2019

Distributed multi-label learning on Apache Spark

Jorge Gonzalez Lopez
Virginia Commonwealth University

DISTRIBUTED MULTI-LABEL LEARNING ON APACHE SPARK

APRENDIZAJE MULTI-ETIQUETA DISTRIBUIDO EN APACHE SPARK

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University and the University of Córdoba.

by

JORGE GONZALEZ LOPEZ

Ph.D. Candidate

Director: Alberto Cano,

Assistant Professor, Department of Computer Science,

Virginia Commonwealth University

Director: Sebastian Ventura,

Professor, Department of Computer Science & Numerical Analysis,

University of Córdoba

Virginia Commonwealth University

Richmond, Virginia

University of Córdoba

Córdoba, Spain

April 2019

The document entitled "Distributed multi-label learning on Apache Spark", reported by Jorge González López to qualify for the doctoral degree, has been conducted under the program 'Engineering, Doctor of Philosophy (Ph.D.) with a concentration in computer science/Computer Science Ph.D. with the University of Cordoba" at the Virginia Commonwealth University, under the supervision of the doctors Alberto Cano Rojas (Virginia Commonwealth University) and Sebastián Ventura Soto (University of Córdoba) fulfilling, in their opinion, the requirements demanded by this type of works and respecting the rights of other authors to be cited, when their results or publications have been used.

Richmond, April 2019

El Doctorando

Fdo: Jorge González López

El Director

Fdo: Alberto Cano Rojas

El Director

Fdo: Sebastián Ventura Soto

La memoria titulada "Distributed multi-label learning on Apache Spark", que presenta Jorge González López para optar al grado de doctor, ha sido realizada dentro del programa dual de doctorado "Doctorado (Ph.D.) en Ciencias de la Computación" de la Virginia Commonwealth University, bajo la dirección de los doctores Alberto Cano Rojas (Virginia Commonwealth University) y Sebastián Ventura Soto (University of Córdoba) cumpliendo, en su opinión, los requisitos exigidos a este tipo de trabajos y respetando los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Richmond, Abril de 2019

El Doctorando

Fdo: Jorge González López

El Director

Fdo: Alberto Cano Rojas

El Director

Fdo: Sebastián Ventura Soto

"Shoot for the moon. Even if you miss, you'll land among the stars."

— Norman Vincent Peale

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Cano, for granting me the incredible opportunity to move to the U.S. where I could carry out my doctoral studies. Additionally, I would like to thank all the people involved in the execution of this thesis.

My deepest gratitude goes out to my parents and my brother. This dissertation would not have been possible without their long-distance support through all these years.

All my appreciation to my friends, which fortunately are too many to be listed here. I am thankful to my friends in Spain for welcoming me every time I go there as if I never left and to my friends in the U.S. for helping me feel at home and patiently listening every time.

I am also grateful to Dylan for his friendship and his ability to cheer me up.

The last words of acknowledgment are saved for my dear wife, Kwan, for her warm love, continued patience, and endless support.

Table of Contents

List of Tables

List of Figures

Abstract (English)

This thesis proposes a series of multi-label learning algorithms for classification and feature selection implemented on the Apache Spark distributed computing model.

Five approaches for determining the optimal architecture to speed up multi-label learning methods are presented. These approaches range from local parallelization using threads to distributed computing using independent or shared memory spaces. It is shown that the optimal approach performs hundreds of times faster than the baseline method.

Three distributed multi-label k nearest neighbors methods built on top of the Spark architecture are proposed: an exact iterative method that computes pair-wise distances, an approximate tree-based method that indexes the instances across multiple nodes, and an approximate local sensitive hashing method that builds multiple hash tables to index the data. The results indicated that the predictions of the tree-based method are on par with those of an exact method while reducing the execution times in all the scenarios.

The aforementioned method is then used to evaluate the quality of a selected feature subset. The optimal adaptation for a multi-label feature selection criterion is discussed and two distributed feature selection methods for multi-label problems are proposed: a method that selects the feature subset that maximizes the Euclidean norm of individual information measures, and a method that selects the subset of features maximizing the geometric mean. The results indicate that each method excels in different scenarios depending on the type of features and the number of labels.

Rigorous experimental studies and statistical analyses over many multi-label metrics and datasets confirm that the proposals achieve better performances and provide better scalability to bigger data than the methods compared in the state of the art.

Abstract (Spanish)

Esta Tesis Doctoral propone unos algoritmos de clasificación y selección de atributos para aprendizaje multi-etiqueta distribuidos implementados en Apache Spark.

Cinco estrategias para determinar la arquitectura óptima para acelerar el aprendizaje multi-etiqueta son presentadas. Estas estrategias varían desde la paralelización local utilizando hilos hasta la distribución de la computación utilizando espacios de memoria compartidos o independientes. Ha sido demostrado que la estrategia óptima permite ejecutar cientos de veces más rápido que el método de referencia.

Se proponen tres métodos distribuidos de "k nearest neighbors" multi-etiqueta sobre la arquitectura de Spark seleccionada: un método exacto que computa iterativamente las distancias, un método aproximado que usa un árbol para indexar las instancias, y un método aproximado que utiliza tablas hash para indexar las instancias. Los resultados indican que las predicciones del método basado en árboles son equivalente a aquellas producidas por un método exacto a la vez que reduce los tiempos de ejecución en todos los escenarios.

Dicho método es utilizado para evaluar la calidad de un subconjunto de atributos. Se discute el criterio para seleccionar atributos en problemas multi-etiqueta, proponiendo: un método que selecciona el subconjunto de atributos cuyas medidas de información individuales poseen la mayor norma Euclídea, y un método que selecciona el subconjunto de atributos con la mayor media geométrica. Los resultados indican que cada método destaca en escenarios diferentes dependiendo del tipo de atributos y el número de etiquetas.

Los estudios experimentales y análisis estadísticos utilizando múltiples métricas y datos multi-etiqueta confirman que nuestras propuestas alcanzan un mejor rendimiento y proporcionan una mejor escalabilidad para datos de gran tamaño respecto a los métodos de referencia.

Resumen

La aparición de nuevas tecnologías ha dado lugar a un incremento exponencial del volumen de datos almacenados en los sistemas modernos. La cantidad de datos generados por los consumidores continúa creciendo en números absolutos. Por otra parte, el volumen de datos generados por otros sistemas de computación también está sufriendo un rápido incremento. Al mismo tiempo que los datos aumentan en volumen, también lo hacen en complejidad, lo cual supone un obstáculo a la hora de utilizarlos para diferentes fines. El crecimiento exponencial de los datos, tanto en tamaño como en complejidad, han producido la necesidad de encontrar técnicas que puedan extraer la información útil de forma precisa, eficiente y escalable.

Aprendizaje Automático (Machine learning) engloba el conjunto de técnicas más avanzadas para la extracción de información de una serie de datos. Esta metodología extrae la información generalizando por experiencia, es decir, es capaz de aprender reglas y relaciones entre datos ya conocidos y extrapolarlos a otros datos. Uno de los paradigmas dentro de este campo es conocido como aprendizaje multi-etiqueta, en el cual cada instancia de los datos es asociada con múltiples variables (etiquetas) simultáneamente.

Desafortunadamente la capacidad de cómputo de los procesadores no ha incrementado de la misma forma que el volumen y la complejidad de los datos. Por lo que la mayoría de los investigadores y empresas se ha visto forzada a migrar el cómputo de las tareas a entornos compuestos por múltiples máquinas. Estos entornos requieren de nuevas herramientas de programación orientadas a sistemas distribuidos. El modelo de programación más popular orientado a la computación de datos a gran escala es MapReduce. Este modelo define cómo la

computación se puede distribuir entre varias máquinas mediante un particionamiento de los datos. Una de las primeras implementaciones de este modelo es Hadoop, desafortunadamente Hadoop depende del uso de almacenamiento secundario lo cual introduce una alta latencia. Apache Spark es un framework que soluciona este problema mediante la posibilidad de mantener y operar con los datos en memoria. Esto hace que Spark se proclame como la mejor opción en la actualidad para la ejecución de aplicaciones con una alta intensidad de cómputo, como por ejemplo Aprendizaje Automático.

La presente Tesis Doctoral propone una serie de algoritmos para problemas multi-etiqueta utilizando Apache Spark como modelo de programación distribuido. El objetivo es proponer nuevos métodos para el aprendizaje y procesamiento de datos multi-etiqueta caracterizados por una gran cantidad de instancias, atributos, y/o etiquetas. Para ello se han estudiado una serie de propuestas, con un enfoque ascendiente en el cual cada una ha establecido la base de la siguiente.

En primer lugar, se han estudiado diferentes estrategias para la distribución del aprendizaje de datos multi-etiqueta. Para ello se propusieron el estudio de cinco estrategias diferentes: una implementación base que utilizaba unúnico hilo en una única máquina, una versión que utilizaba múltiples hilos en una misma máquina, una versión distribuida que utilizada la versión mono-hilo en múltiples máquinas, otra versión distribuida que utilizaba la versión multi-hilo en múltiples máquinas, y, por último, una versión que extendía los métodos nativos de Spark. Esta versión construye modelos colaborativos distribuyendo las instancias, mientras que las anteriores requieren de todas las instancias en todas las máquinas y cada una utiliza diferentes conjuntos de etiquetas. El análisis de los resultados demuestra que la distribución de las instancias en el modelo nativo de Spark produce un mayor rendimiento y mejor escalabilidad.

