1% (48.984 elements) of KDD99 [99] and CT_{small} is the top 10% of the CoverType [52] dataset. Nonetheless, to assess the suitability of our method for processing large datasets, we used the full *Higgs* dataset in our second experiment.

4.4.2. Methodology

The first experiment consisted in comparing the results obtained with ReliefF-LSH with those obtained by DiReliefF and with the ReliefF exact version. We used three

[5]All datasets are publicly available for download at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/` except KDD99, which can be downloaded from `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

different measures for this purpose. First, to gain insight on the time efficiency of each method, we measured the execution time needed to process each dataset. Then we set to establish the accuracy of the obtained results. To assess how exact the retrieved ranking of attributes is, we used the recall measure at various thresholds. Since the final purpose of the ReliefF ranking is obtaining a subset of attributes that are relevant, comparing the obtained subsets gives a clear account of the effectivity of the approximate method. Given a selection level t, a ranking $\mathcal{E}$ of attributes calculated by the exact ReliefF and a ranking $\mathcal{A}$ of attributes calculated by the approximate algorithm, we define the recall as:

$$recall(t) = \frac{|\mathcal{E}.\text{FIRST}(t) \cap \mathcal{A}.\text{FIRST}(t)|}{t} \tag{4.2}$$

where $\mathcal{X}.\text{FIRST}(t)$ represents the first t elements in list $\mathcal{X}$. Additionally, to further explore the accuracy of the obtained rankings, a distinction between sets of attributes that contain the same number of wrong selections need to be made. Due to the nature of ReliefF, the selection is made by sorting the attributes by weight and selecting the top N. Depending on the characteristics of the dataset, many attributes may have similar weights, and selecting one of these similar attributes will not have as adverse an effect as selecting an attribute with a much smaller weight. To quantify the importance of the mistakes made at each selection level we used a difference in total weight of the selected attributes, defined as

$$WD(t) = -(\sum_{a \in \mathcal{E}.\text{FIRST}(t)} \mathcal{E}(a) - \sum_{a \in \mathcal{A}.\text{FIRST}(t)} \mathcal{E}(a)) \tag{4.3}$$

where $\mathcal{E}$ represents the attribute ranking computed by the exact ReliefF, $\mathcal{A}$ represents the attribute ranking computed by the approximate method being measured and $\mathcal{E}(a)$ represents the weight assigned to attribute a by the exact ReliefF. The sign change is a merely esthetic choice to obtain a positively valued measure that should be minimized.

Additionally, both DiReliefF and ReliefF-LSH contain aleatory steps and, therefore, yield non-deterministic results, that is, the rankings obtained can differ between runs. To dampen the effect of randomness in the measurements we represent the average value over four separate executions for both methods. In the case of ReliefF-LSH, the random process included in VRLSH makes its execution time also vary slightly in different runs, but this is also mitigated by listing the average execution time over four separate executions for both methods.

Finally, DiReliefF defines how exact the computed ranking will be by establishing a sampling level which is selected by the user. We used 1000 samples in all experiments

with DiReliefF, in an attempt to obtain the most precise ranking possible without using more time than the exact ranking computation in any case.

4.5. Experimental results

As stated in Section 4.3, VRLSH needed to be modified to be able to tackle classification problems and ensure that all elements are linked to their k nearest neighbors of each class. This constitutes a challenge for the original algorithm that is magnified when the number of classes grows. For this reason we split our first set of experiments in two groups: 1) regression and binary classification datasets and 2) multiclass classification datasets.

4.5.1. Regression and binary classification

First we compared the execution times taken to process each dataset, which are depicted in Figure 4.2. Notice that DiReliefF does not support regression datasets and therefore no results are listed for DiReliefF for the $Year_{small}$ dataset. It is apparent that the execution times of ReliefF-LSH are always dramatically lower than those of its exact counterpart. When compared to DiReliefF, the execution time of ReliefF-LSH is significantly lower for the dataset with higher dimensionality, while it is slightly higher for the $Higgs_{small}$ dataset, which only has 28 attributes. This effect, exclusive to small datasets, can be explained because the overhead created by the hashing and grouping steps in VRLSH is noticeable only in small and low dimensional datasets, as is the case with $Higgs_{small}$, but becomes negligible in real life scenarios when the dataset is high dimensional and therefore each pairwise comparison is more costly or when the dataset contains a high number of elements, and the time saved by the number of pairwise comparisons avoided by ReliefF-LSH greatly surpasses the mentioned overhead.

Figures 4.3 and 4.4 depict the recall and weight error, respectively, at various threshold selection levels for both the approximate algorithms. The results obtained by ReliefF-LSH are clearly superior to those achieved by DiReliefF on both $Epsilon_{small}$ and $Higgs_{small}$. It is worth noting that in the case of $Epsilon_{small}$ the time invested in computing the ranking was notably smaller when using ReliefF-LSH and still the results obtained are clearly superior. The results for ReliefF-LSH on the regression

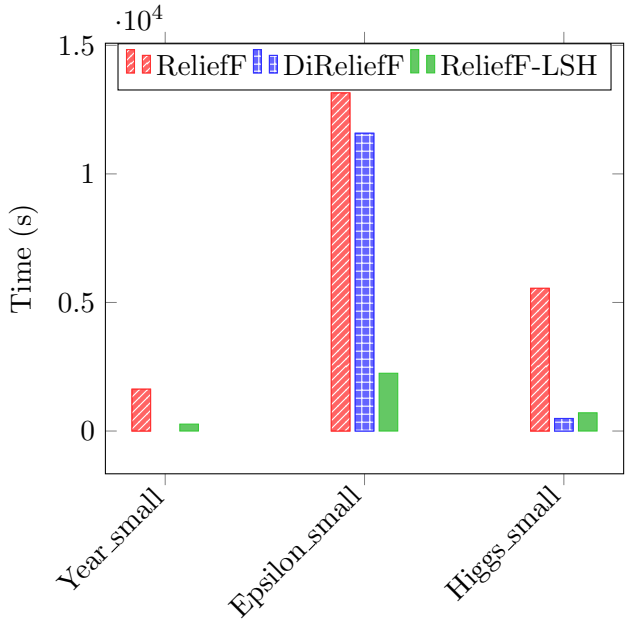


Figure 4.2: Execution times of ReliefF, DiReliefF and ReliefF-LSH for datasets $Year_{small}$, $Epsilon_{small}$, $Higgs_{small}$

dataset are also good, recalling perfectly the first 5 attributes and retrieving the rest of the list with good accuracy (the least accurate recall is obtained when selecting 15 attributes and is a still high value of 0.82, i.e. 12 or 13 correct attributes out of 15 depending on the execution, with a weight error of only $5 * 10^{-5}$).

4.5.2. Multiclass datasets

In the case of multiclass datasets, the execution times shown in Figure 4.5 reflect that ReliefF-LSH is significantly faster than the exact version except in the case of the smaller datasets, for which the previouly mentioned overhead introduced by ReliefF-LSH becomes apparent, but these cases should be considered a product of using small datasets for this experiment. As the dataset size and/or dimensionality grows, the time reduction becomes larger. In constrast with the case of binary classification datasets, DiReliefF is clearly faster than ReliefF for the smaller datasets, showing that the multiclass scenario requires more effort of ReliefF-LSH. Nevertheless, the available implementation of DiReliefF is unable to process *Cifar10* and *SVHN*, which are more computationally intensive, because its approach requires more memory than is available

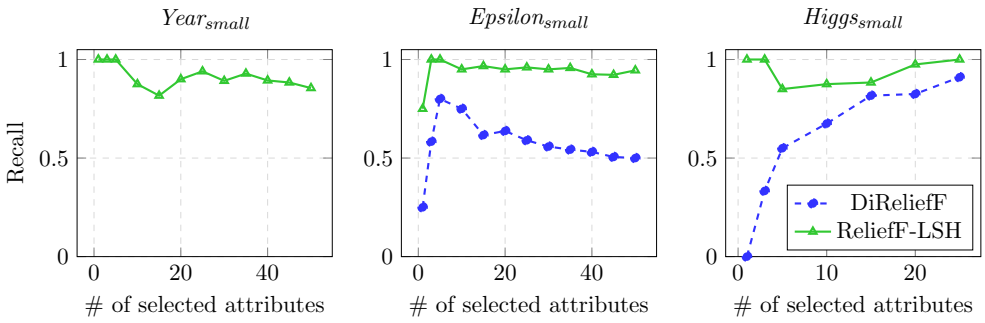


Figure 4.3: Recall obtained for datasets $Year_{small}$, $Epsilon_{small}$ and $Higgs_{small}$.

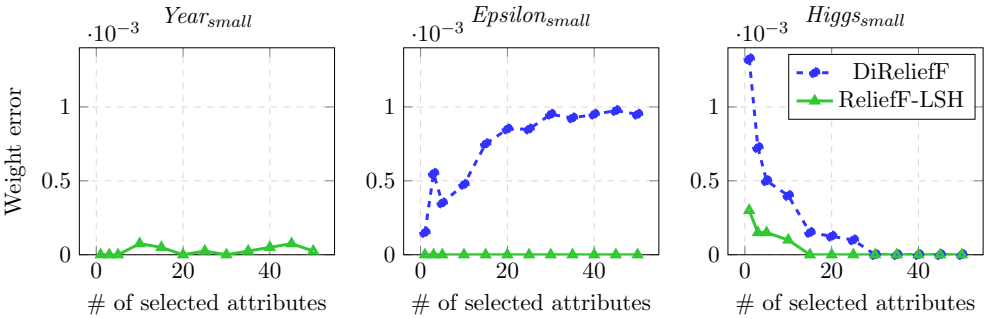


Figure 4.4: Weight error for datasets $Year_{small}$, $Epsilon_{small}$ and $Higgs_{small}$.

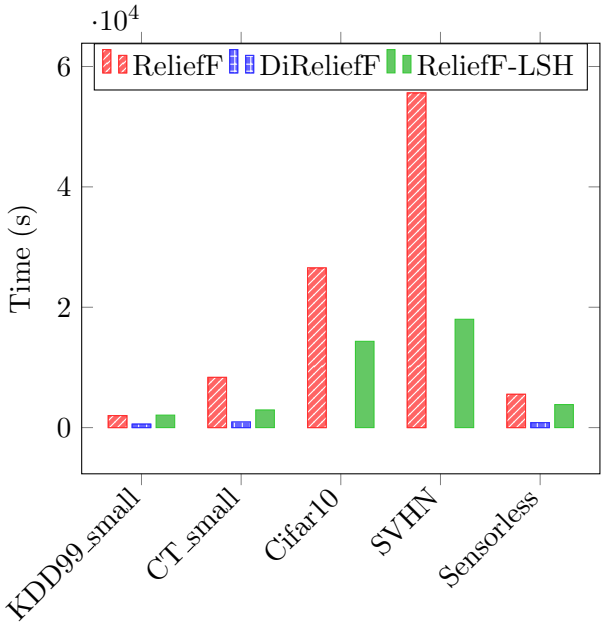


Figure 4.5: Execution times of ReliefF, DiReliefF and ReliefF-LSH for datasets $KDD99_{small}$, CT_{small}, *Cifar10*, *SVHN* and *Sensorless*.

in the computational nodes, regardless of the number of partitions of the distributed task used. In terms of accuracy, Figures 4.6 and 4.7 show that the results obtained by ReliefF-LSH are very good, although in the case of CT_{small} DiReliefF obtains better results for selection levels between 3 and 30.

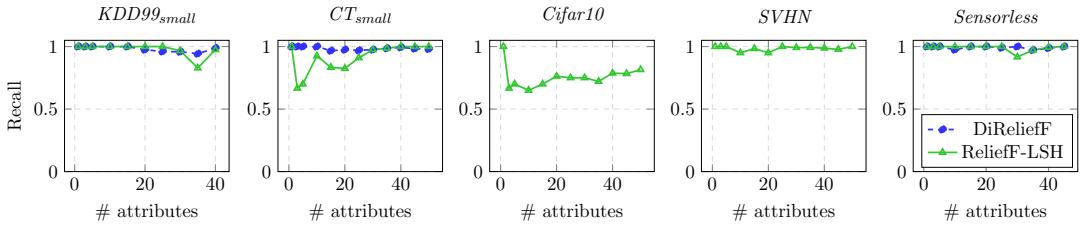


Figure 4.6: Recall obtained for datasets $KDD99_{small}$, CT_{small}, *Cifar10*, SHVN and *Sensorless*.

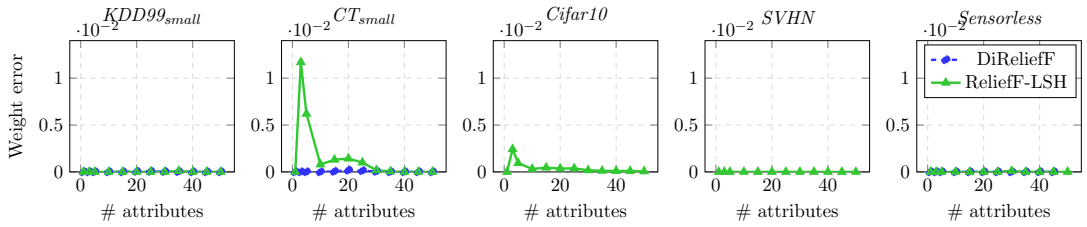


Figure 4.7: Weight error for datasets $KDD99_{small}$, CT_{small}, *Cifar10*, *SVHN* and *Sensorless*.

4.5.3. Scalability

Finally, we performed an experiment to study the scalability of ReliefF-LSH. To this end, we used the full version of the *Higgs* dataset, containing 11 million elements, and computed the attribute ranking repeatedly, using for each execution a growing number of computing cores. The resulting execution times listed in Table 4.3 show that ReliefF-LSH is able to handle datasets that are completely out of reach for the exact version of the algorithm. While the exact ReliefF needed 5.550 seconds to process $Higgs_{small}$ (i.e. 0.5% of the examples in the full *Higgs*), ReliefF-LSH was able to process the whole dataset (200 times larger) in only 48.283 seconds using the same amount of computing cores. It is worth noting that, since the computational complexity of the original ReliefF is quadratic, the expected execution time would be of the order of 10^7 seconds, which is unusable in practice. These results also show that, thanks to the fact that many of the calculations performed by ReliefF-LSH can be performed independently in parallel, the computational nodes that are involved in the calculation are efficiently used, which results in an inversely proportional relation with slope close to -1 between the number of nodes and the execution time, as desired.

4.6. Conclusions

This chapter presents an adaptation of the popular feature selection algorithm ReliefF to allow the processing of large datasets, demonstrating the usefulness of computing approximate data structures as a means to increase the scalability of machine learning algorithms. The currently available alternatives to the exact ReliefF are either unable to handle large-scale datasets or are restricted to a particular type of input

# units	Time (s)	Scan rate	Ops/s	Speed-up
1	48,283	$2.42 \cdot 10^{-3}$	$3.03 \cdot 10^{6}$	1.00
2	19,864	$1.71 \cdot 10^{-3}$	$5.19 \cdot 10^{6}$	1.72
4	15,402	$2.54 \cdot 10^{-3}$	$9.96 \cdot 10^{6}$	3.29

Table 4.3: Scalability vs number of computational units for applying ReliefF-LSH on the full *Higgs* dataset. The speed-up listed is the ratio between the operations per minute obtained and the operations per minute performed with a single computational unit (12 cores).

data. In our approach, the costly step of computing the kNN graph is approximated using the a variation of the VRLSH algorithm, greatly reducing execution time while maintaining accuracy. The resulting algorithm, called ReliefF-LSH, is implemented in the distributed computing framework Apache Spark and available for download. We report experiments that demonstrate the adequacy of the method and its scalability. These experiments show that ReliefF-LSH can be used to process datasets that are out of reach for the exact ReliefF and that it shows superior performance when compared to the available approximate alternative, DiReliefF. Moreover, the memory efficiency of ReliefF-LSH allows it to handle very high dimensional datasets that are out of reach for DiReliefF. Also, ReliefF-LSH does not impose restrictions on the data that it can handle and supports regression datasets, so it is an apt substitute for ReliefF in any problem, in contrast with the previously available alternatives. Furthermore, the lack of additional hyperparameters of our method avoids the need for tuning steps that are very costly for large datasets. Finally, ReliefF-LSH can be configured to prioritize precision over execution time or vice versa, enabling the users to obtain results according to their needs.

As future work, the behaviour of the algorithm when processing datasets containing several classes could be further analyzed in order to explore any possible improvements that could lead to a reduction of the computational effort needed.

CHAPTER 5

Ad-hoc models: Large Scale Anomaly Detection in Mixed Numerical and Categorical Input Spaces

5.1. Introduction

In this chapter we will explore how algorithms can be created to be scalable by design. This approach is more complex than the previously analyzed alternatives since it does not rely in a pre-existing algorithm or data structure. Instead, what is required is designing a model that can be trained efficiently regardless of the dataset size.

To test this approach we will focus on the problem of anomaly detection. An anomaly or outlier can be defined as "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism"[74]. Detecting anomalies is an old discipline for statisticians [53], denominated *outlier detection*. Since those days, this type of method has become increasingly important. Anomaly detection is especially useful in practical situations where the dataset is both numerous and contains unexpected events that carry the most important information. Several challenges stand in the way of developing a general technique for anomaly detection: the growing number of domains of application (detection of intrusions [96], surveillance [138], fraud [5], machine faults [44, 60, 112]) adds high variability to the proposed solutions, while the scarcity of labeled data from real-world processes [33, 36] makes it difficult to test the generalization of new solutions. Additionally, the data available in practice for building the model is usually unlabeled [36, 57, 136]. In addition, the regions of the input space that are likely to contain only non-anomalous elements can be very complex in nature. Therefore, deciding a prior shape for this region through the choice of a specific distribution or geometric shape is a potentially difficult task that can introduce bias, preventing the generation of a meaningful model.

Another major difficulty, on which we focus here, is the type of input data. The introduction of mixed numerical/categorical data can complicate the modeling of correlations between input variables. This means that many of the popular anomaly detection techniques: (a) can only deal with categorical or numerical data [144, 43, 4], (b) leave to the practitioner the responsibility of dealing with this issue through (non-formal) bespoke processes, or (c) introduce heuristic criteria to deal with mixed nature data [62, 119].

With this research, which was published in Information Sciences, we aim to explore a strategy to model anomaly detection problems in which the data are numerous and contain categorical and numerical input variables. We leverage that characteristic to design a model composed of two smaller parts, which reduces the computational requirements, therefore enhancing the scalability of the algorithm. We adopt a probabilistic view of the problem and deal with each kind of variable individually in order to approximate the joint probability measure function with a parametric model. This approach differs from the state of the art in this field in that, instead of departing directly from a heuristic concept of *outlierness* in mixed categorical/numerical spaces, it starts from a formal formulation of the problem in terms of a joint probability measure function approximation and adopts a parametric structure that makes this feasible to compute. This makes the proposed algorithm both theoretically and technically sound. The whole model is trained through a maximum likelihood objective function optimized with stochastic gradient descent. Therefore, the algorithm lends itself well to parallel computation, which allows the model to scale up to large datasets, both in terms of features and number of instances, making it an appealing option for big data applications. To demonstrate this, an implementation of the algorithm (denominated Anomaly Detector for Mixed Numerical and Categorical inputs, ADMNC) in the popular cluster computing framework Apache Spark is provided.[6]

Section 5.2 reviews related work in this area. Section 5.3 presents the formal framework and Section 5.4 introduces the parametric formulation of the problem. Section 5.5 reports a collection of experiments that show the properties of the proposed method in real datasets, as well as the definition of a synthetic dataset generator and further experiments with the resulting datasets. Section 5.6 summarizes the main conclusions and future work.

[6]The implementation of ADMNC in Apache Spark is available for download at `http://github.com/eirasf/ADMNC/`.

5.2. Related work

Numerous anomaly detection techniques have been developed, either from an application specific or a more general-purpose point of view. Anomaly detection application domains impose restrictions which dramatically determine the design of the algorithms. Consequently, the research in this area has yielded only a few general algorithms in recent years [86].