Utilizando esta estrategia para el aprendizaje de modelos multi-etiqueta, se decide estudiar uno de los métodos con más aplicaciones pero que a su vez sufre de grandes problemas de rendimiento: multi-label k Nearest Neighbor (Ml-knn). Este método hace

predicciones en base a la distancia que separa diferentes instancias. Por lo tanto, requiere del almacenaje de todas las instancias conocidas. Para discernir de la mejor estrategia para poder incorporar este método a sistemas distribuidos y que aprenda de un gran volumen de datos, se estudian diferentes implementaciones: una implementación iterativa que comprueba las distancias entre todas las instancias, otra que utiliza un árbol para indexar las instancias a través de múltiples máquinas, y, por último, una que utiliza una serie de hashes para indexar y agrupar las instancias. Estos métodos son comparados y evaluados respecto a sus predicciones, sus tiempos de ejecución, y su escalabilidad respecto al número de instancias, atributos, y etiquetas. Los resultados del estudio indican que el método basado en un árbol permite ejecutar cientos de veces más rápido que los otros métodos manteniendo una precisión equivalente a la de los métodos exactos.

Una de las características que más dificulta el aprendizaje de los modelos multi-etiqueta es la gran dimensionalidad de los datos, es decir, el gran número de atributos asignado a las instancias. Por otra parte, se ha demostrado en múltiples ocasiones que la calidad de los modelos aumenta descartando atributos irrelevantes y/o redundantes. Pero aún existe mucho debate en torno a la forma en la que se puede evaluar cada uno de los atributos respecto a múltiples etiquetas. Por lo tanto, realizamos un análisis detallado de las diferentes estrategias discutiendo sus ventajas y desventajas. Tras lo cual seleccionamos el método que consideramos más conveniente, especialmente para datos caracterizados por su gran número de instancias y atributos. En base a esta estrategia proponemos dos métodos nuevos, los cuales no requieren de ningún tipo de transformación de datos y son capaces de utilizar múltiples etiquetas simultáneamente. El primer método selecciona los atributos cuyas medidas individuales de información forman la mayor normal Euclídea, mientras que el segundo método selecciona aquellos que presentan la mayor media geométrica. Los resultados indican que el primer método presenta los mejores resultados para datos binarios y con un menor número de etiquetas, mientras que el segundo método en preferible para datos continuos o con un mayor número de etiquetas.

Todos los algoritmos y métodos de la presente Tesis Doctoral han sido evaluados mediante una serie de test no paramétricos. Los algoritmos propuestos han sido comparados frente a los algoritmos más utilizados en el estado del arte en cada una de las tareas, por lo tanto, validando la eficiencia de cada una de las propuestas.

Finalmente se presentan una serie de líneas de investigación orientadas a ampliar o mejorar las conclusiones obtenidas en la presente Tesis Doctoral. Todas estas líneas de investigación están relacionadas con el aprendizaje multi-etiqueta y/o el aprendizaje en sistemas distribuidos.

CHAPTER 1

INTRODUCTION

The emergence of new technologies has led to an exponential increase in the volume of data that is produced worldwide. Some of the most relevant factors that leading this growth are the incremental use of the Internet and social networks, the falling costs of the technology devices, the migration from analog to digital systems, the growth of machine-generated data, among others. According to International Data Corporation (IDC), the amount of data created, replicated and consumed worldwide was about 16 ZB (1 ZB = 10^{12} GB) in 2016 [1]. Moreover, the current predictions forecast that by 2025, the global data will grow to 163 ZB [2], which is more than 10 times the amount of data generated so far.

Consumers usually have accounted for around 70% of the total data created and consumed per year. While the amount of data created by consumers will continue to grow in absolute numbers, the volume of data generated by machines continues to grow quickly. The amount of information created by consumers is being replaced by the information being created about them [3]. Consequently, the percentage of data that is generated by machines per year is increasing. This presents a promising scenario since the readings from machines monitoring our world usually are more valuable than the data generated by consumers, which many times is entertainment related. IDC estimates that by 2020, as much as 33% of the digital data will contain valuable information if analyzed.

The rise of the big data age poses serious problems and challenges besides the obvious benefits. As data becomes larger, it also becomes more complex and inexplicable, which would pose substantial difficulties in deciphering and interpreting it by the limited mental

capabilities of humans [4]. Additionally, almost every business has a high demand for data processing in real-time, because of business demands and competitive pressure. As a result, the first problem is how to mine valuable information from massive data efficiently and accurately. The second problem is to choose appropriate techniques that may harness the worthwhile information in the vast collections of data.

Data scientists were the early pioneers in the big data research, and it has been one of the most popular topics due to commercial and political values [5]. The analysis of the data is often done using *data mining* techniques, which is the cross-disciplinary field that focuses on describing the properties and finding patterns in the data. However, people usually incur mistakes during analysis, or when trying to establish relationships in large and complex data. *Machine learning* can be successfully applied to these problems, which is a technique that extracts information from data by generalizing from experience. Data mining algorithms build models that are a representation of the previously known data.

Every real-world object (instance) used on machine learning is represented by a set of features. If the instances are given known labels then the type of learning is called *supervised*, in contrast to whenever the instances are unlabeled, then it is called *unsupervised*. Traditional supervised learning is the most popular task in machine learning and associates each instance to their corresponding output. The problem is known as *classification* whenever the output belongs to a set of categories, whereas a real-valued output is known as *regression*.

Although traditional supervised learning is prevailing and widespread, the assumption that each object has only one unique label arises issues with many real-world scenarios. Modern data is characterized by its ever-increasing volume and complexity, where an object might be associated with multiple labels simultaneously. Following that consideration, the *multi-output learning* paradigm emerges. Multi-output learning is a generalization of traditional supervised learning, that ignores the constraint on how many outputs an instance is assigned to.

This paradigm can consider a fixed number of real-valued outputs, therefore it is known in the literature as multi-target [6]–[10], multi-variate [11]–[13], or multi-response [14], [15] regression. On the other hand, whenever the paradigm assigns to each instance a set of discrete labels, it is known as multi-label classification.

Multi-label classification was originally conceived to solve automatic text categorization problems [16]. However, researchers later realized of the presence of multi-label tasks in many other real-world problems which then draw more and more attention to this paradigm. Multi-label objects, which are annotated with multiple labels, are found in many scenarios from automatic annotation of multimedia contents [17]–[20], to bioinformatics [21]–[25], web mining [26]–[28], rule mining [29]–[31], information retrieval [32], [33], tag recommendation [34], [35], sentiment classification [36], music genre classification [37]–[39], etc.

Unfortunately, the processing capabilities of single machines have not kept up with the ever-increasing volume and complexity of modern data. As a result, many organizations and researchers are migrating their computations across multiple machines, i.e., clusters. This distributed environment comes with several issues and challenges [40], namely: *Heterogeneity*, which describes a system consisting of multiple distinct components; *Openness*, the property of each subsystem to be open for interaction with other systems; *Security*, the information should be safe and protected against any corruption; *Failure handling* detects failures and allows the system to recover; *Transparency*, making the system to be perceived as a single machine by the users, thus local and remote resources are accessible in the same way; *Concurrency*, the capability to handle several requests to access a shared resource simultaneously; *Scalability*, a system is scalable if it remains effective as the number of users, data, or resources increase.

Consequently, a wide range of programming models has been designed for distributed environments. At first, Google introduced the *MapReduce* [41] programming model in 2003, which is considered one of the first distributed frameworks for large-scale data processing. The term MapReduce refers to the two individual tasks that are performed. First, the *map*

task takes a set of data and converts it into another set of data, where the elements are broken down into key-value tuples. Second, the *reduce* task takes the output from a map as input and combines those previous results. Programs written in this functional style are automatically parallelized and executed on a large cluster. The run-time system takes care of the details of partitioning the data, scheduling the execution across a set of machines, handling the failures, and managing the required inter-machine communication.

One of the first implementations of the MapReduce model was Apache Hadoop [42]. Hadoop started as a Yahoo project in 2006, becoming a top-level Apache open-source project later on. Despite the Hadoop ecosystem has grown and matured over the years, it presents some important weaknesses. Some of these issues are the impossibility to maintain data in-memory, thus it reads the same data iteratively and materializes intermediate results in local disks in each iteration, requiring lots of disk accesses and unnecessary computations [43]–[45]. Consequently, these issues lead to poor performance on online, interactive, and/or iterative methods, which are crucial for data scientists in machine learning and data mining.

Apache Spark [46] was initially developed by the AMPLab at UC Berkeley in 2012. It is also a top-level Apache project based on the MapReduce programming model to process data in parallel across a cluster. Spark has become the most powerful engine for the big data scenario, overcoming the limitations of Hadoop. Spark is able to cache intermediate results and reused data in-memory, to query it repeatedly using in-memory primitives, thus making it suitable for large iterative jobs. Using in-memory data has been proved to be of the most relevance in machine learning scenarios. In fact, Spark has been shown to outperform Hadoop in many cases (up to 100x in memory) [47].

In this thesis, several approaches have been devised for scalable multi-label learning on large-scale data. Especially, the high-dimensionality of the multi-label data has been addressed by implementing, on the proposed distributed architectures, a classifier sensitive to the quality of the features and a series of selection methods to select the most relevant features.

1.1 Contributions of the Thesis

Most of the current multi-label methods are built using the capabilities of single machines, which considerably limits the computational resources of the system. Multi-label learning is a challenging task characterized by the properties of its data, such as a large number of instances, the high-dimensionality of the features, the number of labels, and the dependencies between labels, among others. These characteristics also present the opportunity to capture and exploit possible dependencies between labels, at an increased computational complexity. Therefore, there is a pressing need to develop scalable algorithms to harness the worthwhile information present in the label dependencies. This thesis aims to define a distributed architecture that would allow scaling traditional and our novelty processing techniques and learning methods efficiently according to the properties of multi-label data.

The core of this thesis identifies multi-label data as an aggregation of the information of all the labels. This phenomenon leads to the characteristic high-dimensionality of the multi-label data. As a result, there is plenty of irrelevant information that could be filtered in order to improve the learning process. The final goal of this thesis is to define the best methodology to learn multi-label data by detecting and filtering this information. However, due to the novelty of multi-label learning on Apache Spark, it was required to build everything from scratch. Therefore, the research carried out has to lead to the following contributions to the field of multi-label classification:

- **Parallel and distributed multi-label architectures**: Evaluating the performance of five approaches to speed up a multi-label ensemble method: the baseline Mulan implementation using a single thread on a single machine, a multi-threading version of Mulan on a single machine, a distributed version of Mulan where multiple instances of Mulan are deployed in each machine, a distributed version of Mulan where each machine has a multi-threading version of Mulan, and a Spark native implementation of the multi-label learning paradigm. The analysis compares the quality of the predictions, as well as the execution time and scalability of the different approaches.

- **Distributed multi-label k nearest neighbors**: Design and evaluation of a distributed multi-label k nearest neighbors method. This method considered three approaches, which represent the main strategies to find nearest neighbors in a distributed environment: an iterative pair-wise distance computation, a tree-based method that index the instances across multiple nodes, and a local sensitive hashing method that builds multiple hash tables to index the data. Each of these methods was compared and evaluated with respect to the reliability of the predictions, execution times for *train* and *test* phases, and a study of scalability with respect to the number of instances, features, and labels.
- **Multi-label mutual information measure**: Analysis of the feature selection strategies for multi-label learning and the mutual information adaptation for multi-label data. The best strategy to adopt mutual information was discussed, in order to clarify the debate into how to consider the information between a feature and a set of labels.
- **Distributed multi-label selection methods for continuous features**: Proposing the implementation of two distributed feature selection methods on continuous features for multi-label data: a method that maximizes the mutual information of the selected feature subset (MIM), and a method that maximizes the relevance of the selected features while minimizing the redundancy among the selected features (mRMR). These methods handled continuous features directly, and their performance was compared to three multi-label feature selection methods which require the values discretization.
- **Distributed multi-label selection methods for discrete features**: Proposing two distributed feature selection methods on discrete features for multi-label data: a method that selects the features with the largest L^2-norm (ENM), and a method that selects the features with the largest geometric mean (GMM). These methods study two opposite approaches to aggregate the MI of multiple labels. Their performance was compared to three multi-label feature selection methods and the two previously proposed methods.

- **Comparison of proposed methods to traditional methods**: Comparison of our proposed methods against three distributed implementations of traditional feature selection methods. Traditional methods require discrete features, therefore we applied the Freedman-Diaconis rule [48] for the continuous features, which to the best of our knowledge is the first time it was applied to a multi-label scenario. The results were evaluated using the distributed multi-label k nearest neighbors method. The analysis studies the accuracy and execution time produced by an increasing feature subset.

This thesis is structured as follows:

- **Chapter 2** provides an in-depth study of multi-label learning and distributed systems.
- **Chapter 3** discusses the different approaches to build a distributed multi-label architecture.
- **Chapter 4** presents the distributed multi-label k nearest neighbors.
- **Chapter 5** introduces a series of distributed feature selection methods for multi-label problems.
- **Chapter 6** provides a synthesis of the contributions and concluding remarks.
- **Chapter 7** discusses future lines of research and work.

CHAPTER 2

BACKGROUND

This chapter presents the theoretical background and establishes the foundations of our thesis based on multi-label classification and distributed computing. First, the multi-label learning paradigm is discussed, introducing the definition of the problem, the evaluation metrics, and the different strategies to solve it. Second, the distributed computing background is presented, and the MapReduce programming model is introduced.

2.1 Multi-label learning

Multi-label learning is considered a case of traditional supervised learning, i.e., learning from a training set of instances correctly identified with their label. However, multi-label classification generalizes this problem, by relaxing the property that assigns each instance with a label and allowing to each instance to belong to multiple labels simultaneously [49]. This paradigm reflects the true nature of modern data, and as a result, it has attracted a growing interest in the last decade from industry and academia [17]–[19], [27], [37]–[39]

Figure 2.1 shows two movies selected from the *IMDb* website[1], which contains more than 4 millions titles. Each of the movies has assigned a score out of ten, the MPAA rating, and some genres, among other information. There is a finite number of rating categories (*G, PG, PG-13, R, NC-17*), and each movie can only be categorized with only one out of those. Therefore, classifying a movie within those rating categories is considered a multi-

[1]https://www.imdb.com

(a) $Movie_1$ = {Adventure, Drama, Sci-Fi} (b) $Movie_2$ = {Drama, Horror, Thriller}

Fig. 2.1.: Example annotation of movie genre annotation

class classification problem. Similarly, predicting the numerical score of a movie would be a regression problem. On the other hand, predicting genre tags represent a different problem. Each movie can be assigned with multiple categories, e.g., the first movie belongs to {*Adventure, Drama, Sci-Fi*} genres, while the second movie belongs to {*Drama, Horror, Thriller*}. The task of the multi-label classification is to predict the different genres that a movie belongs to.

Figure **??** presents two pictures containing a beach in a semantic image annotation scenario. In traditional classification, both of these pictures would belong to the class *Beach*, despite representing complete different scenes. The first picture could be associated with the labels {*Beach, Mountain, Clear Sky*}, while the second picture belongs to a different scenario and is associated with the labels {*Beach, Urban, Cloudy Sky*}. Multi-label data is instinctively associated with a semantic image classification scenario, where the pictures would always belong to multiple categories [17], [20].

(a) $Image_1$ = {Beach, Mountain, Clear Sky}

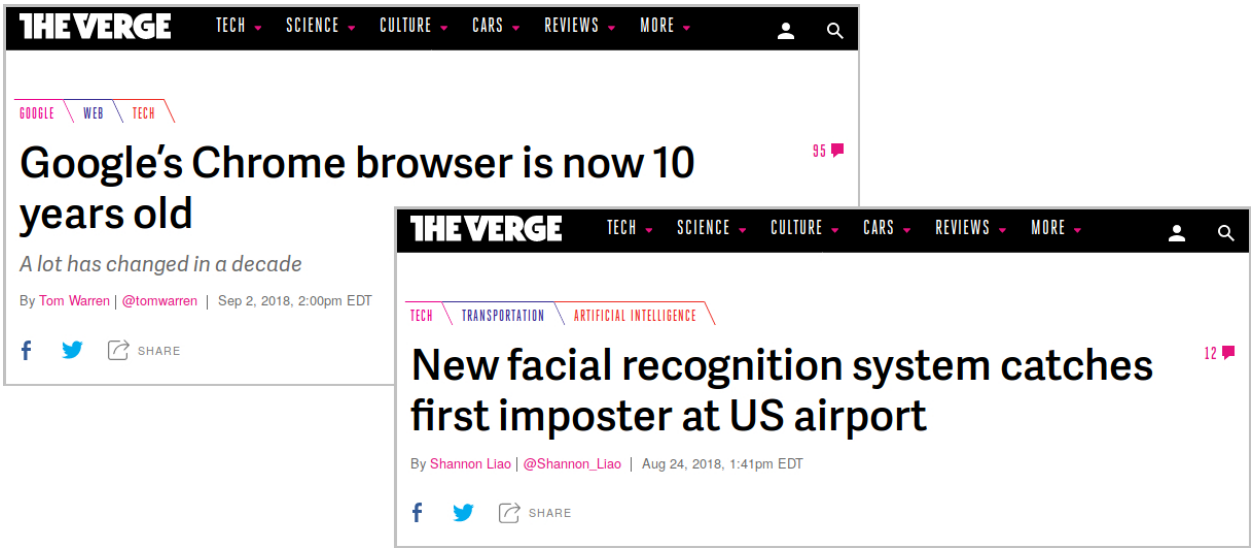
(b) $Image_2$ ={Beach, Urban, Cloudy Sky}

Fig. 2.2.: Example annotation of semantic scene classification

Another popular multi-label application uses text data. Text analytics is a difficult task since it involves structuring the input text and deriving patterns within the structured data. Typical text tasks include entity extraction, sentiment analysis, document summarization, text categorization, text clustering, etc. Many of these task associate multiple outputs to each text, for example, text categorization involves the assignment of one or more predefined categories based on their content. Figure 2.3 presents two news headlines which have been categorized as {*Google, Web, Tech*} and {*Tech, Transportation, Artificial Intelligence*}, respectively. Therefore, the task to learn a model that assigns these categories to each news fall under the multi-label paradigm.