Anomaly detection approaches can be classified according to the nature of the input data. We have assumed that each instance can be described using a set of attributes. These can be of different types, such as binary, categorical or numerical. The nature of the attributes determines the applicability of anomaly detection techniques. Different statistical models and algorithms have been designed for numerical and categorical data [35]. Some anomaly detection models can only deal with categorical data [144, 43, 4], whereas numerical variables have been treated mainly through statistical parametric and non-parametric models [131, 19], geometrical approximations [112], using binary trees [105] and autoencoder neural networks [75, 61, 118]. In addition, there have been numerous efforts to deal with the problem of mixed numerical/categorical anomaly detection. Current approaches in this last group can be classified in one of the following abstract strategies:

- *Categorical space* techniques. These algorithms build an anomaly detection model specially devised for categorical variables and transform any numerical variable into a categorical space through a previous discretization phase. In this group we can find HOT [149], in which the set of outliers in a dataset is detected using a specially devised data structure called a hypergraph and a local test for outliers based on a frequent itemset counting strategy, and OutRank [115] that detects anomalies using random walks on an adjecency graph. Another approach consists in tackling the the problem using information theory concepts [76, 151] but, again, only categorical attributes are considered and numerical attributes need to be circumvented through discretization.

- *Metric-centered* techniques. These methods define an anomaly as a point which lies in a low density region in comparison with its neighborhood. They rely on a function that calculates the similarity between elements in the input space and so can be extended to the mixed numerical/categorical case and other types of structured data [132] through a tailored similarity function. LOF (Local Outlier

Factor) [30] can be considered the seminal work in this area. The basic criteria of these methods has also inspired subsequent improvements for high dimensional spaces [97] and improved density criteria such as [84]. LOCI (LOcal Correlation Integral) is a similar technique that improves on LOF, as it is able to detect outliers and also groups of outliers without user-required cut-offs [121]. These techniques present challenges in (a) devising effective similarity measures for mixed numerical/categorical input spaces and (b) scalability, since the similarity matrix needs to be computed before moving on to the detection phase.

- *Mixed-criteria* techniques. This group of algorithms tackles the nature of numerical and categorical data separately, by trying to design a criterion which encompasses the analysis of an element in both spaces. Solutions that tackle this problem using adaptations of supervised learning techniques like AdaBoost have been described [81], although its requirement for labeled samples prevents its use in the most common use cases. In this group we can also classify LOADED [62], which blends categorical-categorical, categorical-numerical and numerical-numerical correlations in a single criterion using frequent itemset concepts and local correlation matrices. This algorithm has the drawback of high execution times for high dimensional datasets, because although its computational complexity scales linearly with the number of data points, it scales quadratically with the number of numerical attributes, and grows even more computationally costly with the number of categorical attributes. Although a more efficient version named RELOADED was introduced in [119], this still requires more computational effort than more recent methods like ODMAD [92], which first identifies elements with an anomalous categorical part by finding unusual values or combinations of values, and then, for the elements not deemed as anomalies, computes the pairwise similarity of their numerical part to that of points sharing the same categorical values. This takes into account the relationship of the numerical part with each of the categorical values, but disregards any possible dependence on a combination of categorical values. Moreover, its computational complexity, even though less than that of the aforementioned RELOADED, still scales exponentially with the number of categorical attributes, limiting its use to datasets with few such variables. Another algorithm in this category is POD [154], but its reliance on k-nearest neighbor distances entails a computational complexity that renders it unsuitable for large datasets. There have also been efforts to provide a statistical foundation to this problem, like MITRE [107], which uses a generalized linear model with additional latent variables to model correlations and error. Large magnitudes of the error are then used to identify anomalies. Despite

being a sound model, its inference is computationally costly [49], particularly when the number of correlations modeled is high. Moreover, the method relies on the user providing the domain knowledge to identify explanatory and dependent variables; the alternative of assuming all variables to be dependent would drive the execution time up. These characteristics limit its scalability. Finally, the use of deep belief networks for anomaly detection has been proposed in a method named MIXMAD [49] but, although this method scales well in terms of number of variables, its computational complexity makes it unfit for processing large datasets.

In this research we focused on devising a probabilistic strategy able to solve anomaly detection problems, where input elements belong to an input space which mixes numerical and categorical variables. The proposed algorithm is closely related to *mixed-criteria* techniques in the sense that correlations between categorical and numerical variables are explicitly modeled. In addition, the method overcomes the scalability problems of previous approaches, and can tackle both large-scale and high-dimensional problems.

5.3. Basic formulation

We aim to obtain the best fit of a probability measure function for the data under normal conditions. In subsequent monitoring of new data elements, these are assigned a score and those whose score does not reach a pre-specified threshold are considered anomalies. Formally, given a dataset $\mathcal{D} = \left\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{|\mathcal{D}|}\right\}$, we need to estimate $\mathrm{P}(\boldsymbol{x})$.

If we have a homogeneous set of variables, this problem can be reduced to probability density function (pdf) parameter learning. For instance, if all the variables under normal conditions can be well represented by a Gaussian, we can directly elicit the moments of a complete Gaussian from a dataset by maximizing the likelihood of the data, or alternatively, follow a Bayesian approach with an adequate prior.

However, in many situations, datasets have a mix of categorical and numerical variables. For ease of reference we rewrite the dataset as

$$\mathcal{D} = \left\{(\boldsymbol{x}_0, \boldsymbol{y}_0) \ldots, (\boldsymbol{x}_{|\mathcal{D}|}, \boldsymbol{y}_{|\mathcal{D}|})\right\},$$

where $\boldsymbol{x}_i$ is the numerical or continuous component, and $\boldsymbol{y}_i$ is the categorical or nominal

counterpart of the i-th instance of the dataset.

In this case, the model should individually take into account the nature of the two types of variables. In this work, we propose the following heuristic factorization of the pdf:

$$P(\boldsymbol{y}, \boldsymbol{x}) = P(\boldsymbol{y}|\boldsymbol{x})P(\boldsymbol{x}). \tag{5.1}$$

With this partition of the pdf, we can adopt a suitable technique for estimating the parameters of each part independently, while accounting for possible interactions between the two parts.

It was previously mentioned that an incorrect assumption about the shape of the underlying distribution could induce bias that could harm the accuracy of the model. Nevertheless, since we have to make that decision, we try to adopt the most flexible model that is still computable in a closed form. We heuristically adopt a flexible parametric approach for both the conditioned probability of the categorical variables and the marginal probability of the numerical variables. In Section 5.5 we test the adequacy of these heuristic assumptions with results obtained in experiments on several datasets.

Below we describe the models used for each part of the proposed factorization.

5.3.1. Numerical part

Mixture models are the most flexible parametric option for estimating this marginal. The Gaussian mixture model (GMM) is the first appealing option due to its closed form parameter update formulas. In particular, we used the existing implementation of GMM available in Apache Spark, which uses the expectation-maximization algorithm to induce the maximum-likelihood model for the given set of samples. Since GMM is very sensitive to the initial values of the means of the Gaussians, we first performed KMeans clustering on a small sample from the dataset. We then used the resulting centroids as the initial values for these means and computed the empirical standard deviations of the clusters to obtain the initial diagonal covariance matrix. For the KMeans algorithm we used the default implementation included in Spark [11].

5.3.2. Categorical part

The categorical part of each element in the dataset can be assumed, without any loss of generality, to be a binary vector

$$\boldsymbol{y} = (y^0, \ldots, y^k), \;\; y^j \in \{0, 1\}$$

which can be obtained by one-hot encoding of the categorical variables of the element.

With this in mind, we can estimate $P(\boldsymbol{y}|\boldsymbol{x})$ as

$$P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}) = \prod_{j=0}^{k} P(Y = y^j|(\boldsymbol{x}, \boldsymbol{m}_j), \boldsymbol{w}) \tag{5.2}$$

where $\boldsymbol{m}_j$ is just the one-hot representation of "j", and $\boldsymbol{w}$ is a parameter to be learned from the dataset using a logistic regression (LR) model. Specifically, the LR model computes the conditional probability by means of

$$P(Y = y^j|(\boldsymbol{x}, \boldsymbol{m}_j), \boldsymbol{w}) = \frac{1}{1 + e^{-(2y^j - 1)\langle \boldsymbol{w}, (\boldsymbol{x}, \boldsymbol{m}_j)\rangle}} \tag{5.3}$$

where the notation $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ represents the dot product of $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, the learning process is reduced to obtaining the optimal parameters for both the LR model (which approximates the conditional probability of the categorical part of the elements) and the mixture model (for the marginal pdf of the numerical part of the elements). In the next section we show how a maximum likelihood strategy can effectively carry out the optimal parameter search.

5.4. Maximum likelihood parameter estimation

Let $\mathcal{D}$ be a dataset of points $(\boldsymbol{x}, \boldsymbol{y})$. Taking into account the expression (Eq. 5.1) for the factorization of the joint pdf, the data log-likelihood has the following form:

$$\log L(D) = \sum_{i=1}^{|D|} \log P(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w}) + \sum_{i=1}^{|D|} \log P(\boldsymbol{x_i}) \tag{5.4}$$

It is important to note that, since the first part of the data log-likelihood is completely independent of the parameters of the second addend, both optimization processes can be run in parallel and so exploit the structure of each problem separately. This can be achieved separately for each part and then combined in a unified algorithm.

Using the first term of (Eq. 5.4), we aim to obtain the parameters $\boldsymbol{w}^*$ that maximize

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}} \left\{ \sum_{i=1}^{|D|} \log P(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w}) - \nu \frac{\boldsymbol{w}^2}{2} \right\} \tag{5.5}$$

where ν is a hyperparameter that controls an optional regularization term that is proportional to the norm of the parameter vector, added to cope with possible overfitting issues.

This is a well-known convex optimization problem that can be solved using a stochastic gradient descent algorithm [25].

For the implementation of SGD in our experiments, at each iteration t, we updated the learning rate, as it is usually done, according to this formula:

$$\lambda_t = \frac{\lambda_0}{1 + \lambda_s(t-1)} \tag{5.6}$$

Therefore, controlling the constants λ_0 and λ_s it is possible to tune the convergence of the algorithm, which is crucial in order to obtain a good model.

The implementation in Apache Spark parallelizes the minibatch step in the SGD process, potentially accelerating the whole process if there are several computational units available. The code is available for download at `http://github.com/eirasf/ADMNC`.

5.5. Experimental settings and results

In this Section we describe a set of experiments comparing our algorithm with other state-of-the-art methods in terms of both accuracy and time. The first subsection describes methodological issues related to how we measured the scores of the anomaly detection algorithms and the datasets and algorithms used in the experiments. We also describe a synthetic dataset generator and introduce the synthetic datasets used in our experiments. We then report performance results obtained with real world datasets and next make a comparison of the scalability of all algorithms. Finally, we report the results of experiments that show the effect of the complexity of the dataset on the results obtained by each algorithm.

5.5.1. Methodology

To measure the performance of the different methods we tried to simulate a real world environment. Thus, the learning algorithms were trained using only non-anomalous samples. The models thus obtained were then tested with a mixture of anomalous and non-anomalous samples. The performance scores were measured computing the area under the ROC (*Receiving Operator characteristic*) curve (AUC).

We account for the fact that in real situations we typically do not have anomalous samples to train learning algorithms. Additionally, this setup allows the use of datasets with an arbitrary fraction of anomalies, which is useful for experimental purposes given the scarcity of datasets. Therefore, we were able to use binary classification tasks, selecting one of the classes as anomalous, as is common practice [118, 92, 107, 49]. With this transformation, the obtained anomalies comply with the definition given in the Introduction.

To test the strength of our algorithm, we compared the scores with those achieved by the following state-of-the-art algorithms. First, we considered two algorithms that make a differential treatment of numerical and categorical variables: the well-known LOF and LOCI algorithms using Euclidean, Jaccard and Hamming distances, for which we used a Matlab implementation[7]. Additionally, we compared the results with other anomaly detection algorithms that do not differentiate between numerical and categorical variables. For this purpose we selected one-class support vector machine (OC-SVM) (with a radial basis function—RBF— and linear kernels), for which we used the Matlab interface of LibSVM[8]. It is worth noting that the complexity of this family of algorithms approaches quadratic time regarding the number of samples in favorable cases [27]; this makes them poor candidates for handling large amounts of data. Therefore, we also tested DOC-SVM [34], which is a distributed version of the same algorithm that can handle large datasets by splitting them, also implemented in Matlab[9]. Finally, we included in testing the recent iForest [105], implemented in R[10]; and PA-I [113][11], also written in Matlab. For those algorithms that do not make a distinction

[7] https://github.com/jeroenjanssens/lof-loci-occ

[8] https://www.csie.ntu.edu.tw/ cjlin/libsvm/#matlab

[9] To the best of the authors knowledge, there are no other implementations of OC-SVM that can use non-linear kernels with a time complexity inferior to $\mathcal{O}(n^2)$.

[10] https://sourceforge.net/projects/iforest/

[11] We tried to include LOADED in the comparison but despite our best efforts we could not find an implementation; also, the code obtained by strictly following the description provided by the authors resulted in an algorithm that performed very poorly.

Table 5.1: Hyperparameters explored for each algorithm in the experiments reported in this section

Algorithm	Hyperparameter range
LOF	P $\in \{0.01, 0.03, 0.05\}$, K $\in \{2, 3, 5, 10\}$,
	(only Jaccard and Hamming) $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$
LOCI	$\alpha \in \{0.1, 0.3, 0.5\}$,
	(only Jaccard and Hamming) $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$
OC-SVM	$\nu \in \{0.01, 0.05, 0.1, 0.3\}$,
	(only RBF) $\gamma \in \{0.01, 0.05, 0.1, 1, 3, 10\}$
iForest	rFactor $\in \{0.01, 0.1, 0.5, 0.8, 1\}$,
	Row Samples $\in \{0.01, 0.025, 0.05, 0.1\}$
PA-I	$\sigma \in \{1, 2, 3, 4, 5\}$, C $\in \{0.01, 0.025, 0.05, 0.1\}$,
	R $\in \{0.97, 0.99\}$
ADMNC	$\nu \in \{0.1, 1, 10, 100, 1000\}$,
	$\lambda_0 = 1$
	$\lambda_s \in \{0.0001, 0.001, 0.01, 0.1, 1\}$
	Number of gaussians $\in \{2, 4\}$

between categorical and numerical variables, the categorical variables in the datasets were transformed using one-hot encoding.

The algorithms used for comparisons typically have several hyperparameters. To find the best combination for each dataset, we performed a cross-validation (CV) with 5 folds for each possible set of hyperparameter values. For each fold, the algorithm was trained with the non-anomalous examples from the training set and evaluated on all the samples of the test set. The hyperparameters explored are listed in Table 5.1. The scores discussed in Subsection 5.5.2 are the best average AUC obtained in the CV procedure.

Below we describe the datasets employed in the experiments.

Table 5.2: Real datasets used for the comparative study. The *Anomaly ratio* is the quotient of anomalous examples over the number of examples. The numbers of numerical / categorical features are in parentheses.

Dataset	# Features(N/C)	# Instances	Anomaly ratio
Arrhythmia (Arrhyth)	278 (271/7)	420	0.4357
German Credit (GC)	20 (7/13)	1000	0.3000
Abalone 1-8 (Ab. 1)	10 (7/3)	4177	0.3368
Abalone 9-11 (Ab. 9)	10 (7/3)	4177	0.3167
Abalone 11-29 (Ab. 11)	10 (7/3)	4177	0.3464
CoverType (CT)	12 (10/2)	286048	0.0096
KDD99 (full) (KDD)	41 (32/8)	4898431	0.8000
KDD99 (10%) (KDD10)	41 (32/8)	494021	0.8000
KDD99 (http) (KDDh)	40 (32/7)	623091	0.0065
KDD99 (smtp) (KDDs)	40 (32/7)	96554	0.0123
IDS	27 (8/19)	2071657	0.0333

5.5.1.1. Real datasets

As stated above, it is very difficult to come by real-world datasets with labeled anomalies. Thus, the datasets used are commonly employed for classification tasks, but re-purposed for anomaly detection. They were downloaded from the UCI Machine Learning Repository [102].

The datasets are reported in Table 5.2, where we distinguish between two groups. The first group includes small-medium sized datasets: *Arrhythmia (Arrhyth)*, *German Credit (GC)*, and 3 versions of *Abalone*. These versions were built choosing different classes as anomalous and non-anomalous; thus, *Abalone 1-8 (Ab. 1)*, *Abalone 9-11 (Ab. 9)* and *Abalone 11-29 (Ab. 11)* were obtained using, respectively, classes 1, 9 and 11 as non-anomalous and classes 8, 11 and 29 as anomalous.

The second group of datasets, with a larger number of samples, contains versions of *CoverType* [20] and *KDD99* [78] datasets. To transform *CoverType (CT)*, instances of

class 2 were assumed to be normal, while instances of class 4 were selected as anomalies. With respect to *KDD99* it should be noted that, although there has been some criticism that it does not accurately represent an intrusion detection task, those discrepancies have no impact on the validity of the dataset for our purposes. Although the anomaly condition of the elements that are labeled as such may be disputed with the argument that the dataset constitutes a biased sample, our aim is to identify a minority set of instances that were generated by a different process than the rest. Note also that this dataset has been used extensively for anomaly detection [33, 62, 119, 105, 81, 130]. For this research we transformed *KDD99* into an anomaly detection dataset by assuming that attacks of any class are anomalies. To cope with the lack of labeled large datasets, as has been done in similar studies [105] three additional datasets were obtained by transforming the full *KDD99*. Thus, (1) *KDD99 (10%)* is the reduced dataset available at the UCI Repository, which contains only 10% of the instances, (2) *KDD99 (http)* is the result of filtering the full dataset to keep only http connections, and, analogously, (3) *KDD99 (smtp)* only contains smtp connections.

Finally, we used the *IDS 2012* dataset [135], which covers the same domain as *KDD99* but solving its weak points. Again, we consider any sort of attack as an anomaly, as opposed to normal traffic.

5.5.1.2. Synthetic dataset generator

In order to be able to exhaustively test the anomaly detection methods on datasets of diverse sizes and difficulty levels, we decided to create a synthetic dataset generator that could be parametrized. Previous works in the field, which have stated the need for such a publicly available generator to provide the data on which to perform experiments, have resorted to devising their own ad-hoc generators [92, 107], a situation which complicates comparisons across studies. Our configurable generator (available for download at `http://github.com/eirasf/ADMNC/`) offers that capability to data science practitioners and it can be used by researchers in the field to construct benchmarks.

The data generated by this method, while inspired by the data that would be created by a set of users interacting with a set of documents, was simplified to achieve a more general dataset. Each element of the dataset consists of a random binary vector, which symbolizes a bag-of-words representation of a document. Another binary vector

and a numerical vector are generated from the existing random vector using a set of rules that account for statistics regarding the viewers of said document. Note that in a dataset designed this way, the numerical variables depend on the binary variables, which is the opposite assumption of our model that the categorical variables depend on the numerical variables. This design choice is intended to test the reliability of our model in detecting dependencies between the variables.

To obtain the dataset first we must choose the size of the vectors. The generator then creates two sets of random rules, one used to produce a binary vector and the other used to produce a numerical vector. Lastly, generated for the dataset are as many elements as requested by the user, each of which consists of a random binary vector together with the vectors resulting from the application of the mentioned sets of rules.

In the equations, the generated dataset $\mathcal{D}$ is described as a set of vectors over a set of indices

$$\mathcal{D} = \{(\boldsymbol{u}_i, \boldsymbol{b}_i, \boldsymbol{n}_i) \mid i \in I\} \tag{5.7}$$

where $\boldsymbol{u}_i$ is a binary vector generated at random with uniform probability for each component and where the jth component in $\boldsymbol{b}_i$ is generated from $\boldsymbol{u}_i$ by a function f_j

$$\boldsymbol{b}_{ij} = f_j(\boldsymbol{u}_i) \tag{5.8}$$

which assigns 1 with a probability proportional to the fraction of conditions of the rule $\boldsymbol{r}_j$ satisfied by $\boldsymbol{u}_i$

$$f_j(\boldsymbol{x}) : \{0,1\}^n \to 0, 1 = a \mid a \sim Be\left(\frac{\langle \boldsymbol{r}_j, \boldsymbol{x}\rangle}{|\boldsymbol{r}_j|}\right) \tag{5.9}$$

where $Be(x)$ is a Bernoulli distribution with probability x. The jth component in $\boldsymbol{n_i}$ is sampled from a normal distribution whose mean and standard deviation are dictated by a function g_j

$$\boldsymbol{n}_{ij} \sim N\left(g_j((\boldsymbol{u}_i, \boldsymbol{b}_i)), \frac{1}{1 + g_j((\boldsymbol{u}_i, \boldsymbol{b}_i))}\right) \tag{5.10}$$

which simply indicates the fraction of conditions in rule $\boldsymbol{s}_j$ that the concatenation of $\boldsymbol{u}_i$ and $\boldsymbol{b}_i$ meets:

$$g_j(\boldsymbol{x}) : \{0,1\}^n \to \mathbb{R} = \frac{\langle \boldsymbol{s}_j, \boldsymbol{x}\rangle}{|\boldsymbol{s}_j|} \tag{5.11}$$

The rule sets R and S are randomly generated at the beginning of the generation process and kept constant for all elements. R must hold a vector $\boldsymbol{r}_j$ for each component

Table 5.3: Families of synthetic datasets used for the comparative study. NV represents the number of variables affected by each anomaly. In Synth1 and Synth2, the number of samples and NV vary, respectively, with the values $i \in |0, 5|$

Dataset	# Samples	$\|u\|$	$\|b\|$	$\|n\|$	NV	Anomaly ratio
Synth1	$100 * 5^i$	20	10	100	4	0.5
Synth2	500	20	10	100	2^i	0.5

in $\boldsymbol{b}$ and, analogously, S must contain as many rules as components are desired in $\boldsymbol{n}$. Each rule simply consists of a binary vector as long as $\boldsymbol{u}$ and $(\boldsymbol{u}_i, \boldsymbol{b}_i)$, respectively, indicating which components are affected by the rule.