These are just a few examples of the many scenarios where multi-label learning can be applied. In these scenarios, extra information can be derived from the dependencies between labels. However, this is a non-trivial task due to many factors such as the high-dimensionality of the features and labels, as well as the size of the data.

There are some variations to the multi-label learning paradigm, where each instance is represented by a bag of instances [50], or where the output is structured following a hierarchy [24], [51]. In order to avoid confusion, we only consider *single-instance* representation associated with unstructured label sets.



(a) $News_1 = \{\text{Google, Web, Tech}\}$ (b) $News_2 = \{\text{Tech, Transportation, Artificial Intelligence}\}$

Fig. 2.3.: Example annotation of text categorization

2.1.1 Formal definition and notation

Let $\mathcal{X} = \mathbb{R}^d$ (or $\mathbb{Z}^d$) be a d-dimensional input space of numerical or categorical features, and $\mathcal{Y} = \{y_1, y_2, \ldots, y_q\}$ be the label space with q possible class labels. Each multi-label instance (x_i, Y_i), $x_i \in \mathcal{X}$ is a d-dimensional feature vector $(x_{i1}, x_{i2}, \ldots, x_{id})$ and $Y_i \subseteq \mathcal{Y}$ is the set of labels associated with x_i. The set Y_i is also known as the set of relevant labels of x_i, thus the complement $\overline{Y}_i$ is the set of irrelevant labels. Labels associations usually are represented as a q dimensional binary vector $y = (y_i, y_2, \ldots, y_q)$, where its j-th component y_j is 1 if its a relevant label ($y_j \in \mathcal{Y}$) and 0 otherwise. Table 2.1 summarizes the notation, based on [52], which is used to establish the formal definitions from previous works, as to explain into detail our contributions.

Multi-label classification is concerned with learning the function $h : \mathcal{X} \to 2^{\mathcal{Y}}$, which outputs a bi-partition of the set of labels into relevant and irrelevant. For any unseen instance $x \in \mathcal{X}$, the multi-label classifier $h(\cdot)$ predicts $h(x) \subseteq \mathcal{Y}$ as the set of relevant labels for x. On the other hand, label ranking defines a function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, which outputs the ordering of the labels according to their relevance. Thus, $f(x, y)$ can be regarded as the

Table 2.1.: Summary of symbols and notation

Notation	Definition
$\mathcal{X}$	Instance space $\mathbb{R}^d$ (or $\mathbb{Z}^d$)
x	d-dimensional feature vector $x_i = (x_{i1}, \ldots, x_{id})$, $x_i \in \mathcal{X}$
d	Number of features
$\mathcal{Y}$	Label space $\{y_1, y_2, \ldots, y_q\}$
Y	Label set associated to x ($Y \in \mathcal{Y}$)
$\overline{Y}$	Complementary set of Y
Z	Predicted label set ($Y \in \mathcal{Y}$)
q	Number of labels
$\mathcal{D}$	Multi-label training set $\{(x_i, Y_i),\ 1 \leq i \leq m\}$
m	Number of training instances
$\mathcal{S}$	Multi-label test set $\{(x_i, Y_i),\ 1 \leq i \leq p\}$
p	Number of test instances
$h(\cdot)$	Multi-label classifier $h : \mathcal{X} \to 2^{\mathcal{Y}}$, where $h(x)$ returns the relevant labels for x
$f(\cdot, \cdot)$	Real-valued function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, where $f(x, y)$ returns the confidence of y
$t(\cdot)$	threshold function $t : \mathcal{X} \to \mathbb{R}$, where $h(x) = \{y \mid f(x, y) > t(x), y \in \mathcal{Y}\}$
$\delta(\cdot)$	$\delta(y)$ returns the frequency of label y
$\lvert \cdot \rvert$	$\lvert \mathcal{A} \rvert$ operator that returns the cardinality of a set $\mathcal{A}$
$[\![\cdot]\!]$	$[\![\pi]\!]$ operator that returns 1 if the predicate π holds, and 0 otherwise
$\phi(\cdot, \cdot)$	$\phi(Y, y)$ returns 1 if $y \in \mathcal{Y}$, and 0 otherwise
$\mathcal{D}_j$	Binary training set $\{(x_i, \phi(Y_i, y_j))\}$ derived from $\mathcal{D}$ for label y_j
$\psi(\cdot, \cdot, \cdot)$	$\psi(Y, y_j, y_k)$ returns 1 if $y_j \in Y$ and $y_k \notin Y$, and 0 if $y_j \notin Y$ and $y_k \in Y$
$\mathcal{D}_j k$	Binary training set $\{(x_i, \psi(Y_i, y_j, y_k))\}$ derived from $\mathcal{D}$ for labels (y_j, y_k)
$\sigma(\cdot)$	Injective function $\sigma_{\mathcal{Y}} : 2^{\mathcal{Y}} \to \mathbb{N}$ mapping the subsets of $\mathcal{Y}$ to natural numbers
$\mathcal{D}^{\dagger}_{\mathcal{Y}}$	Multi-class training set $\{(x_i, \sigma_{\mathcal{Y}}(Y_i))\}$ derived from $\mathcal{D}$
$\mathcal{B}$	Binary learning algorithm
$\mathcal{M}$	Multi-class classification algorithm

confidence of $y \in \mathcal{Y}$ being the proper label of x. Note that a classifier $h(\cdot)$ can be derived from the function $f(\cdot, \cdot)$ by: $h(x) = \{y \mid f(x, y) > t(x), y \in \mathcal{Y}\}$, where $t : \mathcal{X} \rightarrow \mathbb{R}$ is a threshold function which partitions the label space into relevant and irrelevant labels.

This document focuses primarily on the multi-label classification problem since all the proposed methods and contributions aim to improve the quality of the predictions over the relevance set of labels. The label ranking problem concerns about finding the right ordering which requires specific processing and learning methods.

2.1.1.1 Threshold calibration

Although multi-label classification and label ranking are two tasks whose aim is to solve different problems, in practice most of the multi-label learners build a real-valued function $f(\cdot, \cdot)$ as the inferred model [52]. These models can be applied directly to the label ranking problems; or they can be used into multi-label classification through the use of a threshold function, i.e., the real-valued function $f(x, y)$ on each label should be calibrated with the threshold function $t(x)$ in order to predict the set of relevant labels $h(x)$.

The calibration of the threshold function is usually done by setting a constant value. Assuming that $f(x, y)$ returns values in $\mathbb{R}$, the most straightforward approach is to set the value to zero [17]. However, the default calibration value in most of the implementations is 0.5, considering that function $f(x, y)$ returns the posterior probability of y being a relevant label of x [21]. Furthermore, the threshold value can be set up to minimize some multi-label metric between the training and test instances, such as the label cardinality [53].

Consequently, the threshold value can be determined using more complex approaches. For example, a linear model for the $t(x)$ can be built stacking the real-valued outputs from the function $f(x, y)$ for each label. Then, the weights of the model can be found solving the *linear least square* problem that uses the target output for each instance that partition $\mathcal{Y}$ and minimizes the number of miss-classifications [23], [54].

2.1.1.2 Label correlations

Multi-label classification poses the challenge of learning an output space whose label set space grows exponentially with the number of labels. However, it also presents the possibility of exploiting the correlations (or dependencies) among labels, in order to considerably improve the final predictions [55], [56]. For example, in the semantic scene scenario, the beach images have a high probability of being annotated with other labels such as *Sea, Ship, Sand, Volley*, but probably not with labels like *River*, *Truck*, *Road*, or *Basketball*. Using the *IMDb* website scenario, a movie being classified as *Horror* will possibly be associated with other genres such as *Thriller* or *Drama*, but possibly not with *Family* or *Animated*.

Therefore, additional and valuable information about what the object *is* can be provided by the labels, in contrast to information about the *characteristics* of the object provided by the features. Using this extra information is crucial, but as it has been shown, the more labels the more possible label sets with many levels of interactions can be considered. These interactions are known as correlations, and depending on how they are exploited the learning techniques can be categorized into the following groups [52], [55], [56]:

- *First-order strategies*: It refers to the techniques that address the multi-label learning by considering the labels independently, thus ignoring interactions between different labels. The most common approach decomposes the multi-label problem into multiple binary classification problems [17], [21]. The main advantage of these methods lies in its simplicity. However, inferred models will ignore any correlations between labels.