After $\mathcal{D}$ is generated, a fraction of the elements are turned into anomalies by altering a number of its randomly selected components. Binary components are altered by flipping their value, while numerical components are incremented with a value randomly sampled from a standard normal distribution. When the dataset is constructed this way, the number of variables affected by an anomaly acts as a proxy for dataset difficulty: intuitively, the fewer components that are altered by an anomaly, the more difficult it is to spot it.

With this methodology, we created two datasets for our experiments, as described on Table 5.3: in the first dataset we left the number of variables affected by an anomaly as a parameter, to study the effect of dataset difficulty on the algorithms; while in the second dataset we varied the number of elements to analyze the scalability of the different methods.

5.5.2. Results and discussion

The results obtained for the small-medium sized datasets are shown in Table 5.4. It is hard to draw a conclusion from this table regarding the best algorithm. In fact, using the Nemenyi post-hoc test [47] with $\alpha = 0.05$, the scores achieved by ADMNC cannot be significantly differentiated from those for any of the other algorithms; see Figure 5.1. Consequently, we expanded on the study of these algorithms with further experiments.

Table 5.4: AUC of the proposed approach (ADMNC) compared with LOF and LOCI algorithms (using Euclidean (E), Hamming (H) and Jaccard (J) distances), OC-SVM using linear (SVM-L) and RBF (SVM-R) kernels, DOC-SVM, iForest and PA-I. The best results for each dataset are highlighted in boldface.

	Arrhyth	*GC*	*Ab. 1*	*Ab. 9*	*Ab. 11*
LOF (E)	0.6670	0.5847	0.6936	0.6029	0.5927
LOF (H)	0.6983	0.5646	0.6936	0.6029	0.5927
LOF (J)	0.7010	0.5681	0.6936	0.6029	0.5927
LOCI (E)	0.6735	0.5917	0.8524	0.6756	0.7155
LOCI (H)	0.7141	0.5709	**0.8526**	0.6856	0.7155
LOCI (J)	0.7144	0.5663	0.8512	**0.6874**	0.7159
SVM-L	0.6794	0.5697	0.7944	0.6140	0.7670
SVM-R	**0.7479**	**0.6452**	0.8121	0.6756	0.7448
DOC-SVM (RBF)	0.6530	0.5419	0.5561	0.5748	0.5502
iForest	0.7133	0.5792	0.6519	0.5966	0.5984
PA-I	0.6932	0.6216	0.8498	0.6511	0.7113
ADMNC	0.6140	0.6276	0.8453	0.6120	**0.7930**

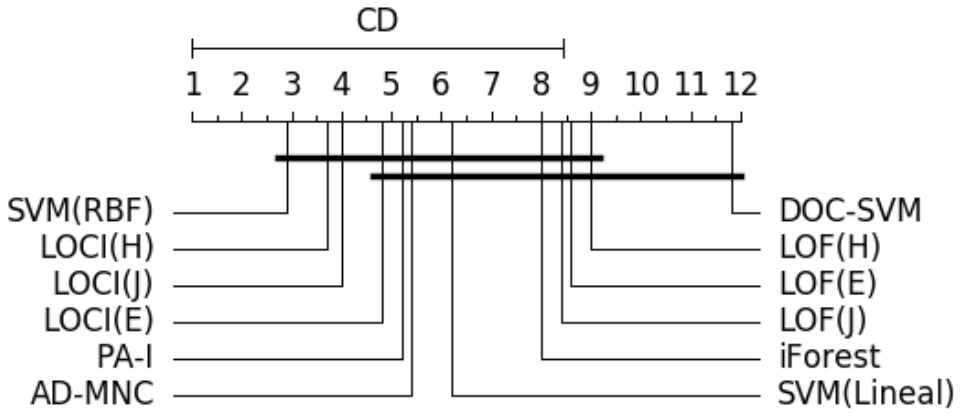


Figure 5.1: Nemenyi test with the scores for Table 5.4. ADMNC is in the group of the best algorithms, although no algorithm is significantly better than the others.

A few larger datasets were also used. Here the fact that LOCI and LOF are quadratic algorithms regarding the number of examples makes them far more computationally costly than the other algorithms, and thus unable to manage large datasets. LOF and LOCI were therefore excluded from this comparison. Therefore, in the context of large-scale learning only ADMNC and algorithms that make no distinction between categorical and numerical variables can be used.

The results obtained with these larger datasets are shown in Table 5.5. To overcome computational difficulties, we performed these experiments using only 2 folds instead of 5. In addition, for all algorithms the best parameters for *KDD* and *IDS* were determined for their respective variants with only 10% of elements and those same values were used for all the variants. Four of the algorithms struggled with the two largest datasets: (1) the implementation of iForest in R could not handle the memory requirements of *KDD* or *IDS*, (2) PA-I took more than 10 hours to explore a single hyperparameter combination with *KDD*, (3) OC-SVM (RBF) required quadratic memory space (in terms of number of samples), which made handling the full *KDD* or *IDS* datasets impossible, and (4) even though the distributed nature of DOC-SVM allows it to process arbitrarily large datasets, the reliance on Java of its Matlab implementation meant it failed when trying to split a large dataset and, consequently, it could not process any of these datasets. With those methods unable to handle large datasets, OC-SVM with a linear kernel and ADMNC rendered results very favorable to our method. Additionally, the parallel implementation of ADMNC made handling large datasets much easier. It is worth noting that, for the largest datasets OC-SVM-L took several hours to compute, while ADMNC took just a few minutes. Even though they are implemented in different platforms, we illustrate this difference in scalability with our next experiment.

Since there was great disparity in the computational costs of the tested algorithms, we performed additional experiments to more thoroughly assess their scalability in terms of dataset size. To be able to adequately run this test and due to the aforementioned lack of real datasets with the necessary characteristics, we used the synthetic datasets described in Section 5.5.1.2 to allow us to control the size and difficulty of the dataset. The process used to generate these datasets is described in Section 5.5.1.2.

The results of testing the algorithms on the *Synth1* family of datasets, shown in Figure 5.2, show that our method is clearly superior in terms of scalability of the dataset size. Since the compared algorithms are implemented in diverse platforms and, therefore, their absolute times cannot be compared, times are presented as the ratio between the time taken to process 100 elements with that algorithm and the time

Table 5.5: AUC of the proposed approach (ADMNC) compared with OC-SVM using linear (SVM-L) and RBF (SVM-R) kernels, iForest, and PA-I for large datasets. In all cases "-" indicates that results could not be obtained due to excessive time and/or memory requirements.

	CT	*KDD10*	*KDD*	*KDDh*	*KDDs*	*IDS*
SVM-L	0.9975	0.8712	0.8806	0.9139	0.9959	0.7300
SVM-R	0.9988	0.9965	-	0.9961	0.9859	-
iForest	0.9652	-	-	-	-	-
PA-I	**0.9989**	-	-	-	0.9903	-
ADMNC	0.9763	**0.9968**	**0.9975**	**0.9993**	**0.9972**	**0.9254**

taken for a given dataset size, which allows us to get an idea of the time complexity of each method. The time reported is the average execution time per fold for the algorithm using all the hyperparameter combinations described in Table 5.1, except when the execution time was too high, when just one hyperparameter combination was used and where the time therefore corresponds to a single execution. Even with this simplification, for some methods the absolute times were unmanageable for the largest versions of the dataset, so times could only be measured for the smaller versions. While the times of LOF and LOCI approach cubic complexity, PA-I displays quadratic complexity, OC-SVM exhibits super-linear complexity that approaches quadratic when the dataset is large, and DOC-SVM presents linear complexity, although its current implementation does not allow the use of large datasets. The time complexity of iForest approaches linear when the dataset is large. Our method exhibits clearly sub-linear complexity, which makes it the only candidate for very large datasets. Moreover, the ADMNC complexity increase is stopped when the dataset reaches 12500 elements. This is because most of the complexity is due to the KMeans initialization step described in Section 5.3.1 which, once the dataset is large enough, works only with a fixed-size sample, therefore limiting the impact of dataset size on execution time. It is worth noting that, although the parallel implementation of our method potentially allows for additional speed-up using more computing cores, the scalability shown in these experiments does not stem from the addition of more computing cores. All experiments reported in this chapter were executed using 12 computing cores on a single machine.

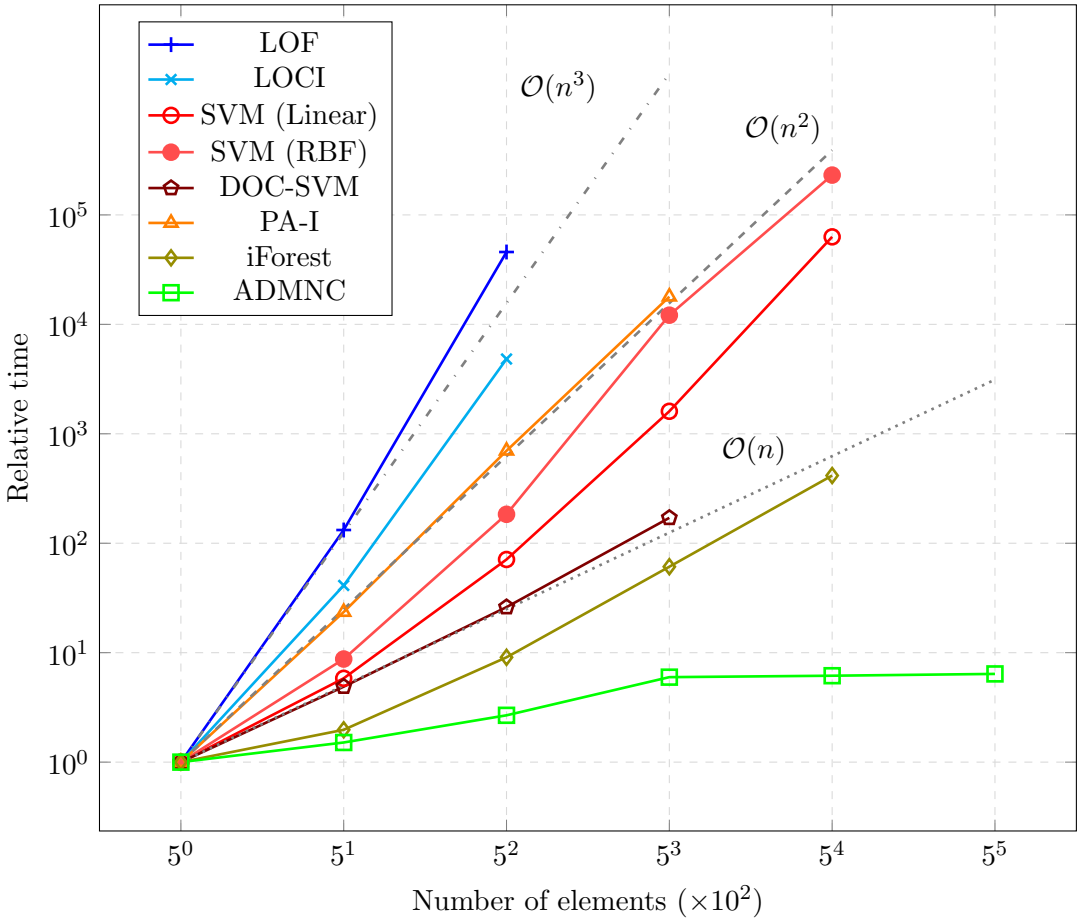


Figure 5.2: Execution time for each algorithm on datasets of incrementing size (*Synth1*). Times are presented as a ratio of the time taken for a given execution and the time for the same algorithm on a 100-element dataset. Both axes are represented using a logarithmic scale.

Finally, we compared the performance of each algorithm on *Synth2*, a family of datasets with an ascending order of difficulty. Since the three variants of LOF and LOCI offered very similar results, only the best performer for each method is reported. Results shown in Figure 5.3 indicate that our method outperforms the rest of algorithms when the dataset is very complicated, while as the difficulty decreases the results even out. It is worth noting that the fact that the dataset is constructed with numerical variables depending on the binary variables is no obstacle to the performance of ADMNC, even though it models the probability the other way around, that is, the categorical variables depend on the numerical variables. It is also interesting to highlight that the methods with a comparable AUC to ADMNC (LOF, LOCI, SVM-L and SVM-R) all have time complexity $\mathcal{O}(n^2)$ or superior, which makes their results unavailable for large datasets. This leaves our method as the clearly superior option for those datasets.

5.6. Conclusions

In this chapter the use of an ad-hoc method to perform scalable anomaly detection is described. We present a new method capable of handling large datasets and high dimensionality scenarios and of dealing with data having both categorical and continuous variables. This characteristic of the input space, which complicates the problem, is transformed into an advantage by using it to split a complicated model into two simpler ones. The resulting algorithm constitutes a useful tool for an emerging problem that currently lacks capable solutions.

The approach presented uses a probabilistic perspective. The continuous part is modeled using a Gaussian mixture model, while the categorical part is estimated using a logistic model that uses a maximum likelihood approach optimized with a stochastic gradient descent algorithm. The whole method is thus scalable to large datasets which is further enhanced by its distributed computing implementation.

Several experiments show that this method obtains better or similar results than those of state-of-the-art anomaly detection methods for small-medium datasets and that it performs well with datasets that are beyond the reach of other methods because of their computational demands. These favorable results would point to the viability of further research on the capabilities of this method as new datasets become available in the future. To facilitate further research we provide a working implementation of

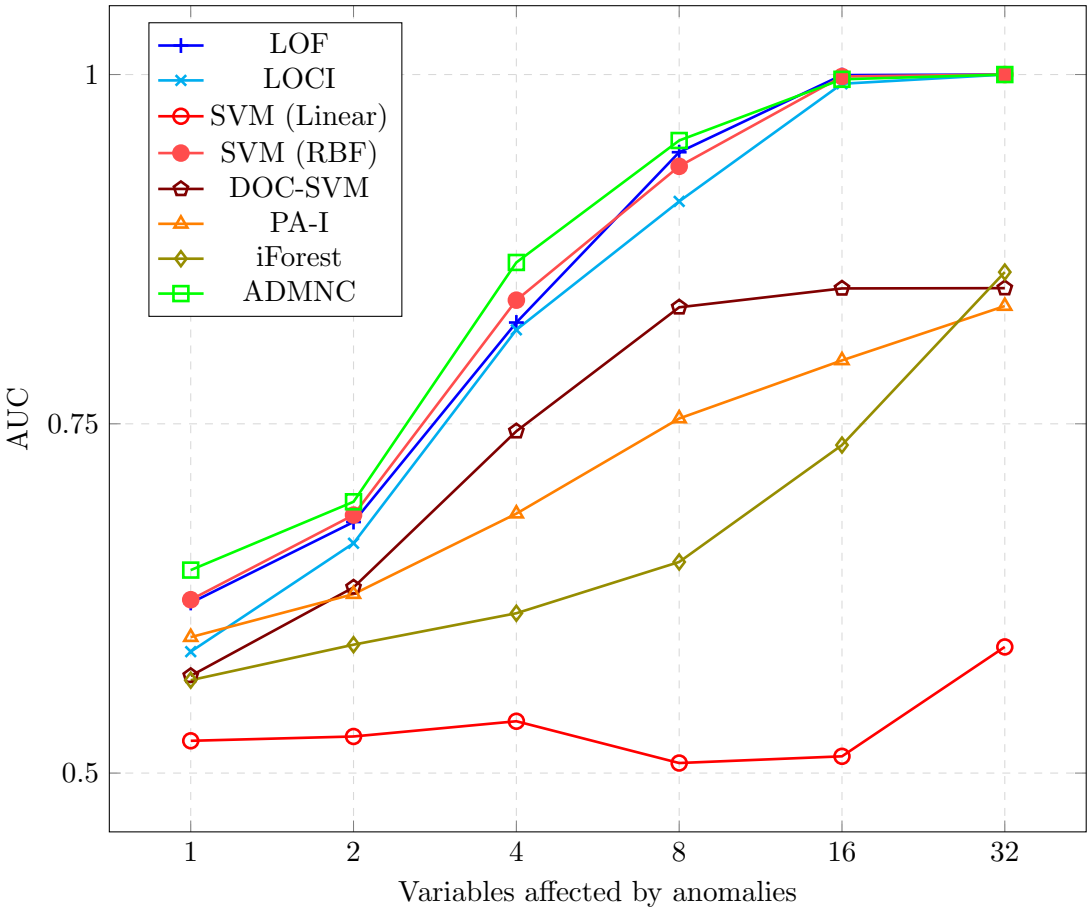


Figure 5.3: Area under the ROC curve obtained on the *Synth2* dataset. The number of variables affected by anomalies acts as a proxy for difficulty (a higher number indicates an easier dataset). The X axis is represented using a logarithmic scale.

the algorithm in the popular Apache Spark framework.

In real world applications, and especially in the case of recent large datasets, the existence of missing data points is relatively common. Thus, and as future work, we plan to extend our probabilistic model to deal with these situations. We will also explore the interpretability of this model and the possibility of justifying each example labeled as an anomaly to users.

CHAPTER 6

Speculative computation: Explaining large-scale dyadic data

6.1. Introduction

Dyadic data [80] hold information regarding the interaction of two entities of any kind. They are pervasive in a variety of popular problems such as recommender systems [90], social science application analysis, market segmentation [91], computational linguistics, information retrieval, and preference learning [108], but also in more specific areas, such as automatic exam grading [109]. The large number of elements of each entity in such datasets yields an immense number of possible pair-wise interactions that is much larger than the recorded data. The sparsity of measurements can be overcome by using available data points to learn a *utility function* that generalizes the recorded data and predicts the outcome of each possible interaction. The very common problem associated with the learning of this utility function has usually been resolved using matrix factorization [90]. However, this procedure usually results in thousands of parameters and, despite the prediction accuracy for any given pair of elements, offers little insight into the nature of the relationship between two entities. A major problem in dealing with dyadic data therefore consists of identifying groups of entities with similar behaviour, so as to obtain a high-level model of the studied environment. For instance, when analysing data from a book recommender system, a data scientist could look for groups of books that attract similar groups of readers with common reading interests. Characterizing such groups in terms of relevant information is a problem of great commercial interest since information on such small homogeneous groups would enable the development of more effective strategies tailored to specific groups. Such information, highly coveted by companies, is a difficult to obtain in practice, even when there is abundant data to analyse. The algorithms used to process this data should be computationally efficient in their capacity to handle large amounts of data, yet should

be accurate enough to retrieve meaningful information and to yield results that are actionable and comprehensible for decision makers.

On a different note, as the complexity of machine learning models grows, predictions are becoming more accurate, yet these models are often hard to interpret and do not provide global actionable information. It is therefore becoming increasingly important that the results of machine learning algorithms can be understood by a human supervisor. In some cases this is even required by law, as in the case of the right to explanation included in the new General Data Protection Regulation (GDPR) of the European Union[12]. The goal of Explainable Artificial Intelligence (XAI) (apparently the most common moniker, although sometimes also called Interpretable AI[13] or Transparent AI) is to obtain models that ideally should [103]:

1. Allow supervisors to interpret results so that it can be confirmed that the model goals are aligned with the desired goals (for instance, a credit rating system should not have gender or racial biases).
2. Justify predictions so as to enable a supervisor to formulate hypotheses that can be later verified, thereby excluding mere correlations due to randomness or dependence on some external factor,
3. Explain outputs in such a way that their generalizability can be established.
4. Be informative, that is, it should offer the supervisor new information regarding the studied variables.

Interpretability can be achieved in one of two ways: (1) in the form of a transparent model that allows the supervisor to follow the "logic" behind every prediction, as in the case of production rules, or (2) as post-hoc interpretability, which consists of justifying a prediction through similar cases or through visualizations or other methods that identify the input features that led to a prediction. While post-hoc interpretability is the most common approach, it is limited to explaining individual cases and does not provide the supervisor with new general information about the modelled environment.

Finally, scalability is also a problem in this context. Despite the immense amount of data available in some cases holding a great amount of valuable information, processing

[12]https://ec.europa.eu/info/law/law-topic/data-protection_en

[13]Although some authors [63] make a distinction between the terms *interpretability* and *explainability*, in this work we use them interchangeably.

this data can be challenging because of its sheer volume. As a result, there is a need for scalable algorithms that can deal with very large datasets.

This work, which was published in Decision Support Systems, explores the use of the techniques used to create scalable algorithms to increase the efficacy of simple models by increasing the amount of search space that is explored. In particular, distributed computing is used to perform independent parallel explorations of the search space, increasing the probabilities of finding a good solution.

The presented approach tackles the formal problem of obtaining relevant information from a utility function that codes the relationships existing between entities in a dyadic dataset. The proposed method splits the actors into easily interpretable groups with homogeneous behaviour. This high-level summary of the data explains existing relationships and provides supervisors with meaningful information that will improve decision making processes. Moreover, the implementation in the Apache Spark scalable distributed computing framework [153] enables the processing of large amounts of data to obtain relevant information within a reasonable timeframe.

The rest of the chapter is structured as follows. Section 6.2 gives an account of existing methods in this field, Section 6.3 contains definitions of key concepts used in the proposed system, Section 6.4 describes the algorithm, Section 6.5 describes how the experiments performed to assess the suitability of the method were designed, Section 6.6 reports on and comments the results, and, finally, Section 6.7 summarizes the conclusions drawn and indicates future lines of research.

6.2. Related work

Although XAI is a nascent field, the number of proposed methods is increasing rapidly as shown in the latest survey papers [64, 63, 116]. The goal of obtaining an alternative to an opaque predictor, called the *Open the Black Box* problem, can be tackled in four different ways according to Guidotti et al. [64]:

1. *Transparent box design* consists of obtaining a classifier that uses a logic or methodology that can be directly interpreted by the supervisor.
2. *Model explanation* obtains a surrogate interpretable predictor that mimics the

behaviour of the opaque model as closely as possible for any input.

3. *Outcome explanation* yields an explanation for a particular prediction of the classifier, offering post-hoc explainability.

4. *Model inspection* manipulates black box inputs to assess the magnitude of the effect of each variable in the prediction.

Our proposed method can be classified as either a *model explanation* of the utility function that predicts the nature of the relationship between any possible pair of actors, or as a *transparent box design* approach that obtains a grouping of the actors in a dyadic dataset. The resulting model is a shallow decision tree easily interpreted by the supervisor. Decision trees, along with rule systems and linear models, are considered to be easily interpreted, and using a single decision tree model as a surrogate for the black box model requiring explanation is a popular approach that started with the classic Trepan algorithm [42]. While several authors have iterated this approach [94, 28, 13, 85], none of their algorithms can be used to explain dyadic data.

For dyadic data, the most widely used methods to identify large-scale trends are segmentation and clustering techniques. Often problem-specific, they span market segmentation, document clustering and topic modelling, web user clustering and similar related fields. Our problem has been tackled using clustering algorithms [18, 106], self-organizing maps [87, 71], dimensionality reduction algorithms [110], evolutionary algorithms [1] and co-clustering algorithms [134]. Regarding interpretability, the mentioned algorithms have varying features. While evolutionary algorithms and co-clustering methods are not entirely suitable if interpretability is a goal, self-organizing maps and dimensionality reduction algorithms, although they do offer *post-hoc* explanations to the supervisor, do not clearly reveal links between the input variables that describe each entity, which reduces their effectiveness in motivating predictions and explaining outputs.

Lastly, a number of works have tackled explainability in the context of dyadic data. The importance of providing explanations for recommendations in the context of recommender systems has been established, and the effectiveness of different forms of explanation has been studied [77, 93]. An algorithm to obtain explainable clusters of users in the specific context of short text streams has been proposed [156], as a specific version of the topic modelling problem that cannot be used for generic dyadic data. Finally, the most similar work to our own is TEM [148] which consists of an embedding

model enhanced with a classification tree in order to obtain explainable outputs. The algorithm uses an attention network to highlight the most relevant embeddings, which are then explained using the classification tree. However, TEM is limited to obtaining post-hoc explanations since the attentive embedding precludes the attainment of global explanations.

In conclusion, although the analysis of dyadic data is a popular field with many different approaches in the literature, the interpretable methods available are limited to post-hoc explanations of outputs. When tasked with obtaining an explanation of the relationships encoded in a utility function, the only possibility available to the practitioner is to use a generic clustering algorithm and then open the black box with an interpretable predictor.

6.3. Definitions

Dyadic data $\mathcal{X}$ describe interactions between two entities $\mathcal{U}$ and $\mathcal{I}$. Each data point $x \in \mathcal{X}$ is a (u, i, v) tuple, where $u \in \mathcal{U}$ and $i \in \mathcal{I}$ are the elements of each entity involved in the interaction and v is a value that informs of some characteristic of that interaction. For every u, i there is a data point x describing their relationship (x can be observed or predicted). Consequently, $\mathcal{X}$ can be represented with a function $f : (\mathcal{U}, \mathcal{I}) \rightarrow \{-1, +1\}$, commonly called a utility function. Note that v can take any value, but here we simplify the problem by transforming v to -1 or 1. Obtaining a prediction of this utility function is a very common problem that has usually been solved in the literature using matrix factorization [90, 104]. In addition, we define a clustering $Cl(\mathcal{U})$ over a dataset $\mathcal{U}$ as a set of m disjoint groups that contain every element in $\mathcal{U}$. The formulation is as follows:

$$Cl(\mathcal{U}) = \{Clu_1, \ldots, Clu_m\}\,. \tag{6.1}$$

The homogeneity of the utility function inside each group Clu_k can be used to establish the fitness of the clustering [48]. With this goal, we define the ratio p of positive elements in a group k for a given i_j that represents the j-th element in $\mathcal{I}$ as:

$$p_{kj} = Pr(+1|Clu_k, i_j) = \frac{|\{u \in Clu_k : f(u, i_j) = +1\}|}{|Clu_k|} \tag{6.2}$$

A group Clu_k is said to be *consistent* when there is good agreement in the values of f for the elements contained in the group, that is, p approaches 0 or 1. To measure that

consistency as intended, we use the entropy of p.

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p). \tag{6.3}$$

$H(p_{kj})$ measures the consistency of a single group Clu_k with respect to i_j. To extend this measure to the whole clustering $Cl(\mathcal{U})$, every group and every element in $\mathcal{I}$ must be taken into account. Formally, we define the *weighted entropy (WE)* of a clustering $Cl(\mathcal{U})$ as:

$$WE(Cl(\mathcal{U})) = \sum_{k,j} \frac{|Clu_k|}{|\mathcal{U}||\mathcal{I}|} H(p_{kj}). \tag{6.4}$$

Bearing in mind the interpretability characteristics described in Section 6.1, here we focus on the ability of algorithms to motivate their predictions in an interpretable way using the input variables. We must therefore add a value to the fitness measure to evaluate the complexity of the explanation required to define each group, which we will estimate with the number of variables that describe the group. Consequently, we define the *quality* of a clustering as:

$$quality(Cl(\mathcal{U})) = -WE(Cl(\mathcal{U})) - \lambda \sum_{Clu_k \in Cl(\mathcal{U})} NV(Clu_k). \tag{6.5}$$

where NV represents the number of variables needed to describe Clu_k and λ is a hyperparameter that allows the supervisor to balance the entropy of the clustering with its interpretability. It can be shown that we can intuitively expect a balance between the complexity of the explanation and the precision of the obtained clustering. Thus, to obtain very uniform groups one will generally need to form a large number of such groups which, in turn, will require a larger number of variables; however, both requirements decrease the interpretability of the clustering. Managing the trade-off between interpretability and accuracy is a desirable feature that avoids obtaining misleadingly oversimplified explanations while keeping interpretability at acceptable levels [63]. Hyperparameter λ allows the supervisor to manage this trade-off. This idea of factoring in both model accuracy and complexity is reminiscent of the well-known Akaike information criterion [3] and the Bayesian information criterion [133], although with significant differences that generalize the model to allow dyadic data handling and explainability through trees.

It is worth noting that it is irrelevant that the *quality* measure is a negative number. The goal of the algorithm is maximizing its value to approach 0. This general quality measure can be applied in any clustering built with whatever method for a dyadic dataset.

6.4. Proposed algorithm

We describe a new explanatory algorithm for dyadic data that obtains groups that are as homogeneous as possible and that are simultaneously explained using as few of the input variables as possible. To achieve this, a binary decision tree is built and a clustering $Cl(\mathcal{U})$ is defined by considering each leaf node as a separate group that is described by the variables that lead to that node. Note that this contrasts the available alternative, which is to use separate models for the clustering and the explanation tree, producing inferior results. It is also worth noting that the obtained groups are described with a varying number of attributes and the result can therefore be considered as a subspace clustering of the data.

The main process consists of finding the tree that maximizes the *quality* of the resulting clustering defined using Eq. 6.5. Certain simplifications are needed in order to efficiently explore the solution space. First, as mentioned above, the decision tree is binary because only dichotomous splits are contemplated (a common simplification when building decision trees). Also, to facilitate the calculation of the *quality* of a clustering, given that the number of elements in $\mathcal{I}$ can prevent accurate calculation in a reasonable time, a significant random sample of $\mathcal{I}$ is considered instead of the entire $\mathcal{I}$ set. This is done by performing a previous clustering on $\mathcal{I}$ using a standard algorithm such as K-Means and using centroids ci_j as representative points; in the event that the input space structure does not allow for K-Means to be computed, any sampling procedure that obtains a reduced number of representatives of $\mathcal{I}$ could be used. Once the representatives are computed, the ratio p is estimated using a variation of Eq. 6.2 where i_j includes only the selected representatives rather than each possible item. Analogously, addends in Eq. 6.4 are computed for each representative and divided by the number of items belonging to the represented cluster.

In addition, since exploring every possible decision tree built as described is unfeasible, a search strategy is mandatory. First, to prevent the tree from splitting at any possible value of each input variable, the number of split points must be reduced. A maximum number of split points is therefore established for each variable. In the case of numerical variables these points are determined using discretization. We modelled the search procedure after the C4.5 algorithm [124], adapting the entropy calculation to a multi-label context. Our implementation differs from existing multi-label versions of C4.5 [39] in that the weighted entropy measure that we use factors in the size of the groups directly.

A single tree of L_{MAX} levels is built by performing a greedy search in which, for each node, the candidate with the best weighted entropy is selected. This process is recursively repeated for the new groups obtained after the split, until a given L_{MAX} level is reached, as previously selected by the user. To expand the reach of this solution space exploration and so increase the possibilities of achieving a good solution, at each step the proposed algorithm explores not only the candidate with the best weighted entropy but the N best candidates. This spawns N possible trees that, when exploring the next level, will each generate $2 * N$ different possibilities. This process makes the number of explored trees grow exponentially with L_{MAX}. For this reason, the selected values for the N and L_{MAX} hyperparameters should be low. Once all possible trees are generated, the tree that defines the clustering with the highest quality is selected. The resulting clustering defined by this tree will consist of a maximum of $2^{L_{MAX}}$ groups.

This entire process is described in Algorithm 12.

Lastly, in some cases the clustering defined by the retrieved tree may have less quality than the clustering defined by a subset of that tree. To address this, pruning is implemented; nodes are examined from level $L_{MAX} - 1$ to the root and any splits that do not have a positive impact on the overall *quality* are removed, as described in Algorithm 13.

This algorithm consists of calculations that can be performed in parallel since they are independent of each other, so, to take advantage of this feature, the algorithm was implemented in the Apache Spark distributed computing framework. By leveraging distributed computing, the scalability of the algorithm is greatly increased, which, in turn, enables the analysis of large datasets in manageable times. The Apache Spark implementation of the clustering algorithm and all data transformation procedures are available for download from `https://github.com/eirasf/Dyadic-Explanation-Tree`.

6.5. Experimental setup

To assess the validity of the algorithm we performed two sets of experiments. We applied the method first to two real-world large datasets, measured the quality of the obtained explanation and compared the results with those for an alternative approach consisting of using two separate models: a generic clustering algorithm and an inter-

Algorithm 12: Explanatory tree construction algorithm.

Data: $\mathcal{U}, L_{MAX}, N$

Result: Decision tree that determines the clustering.

function BUILDTREE($\mathcal{U}, level, splitPs, L_{MAX}, N$) $\rightarrow$ *best*

    if $level > L_{MAX}$ **then**
        return $\emptyset$
    end
    candidates $\leftarrow$ sorted list with capacity N;
    for $(variable, value) \in splitPs$ **do**
1       $left \leftarrow \{u \in \mathcal{U} : u[variable] < value\}$;
        $right \leftarrow \{u \in \mathcal{U} : u[variable] > value\}$;
        if WE($left$)$*left.size+$WE($right$)$*right.size < candidates.max$ **then**
2           $candidates$.ADD($(variable, value)$);
        end
    end
    $best \leftarrow \emptyset$;
    for $(variable, value) \in candidates$ **do**
3       $left \leftarrow \{u \in \mathcal{U} : u[variable] < value\}$;
4       $right \leftarrow \{u \in \mathcal{U} : u[variable] > value\}$;
5       $leftTree \leftarrow$ BUILDTREE($left, level + 1, splitPs, L_{MAX}, N$);
6       $rightTree \leftarrow$ BUILDTREE($right, level + 1, splitPs, L_{MAX}, N$);
7       $newTree \leftarrow (variable, value, leftTree, rightTree)$;
8   **if** WE($newTree$) $>$ WE($best$) **then**
9           $best = newTree$;
        end
    end
    return *best*;

end

splitPs $\leftarrow$ list of split points for every variable;

return BUILDTREE($\mathcal{U}, 0, splitPs, L_{MAX}, N$);

Algorithm 13: Pruning algorithm.

Data: $tree$, λ

Result: Pruned tree.

function PRUNE*(tree,λ)* $\rightarrow tree$

if ISLEAF*(tree)* **then**

  return $tree$;

end

$left \leftarrow$PRUNE$(tree.leftBranch, \lambda)$;

$right \leftarrow$PRUNE$(tree.rightBranch, \lambda)$;

$splitEntropy \leftarrow \frac{left.entropy*|left|+right.entropy*|right|}{|tree|}$;

$\Delta E = splitEntropy - tree.entropy$;

$\Delta NV = left.numVars + right.numVars - tree.numVars$;

if $-\Delta E - \lambda \Delta NV <= 0$ **then**

  $tree.leftBranch \leftarrow \emptyset$;

  $tree.rightBranch \leftarrow \emptyset$;

return $tree$;

end

pretable predictor that opens the black box (see Section 6.2). We selected K-Means and a single CART tree with entropy as the impurity measure, to ensure that the obtained results achieved a quality measure as high as possible. Another experiment was performed to measure the effect on execution time of adding further computational nodes to the distributed calculation; this was done to test the scalability of the presented algorithm in the implementation in Apache Spark.

The first dataset we selected was one published by the advertising company Outbrain, made public as the subject of a competition hosted in the popular machine learning site Kaggle.com. Outbrain suggests new news content that may be of interest to readers. The dataset[14] records the page views of a number of users in a variety of news-related websites over the span of 14 days and, since the documents refer to current issues, the recorded views vary in terms of topic at different times; consequently, it was advisable to split the dataset into smaller parts that covered a shorter time span. For the purposes of this research, we used some 1,450,000 records consisting of 667 variables that were collected on the first day. Our second choice was the popular *MovieLens 20M* dataset [72], commonly used to test recommendation systems. This dataset reflects interactions, in the form of some 20 million ratings, between 138,000 users and 27,000 movies. After the required transformations (described in the next subsection), the dataset contained some 20 million examples with 2,154 variables. The dimensions of the datasets are summarized in Table 6.1.

6.5.1. Dataset transformation

In order to apply the algorithms, the datasets needed to be formatted appropriately so that each sample represents an interaction between an element u in an entity $\mathcal{U}$ and all the representatives ci_j of the opposing entity $\mathcal{I}$. Therefore, each sample contained the variables that characterize u, which we refer to as *explanatory* variables, and a vector of values $(r(u, ci_0), \ldots, r(u, ci_j))$ where $r(u, ci_j) \in \{-1, 1\}$ qualifies the relationship between u and representative ci_j, which we call *defining* variables.

The Outbrain dataset consists of elements that represent a page view by a user and that contain information about the user, the viewed document and the result of the interaction between the user and the offered sponsored links referring to other documents. While documents are characterized by numerous variables, including the

[14] Available for download from `https://www.kaggle.com/c/outbrain-click-prediction`

Table 6.1: Dataset description.

Original datasets			
Dataset	Features		Instances
Outbrain_DAY1	667		1,445,196
MovieLens 20M	2,154		20,000,263
Transformed datasets			
Dataset	Explanatory	Defining	Samples
Outbrain_DAY1	667	100	1,445,196
UsersEx	1005	100	138,493
UsersExWithPrediction	1005	100	138,493
MoviesEx	1149	100	10,369

publisher, category, topics covered and entities mentioned, users are solely described by their position (latitude and longitude) and the type of device used. To overcome the lack of information regarding users, we used the transformed dataset obtained by Luaces *et al.* [48] in which the attributes of the viewed document are added to the characterization of the user. Also, although the aim of the original Outbrain competition was to predict the most effective sponsored links in a given situation, the problem tackled here is different, namely, to obtain an explanation of the relationships between the user-document pairs and the sponsored links.

The user-document pairs were thus grouped according to their behaviour in order to obtain a high-level summary. Records corresponding to two consecutive page views by the same user were located and the preference of the user for one document over others was recorded so as to obtain preference tuples. These tuples conformed the dyadic dataset from which the utility function learned using matrix factorization [48]. The interactions of each tuple with a set of document representatives were selected to form the defining vector. These document representatives were the centroids of a $k = 100$ K-Means of the documents. This process is represented in Figure 6.1.

The *MovieLens 20M* dataset consists of a large number of ratings that directly describe relationships between users and movies. Movies are characterized by the year of release, a vector of 20 possible non-exclusive genres and a vector of 1,128 tags,

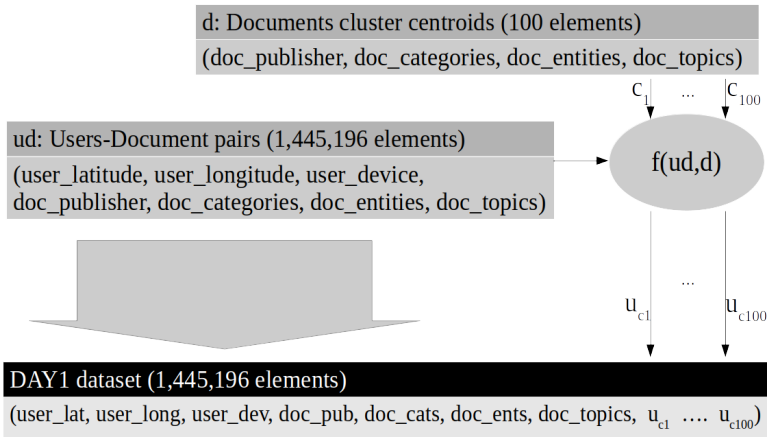


Figure 6.1: Transformation performed to obtain the Outbrain_DAY1 dataset.

totalling 1,149 variables per movie. Users, in contrast, are only identified with an ID. To solve this problem of a lack of user information, we represented users with a vector containing their ratings of the most popular movies, considering popular movies to be those rated at least 5,000 times. This yielded a total of 1,005 popular movies. Users u were therefore described by a vector spanning 1,005 components of the form $\{-1, 0, 1\}$ where -1 represents a negative rating for a movie, 0 represents no rating and 1 represents a positive rating m. We modelled the utility function $f(u, m)$ that predicts ratings to be of the form

$$f(u, m, W, V) = \sigma(< Wu, Vm >) = \frac{1}{1 + e^{-<Wu, Vm>}} \tag{6.6}$$

where W,V are parameter matrices to be learned that project users and movies in a common space with fewer dimensions than the input space; in this case we selected a space with a dimension equal to 200. By establishing a cost function:

$$J(W, V) = \sum_{(u,m)\in\mathcal{X}} -\log \sigma(r(u, m)\dot{<}Wu, Vm >) \tag{6.7}$$

where $r(u, m) \in \{-1, 1\}$ is the rating given to movie m by user u, the parameter matrices can be learnt using stochastic gradient descent, a very common approach to this problem analogous to that used in similar works [48].