- *Second-order strategies*: This approach tackles the label correlations in a pairwise manner. Either by considering the ranking between the relevant labels and irrelevant labels [22], [23], [57] or by considering the interactions between any pair of labels [18], [33], [58], [59]. This strategy exploits the direct correlations between labels. However, it ignores deeper dependencies such as those arising between all the labels, or those that influence other correlations, in turn.

- *High-order strategies*: This approach can either consider the influence that all the labels have on each other [60]–[63] or address the correlations among certain subsets of labels instead [53], [64]–[66]. This strategy allows a stronger influence between the labels, thus eventually leading to harnessing more worthwhile information from the data. However, this comes at a high cost since the more correlations are considered, the more challenging and less scalable becomes on high-dimensional label spaces.

The previous categorization groups the strategies by the level of correlations. The correlations, in turn, can be categorized from a probabilistic point of view into *conditional* and *unconditional* (marginal) label dependences [67]–[70]. The conditional dependences capture the correlations between labels given a certain instance [65], [67]. While, marginal dependences capture the correlations between labels, or how likely are they to occur together, independently of any instance [60], [71], [72].

2.1.2 Learning algorithms

This section covers the most representative methods of each category. The methods were selected with respect to the following criteria: broad spectrum that covers a wide range of design strategies, innovative discoveries which have led to a number of follow up methods, direct or indirect relation to our contributions, and highly cited algorithms in the field.

The methods detailed are strictly restricted to the multi-label classification problem, therefore ignoring those targeting the label ranking problem. Additionally, we do not include multi-label hierarchical classification methods since it has been shown that they perform similarly to those with a flat representation of the labels [73]. For example, [74] compared the performance of a naive Bayes method using a flat and a hierarchical structure, to find out that they performed similarly. Furthermore, [73] compared a regular flat classification against two types of hierarchies. The results indicated that the hierarchies offer little to no benefit to the predictions, while a the same time introduced an overhead which causes memory problems in the larger datasets. The lack of considerable prediction improvement

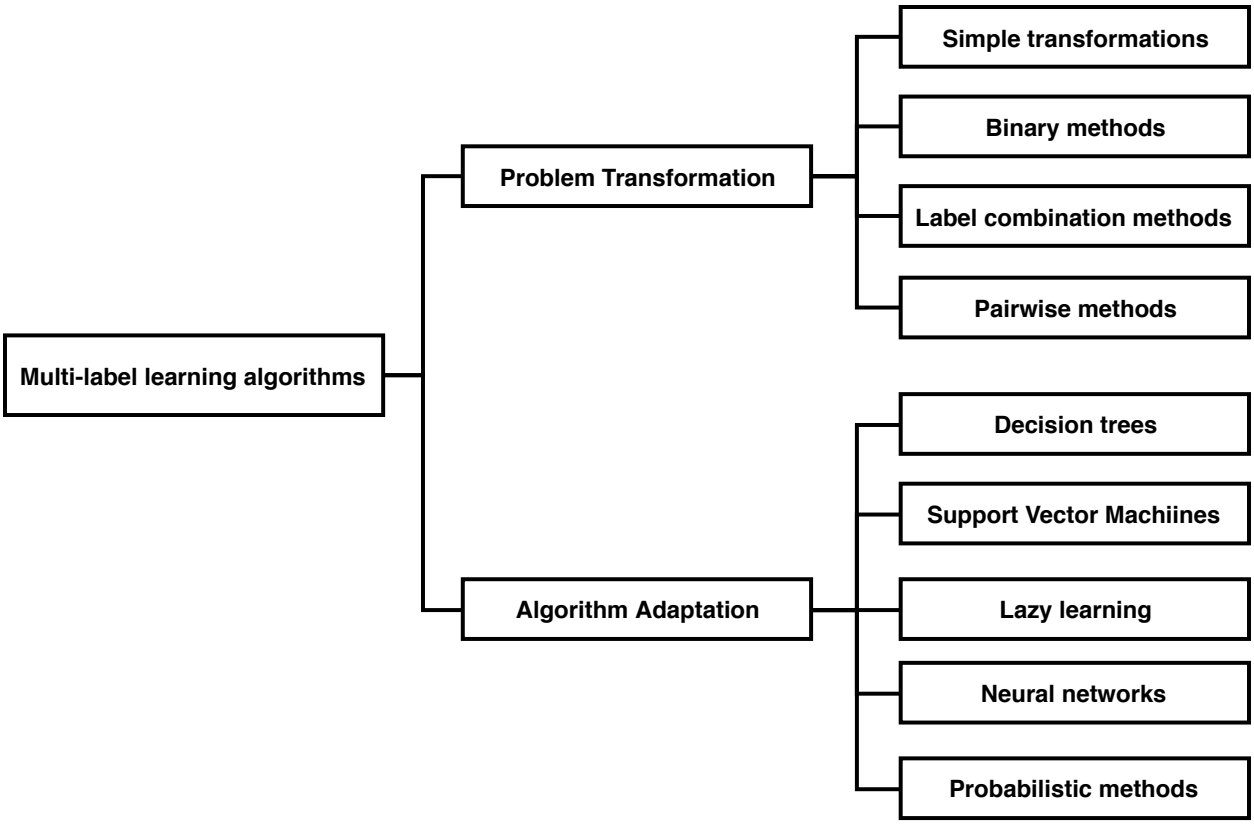


Fig. 2.4.: Multi-label learning algorithm categorization

is due to the fact that the errors propagate down the hierarchy, and the label correlations being narrowed down to concerned label subset.

The most accepted taxonomy for the multi-label learning algorithms categorize the methods in two main approaches: *problem transformation* and *algorithm adaptation* [75]. The former transforms the multi-label problem into one or more single label problems, thus it maps the predictions of single label algorithms onto multi-label. The latter consists of extending existing single label algorithms in order to handle multi-label data directly. Figure 2.4 presents the taxonomy of the multi-label learning algorithms covered in this section. The notation and definitions of the multi-label learning methods are based on on [52], [76], [77].

Table 2.2.: Example multi-label dataset

Instance	Multi-label output				
	Labels	y_1	y_2	y_3	y_4
1	$\{2,3,4\}$	0	1	1	1
2	$\{1\}$	1	0	0	0
3	$\{1,3\}$	1	0	1	0
4	$\{1,4\}$	1	0	0	1
5	$\{1,2,3\}$	1	1	1	0

2.1.2.1 Problem transformation methods

Table 2.2 presents a multi-label training set $\mathcal{D}$ with five instances, this example data will be used to describe the main problem transformation methods. Note that the feature values of each instance are ignored here since the transformations are done on the label space.

Simple transformations

The most straightforward approach is based on transforming the multi-label problem into a multi-class classification problem by just using a single label. These transformations define a strategy to pick a label in each instance, or even none (thus discarding the instance).

The *ignore transformation* simply ignores all the instances that belong to more than one label simultaneously. Therefore, it selects the instances whose relevant label set has a cardinality equal to one.

$$\mathcal{D}^{\dagger}_{\mathcal{Y}} = \{(x_i, y_1) \mid |Y_i| = 1,\ 1 \leq i \leq m\} \tag{2.1}$$

The *copy transformation* takes each multi-label instance and makes as many copies as relevant labels are present. Each new instance retains the information of a single label.

$$\mathcal{D}^{\dagger}_{\mathcal{Y}} = \{(x_i, y_j) \mid y_j \in \mathcal{Y},\ 1 \leq i \leq m,\ 1 \leq j \leq q\} \tag{2.2}$$

This method can be extended by incorporating the weight of each instance, which would be inversely proportional to the number of relevant labels of that instance.

$$\mathcal{D}_{\mathcal{Y}}^{\dagger} = \{(x_i, y_j, \frac{1}{|Y_i|}) \mid y_j \in \mathcal{Y},\ 1 \leq i \leq m,\ 1 \leq j \leq q\} \tag{2.3}$$

The *select transformation* replaces the set of relevant labels of each instance by selecting one of its members. There are three main strategies: *select-max transformation* which uses the member that is most frequent in the data, *select-min transformation* that selects the most infrequent member, and *select-random transformation* that selects a member randomly.

$$\mathcal{D}_{\mathcal{Y}}^{\dagger} = \{(x_i, y_j) \mid \arg\max_{y_j \in Y_i} \delta(y_j),\ 1 \leq i \leq m\} \tag{2.4}$$

$$\mathcal{D}_{\mathcal{Y}}^{\dagger} = \{(x_i, y_j) \mid \arg\min_{y_j \in Y_i} \delta(y_j),\ 1 \leq i \leq m\} \tag{2.5}$$

$$\mathcal{D}_{\mathcal{Y}}^{\dagger} = \{(x_i, y_{rand}) \mid 1 \leq i \leq m\} \tag{2.6}$$

For an unseen instance x, these methods just predict the most relevant label, e.g., the label with the highest confidence in the multi-class problem. Therefore, the predicted set of labels Y will only have one member. Table 2.3 shows the results of applying the different transformations according to the example data in Table 2.2.