This approach can be used with any dyadic dataset for which there is little or no information describing one of the entities. Nevertheless, in some cases, for datasets where the data available is sparse, the number of zeros in the user coding vector can become too large, increasing the similarity between users and complicating the decision tree task. To circumvent this problem, the user coding can be used to learn the utility

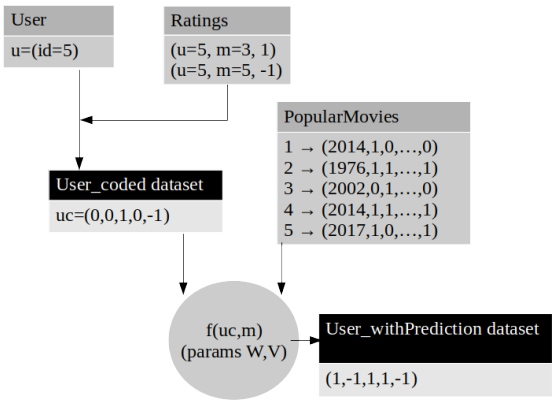


Figure 6.2: Example demonstrating the two options regarding the codification of a user.

function and can then be substituted by the predicted ratings for the most popular movies for that user. For a given popular movie pm_i, the user u coding vector will be 1 in component i if $f(u, pm_i) > 0.5$ and -1 otherwise. Both options are compared in Fig. 6.2 and their effectiveness for this particular dataset was tested as reported in Section 6.6.

Once users were coded and the utility function was learned, the projected movies Vm were clustered using K-Means with $k = 100$, yielding 100 movie representatives. A dataset, named *UsersEx*, was constructed by appending to the coding for each user (explanatory) their rating of each movie representative (*defining*). Once the users were coded using the prediction, the dataset obtained using the same procedure was called *UsersExWithPrediction*. Analogously, 100 user representatives were obtained by clustering the projected users Wu using K-Means with $k = 100$. A third dataset, named *MoviesEx*, was constructed by appending, to the coding of each movie, the utility of each projected user representative. Both these datasets allowed us to obtain two complementary explanations of the data, as will be further explained in Section 6.6. The complete pipeline is depicted in Fig. 6.3.

The datasets resulting from the transformations are described in Table 6.1. Note that the number of movies was reduced to 10,369, since many of them did not have any ratings in the dataset. Also, although the number of explanatory variables depends on the number of attributes characterizing each entity, the number of defining variables was always 100 since in all cases we used the relationship of each element with 100 representatives of the opposing entity to characterize behaviour.

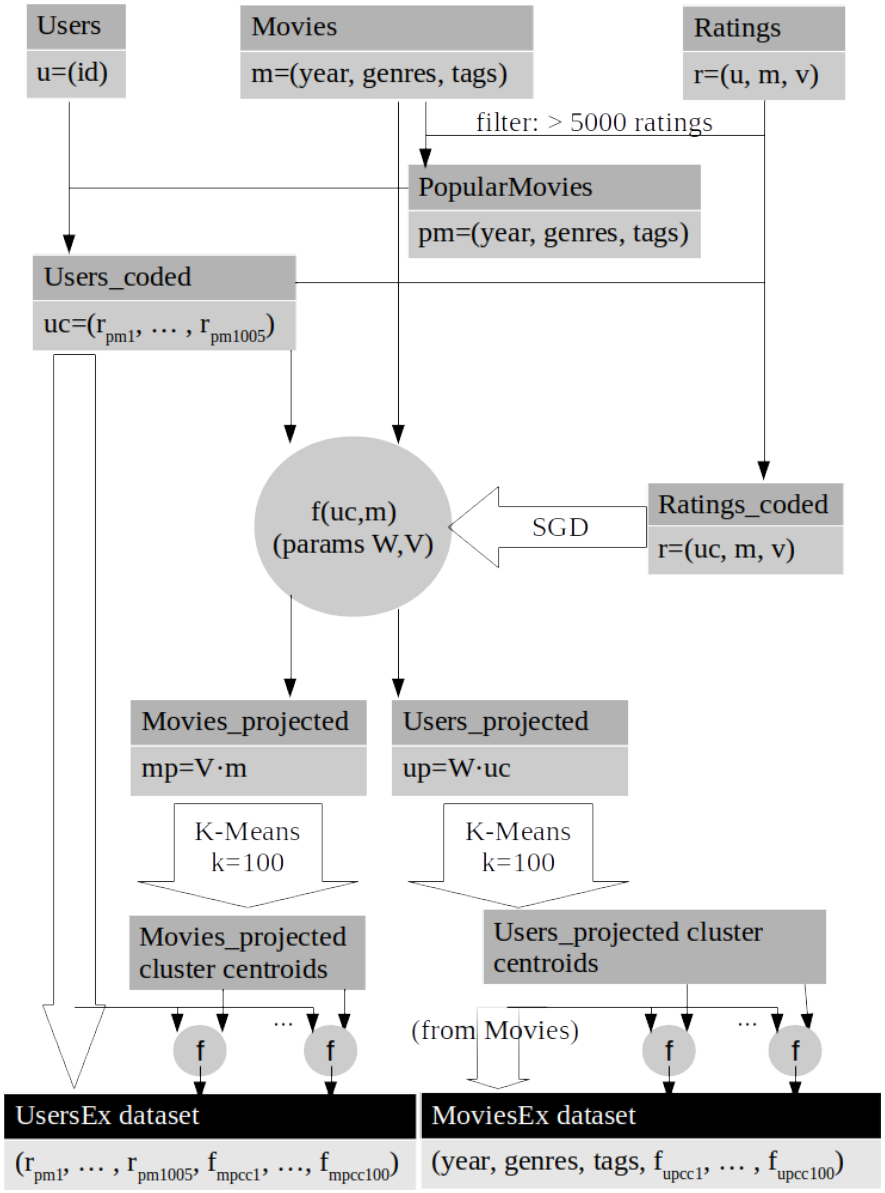


Figure 6.3: Transformations performed to convert the *MovieLens 20M* dataset in the *MoviesEx* and *UsersEx* datasets.

6.6. Results

In our first set of experiments we undertook the construction of an explanatory tree for the datasets and compared its quality to that of the explanation obtained by using a clustering algorithm and a separate model explainer. In order to perform these experiments, the values of hyperparameters λ, N and L_{MAX} needed to be set. We used $N = 5$ and $L_{MAX} = 5$, which originated a 5-level binary tree that, consequently, described 32 clusters characterized by 5 variables each – considered to be a reasonable upper threshold for the complexity of the explanatory tree.

6.6.1. Effect of the λ hyperparameter

Using $N = 5$ and $L_{MAX} = 5$ in Eq. 6.5 we obtained $NV = 160$. As described in Section 6.3, the λ hyperparameter regulates the pruning process, balancing the weighted entropy, which measures the effectiveness of the clustering and is in the $[0, 1]$ range, with $NV(Cl(\mathcal{U}))$, which was in the $[0, 160]$ range. We selected $\lambda = 0.001$ for our comparisons so that the obtained trees would be highly pruned and so could be easily represented. Nonetheless, this process is inexpensive enough to be performed rapidly with hundreds of different λ values. Fig. 6.4 shows a plot of λ vs. the quality of the explanation for dataset *Outbrain_DAY1*. It can be seen that as λ grows, the quality of the full tree decreases linearly, since the importance of the second component in Eq. 6.5 becomes larger. When λ is large enough, the pruning process can get rid of nodes that do not decrease the weighted entropy sufficiently to offset the quality penalty associated with having more nodes. Consequently, the number of groups in the clustering decreases, while the quality improves with respect to that of the full tree. Similar results were obtained for datasets *MoviesEx*, *UsersEx* and *UsersExWithPrediction* (see the supplementary material). The λ hyperparameter allows the supervisor to control the aggressiveness of the pruning process and, therefore, the complexity and accuracy of the explanatory model.

6.6.2. Suitability of the method

The results listed in Table 6.2 show that the models obtained with our method were both more accurate and more explainable than those obtained using the alternative

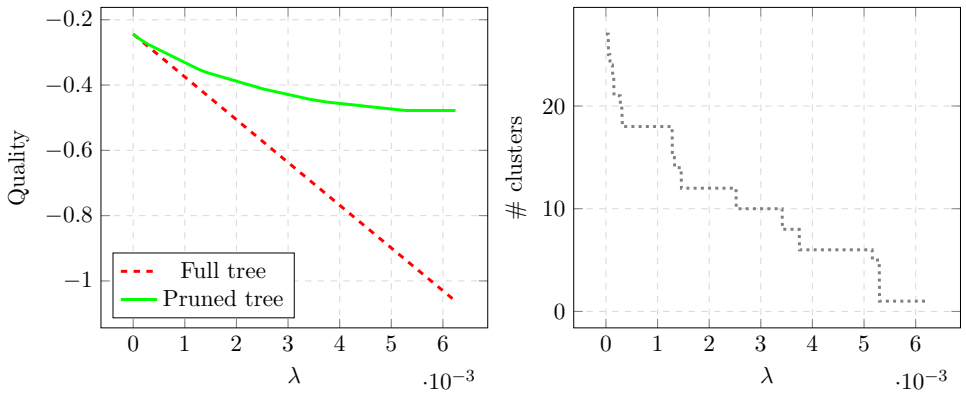


Figure 6.4: Pruned vs. original tree quality for the *Outbrain_DAY1* dataset (left). Number of nodes in the resulting pruned tree (right).

Table 6.2: Results comparison for the experimental datasets. Best results are highlighted in bold face. $\lambda = 0.001$ was used for the quality measurements.

Dataset	ExplainTree		Clustering+Explanation	
	WE	Quality	WE	Quality
Outbrain_DAY1	**0.244**	**-0.331**	0.359	-0.519
MoviesEx	**0.260**	**-0.353**	0.316	-0.476
UsersEx	**0.290**	**-0.333**	0.301	-0.461
UsersExWithPrediction	0.047	**-0.091**	**0.027**	-0.187

method. Only for dataset *UsersExWithPrediction* did the $k = 32$ K-Means+CART build a model with smaller entropy (0.027) than our method (0.047), although it needed more groups and so resulted in considerably inferior quality (-0.187) than our method (-0.091). Note that our method yielded more homogeneous groups in most cases than the alternative method, and this advantage was further enhanced when the explainability of the model was taken into account. This highlights the superiority of our approach that couples tree construction with a clustering process over the approach that uses independent algorithms for each step.

6.6.3. Analysis of the explanations

The information that a supervisor can extract regarding the characteristics defining the behaviour of an Outbrain user reading a given document is limited by the fact that the variables that characterize the user-document pair (topic, tags, etc.) are anonymized and only referred to by identifiers. Without the variable names no conclusions can be extracted from the explanatory tree. In contrast, the *MovieLens* dataset contains known variables that help identify trends in the data. Fig. 6.5 shows the explanatory tree for dataset *MoviesEx* and indicates which characteristics of a movie best define how different user types will react to them. It is apparent that a movie in a top list (variables *"movielens top pick"* and *"imdb top 250"*) was the most defining factor, after which certain movie qualities (variables *"affectionate"*, *"earnest"* or *"predictable"*) determine different user group responses. This information would give a supervisor insight into, for instance, what sort of movies should be added/removed from a catalogue. Another relevant piece of information in Fig.5 is the fact that the initial weighted entropy of the dataset (0.86) decreases greatly (to 0.3) with this clustering, indicating both that the response of users to different movies was very diverse and that the input variables allowed the uncertainty of the user response to a given movie to be decreased; this reflects the high value of the information extracted. The explanatory model for the *UsersEx* dataset represented in Fig. 6.5 offers additional insights to the same data. The first piece of information that stands out is that the weighted entropy of the full dataset is not very large (0.38), which indicates that users are somewhat homogeneous in their behaviour towards movies. Moreover, clustering does not manage to significantly decrease the weighted entropy of the data and, in consequence, the pruning process was very aggressive, yielding only 6 nodes. This was because, as stated in Section 6.5.1, the users are characterized by their rating $(-1, 0, 1)$ of the 1,005 most popular movies. However, for a given user, most movies are not rated, so users are defined by very sparse vectors. The large number of coincidences between users (most movies are unrated for a large set of users) made the task of the decision tree a difficult one. Nonetheless, this information could still be used, for instance, to rapidly determine the user type of a new user in a cold-start situation by simply asking them to rate a few movies selected from this decision tree. Furthermore, more information could be extracted from this approach by using the learned utility function to eliminate undetermined values in the characterization of users, as described in Section 6.5.1. Using the predicted values, the clustering tree corresponding to dataset *UsersExWithPrediction*, represented in Fig. 6.5, was much more effective and decreased the weighted entropy to 0.05. Such a decision tree could be useful in a cold-start scenario,

Table 6.3: Computer cluster overview.

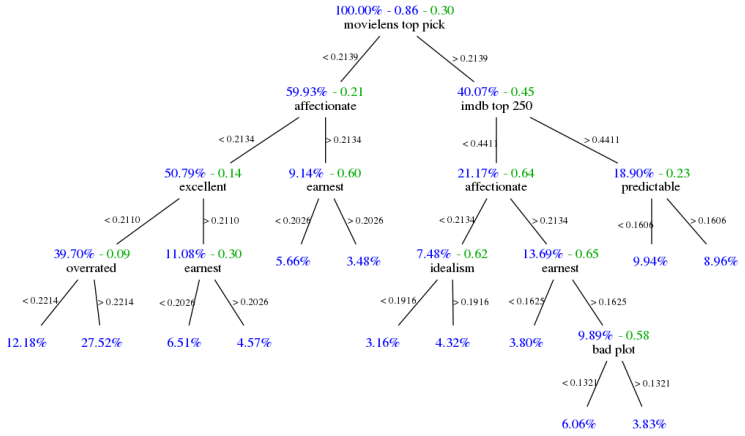
8 nodes with the following characteristics:	
Processor:	2 × Intel Xeon E5-2620 v3 at 2.40Ghz
Cores:	6 per processor (12 per node)
Threads:	2 per core (24 total per node)
Storage:	12 × 2TB NL SATA 6Gbps 3.5" G2HS
RAM:	64 GB
Network:	1x10Gbps + 2x1Gbps

similar to that described above but in which one could expect an unambiguous rating of each presented item, as could be the case when items can be rated on the spot.

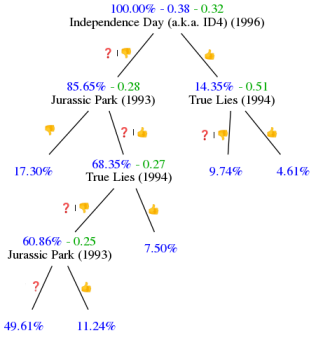
6.6.4. Scalability of the method

We performed an experiment with the aim of measuring the scalability of the method and of the Apache Spark distributed implementation. The same computation was performed for varying numbers of computing nodes. The experiments were run in a computer cluster formed by 8 machines with 12 computing cores each. The technical specifications for each node are provided in Table 6.3. The Spark version used was 2.4.0, on Hadoop 3.0.0-cdh6.1.0. The operating system of the machines was CentOS Linux release 7.4.1708.

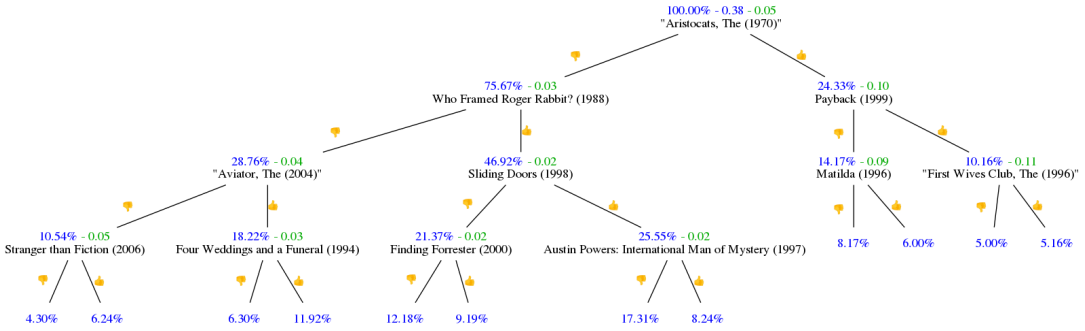
The times invested to compute a three-level clustering tree for dataset *Outbrain_DAY1*, listed in Table 6.4, indicate that the Spark implementation takes advantage of the fact that most calculations are mutually independent and can, therefore, be performed in parallel. Consequently, the execution time decreased at a similar rate to the increase in the number of computing nodes, which would be the ideal. The implementation allows the processing of large amounts of data in a reasonable time, provided the user supplies enough computing resources for the calculation.



(a) *MoviesEx*



(b) *UsersEx*



(c) *UsersExWithPrediction*

Figure 6.5: Explanatory trees for the *MovieLens* datasets after pruning with $\lambda = 0.001$. Each node shows, respectively, the proportion of elements that it represents w.r.t. the full dataset (shown in blue), the weighted entropy at that node (shown in green) and the variable used in the next split (black). Split values are shown next to each split line. The root node also indicates (in blue) the weighted entropy of the whole dataset.

Table 6.4: Execution time for computing a three-level tree for varying numbers of computing nodes.

	Time (H:M:S)		
# cores	12	2x12	4x12
Outbrain_DAY1	3:19:46	2:35:33	1:33:45

6.7. Conclusions

In this chapter we leverage the use of distributed computation to increase the efficacy of the training of a simple model. We describe a method to obtain a global explanation of the information encoded in a dyadic dataset. The computed model consists of a single decision tree that partitions one of the entities into groups with homogeneous behaviour; this decision tree is computed using an adaptation of a measure documented in the literature. The presented method is formulated in such a way that it can be applied to any dyadic dataset in any field that processes dyadic data. For instance, for data representing interactions in a market it could be used to perform large-scale market segmentation that provides supervisors with valuable insights for informed decision making. It could also be used to identify global trends in the data corresponding to recommender systems, topic modelling, social network analysis and other similar data problems.

Also described is the implementation of the presented algorithm in the popular Apache Spark distributed computing framework, which allows the processing of large volumes of data. Our experiments point to both the validity and the scalability of the approach, while also demonstrating various approaches to analysing diverse dyadic data. A brief analysis of how the retrieved information can be used is also presented.

In the future we plan to adapt this algorithm so that it becomes incremental and so it allows the use of a previously trained model to accelerate the calculation of an updated version when new data becomes available. We also plan to revise the search strategy used to build the tree so that the algorithm can be interrupted at any time and still yield meaningful results.

CHAPTER 7

Conclusions and future work

There is great opportunity in this new world overflowing with data. The efficacy of machine learning to produce accurate results on small datasets hinted great promise with the increase in the volume of data. As a consequence, great expectations have been placed in machine learning to play a crucial role in the advancement of the economy in the next century. A "data revolution" with effects comparable to the industrial revolution has been forecast, and machine learning is expected to revolutionize everything from employment to sustainability and industrial processes. Certainly, success cases are arriving at an increasing rate, with machine learning algorithms beating humans at diverse tasks, but we are a long way away from delivering on the hope that society has placed in machine learning.

Many of the most effective machine learning algorithms struggle when tasked with analyzing large datasets. A variety of problems arise when dealing with such data, from the more mundane logistical complications of handling heavy files to more structural ones like the curse of dimensionality. Even other yet unknown difficulties that can threaten the viability of a successful method. However, the factor that yields more methods useless in the context of large-scale learning is computational and spatial complexity. Methods that have computational or spatial complexity that exceeds $\mathcal{O}(n)$ are guaranteed to struggle with a dataset large enough. Even the success cases of state-of-the-art deep learning models, which can be learned with an effort that does not exceed $\mathcal{O}(n)$, require using very large computational resources for an extended time, which has great economic and environmental impact [143]. This high cost stifles research progress since many researchers simply lack the resources needed to obtain competitive results and it also negatively impacts the implantation of machine learning solutions in industry. Much work is still needed in academia to obtain more cost efficient and scalable methods.

In this thesis we have humbly attempted to contribute to tackling this problem by exploring four different approaches that can be used to increase the scalability of ma-

chine learning algorithms. We have done so while focusing in problems other that the heavily explored classification and regression and we have presented scalable algorithms for feature selection, graph construction, anomaly detection and model explainability. We have relied on the use of distributed computation for all of our algorithms, which provides a solution to the logistical problems of handling large datasets while also enabling the possibility of supplying additional computational resources that can be taken advantage of with a careful rewriting of the algorithms. This solution and the associated problems were studied in detail in Chapter 2 by obtaining distributed versions of various popular feature selection algorithms. A variety of strategies were used to obtain implementations in which most of the effort can be performed in parallel across several computational units. The experiments performed showed that the obtained implementations have very good scalability and highlighted that this approach is not only able to manage the handling of large-scale datasets to provide good scalability to algorithms with linear complexity, but also can be applied to methods with higher complexity, allowing them to handle the computational load associated with large datasets by adding more computational units to the calculation. However, although this trade-off of computation time for computational resources enables using complex methods, the associated cost encourages the search for alternatives. With that in mind, in Chapter 3 we explored the use of an approximation of the exact solution as an alternative for methods that are computationally expensive. In particular, we presented an algorithm that leverages Locality Sensitive Hashing to obtain an approximate k nearest neighbors graph, greatly reducing the original quadratic complexity. The development of this algorithm revealed the importance of obtaining a good set of hyperparameters to achieve accurate results, and the relevance of tuning these hyperparameters with as little effort as possible in the context of large-scale learning. The obtained algorithm, named VRLSH, quickly and automatically tunes these hyperparameters to obtain a very accurate approximate k-NN graph in a small fraction of the time needed to calculate the exact one. The exploration of this approach was furthered in Chapter 4 with the use of the obtained approximate k-NN graph in the calculations involved in the ReliefF feature selection algorithm. Our experiments show that the use of an approximate graph does not have significant impact in the accuracy of the results and the resulting algorithm, which we called ReliefF-LSH, obtains this results much quicker than the original ReliefF. Moreover, when compared to the existing alternative approximations to ReliefF, our experiments point to the superiority of the results of ReliefF-LSH, which, additionally, supports all data that ReliefF can manage, in contrast to the existing alternatives. Another course of action to deal with large-scale learning that we studied is the design of specific models that take advantage of some peculiarity of the data to obtain simpler and more effective models. To that effect,

in Chapter 5 we present an anomaly detection method that tackles the very common case in which the input data consists of both numerical and categorical variables. Exploiting this structure we designed a model that factorizes the probability distribution function of the data into two interrelated components that are learned using simple models. The resulting combined model, implemented into the ADMNC algorithm, has very good scalability that enables it to tackle datasets that are out of reach of the existing alternatives, and our experiments show that it obtains state-of-the-art results in terms of accuracy. Finally, we delved into the nascent field of Explainable AI to test one last approach when dealing with large datasets: using the computational power of distributed computing to enhance a simple method. In Chapter 6 we describe a new method to obtain an explanation of the relationships encoded in dyadic data in the form of a decision tree. The resulting tree effectively partitions one of the involved entities in groups of homogeneous behaviour that are simply described in terms of the input variables, constituting a high-level summary of the information encoded in the data. The proposed algorithm can obtain an accurate result quickly and in a scalable manner thanks to its use of distributed computation to broaden the exploration of the search space.

The aim of this work was to explore lines of action that lead to an increase in the scalability of algorithms. We studied four separate strategies and showed their viability by using them to implement algorithms that we then tested experimentally to show their fitness. But, in doing so, we also tried to obtain new scalable algorithms and improved versions of existing ones that enable the practitioner to deal with the problems of large-scale learning. While intentionally leaving aside the much more explored fields of classification and regression, we have presented in this thesis seven different implementations of four feature selection algorithms that can be used in single machine environments and in computer clusters, with an additional implementation of a popular discretization method. We have also presented an algorithm that detects anomalies in data that has a mix of numerical and categorical variables. This is a very common setting for which few of the existing alternatives could be used to analyze large datasets. Additionally, we developed a new method to compute the k nearest neighbors graph for large datasets. This data structure needs to be computed in many machine learning methods, and we proved its fitness by using it in an adaptation of the ReliefF feature selection algorithm. Finally, we designed an explanation algorithm that can be used in the analysis of dyadic data, a problem that has raised much interest in the industry. All of the implementations of the methods are available for download. It is our hope that our contributions enable practitioners to approach large-scale learning a bit better prepared and that they can be used to further the research in this area.

This thesis can only explore a tiny fraction of the vast field of the scalability of machine learning algorithms. Consequently, many interesting ideas were left unpursued. Some of those we intend to explore in our future work and have been described in each chapter. Additionally, in the explorations needed to complete this thesis we have identified promising lines that we plan to research further. In particular, we have obtained preliminary results that have encouraged us to explore the use of Locality Sensitive Hashing to detect anomalies at large scale. Some effective anomaly detection methods rely on the computation of the density distribution of the input space, but this calculation is very computationally intensive. We intend to use LSH to obtain an estimation of density at a fraction of the cost that can, hopefully, be useful in identifying anomalies. Additionally, we want to research the possibilities of using VRLSH for information retrieval, implementing a nearest neighbor search, and also for anomaly detection by traversing the approximated kNNG. Finally, we would like to dedicate some effort to finding new explainability algorithms that work at large scale. In this regard, we have obtained promising preliminary results explaining the anomaly detections of ADMNC by applying a similar approach to that described in Chapter 6. We believe that XAI is a field that will gain relevance since it offers a much needed link between machine learning and human reasoning.

However, we realize that the pervasiveness of the scalability problem and the vastness of the machine learning field mean that our contributions can only get us so far. In consequence, we have made a point of always making available for download working implementations of our algorithms, hoping that they can be of use to the research community. The observation that "the ability to collaborate and organize large groups of individuals to accomplish a task that would be unachievable individually is one of the trademarks of the human species" led us in Chapter 2 to the exploration of distributed computation solutions to complete immense tasks. Only our inherently human ability to collaborate at large scale can create a research community that brings mankind closer to the extraordinary goal of creating machines that can think.

APPENDIX

Best hyperparameters for anomaly detection methods

This is the list of the best hyperparameter combination for each method for each dataset in the experiments reported in Chapter 5. Hyperparameters for Synth1 are not listed since they are not relevant for the execution time, which is the only measure reported for that dataset family.

Table I.1: Best hyperparameters for LOF

	E (**K, P**)	H (**K, P,** λ)	J (**K, P,** λ)
Arrhyth	10, 0.01	10, 0.01, 0.9	10, 0.01, 0.7
GC	10, 0.01	10, 0.01, 0.3	10, 0.01, 0.1
Ab. 1	10, 0.01	10, 0.01, 0.3	10, 0.01, 0.3
Ab. 9	10, 0.01	10, 0.01, 0.3	10, 0.01, 0.3
Ab. 11	10, 0.01	10, 0.01, 0.3	10, 0.01, 0.3
Synth1-100	5, 0.01	3, 0.01, 0.9	5, 0.01, 0.5
Synth1-500	10, 0.01	10, 0.01, 0.9	10, 0.01, 0.7
Synth1-2500	Single test repeating values above		

Table I.2: Best hyperparameters for LOCI

	LOCI		
	E (α)	H (α, λ)	J (α, λ)
Arrhyth	0.3	0.5, 0.9	0.3, 0.5
GC	0.3	0.1, 0.5	0.1, 0.1
Ab. 1	0.1	0.1, 0.7	0.1, 0.3
Ab. 9	0.3	0.1, 0.9	0.1, 0.7
Ab. 11	0.5	0.5, 0.7	0.5, 0.3
Synth1-100	0.3	0.1, 0.9	0.1, 0.9
Synth1-500	0.1	0.5, 0.9	0.3, 0.1
Synth1-2500	Single test repeating values above		

Table I.3: Best hyperparameters for SVM

	Linear (ν)	RBF (γ, ν)	DOC-SVM(γ, ν)
Arrhyth	0.3	1, 0.1	1
GC	0.01	1, 0.01	3
Ab. 1	0.01	1, 0.3	5
Ab. 9	0.05	10, 0.1	1
Ab. 11	0.3	3, 0.05	1
CT	0.3	1, 0.3	4
KDD	0.1	10, 0.01	2
Synth1-100	0.3	0.01, 0.1	0.01, 0.3
Synth1-500	0.01	0.01, 0.01	0.01, 0.3
Synth1-2500	0.01	0.05, 0.01	Same values
Synth1-12500	Single test repeating values above		
Synth1-62500	Same values		*

Table I.4: Best hyperparameters for PA-I and iForest.

	PA-I			iForest	
	σ	**C**	**R**	**rF**	**rS**
Arrhyth	1	0.01	0.97	1	0.2
GC	3	0.01	0.97	1	1
Ab. 1	5	0.01	0.97	0.1	0.01
Ab. 9	1	0.05	0.97	1	0.5
Ab. 11	1	0.01	0.97	0.8	0.01
CT	4	0.025	0.97	1	0.01
KDD	2	0.05	0.97	0.1	0.5
Synth1-100	4	0.05	0.97	0.5	0.025
Synth1-500	4	0.01	0.97	0.8	0.01
Synth1-2500	4	0.01	0.97	1	0.01
Synth1-12500	4	0.01	0.97	1	0.025

Table I.5: Best hyperparameters for ADMNC.

	ν	λ_s	# gaussians
Arrhyth	1	1	4
GC	0.1	0.001	4
Ab. 1	100	0.01	4
Ab. 9	10	0.001	4
Ab. 11	0.1	0.001	4
CT	0.1	0.0001	4
KDD	1	0.1	2
IDS	1	0.1	4
Synth1-100	0.1	1	4
Synth1-500	1	1	4
Synth1-2500	10	1	4
Synth1-12500	10	0.001	4
Synth1-62500	100	0.1	4
Synth1-312500	1	0.1	4

APPENDIX II

Additional results of the scalable dyadic data explanator

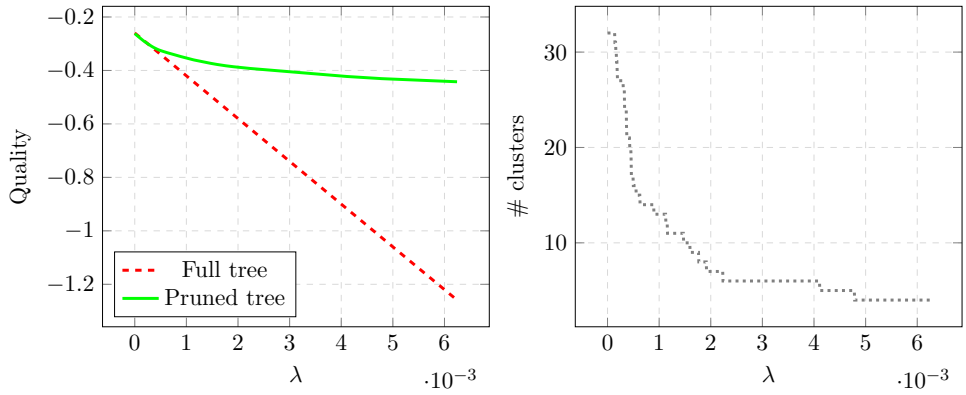


Figure II.1: Pruned vs. original tree quality for the *MoviesEx* dataset (left). Number of nodes in the resulting pruned tree (right).

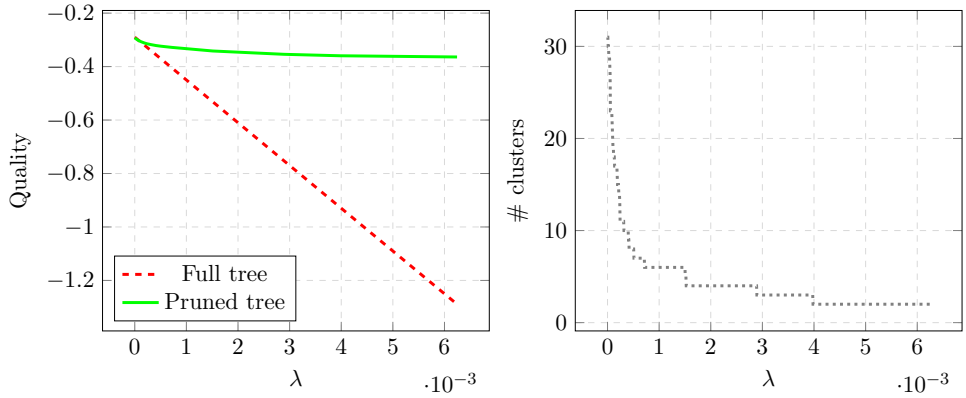


Figure II.2: Pruned vs. original tree quality for the *UsersEx* dataset (left). Number of nodes in the resulting pruned tree (right).

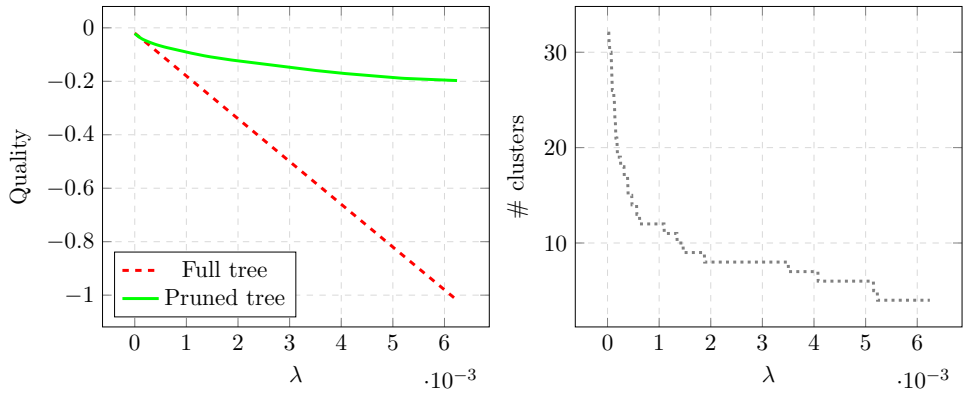


Figure II.3: Pruned vs. original tree quality for the *UsersExWithPrediction* dataset (left). Number of nodes in the resulting pruned tree (right).

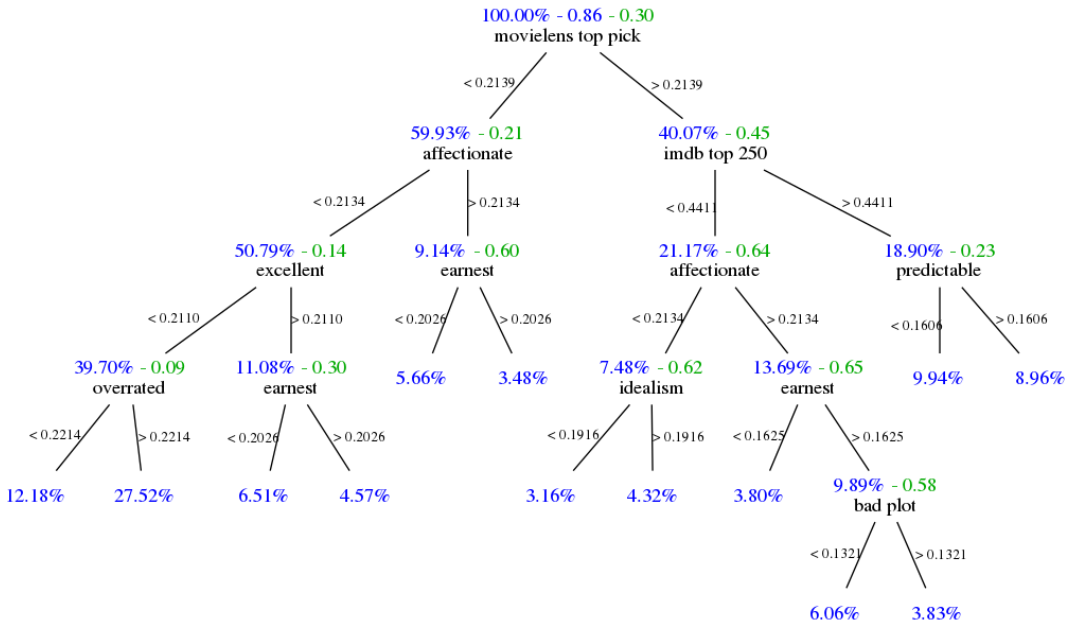


Figure II.4: Explanatory tree for the *Outbrain* dataset after pruning with $\lambda = 0.001$. Each node shows, respectively, the proportion of elements that it represents w.r.t. the full dataset (shown in blue), the weighted entropy at that node (shown in green) and the variable used in the next split (black). Split values are shown next to each split line. The root node also indicates (in blue) the weighted entropy of the whole dataset.

APPENDIX III

Publications supporting this thesis

- Eiras-Franco, C., Bolón-Canedo, V., Ramos, S., González-Domınguez, J., Alonso-Betanzos, A., & Tourino, J. (2015). Paralelización de algoritmos de selección de caracterıticas en la plataforma weka. In CAEPIA (Vol. 2015, pp. 949-958).
- Eiras-Franco, C., Bolón-Canedo, V., Ramos, S., González-Domínguez, J., Alonso-Betanzos, A., & Tourino, J. (2016). Multithreaded and spark parallelization of feature selection filters. Journal of Computational Science, 17, 609-619.
- Eiras-Franco, C., Kanthan, L., Alonso-Betanzos, A., & Martínez-Rego, D. (2017). In 25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, (pp.535-540).
- Eiras-Franco, C., Guijarro-Berdiñas, B., Alonso-Betanzos, A., & Bahamonde, A. (2018). Interpretable market segmentation on high dimension data. Multidisciplinary Digital Publishing Institute Proceedings, 2(18), 1171.
- Eiras-Franco, C., Guijarro-Berdinas, B., Alonso-Betanzos, A., & Bahamonde, A. Segmentacion de mercado explicable sobre datos de alta dimension. (2018). In CAEPIA (Vol. 2018, pp. 1099-1104).
- Eiras-Franco, C., Guijarro-Berdinas, B., Alonso-Betanzos, A., & Bahamonde, A. Selección de características escalable con ReliefF mediante el uso de Hashing Sensible a la Localidad. (2018). In CAEPIA (Vol. 2018, pp. 1123-1128).
- Eiras-Franco, C., Martínez-Rego, D., Guijarro-Berdiñas, B., Alonso-Betanzos, A., & Bahamonde, A. (2019). Large scale anomaly detection in mixed numerical and categorical input spaces. Information Sciences, 487, 115-127.
- Eiras-Franco, C., Guijarro-Berdiñas, B., Alonso-Betanzos, A., & Bahamonde, A. (2019). A scalable decision-tree-based method to explain interactions in dyadic data. Decision Support Systems. (In press).

Bibliography

[1] A. Fernandez O. C. F. M., M.J. del Jesus and Herrera. F. Evolutionary fuzzy systems for explainable artificial intelligence: Why, when, what for, and where to? *IEEE Computational Intelligence Magazine* **14**(1), 69–81 (2019).

[2] Ahle T. D., Aumüller M., and Pagh R. Parameter-free locality sensitive hashing for spherical range reporting. In "Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms", pages 239–256. SIAM (2017).

[3] Akaike H. Information theory and an extension of the maximum likelihood principle. In "2nd International Symposium on Information Theory", pages 267–281. Akademiai Kiado (1973).

[4] Akoglu L., Tong H., Vreeken J., and Faloutsos C. Fast and reliable anomaly detection in categorical data. In "Proceedings 21st ACM International Conference on Information and Knowledge Management, CKIM 2012", New York, NY, USA (2012). ACM.

[5] Aleskerov E., Freisleben B., and Rao B. CARDWATCH: A neural network based database mining system for credit card fraud detection. In "Proceedings of the IEEE Conference on Computational Intelligence for financial engineering", pages 220–226 (1997).

[6] An infotheoretic feature selection framework for Apache Spark. `http://spark-packages.org/package/sramirez/spark-infotheoretic-feature-selection`. Accessed: 2016-04-19.

[7] Anastasiu D. C. and Karypis G. L2knng: Fast exact k-nearest neighbor graph construction with l2-norm pruning. In "Proceedings of the 24th ACM International on Conference on Information and Knowledge Management", pages 791–800. ACM (2015).

[8] Andoni A. and Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In "Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on", pages 459–468. IEEE (2006).

[9] Atzori L., Iera A., and Morabito G. The internet of things: A survey. *Computer networks* **54**(15), 2787–2805 (2010).

[10] Aumüller M., Bernhardsson E., and Faithfull A. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In "International Conference on Similarity Search and Applications", pages 34–49. Springer (2017).

[11] Bahmani B., Moseley B., Vattani A., Kumar R., and Vassilvitskii S. Scalable k-means++. *Proceedings of the VLDB Endowment* **5**(7), 622–633 (2012).