These methods transform multi-label problems using simple approaches to single-label problems. However, none of these approaches is likely to maintain the underlying data distribution and therefore will probably make less accurate predictions. Although most of these transformations usually are never present in literature comparisons, due to their poor performance, some of these techniques have been used for other purposes such as feature selection [78].

Table 2.3.: Simple transformations

Instance	Label
2	4

(a) Ignore transf.

Instance	Label
1	2
1	3
1	4
2	1
3	1
3	3
4	1
4	4
5	1
5	2
5	3

(b) Copy transf.

Instance	Label	Weights
1	2	0.33
1	3	0.33
1	4	0.33
2	1	1.0
3	1	0.5
3	3	0.5
4	1	0.5
4	4	0.5
5	1	0.33
5	2	0.33
5	3	0.33

(c) Copy-weight transf.

Instance	Label
1	3
2	1
3	1
4	1
5	1

(d) Select-max transf.

Instance	Label
1	2
2	1
3	3
4	4
5	2

(e) Select-min transf.

Instance	Label
1	4
2	1
3	3
4	1
5	2

(f) Select-random transf.

Binary methods

The **Binary Relevance (BR)** method decomposes the multi-label data into $q = \mid \mathcal{Y} \mid$ binary independent single-label datasets, where each one corresponds to a label in the label space [17]. Each of these datasets keeps the original number of instances and categorizes each of them depending on whether they belonged to the corresponding label or not.

BR constructs a binary classification dataset $\mathcal{D}_j$ for the j-th label according to the relevance of each training example to y_i, thus:

$$\begin{aligned} &\mathcal{D}_j = \{(x_i, \phi(Y_i, y_j)) \mid 1 \leq i \leq m\} \\ &\text{where } \phi(Y_i, y_i) = \begin{cases} 1, & \text{if } Y_i \in y_j \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2.7}$$

Table 2.4.: Binary Relevance data transformation

Instance	Label
1	0
2	1
3	1
4	1
5	1

(a) $\mathcal{D}_1 : y_1$

Instance	Label
1	1
2	0
3	0
4	0
5	1

(b) $\mathcal{D}_2 : y_2$

Instance	Label
1	1
2	0
3	1
4	0
5	1

(c) $\mathcal{D}_3 : y_3$

Instance	Label
1	1
2	0
3	0
4	1
5	0

(d) $\mathcal{D}_4 : y_4$

Once the label space has been split into individual binary labels, a binary learning algorithm $\mathcal{B}$ can be applied to induce a classifier $\mathcal{B}(\mathcal{D}_{|}) \rightarrow g_j$, i.e., $g_j : \mathcal{X} \rightarrow \mathbb{R}$. As a result, any instance (x_i, Y_i) will be involved in the training processing of q binary classifiers. Whether they will be labeled positively if $y_i \in Y_i$ and zero otherwise. Table 2.4 shows the transformation results of applying BR to the example multi-label data in Table 2.2.

Once all the binary classifiers have been built, BR can predict the associated label set of an unlabeled instance by combining the relevant labels of the individual predictions.

$$\mathcal{Y} = \{y_j \mid g_j(x), 1 \leq j \leq q\} \tag{2.8}$$

BR is a straightforward approach with relatively low complexity. It scales linearly with the number of labels, and since they are treated independently they could be added and removed in an evolving and dynamic scenario [53]. On the other hand, BR presents a series of disadvantages [49], [76] which are directly related to the fact that is a *first-order* approach, handling each label independently which can fail to predict label combinations and rankings besides the ignoring the label correlations. Additionally, it introduces the problem of instance imbalance since after the transformation it is expected that the negative instances outnumber the positive ones. This issue increases with the number of labels, which also increments the number of binary classifiers.

The **Classifier Chains (CC)** [53] method addresses the incapability of BR to handle label correlations. This algorithm transforms the multi-label learning problem into a *chain* of binary classification problems, i.e., BR transformations, by incorporating in the features of the training instances of each binary classifier the label predictions of the preceding ones.

Let $\rho : \{1, \ldots, q\} \rightarrow \{1, \ldots, q\}$ be a permutation function which is used to define a order of all the labels, i.e., $y_{\rho(1)} \succ y_{\rho(2)} \succ \cdots \succ y_{\rho(q)}$. CC would construct a binary classification dataset $\mathcal{D}_j$ for the j-th label by appending each instance with its relevance to those labels preceding $y_{\rho(j)}$:

$$\begin{aligned} \mathcal{D}_{\rho(j)} &= \{\big([x_i, pre^i_{\rho(j)}], \phi(Y_i, y_j)\big) \mid 1 \leq i \leq m\} \\ \text{where } pre^i_{\rho(j)} &= (\phi(Y_i, y_{\rho(1)}), \ldots, \phi(Y_i, y_{\rho(j-1)})) \end{aligned} \tag{2.9}$$

Here, $pre^i_{\rho(j)}$ represents the binary predictions $\{0, 1\}$ of the preceding labels of $y_{\rho j}$ on x_i. After the predictions have been concatenated to x_i a binary learning algorithm $\mathcal{B}$ is applied to induce a classifier $\mathcal{B}(\mathcal{D}_{\rho(|)}) \rightarrow g_{\rho(j)}$, i.e., $g_{\rho(j)} : \mathcal{X} \times \{0, 1\}^{j-1} \rightarrow \mathbb{R}$. Therefore, $b_{\rho(j)}$ predicts whether $y_{\rho(j)}$ is a relevant label or not.

For an unseen instance x, CC can predict its associated label set Y by querying iteratively the chain of classifiers as follows:

$$\begin{aligned} Y &= \{y_{\rho(j)} \mid \lambda^x_{\rho(j)},\ 1 \leq j \leq q\} \\ \text{where } \lambda^x_{\rho(j)} &= g_{\rho(j)}([x, \lambda^x_{\rho(1)}, \ldots, \lambda^x_{\rho(j-1)}]),\ (2 \leq j \leq q) \end{aligned} \tag{2.10}$$

Predictions of CC highly depend on the ordering specified by the ordering function ρ. This issue has been addressed by using an **Ensemble of Classifier Chains (ECC)** [53] which builds n random chains over the label space, i.e., $\{\rho^1, \ldots, \rho^n\}$. Two different approaches have been proposed over this method, one where the chain ρ^r uses a sampling of $\mathcal{D}$ without replacement ($\mid \mathcal{D}^r \mid = 0.67 \cdot \mid \mathcal{D} \mid$) or with replacement ($\mid \mathcal{D}^r \mid = \mid \mathcal{D} \mid$).

CC is considered an extension of BR, as it keeps the advantages of BR while incorporating the correlations between labels. Therefore, it is considered a *high-order* approach, despite incorporating the correlations in a random manner. However, the iterative nature of this algorithm loses the possibility of parallelization of BR due to the chaining property.

Label combination methods

The **Label Powerset (LP)** method [37] transforms the multi-label data by considering each unique combination of labels as a class. As a result, it transforms the multi-label classification problem into a multi-class problem.

LP uses a function $\sigma_{\mathcal{Y}} : 2^{\mathcal{Y}} \rightarrow \mathbb{N}$ to map each power set of $\mathcal{Y}$ present in the training set into the natural numbers. The algorithm also defines an inverse function $\sigma_{\mathcal{Y}}^{-1} : \mathbb{N} \rightarrow 2^{\mathcal{Y}}$ to map back from the natural numbers into the label space, therefore:

$$\mathcal{D}_{\mathcal{Y}}^{\dagger} = \{(x_i, \sigma(Y_i)) \mid 1 \leq i \leq m\} \tag{2.11}$$

where the set of classes covered, whose cardinality is $min(m, 2^q)$, would be:

$$\Gamma\big(\mathcal{D}_{\mathcal{Y}}^{\dagger}\big) = \{\sigma_{\mathcal{Y}}(Y_i) \mid 1 \leq i \leq m\} \tag{2.12}$$

The new learning problem can be solved by applying a multi-class learning algorithm $\mathcal{M}$ to induce a multi-class classifier $\mathcal{M}(\mathcal{D}_{\mathcal{Y}}^{\dagger}) \rightarrow g_{\mathcal{Y}}^{\dagger}$, i.e., $g_{\mathcal{Y}}^{\dagger} : \mathcal{X} \rightarrow \Gamma(\mathcal{D}_{\mathcal{Y}}^{\dagger})$. Therefore, any instance (x_i, Y_i) will be assigned a new class $\sigma_{\mathcal{Y}}(Y_i)$, and then will be used in the training process of the multi-class classifier. Table 2.5 shows the transformation results of applying LP to the example multi-label data in Table 2.2.

After the multi-class classifier has been trained, the relevant set of labels associated with an unseen instance can be predicted by querying the multi-class classifier and mapping the result back to $\mathcal{Y}$ with the inverse function:

$$\mathcal{Y} = \sigma_{\mathcal{Y}}^{-1}\big(\gamma_{\mathcal{Y}}(x)\big) \tag{2.13}$$

LP succeeds over BR in considering the correlations between labels, since it learns complete subsets of $\mathcal{Y}$. Additionally, it only needs to learn one single-label classifier, in contrast to BR whose number of classifiers is equal to the number of labels. However, LP has a series of major limitations directly related to the dimensionality of the label space. The larger the number of labels, the less frequent most of the combinations will be in the

Table 2.5.: Label Powerset data transformation

Instance	Label
1	4
2	1
3	2
4	3
5	5

training set, thus leading to an extreme class imbalance. Additionally, this low frequency on many label combinations can lead to some of them to do not appear into the training set at all, as a result, the multi-class classifier would never learn those combinations, therefore LP would be limited to predict only the label sets present in the training set. Additionally, the number of classes that the function maps to increases exponentially with the number of labels, since it is bounded by $min(m, 2^{|\mathcal{Y}|})$, thus leading to an exponential complexity in the worst-case scenario. Those issues are the reason for the quick deterioration of the LP performance for larger label sets [60].

The **Pruned Problem Transformation (PPT)** method [79] addresses these issues by pruning away the instances with a label set whose frequency is below than a user-defined threshold τ before applying the LP method.

PPT uses a function $freq(Y_i)$ that computes the frequency of a label set Y_i in the training set. Using this function, an instance (x_i, Y_i) would be removed if the frequency of its label set $freq(Y_i)$ is smaller than the defined threshold τ, as follows:

$$\mathcal{D}_{\mathcal{Y}}^{\dagger} = \{(x_i, Y_i) \mid \delta(Y_1) \geq \tau,\ 1 \leq i \leq m\} \tag{2.14}$$

Once the multi-label training set has been transformed, PPT applies the LP algorithm (although it could apply any other). Therefore, the resulting training set $\mathcal{D}_{\mathcal{Y}}^{\dagger}$ would be used to learn a multi-label classifier. For any unseen instance x, the learned multi-label classifier should be queried to predict the set of relevant labels.

However, since some of the patterns are discarded from the original data, this is an irreversible transformation in which there may be a loss of information. As a result, the performance may be limited by the threshold parameter which is generally unknown in practical situations.

Nonetheless, pruned instances can be reintroduced into the transformed set $\mathcal{D}^{\dagger}_{\mathcal{Y}}$ with a smaller and more frequent label set. The method uses a strategy to prevent an obvious increase of the size of the transformed set or a decrease of the average number of labels per instance which, in turn, can cause too few labels to be predicted for an unseen instance. Therefore, it proposes to generate every subset $\{s_i \in Y_i \mid freq(s_i) \geq \tau\}$ and rank these subsets by the number of labels it contains $\mid s_i \mid$ and select the top-ranked subsets without label repetitions. The loss of information is the main drawback of the PS method, therefore some methods have tried to minimize the impact of this problem. The **Ensemble of Pruned Sets (EPS)** algorithm [79] addresses the issue of information loss by applying a set of PS methods to a sampled training set using bootstrap. This method has been shown to outperform PS and LP methods while remaining competitive in terms of efficiency.

The **RAndom k-labELsets (RAkEL)** method [65] builds an ensemble of LP classifiers, although as an ensemble method it could use any other multi-label classifier. Each of the classifiers is trained with a random subset of $\mathcal{Y}$ of k labels. This strategy avoids the exponential computational cost and the extremely instance imbalance associated with high dimensional label space. Therefore, RAkEL keeps the simplicity and advantages of LP, while at the same time overcoming its major drawback.

RAkEL considers $\mathcal{Y}^k$ the collection of all the random label subsets, where the l-th subset is denoted as $\mathcal{Y}^k(l)$, i.e., $\mathcal{Y}^k(l) \subseteq \mathcal{Y}$, $\mid \mathcal{Y}^k(l) \mid = k$, $1 \leq l \leq \binom{q}{k}$. Therefore, a multi-class classifier can be built using the labels of the training instances that intersect with the selected random subset:

$$\mathcal{D}^{\dagger}_{\mathcal{Y}^k(l)} = \{\big(x_i, \sigma(Y_i \cap \mathcal{Y}^k(l))\big) \mid 1 \leq i \leq m\} \tag{2.15}$$

where the set of classes covered in $\mathcal{D}^{\dagger}_{\mathcal{Y}^k(l)}$ are:

$$\Gamma\big(\mathcal{D}^{\dagger}_{\mathcal{Y}^k(l)}\big) = \{\sigma_{\mathcal{Y}^k(l)}(Y_i \cap \mathcal{Y}^k(l)) \mid 1 \leq i \leq m\} \tag{2.16}$$

After the dataset is transformed, a multi-class learning algorithm $\mathcal{M}$ can be used to induce a multi-class classifier $\mathcal{M}(\mathcal{D}_{\mathcal{Y}^k(l)}) \rightarrow g^{\dagger}_{\mathcal{Y}^k(l)}$, i.e., $g^{\dagger}_{\mathcal{Y}^k(l)} : \mathcal{X} \rightarrow \Gamma(\mathcal{D}^{\dagger}_{\mathcal{Y}^k(l)})$.

For an unseen instance x, RAkEL predicts the relevant set of labels by combining the predictions of the component classifiers. RAkEL regards a certain label y_j as relevant as long as at least half the classifiers, whose random subsets contain it, vote so:

$$\begin{aligned} Y &= \{y_j \mid \frac{\mu(x, y_j)}{\phi(x, y_j)} > 0.5,\ 1 \leq j \leq q\} \\ \phi(x, y_j) &= \sum_{r=1}^{n} [\![y_j \in \mathcal{Y}^k(l_r)]\!],\ (1 \leq j \leq q) \\ \mu(x, y_j) &= \sum_{r=1}^{n} [\![y_j \in \sigma^{-1}_{\mathcal{Y}^k(l_r)}\big(k_{\mathcal{Y}^k(l_r)}(x)\big)]\!],\ (1 \leq j \leq q) \end{aligned} \tag{2.17}$$

Here, $\phi(x, y_j)$ counts the number of classifiers that contain y_j in their label subset space, i.e., the maximum number of possible votes. While $\mu(x, y_j)$ counts the number of classifiers that actually predict y_j as a relevant label, i.e., the actual number of votes. The recommended settings for RAkEL is to use $k = 3$ and $n = 2q$ [65], therefore each label is present on nk/q random subsets on average.

RAkEL is considered a *high-order* approach where the degree of correlations between labels is controlled by the size of the random subsets. This algorithm succeeds to bound the quality of the predictions of its base classifier while remaining competitive in terms of computational complexity. However, each label is present on nk/q random subsets which is probably more than those that are directly correlated with it. As a result, RAkEL successfully takes the label correlations into account at the cost of blindly guessing which subsets could be correlated.

Pairwise methods

The **Ranking Pairwise Comparison (RPC)** [57] method transforms the multi-label classification problem with q possible labels, into a $q(q-1)/2$ (or $\binom{q}{2}$) binary classification problems. Each of these new classifiers will learn from the generated data resulting from comparing each label pair (y_j, y_k) $(1 \leq j < k \leq q)$. Although this method generally aims to obtain the ranking of the labels, the definition here will focus on how it is used to learn the relevant set of labels in the multi-label classification problem.

RPC constructs a binary training set for each label pair (y_j, y_k), $1 \leq j < k \leq q$. The training set will consider the instances that belong to at least one of the labels in the pair, but not both. Therefore, the instances are selected based on the relative relevance of y_j with respect to y_k:

$$\begin{gathered}\mathcal{D}_{j,k} = \{(x_i, \psi(Y_i, y_j, y_k)) \mid \phi(Y_i, y_j) = \phi(Y_i, y_k),\ 1 \leq i \leq m\} \\ \text{where } \psi(Y_i, y_j, y_k) = \begin{cases} 1, & \text{if } y_j \in Y_i \text{ and } y_k \notin Y_i \\ 0, & \text{if } y_j \notin Y_i \text{ and } y_k \in Y_i \end{cases}\end{gathered} \tag{2.18}$$

After all the binary training sets have been built, a binary learning algorithm $\mathcal{B}$ can be applied to each of them to induce a classifier $\mathcal{B}(\mathcal{D}_{j,k}) \rightarrow g_{jk}$, i.e., $g_{jk} : \mathcal{X} \rightarrow \mathbb{R}$. Therefore, any instance (x_i, Y_i) will be involved in the training process of a maximum of $\binom{q}{2}$ binary classifiers. Table 2.6 shows the transformation results of applying RPC to the example multi-label data in Table 2.2.