[12] Baldi P., Sadowski P., and Whiteson D. Searching for exotic particles in high-energy physics with deep learning. *Nature communications* **5**, 4308 (2014).

[13] Basak J. and Krishnapuram R. Interpretable hierarchical clustering by constructing an unsupervised decision tree. *IEEE Transactions on Knowledge and Data Engineering* **17**(1), 121–132 (2005).

[14] Basios M., Li L., Wu F., Kanthan L., and Barr E. T. Darwinian data structure selection. In "Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering", pages 118–128. ACM (2018).

[15] Bawa M., Condie T., and Ganesan P. Lsh forest: self-tuning indexes for similarity search. In "Proceedings of the 14th international conference on World Wide Web", pages 651–660. ACM (2005).

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

[17] Bentley J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM* **18**(9), 509–517 (1975).

[18] Berkhin P. A survey of clustering data mining techniques. In "Grouping multidimensional data", pages 25–71. Springer (2006).

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

[20] Blackard J. A. and Dean D. J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture* **24**(3), 131–151 (1999).

[21] Bloom B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* **13**(7), 422–426 (1970).

[22] Bolón-Canedo V., Remeseiro B., Sánchez-Maroño N., and Alonso-Betanzos A. mc-relieff-an extension of relieff for cost-based feature selection. In "ICAART (1)", pages 42–51 (2014).

[23] Bolón-Canedo V., Sánchez-Maroño N., and Alonso-Betanzos A. "Feature Selection for High-Dimensional Data". Springer (2015).

[24] Bondi A. B. Characteristics of scalability and their impact on performance. In "Proceedings of the 2nd international workshop on Software and performance", pages 195–203. ACM (2000).

[25] Bottou L. and Bousquet O. The tradeoffs of large scale learning. In Platt J., Koller D., Singer Y., and Roweis S., editors, "Advances in Neural Information Processing Systems", volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc) (2008).

[26] Bottou L., Curtis F. E., and Nocedal J. Optimization methods for large-scale machine learning. *Siam Review* **60**(2), 223–311 (2018).

[27] Bottou L. and Lin C.-J. Support vector machine solvers. *Large scale kernel machines* **3**(1), 301–320 (2007).

[28] Boz O. Extracting decision trees from trained neural networks. In "Proceedings of the eighth ACM SIGKDD International Conference on Knowledge discovery and data mining", pages 456–461. ACM (2002).

[29] Bratić B., Houle M. E., Kurbalija V., Oria V., and Radovanović M. Nn-descent on high-dimensional data. In "Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics", page 20. ACM (2018).

[30] Breunig M., Kriegel H., Ng R., and Sander J. Lof: Identifying density-based local outliers. *SIGMOD Rec.* **29**(2), 93–104 (May 2000).

[31] Brown G., Pocock A., Zhao M.-J., and Luján M. Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. *The Journal of Machine Learning Research* **13**(1), 27–66 (2012).

[32] Buhler J. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics* **17**(5), 419–428 (2001).

[33] Campos G. O., Zimek A., Sander J., Campello R. J., Micenková B., Schubert E., Assent I., and Houle M. E. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery* **30**(4), 891–927 (2016).

[34] 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).

[35] Chandola V., Banerjee A., and Kumar V. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering* **24**(5), 823–839 (2012).

[36] Chandola V., Banerjee A., and Kumar V. Anomaly detection: A survey. *ACM computing surveys (CSUR)* **41**(3), 15 (2009).

[37] Chang C.-C. and Lin C.-J. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* **2**(3), 27 (2011).

[38] Chen J., Fang H.-r., and Saad Y. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research* **10**(Sep), 1989–2012 (2009).

[39] Clare A. and King R. D. Knowledge discovery in multi-label phenotype data. In "European Conference on Principles of Data Mining and Knowledge Discovery", pages 42–53. Springer (2001).

[40] Cormode G., Dasgupta A., Goyal A., and Lee C. H. An evaluation of multi-probe locality sensitive hashing for computing similarities over web-scale query logs. *PloS one* **13**(1), e0191175 (2018).

[41] Cover T. and Hart P. Nearest neighbor pattern classification. *IEEE transactions on information theory* **13**(1), 21–27 (1967).

[42] Craven M. and Shavlik J. W. Extracting tree-structured representations of trained networks. In "Advances in neural information processing systems", pages 24–30 (1996).

[43] Das K. and Schneider J. Detecting anomalous records in categorical datasets. In "Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", KDD '07, New York, NY, USA (2007). ACM.

[44] Das S., Matthews B. L., Srivastava A. N., and Oza N. Multiple kernel learning for heterogeneous anomaly detection: Algorithm and aviation safety case study. In "Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", KDD '10, pages 47–56, New York, NY, USA (2010). ACM.

[45] Dasarathy B. V. Data mining tasks and methods: Classification: nearest-neighbor approaches. In "Handbook of data mining and knowledge discovery", pages 288–298. Oxford University Press, Inc. (2002).

[46] Dean J. and Ghemawat S. MapReduce: simplified data processing on large clusters. *Communications of the ACM* **51**(1), 107–113 (2008).

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

[48] Díez J., Pérez P., Luaces O., and Bahamonde A. Readers segmentation according to their preferences to click promoted links in digital publications. Technical Report, Universidad de Oviedo (2018).

[49] Do K., Tran T., and Venkatesh S. Energy-based anomaly detection for mixed data. *Knowledge and Information Systems* pages 1–23 (2018).

[50] Dong W., Moses C., and Li K. Efficient k-nearest neighbor graph construction for generic similarity measures. In "Proceedings of the 20th international conference on World wide web", pages 577–586. ACM (2011).

[51] Dong W., Wang Z., Josephson W., Charikar M., and Li K. Modeling lsh for performance tuning. In "Proceedings of the 17th ACM conference on Information and knowledge management", pages 669–678. ACM (2008).

[52] Dua D. and Graff C. UCI machine learning repository (2017).

[53] Edgeworth F. On discordant observations. *Phylosoph. Mag.* **23**(5), 364–375 (1887).

[54] Eiras-Franco C., Bolón-Canedo V., Ramos S., González-Domínguez J., Alonso-Betanzos A., and Touriño J. Multithreaded and spark parallelization of feature selection filters. *Journal of Computational Science* **17**, 609–619 (2016).

[55] Eiras-Franco C., Guijarro-Berdinas B., Alonso-Betanzos A., and Bahamonde A. Seleccion de caracterısticas escalable con relieff mediante el uso de

hashing sensible a la localidad. In "XVIII Conferencia de la Asociación Española para la Inteligencia Artificial" (2018).

[56] Eiras-Franco C., Kanthan L., Alonso-Betanzos A., and Martinez-Rego D. Scalable approximate k-nn graph construction based on locality sensitive hashing. In "25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning" (2017).

[57] Emmott A. F., Das S., Dietterich T., Fern A., and Wong W.-K. Systematic construction of anomaly detection benchmarks from real data. In "Proceedings of the ACM SIGKDD workshop on outlier detection and description", pages 16–21. ACM (2013).

[58] Eppstein M. J. and Haake P. Very large scale relieff for genome-wide association analysis. In "Computational Intelligence in Bioinformatics and Computational Biology, 2008. CIBCB'08. IEEE Symposium on", pages 112–119. IEEE (2008).

[59] Fayyad U. M. and Irani K. B. Multi-interval discretization of continuous-valued attributes for classification learning. In "13th International Joint Conference on Artificial Intelligence", pages 1022–1029 (1993).

[60] Fernández-Francos D., Martínez-Rego D., O.Fontenla-Romero, and Alonso-Betanzos A. Automatic bearing fault diagnosis based on one-class nu-svm. *Computers & Industrial Engineering* **64**(1), 357–365 (2013).

[61] Fiore U., Palmieri F., Castiglione A., and De Santis A. Network anomaly detection with the restricted boltzmann machine. *Neurocomputing* **122**, 13–23 (2013).

[62] Ghoting A., Otey M., and Parthasarathy S. Loaded: link-based outlier and anomaly detection in evolving data sets. In "Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on", pages 387–390 (2004).

[63] Gilpin L. H., Bau D., Yuan B. Z., Bajwa A., Specter M., and Kagal L. Explaining explanations: An overview of interpretability of machine learning. In "2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)", pages 80–89. IEEE (2018).

[64] Guidotti R., Monreale A., Ruggieri S., Turini F., Giannotti F., and Pedreschi D. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)* **51**(5), 93 (2018).

[65] Guyon I. "Feature Extraction: Foundations and Applications", volume 207. Springer Science & Business Media (2006).

[66] Guyon I., Gunn S., Nikravesh M., and Zadeh L. A. "Feature extraction: foundations and applications", volume 207. Springer (2008).

[67] Guyon I., Weston J., Barnhill S., and Vapnik V. Gene selection for cancer classification using support vector machines. *Machine learning* **46**(1-3), 389–422 (2002).

[68] Apache Hadoop Project. `http://hadoop.apache.org/`. Accessed: 2016-04-19.

[69] Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., and Witten I. H. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter* **11**(1), 10–18 (2009).

[70] Hall M. A. "Correlation-based Feature Selection for Machine Learning". PhD thesis, The University of Waikato (1999).

[71] Hanafizadeh P. and Mirzazadeh M. Visualizing market segmentation using self-organizing maps and fuzzy delphi method–adsl market of a telecommunication company. *Expert Systems with Applications* **38**(1), 198–205 (2011).

[72] Harper F. M. and Konstan J. A. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)* **5**(4), 19 (2016).

[73] Haveliwala T., Gionis A., and Indyk P. Scalable techniques for clustering the web. In "Third International Workshop on the Web and Databases" (2000).

[74] Hawkins D. M. "Identification of outliers", volume 11. Springer (1980).

[75] Hawkins S., He H., Williams G., and Baxter R. Outlier detection using replicator neural networks. In "International Conference on Data Warehousing and Knowledge Discovery", pages 170–180. Springer (2002).

[76] He Z., Deng S., and Xu X. An optimization model for outlier detection in categorical data. In Huang D., X.P. Zhang G., and Huang, editors, "Advances in Intelligent Computing", volume 3644 of "Lecture Notes in Computer Science", pages 400–409. Springer Berlin Heidelberg (2005).

[77] Herlocker J. L., Konstan J. A., and Riedl J. Explaining collaborative filtering recommendations. In "Proceedings of the 2000 ACM conference on Computer supported cooperative work", pages 241–250. ACM (2000).

[78] Hettich S. and Bay S. KDD Cup 1999 data. *The UCI KD Archive, Irvine, CA: University of California, Department of Information and Computer Science* (1999).

[79] Higgs dataset at the UCI Machine Learning Repository. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`. Accessed: 2016-04-19.

[80] Hofmann T., Puzicha J., and Jordan M. I. Learning from dyadic data. In "Advances in Neural Information Processing Systems", pages 466–472 (1999).

[81] Hu W., Hu W., and Maybank S. Adaboost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **38**(2), 577–583 (2008).

[82] Hull J. J. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(5), 550–554 (1994).

[83] Ismo K. et al.. Outlier detection using k-nearest neighbour graph. In "null", pages 430–433. IEEE (2004).

[84] JeffreyXu Y., Qian W., Hongjun L., and Aoying Z. Finding centric local outliers in categorical/numerical spaces. *Knowledge and Information Systems* **9**(3), 309–338 (2006).

[85] Johansson U. and Niklasson L. Evolving decision trees using oracle guides. In "2009 IEEE Symposium on Computational Intelligence and Data Mining", pages 238–244. IEEE (2009).

[86] 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).

[87] Kiang M. Y., Hu M. Y., and Fisher D. M. An extended self-organizing map network for market segmentation—a telecommunication example. *Decision Support Systems* **42**(1), 36–47 (2006).

[88] Kira K. and Rendell L. A. A practical approach to feature selection. In "Proceedings of the Ninth International Workshop on Machine Learning", pages 249–256 (1992).

[89] Kononenko I. Estimating attributes: analysis and extensions of RELIEF. In "Machine Learning: ECML-94", pages 171–182. Springer (1994).

[90] Koren Y., Bell R., and Volinsky C. Matrix factorization techniques for recommender systems. *Computer* **42**(8) (2009).

[91] Kotler P. and Cox K. K. "Marketing management and strategy". Prentice Hall (1980).

[92] Koufakou A. and Georgiopoulos M. A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes. *Data Mining and Knowledge Discovery* **20**(2), 259–289 (2010).

[93] Kouki P., Schaffer J., Pujara J., O'Donovan J., and Getoor L. Personalized explanations for hybrid recommender systems. In "Proceedings of the 24th International Conference on Intelligent User Interfaces", pages 379–390. ACM (2019).

[94] Krishnan R., Sivakumar G., and Bhattacharya P. Extracting decision trees from trained neural networks. *Pattern recognition* **32**(12) (1999).

[95] Krizhevsky A. and Hinton G. Learning multiple layers of features from tiny images. Technical Report, Citeseer (2009).

[96] Kumar V. Parallel and distributed computing for cybersecurity. *IEEE Distributed Systems Online* **6**(10), 1–10 (2005).

[97] Lazarevic A. and Kumar V. Feature bagging for outlier detection. In "Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining", pages 157–166. ACM (2005).

[98] LeCun Y., Bengio Y., and Hinton G. Deep learning. *nature* **521**(7553), 436 (2015).

[99] Lee W. and Stolfo S. J. A framework for constructing features and models for intrusion detection systems. *ACM transactions on Information and system security (TiSSEC)* **3**(4), 227–261 (2000).

[100] LibSVM dataset repository. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`. Accessed: 2016-04-19.

[101] Lichman M. UCI machine learning repository (2013).

[102] Lichman M. UCI machine learning repository (2013). `http://archive.ics.uci.edu/mll` [Last accessed April 2017].

[103] Lipton Z. C. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490* (2016).

[104] Liu C., Yang H., Fan J., He L., and Wang Y. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In "Proceedings of the 19th international conference on World wide web", pages 681–690. ACM (2010).

[105] Liu F. T., Ting K. M., and Zhou Z.-H. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* **6**(1), 3 (2012).

[106] Liu J., Liao X., Huang W., and Liao X. Market segmentation: A multiple criteria approach combining preference analysis and segmentation decision. *Omega* (2018).

[107] Lu Y.-C., Chen F., Wang Y., and Lu C.-T. Discovering anomalies on mixed-type data using a generalized student-t based approach. *IEEE Transactions on Knowledge and Data Engineering* **28**(10), 2582–2595 (2016).

[108] Luaces O., Díez J., Alonso-Betanzos A., Troncoso A., and Bahamonde A. A factorization approach to evaluate open-response assignments in MOOCs using preference learning on peer assessments. *Knowledge-Based Systems* **85**, 322–328 (2015).

[109] Luaces O., Díez J., Alonso-Betanzos A., Troncoso A., and Bahamonde A. Content-based methods in peer assessment of open-response questions to grade students as authors and as graders. *Knowledge-Based Systems* **117**, 79–87 (2017).

[110] Maaten L. v. d. and Hinton G. Visualizing data using t-sne. *Journal of Machine Learning Research* **9**(Nov), 2579–2605 (2008).

[111] Apache Mahout Project. `http://mahout.apache.org/`. Accessed: 2016-04-19.

[112] Martinez-Rego D., Castillo E., Fontenla-Romero O., and Alonso-Betanzos A. A Minimum Volume Covering Approach with a Set of Ellipsoids. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **35**(12), 2997–3009 (Dec 2013).

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

[114] Meng X., Bradley J., Yavuz B., Sparks E., Venkataraman S., Liu D., Freeman J., Tsai D., Amde M., Owen S., et al.. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* **17**(1), 1235–1241 (2016).

[115] Moonesignhe H. and Tan P.-N. Outlier detection using random walks. In "null", pages 532–539. IEEE (2006).

[116] Murdoch W. J., Singh C., Kumbier K., Abbasi-Asl R., and Yu B. Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592* (2019).

[117] Netzer Y., Wang T., Coates A., Bissacco A., Wu B., and Ng A. Y. Reading digits in natural images with unsupervised feature learning. In "NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011" (2011).

[118] Nicolau M., McDermott J., et al.. Learning neural representations for network anomaly detection. *IEEE transactions on cybernetics* **49**, 3074–3087 (2018).

[119] Otey M., Ghoting A., and Parthasarathy S. Fast distributed outlier detection in mixed-attribute data sets. *Data Mining and Knowledge Discovery* **12**(2-3), 203–228 (2006).

[120] Palma-Mendoza R.-J., Rodriguez D., and de Marcos L. Distributed relieff-based feature selection in spark. *Knowledge and Information Systems* pages 1–20 (2018).

[121] Papadimitrou H., S.and Kitagawa, Gibbons P., and Faloutsos C. Loci: Fast outlier detection using the local correlation integral. Technical report irp-tr-02-09, Intel Research Laboratory (2002).

[122] Pham N. Hybrid lsh: Faster near neighbors reporting in high-dimensional space. *arXiv preprint arXiv:1607.06179* (2016).

[123] Platt J. et al.. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods — Support Vector Learning* **3** (1999).

[124] Quinlan J. R. "C4. 5: programs for machine learning". Elsevier (2014).

[125] Quinlan J. R. Induction of decision trees. *Machine Learning* **1**(1), 81–106 (1986).

[126] Ravichandran D., Pantel P., and Hovy E. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In "Proceedings of the 43rd annual meeting on association for computational linguistics", pages 622–629. Association for Computational Linguistics (2005).

[127] Robnik-Šikonja M. and Kononenko I. An adaptation of relief for attribute estimation in regression. In "Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)", pages 296–304 (1997).

[128] Saini S., Chang J., and Jin H. Performance evaluation of the Intel Sandy Bridge based NASA Pleiades using scientific and engineering applications. In "High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation", pages 25–51. Springer (2014).

[129] Sankaranarayanan J., Samet H., and Varshney A. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics* **31**(2), 157–174 (2007).

[130] Sarasamma S. T., Zhu Q. A., and Huff J. Hierarchical kohonenen net for anomaly detection in network security. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **35**(2), 302–312 (2005).

[131] Scholkopf B., Platt J., Shawe-Taylor J., Smola A., and Williamson R. Estimating the support of a high-dimensional distribution. *Neural Computation* **13**(7), 1443–1471 (2001).

[132] Schubert E., Zimek A., and Kriegel H. Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. *Data Mining and Knowledge Discovery* pages 1–48 (2012).

[133] Schwarz G. Estimating the dimension of a model. *The annals of statistics* **6**(2), 461–464 (1978).

[134] Shan H. and Banerjee A. Bayesian co-clustering. In "Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on", pages 530–539. IEEE (2008).

[135] Shiravi A., Shiravi H., Tavallaee M., and Ghorbani A. A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security* **31**(3), 357–374 (2012).

[136] Singh S., Tu H., Donat W., Pattipati K., and Willett P. Anomaly detection via feature-aided tracking and hidden markov models. *IEEE Transactions*

on Systems, Man, and Cybernetics-Part A: Systems and Humans **39**(1), 144–159 (2009).

[137] Slavkov I., Karcheska J., Kocev D., Kalajdziski S., and Džeroski S. Relieff for hierarchical multi-label classification. In "International Workshop on New Frontiers in Mining Complex Patterns", pages 148–161. Springer (2013).

[138] Sodemann A., Ross M., and Borghetti B. A review of anomaly detection in automated surveillance. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* **42**(6), 1257–1272 (2012).

[139] Sonnenburg S., Franc V., Yom-Tov E., and Sebag M. Pascal large scale learning challenge. In "25th International Conference on Machine Learning (ICML2008) Workshop", volume 10, pages 1937–1953 (2008).

[140] Spark implementation of Fayyad's discretizer based on Minimum Description Length Principle (MDLP). `http://spark-packages.org/package/sramirez/spark-MDLP-discretization`. Accessed: 2016-04-19.

[141] Spolaôr N., Cherman E. A., Monard M. C., and Lee H. D. Relieff for multi-label feature selection. In "Intelligent Systems (BRACIS), 2013 Brazilian Conference on", pages 6–11. IEEE (2013).

[142] Stamper J., Niculescu-Mizil A., Ritter S., Gordon G., and Koedinger K. Bridge to algebra data set from KDD Cup 2010 educational data mining challenge (2010).

[143] Strubell E., Ganesh A., and McCallum A. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243* (2019).

[144] S.Wu and Wang S. Parameter-free anomaly detection for categorical data. In "Proceedings of the 7th International Conference on Machine Learning and Data Mining, MLDM 2011. Lecture notes in Computer Science", volume 6871, pages 112–126 (2011).

[145] Urbanowicz R. J., Meeker M., LaCava W., Olson R. S., and Moore J. H. Relief-based feature selection: introduction and review. *arXiv preprint arXiv:1711.08421* (2017).

[146] Veiga J., Expósito R. R., Taboada G. L., and Touriño J. MREv: An automatic mapreduce evaluation tool for big data workloads. *Procedia Computer Science* **51**, 80–89 (2015).

[147] Wang J., Wang J., Zeng G., Tu Z., Gan R., and Li S. Scalable k-nn graph construction for visual descriptors. In "Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on", pages 1106–1113. IEEE (2012).

[148] Wang X., He X., Feng F., Nie L., and Chua T. Tem: Tree-enhanced embedding model for explainable recommendation. In "Proceedings of the 2018 World Wide Web Conference on World Wide Web", pages 1543–1552. International World Wide Web Conferences Steering Committee (2018).

[149] Wei L., Qian W., Zhou A., Jin W., and Yu J. Hot: Hypergraph-based outlier test for categorical data. In Whang K., Jongwoo J., Shim K., and Srivastava J., editors, "Advances in Knowledge Discovery and Data Mining", volume 2637 of "Lecture Notes in Computer Science", pages 399–410. Springer Berlin Heidelberg (2003).

[150] Wu F., Weimer W., Harman M., Jia Y., and Krinke J. Deep parameter optimisation. In "Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation", pages 1375–1382. ACM (2015).

[151] Wu S. and Wang S. Information-theoretic outlier detection for large-scale categorical data. *IEEE transactions on knowledge and data engineering* **25**(3), 589–602 (2013).

[152] Xu S., Li X., and Lu W. F. Randomized kd tree relieff algorithm for feature selection in handling high dimensional process parameter data. In "Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on", pages 1–8. IEEE (2016).

[153] Zaharia M., Chowdhury M., Franklin M. J., Shenker S., and Stoica I. Spark: cluster computing with working sets. In "Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing", volume 10, page 10 (2010).

[154] Zhang K. and Jin H. An effective pattern based outlier detection approach for mixed attribute data. In "Australasian Joint Conference on Artificial Intelligence", pages 122–131. Springer (2010).

[155] Zhang Y.-M., Huang K., Geng G., and Liu C.-L. Fast knn graph construction with locality sensitive hashing. In "Joint European Conference on Machine Learning and Knowledge Discovery in Databases", pages 660–674. Springer (2013).

[156] Zhao Y., Liang S., Ren Z., Ma J., Yilmaz E., and de Rijke M. Explainable user clustering in short text streams. In "Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval", pages 155–164. ACM (2016).

[157] Zimek A., Schubert E., and Kriegel H.-P. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal* **5**(5), 363–387 (2012).

Resumen extendido

El reciente aumento de la cantidad de datos disponibles ha dado lugar a una nueva y prometedora era del aprendizaje máquina. La oleada de sensorización y de toma de datos en todos los contextos, especialmente en el entorno web, han dado lugar a ingentes cantidades de datos que están constituyendo un combustible con el que el aprendizaje máquina está tomando gran velocidad. Los éxitos en este campo se están sucediendo a un ritmo cada vez mayor gracias a la capacidad de algunos algoritmos de aprovechar inmensas cantidades de datos para producir predicciones difíciles y muy certeras. En particular, el aprendizaje profundo o Deep Learning está obteniendo resultados que en algunos casos superar las capacidades humanas para problemas tradicionalmente considerados inasequibles para las máquinas. No obstante, estos progresos llegan con un coste alto. La gran demanda computacional de estos algoritmos está convirtiendo la investigación en este terreno en particular en un coto cerrado reservado a las entidades con los grandes medios monetarios requeridos para el entrenamiento de este tipo de modelos. Además, existe una creciente preocupación por los costes energéticos asociados al uso de esta tecnología. A mayores, muchos de los algoritmos hasta ahora disponibles para los científicos de datos que ofrecían grandes resultados han perdido su efectividad en este nuevo escenario debido a las complicaciones asociadas al aprendizaje a gran escala que han llevado a su abandono en favor de alternativas menos precisas pero que sí pueden manejar estos volúmenes. Trabajar con grandes conjuntos de datos conlleva problemas logísticos, dado que el manejo y almacenamiento de grandes cantidades de datos se escapa de las capacidades de las tecnologías tradicionales. El aprendizaje a gran escala también limita la complejidad computacional y espacial de los algoritmos utilizados, siendo los algoritmos con coste lineal o menor los que mejor se prestan a este escenario, frente a alternativas que ofrecen mejores resultados pero a un coste computacional mayor. Este escenario también favorece los métodos con pocos o ningún hiperparámetro a configurar, dado el alto coste que tiene realizar muchas iteraciones de entrenamientos de prueba para ajustar dichos valores. Por último, el aprendizaje a gran escala muestra complicaciones específicas que dificultan el aprendizaje tales como la maldición de la dimensionalidad en el caso de conjuntos de datos con un gran número de variables.

Existe, por tanto, una oportunidad en el estudio de algoritmos de aprendizaje máquina que puedan realizar aprendizaje a gran escala. Tanto el mundo académico como el empresarial se beneficiarían de la existencia de nuevos algoritmos que se puedan enfrentar a grandes conjuntos de datos. Se conoce como escalabilidad a la capacidad de los algoritmos de mantener su efectividad a medida que la escala del conjunto de datos aumenta. Esta tesis se centra en la escalabilidad de los algoritmos de aprendizaje máquina y en ella exploraremos tanto modos de mejorar la escalabilidad de algoritmos existentes como nuevos desarrollos de algoritmos que tienen la escalabilidad como meta de diseño.

En el panorama actual del aprendizaje máquina, problemas clásicos como la predicción y la regresión cuentan con soluciones muy eficaces, generalmente basadas en aprendizaje profundo, que además son capaces de tratar con grandes conjuntos de datos. Es por ello que en esta tesis ponemos el foco en problemas cuyas soluciones actuales tienen problemas al aumentar la escala. Por tanto, obviando las mencionadas clasificación y regresión, nos centramos en otros problemas. En particular, exploraremos la selección de características, definida como el estudio del valor predictivo de cada una de las variables de entrada con el fin de desechar aquellas que sean redundantes y quedarse con un subconjunto de variables con gran valor predictivo. También trataremos la detección de anomalías, es decir, la identificación de patrones de entrada que no se ajustan a la distribución del resto de datos hasta el punto de hacer sospechar que puedan haber sido generados por un proceso distinto al normal. Además trabajaremos en la construcción de grafos, en particular del grafo de vecinos más cercanos, en su uso en problemas de aprendizaje máquina y, en concreto, a la selección de características. Y, por último, haremos una incursión en el aprendizaje máquina explicable, que está adquiriendo gran auge en tiempos recientes debido a las crecientes preocupaciones respecto a la opacidad de los algoritmos de aprendizaje máquina tradicionales.

Analizamos el uso de cuatro estrategias diferentes para obtener los mencionados algoritmos escalables nuevos o para transformar costosos algoritmos ya existentes para dotarlos de mayor escalabilidad. En primer lugar, nos centramos en el procesamiento distribuido, una tecnología que se encuentra detrás de todos los avances recientes que ha experimentado el aprendizaje máquina. La reestructuración de algoritmos para que los cálculos que sean independientes entre sí se realicen simultáneamente es una manera eficaz de acelerar el aprendizaje de modelos complejos, permitiendo tratar conjuntos de datos mayores. El paradigma de programación Map Reduce facilita este proceso de reestructuración de los algoritmos al homogeneizar la estructura. Plataformas de código abierto como Apache Hadoop o Apache Spark permiten al desarrollador centrarse

en la implementación del algoritmo, dejando los detalles logísticos de coordinación de las máquinas que están realizando el cómputo, gestión de fallos, transferencia y almacenamiento de datos y demás a cargo de la plataforma. En esta tesis utilizaremos procesamiento paralelo y presentamos implementaciones en Apache Spark de todos los algoritmos tratados.

En el segundo capítulo exploramos en detalle las ventajas del procesamiento distribuido frente al procesamiento paralelo en una sola máquina. Lo hacemos mediante el desarrollo de nuevas alternativas escalables para la selección de características. Al centrarnos en este problema no solo podemos hacer la comparativa mencionada entre implementaciones secuenciales, paralelas y distribuidas, sino que también portamos al nuevo escenario de aprendizaje a gran escala herramientas que tradicionalmente habían resultado muy útiles. Paradójicamente, existen pocas alternativas a la hora de realizar selección de características en conjuntos de grandes dimensiones, a pesar de que la utilidad de estos métodos en dicho entorno es, si cabe, aún mayor que en problemas de la escala tradicional. Es por ello que tomamos como objeto de estudio cuatro populares algoritmos de selección de características que se incluyen en la plataforma de aprendizaje máquina Weka con el objetivo de adaptarlos al nuevo escenario. La implementación secuencial provista por Weka, unida al alto coste computacional que tienen estos algoritmos, limitan efectivamente el tamaño de los conjuntos de datos que pueden ser procesados. Para los usuarios de Weka proponemos implementaciones paralelas de los mismos algoritmos que se ejecutan en la máquina del usuario aprovechando toda la capacidad computacional de la máquina, al utilizar todos los núcleos de computación disponible, acelerando sensiblemente el proceso. Para los usuarios con acceso a un clúster de computación, presentamos implementaciones distribuidas de los mismos algoritmos en Apache Spark, que permiten utilizar varias máquinas para poder así enfrentarse a conjuntos de datos muy grandes en un tiempo razonable. Los resultados experimentales detallados muestran que la implementación paralela ofrecen muy notables ventajas respecto a la versión estándar disponible en Weka, mientras que la implementación en Apache Spark ofrece aún mejores resultados para aquellos usuarios que tengan acceso a los recursos computacionales necesarios.

El tercer capítulo lo dedicamos a analizar la posibilidad de utilizar modelos aproximados para acelerar cómputos costosos. La computación distribuida, mencionada anteriormente, desplaza el alto coste temporal de los cómputos hacia unos mayores requisitos de capacidad computacional, es decir, introduce una compensación del coste temporal por coste de hardware. Sin embargo, y dado que la complejidad computacional de algunos algoritmos es alta, incluso una implementación distribuida puede requerir

una capacidad computacional y un tiempo inasequibles. Por ello vale la pena explorar la posibilidad de realizar cálculos que aproximen en poco tiempo el resultado exacto buscado por el algoritmo original y comprobar su eficacia frente al uso del cómputo exacto. Exploramos esta estrategia en el contexto del cómputo del grafo de vecinos más cercanos, una estructura de datos utilizada en gran variedad de campos de la computación en general y del aprendizaje máquina en particular, que consiste en un grafo dirigido donde cada puntos se enlaza con sus k vecinos más cercanos. El cálculo exacto de este grafo en casos generales tiene un coste computacional del orden del cuadrado del número de elementos del conjunto de datos, lo cual lo convierte en demasiado costoso para conjuntos de datos grandes. Proponemos el algoritmo VRLSH (Variable Resolution Locality Sensitive Hashing), que hace uso de Hashing Sensible a la Localidad (LSH) para reducir esta ingente cantidad de datos, evitando mediciones de distancias entre puntos para los que tenemos una probabilidad de vecindad muy baja. El LSH consiste en la utilización de funciones construidas especialmente que envían elementos de un espacio de entrada de alta dimensión y generalmente muy disperso a un espacio de dimensión mucho menor y con mucha mayor densidad. Al hacerlo, tienen la particularidad de que envían al mismo punto de destino aquellos puntos que en el espacio de entrada se encontraban cercanos. Este proceso nos permite obtener agrupaciones de puntos cercanos entre los cuales realizar mediciones de distancia, evitando dicho cómputo para puntos que sabemos distantes con alta probabilidad. El uso de LSH para este problema también tiene la ventaja de que permite trabajar con medidas de distancia muy variables, con el único requisito de que exista una función de hashing apropiada para esa medida. No obstante, en este trabajo nos centraremos en el caso más popular, que es el del uso de la medida de distancia euclídea. La implementación propuesta, que además aprovecha la estructura del problema para distribuir los cálculos para que se puedan efectuar en paralelo, obtiene grafos muy coincidentes con grafo exacto y lo hace utilizando tan solo una pequeña fracción del esfuerzo computacional. Detallamos además experimentos que demuestran que esto es así para distintos conjuntos de datos con características muy variables y que la precisión y velocidad obtenidas superan a los métodos alternativos.

Esta nueva implementación ha sido probada en un caso de uso aplicado al aprendizaje máquina, proceso que es detallado en el capítulo cuarto. En él se explican las modificaciones que se tuvo que hacer a VRLSH para poder aplicarlo al problema de la selección de características con el popular método ReliefF. Este algoritmo utiliza las diferencias y similitudes entre las características de vecinos más cercanos para establecer qué atributos son más determinantes a la hora de diferenciar elementos de distintas clases, estableciendo así un clasificación de las características por orden de importancia

que se puede utilizar para desechar las menos relevantes. Para realizar este proceso, ReliefF debe disponer primeramente del grafo de vecinos más cercanos de cada clase, cuyo coste computacional cuadrático impide que ReliefF se pueda utilizar para seleccionar características de conjuntos de datos muy numerosos. Presentamos, por tanto, ReliefF-LSH, un algoritmo que utiliza una variante de VRLSH que respeta las distintas clases de cada punto y obtiene una aproximación certera del deseado grafo de vecinos más cercanos de cada clase en una pequeña fracción de tiempo requerido por la versión original. Los resultados experimentales demuestran que, en una variedad de conjuntos de datos muy diversos en sus dimensiones, las características seleccionadas por ReliefF-LSH difieren poco de las seleccionadas por el más costoso ReliefF, mientras que el tiempo de computación se reduce sensiblemente. Además, ReliefF-LSH ofrece mejores resultados que otros métodos de aproximar ReliefF, pudiendo, a diferencia de estos, enfrentarse a todos los tipos de datos que se pueden procesar con la versión exacta de ReliefF.

La tercera estrategia para aumentar la escalabilidad de los algoritmos consiste en el diseño de modelos que aprovechan una particularidad de los datos de entrada para simplificar el entrenamiento y se detalla en el capítulo quinto. En particular, tratamos el problema de la detección de anomalías en el contexto particular en que los datos de entrada son una mezcla de variables numéricas y categóricas, un caso muy frecuente. Generalmente esta característica de los datos de entrada se obvia y se transforman los datos de entrada a valores exclusivamente numéricos o, con menos frecuencia, a valores exclusivamente categóricos. Los algoritmos que sí mantienen esa distinción entre tipos de variables tienen un coste computacional que impide su uso en conjuntos de datos grandes, tal como mostramos en la sección experimental del capítulo. Proponemos un algoritmo, llamado ADMNC (Anomaly Detection in Mixed Numerical and Categorical inputs), que transforma esta complicación de los datos de entrada en una ventaja que le permite desgranar el modelo a aprender en dos partes más sencillas. Para ello, se propone una factorización de la probabilidad conjunta de las variables de entradas en el producto de la probabilidad de las variables continuas por la probabilidad de las variables categóricas condicionadas a las continuas, obteniendo para cada punto un estimador que, tras establecer un umbral, se puede utilizar para detectar datos anómalos. Para modelar la probabilidad de las variables numéricas se utiliza una mezcla de gaussianas, mientras que para modelar la probabilidad de las variables categóricas condicionadas a las numéricas se utiliza un modelo de regresión logística. Estos dos modelos son sencillos y se pueden aprender con poco esfuerzo computacional, por lo que el algoritmo resultante consigue mantener la distinción entre los dos tipos de variables y escalar a grandes conjuntos de datos. Los experimentos realizados demuestran que

el algoritmo ofrece resultados comparables al estado del arte para conjuntos de datos pequeños, mientras que en conjuntos de datos grandes obtiene resultados superiores a las alternativas disponibles.

Finalmente, en el capítulo seis detallamos la utilización de la computación especulativa a la que da acceso la programación distribuida para mejorar la eficiencia de modelos que, a priori, serían demasiado sencillos para aplicar en contextos de aprendizaje a gran escala. Aplicamos, además, esta idea en el floreciente campo del Aprendizaje Máquina Explicable, que busca dotar a los algoritmos de aprendizaje máquina, hasta ahora opacos a la hora de justificar sus predicciones, de características que permitan verificar que su comportamiento es el adecuado, que sus predicciones generalizan a datos que divergen mucho de aquellos utilizados para entrenar, que les permitan ofrecer información novedosa y que permitan formular hipótesis que luego se puedan verificar experimentalmente. Estas características no solo son de gran interés comercial por el gran abanico de posibilidades que descubren, sino que incluso son un requisito legal tras la implantación de normativas como el RGPD (Reglamento General de Protección de Datos) de la Unión Europea, vigente desde hace algunos meses. Para atacar este problema, Presentamos un algoritmo que proporciona una explicación a las predicciones realizadas por una función de utilidad que relaciona elementos de dos entidades diferentes. Este tipo de datos, conocidos como datos diádicos, está presente en muchos problemas de computación que son de gran interés no solo académico sino también comercial. Obtener una explicación de estos datos es, por tanto, un objetivo codiciado para el cual existían pocas alternativas disponibles. El algoritmo presentado hace uso de un árbol de decisión para obtener grupos de elementos de una entidad que se comportan de manera homogénea con respecto a la otra entidad. Este planteamiento consigue que los grupos se puedan describir con muy pocas variables de entrada, constituyendo, de facto, una explicación de la información codificada en la función de utilidad que relaciona ambas entidades. La computación distribuida permite la exploración de un gran número de posibles árboles de decisión en un tiempo razonable. Proponemos, además, una medida de calidad que permite evaluar la idoneidad de una explicación sobre datos diádicos teniendo en cuenta no solo su capacidad predictiva sino también su explicabilidad. Los resultados experimentales muestran la superioridad de nuestro método frente a las alternativas disponibles, así como apuntan a posibles casos de uso de la información novedosa extraída en el proceso.

Además, dado que la escalabilidad de los algoritmos de aprendizaje máquina es el tema central de esta tesis, para todos los algoritmos presentados se ha llevado a cabo un estudio de su tiempo de ejecución a medida que se añaden más máquinas al cómputo.

Estos experimentos están descritos y detallados en cada una de las secciones experimentales de los distintos capítulos y muestran, en todos los casos, que los algoritmos obtenidos son altamente escalables, dado que el tiempo de ejecución desciende en la misma proporción en que se añaden máquinas al cómputo. Esto faculta a los algoritmos presentados para tratar con conjuntos de datos masivos, siempre y cuando se dispongan de máquinas suficientes para llevar a cabo los cálculos en un tiempo razonable.

En el desarrollo de esta tesis hemos obtenido desarrollos que hemos puesto a disposición de la comunidad académica y de científicos de datos. Hemos implementado cuatro nuevos algoritmos y seis versiones de algoritmos existentes que tratan los problemas mencionados. Todos los algoritmos presentados han sido puestos a disposición del lector para su descarga en repositorios de código. Además, los resultados se han difundido mediante publicaciones en revistas científicas y presentaciones en congresos nacionales e internacionales para facilitar que tanto investigadores como científicos de datos puedan conocerlos y utilizarlos. A fecha de presentación de la tesis, dos de los trabajos han sido publicados en revistas que se encuentran en el primer cuartil del índice Journal Citation Records, mientras que otro trabajo ha sido publicado en una revista que se encuentra en el segundo cuartil del mismo índice. Además, dos trabajos adicionales están bajo revisión en revistas del primer cuartil, y otros cinco trabajos se han presentado en congresos especializados, uno de ellos en un congreso internacional.

Esta tesis se cierra con un capítulo donde se detallan las principales conclusiones a las que se llegó tras el desarrollo de los trabajos que la componen. En particular, se refrenda la validez de las estrategias exploradas y se indica aquéllas que ofrecen mayores posibilidades de crecimiento. En ese sentido, también se listan las nuevas líneas de investigación abiertas y las posibles avenidas de crecimiento que todavía no se han comenzado a explorar pero que muestran indicios de tener potencial. Esto se une a la sección de conclusiones de cada uno de los capítulos individuales, donde se detallan las mejoras previstas de cada uno de los algoritmos presentados.

En resumen, esta tesis es una exploración de los modos en que afrontar problemas que inicialmente están más allá de las capacidades de un algoritmo mediante la colaboración entre varias máquinas para aunar esfuerzos, entre otras estrategias. Aplicamos esa misma filosofía a la investigación y por ello proporcionamos implementaciones de todos nuestros algoritmos, esperando que sean un granito de arena en el gran esfuerzo colectivo de construir máquinas inteligentes.