For any unseen instance x, RPC will query the $\binom{q}{2}$ trained binary classifiers to combine and average their predictions on each label:

$$\begin{gathered}Y = \{y_j \mid \frac{\zeta(y_j)}{q-1},\ 1 \leq j \leq q\} \\ \text{where } \zeta(y_j) = \sum_{k=1}^{j-1} g_{kj}(x) \leq 0.5 + \sum_{k=j+1}^{q} g_{jk}(x) > 0.5\end{gathered} \tag{2.19}$$

Table 2.6.: Ranking pairwise comparison data transformation

Instance	Label
1	0
2	1
3	1
4	1

(a) $\mathcal{D}_{12} : y_1$ vs. y_2

Instance	Label
1	0
2	1
4	1

(b) $\mathcal{D}_{13} : y_1$ vs. y_3

Instance	Label
1	0
2	1
3	1
5	1

(c) $\mathcal{D}_{14} : y_1$ vs. y_4

Instance	Label
3	0

(d) $\mathcal{D}_{23} : y_2$ vs. y_3

Instance	Label
4	0
5	1

(e) $\mathcal{D}_{24} : y_2$ vs. y_4

Instance	Label
3	1
4	0
5	1

(f) $\mathcal{D}_{34} : y_3$ vs. y_4

Here $\zeta(y_j)$ sums the predictions for y_j of all the binary classifiers where the label is involved, either as a positive value or as zero.

RPC is categorized as a *second-order* approach since it considers the correlations of pairs of labels. This method is usually involved in the label ranking problem since it is straightforward to extend it to produce an ordering of labels by counting the votes on each label [57]. A modification of this method, known as **Calibrated Label Ranking (CLR)**, has been proposed for label ranking problems where a virtual label is included to separate the set of relevant and irrelevant labels.

2.1.2.2 Algorithm adaptation methods

This category of algorithms modifies the traditional classifiers to handle multi-label problems directly. Most of these methods have been proposed with a specific problem in mind, for example, algorithm adaptations for decision trees were presented to tackle the biological datasets [21], [80], while probabilistic models were used on text categorization [81].

Furthermore, algorithm adaptation methods inevitably involve inner transformations, these transformations are carried out internally by the classifier in order to adapt to multi-label data. Despite most of these algorithms do not apply explicitly problem transformations, they can still be categorized as *Binary*, *Label combination* or *Pair-wise* transformations.

There is a wide range of algorithms that have been proposed in the literature, this section will cover the most relevant ones grouped by their nature. Although many algorithms report very successful results in comparison to problem transformation approaches, their formula derivations are based on the concepts used by the problem transformations.

Decision trees

The **Multi-label Decision Tree (ML-DT)** method [21] adapts the popular decision tree algorithm to handle multi-label data directly. This implementation allows multiple labels to be in the leaves nodes and modifies the information gain criterion to describe how much information was needed to represent all the labels that belong to it.

ML-DT modifies the information gain criterion using a multi-label entropy definition to build a decision tree recursively. Given a multi-label training set $\mathcal{D}$ with m instances, the information gain achieved by splitting the dataset along the attribute is:

$$IG(\mathcal{D}, A) = H_{ML-DT}(\mathcal{D}) - \sum_{v \in A} \frac{\mid \mathcal{D}_v \mid}{\mid \mathcal{D} \mid} H_{ML-DT}(\mathcal{D}_v) \tag{2.20}$$

Here, $\mathcal{D}_v$ refers to the subset of $\mathcal{D}$ with a value v for attribute A. For real-value features, the information gain could be adapted by subtracting to the total entropy of $\mathcal{D}$ the sum of the entropies of the subsets with larger and smaller values than v.

ML-DT follows the same strategy to build the tree as the $C4.5$ tree [82]. First, it finds value v that splits the data while maximizing the information gain, and then it creates two child nodes with the corresponding subsets. The process is invoked recursively until a certain stop criterion is met, e.g., the size of a child node is below a certain threshold.

In order to compute the information gain over a specific set of multi-label instances, a multi-label entropy formulation needs to be defined. The proposed multi-label entropy is the sum of the information needed to describe the membership or non-membership of each label. Therefore, using $p(y_i)$ as the probability of membership of a given label in $\mathcal{D}$, the multi-label entropy can be redefined as follows:

$$H_{ML-DT}(\mathcal{D}) = -\sum_{j=1}^{q} \Big(\big(p(y_j)\, log\, p(y_j)\big) + \big((1-p(y_j))\, log\, (1-p(y_j))\big) \Big) \\ \text{where } p(y_j) = \frac{\sum_i^m = 1[\![y_j \in Y_i]\!]}{m} \tag{2.21}$$

For an unseen instance x, ML-DT is queried by traversing the tree from the root node to the leaf nodes. The instance will be sent to a specific child node according to a defined rule in each node based on the splitting value v in A. Once the instance reaches a leaf node, the predicted label set corresponds to:

$$Y = \{y_j \mid p(y_j) > 0,5,\ 1 \leq j \leq q\} \tag{2.22}$$

Here, $p(y_j)$ refers to the probability of the label y_j within the training instances that fell into the same leaf node, therefore the unseen instance will assume the label distribution of those that took the same path of the tree.

This method is a simple extension of the decision tree algorithm for the multi-label problem. ML-DT is considered a *first-order* approach since it assumes the label independence in the multi-label entropy calculation. The main advantage of this method is the efficient computational complexity since it handles multi-label data directly.

A number of learning algorithms have been proposed based on the discoveries of this method. **Clustering Trees (PCT)** [80] considers the construction of the tree by choosing the splitting value which minimizes the variance. Additionally, this method was extended to an ensemble in **Random Forest of Predictive Clustering Trees (RF-PCT)** [83].

Support Vector Machines

Ranking Support Vector Machine (Rank-SVM) [22] adapts the maximum margin concept to multi-label problems directly. Given a multi-label training set with q labels, this method creates q linear classifiers $\mathcal{W} = \{(w_j, b_j) \mid 1 \leq j \leq q\}$, where w_j $\mathbb{R}^d$ stands for the weight vector, and $b_j \in \mathbb{R}$ stands for the bias, for the j-th label y_j. The multi-label definition of the learning margin over an instance (x_i, Y_i) considers the ranking performance over the relevant and irrelevant labels. Thus, the decision boundary for each pair of relevant and irrelevant labels corresponds to the hyperplane $\langle w_j - w_k, x\rangle + b_j - b_k = 0$.

$$\min_{(x_i,Y_i)\in\mathcal{D}} \min_{(y_j,y_k)\in Y_i\times\overline{Y_i}} \frac{\langle w_j - w_k, x_i\rangle + b_j - b_k}{|| w_j - w_k ||} \tag{2.23}$$

Here, the decision boundary is represented by the L_2 distance to each instance. Assuming that the training set is well ranked, the parameter w_j can be normalized such that:

$$\langle w_j - w_k, x\rangle + b_j - b_k \geq 1 \tag{2.24}$$

Therefore, the formulation that maximizes the margin over all the training set S and all the decision boundaries $\mathcal{W}$ can be expressed as:

$$\begin{aligned} &\max_{\mathcal{W}} \min_{(x_i,Y_i)\in\mathcal{D}} \min_{(y_j,y_k)\in Y_i\times\overline{Y_i}} \frac{1}{|| w_j - w_k ||^2} \\ &\text{subject to:} \quad \langle w_j - w_k, x\rangle + b_j - b_k \geq 1,\ (1 \leq i \leq m,\ (y_j, y_k) \in Y_i \times \overline{Y_i}) \end{aligned} \tag{2.25}$$

Assuming that the problem is not ill-conditioned, i.e., for each pair of labels (y_j, y_k) $(j \neq k)$, there exists $(x, Y) \in \mathcal{D}$ such that $(y_j, y_k) \in Y_i \times \overline{Y_i}$. The objective function can be reformulated as:

$$\begin{aligned} &\min_{\mathcal{W}} \max_{1\leq j<k\leq q} || w_j - w_k ||^2 \\ &\text{subject to:} \quad \langle w_j - w_k, x\rangle + b_j - b_k \geq 1,\ (1 \leq i \leq m,\ (y_j, y_k) \in Y_i \times \overline{Y_i}) \end{aligned} \tag{2.26}$$

The maximum operator can be approximated by the sum operator:

$$\min_{\mathcal{W}} \sum_{j=1}^{q} || \ w_j \ ||^2$$
$$\text{subject to:} \quad \langle w_j - w_k, x \rangle + b_j - b_k \geq 1, \ (1 \leq i \leq m, \ (y_j, y_k) \in Y_i \times \overline{Y_i}) \tag{2.27}$$

In pursuance of extrapolating the problem to the cases where the training set cannot be ranked exactly, the ranking loss function can be expressed following the previous constraints. Therefore, if $\langle w_j - w_k, x \rangle + b_j - b_k \geq 1 - \xi_{ijk}, \ (\xi_{ijk} \geq 0, 1 \leq i \leq m, \ (y_j, y_k) \in Y_i \times \overline{Y_i})$, the ranking loss on the training set can be expressed as:

$$\frac{1}{m} \sum_{i=1}^{m} \frac{1}{\mid Y_i \mid\mid \overline{Y_i} \mid} \sum_{(y_j, y_k) \in Y_i \times \overline{Y_i}} \theta(-1 + \xi_{ijk}) \tag{2.28}$$

Here, θ is the Heaviside function. The final objective function consists of two parts, which are balanced by the parameter C. The first part of the function corresponds to the margin width, whereas the second part corresponds to the ranking loss.

$$\min_{\mathcal{W}} \sum_{j=1}^{q} || \ w_j \ ||^2 + C \sum_{i=1}^{m} \frac{1}{\mid Y_i \mid\mid \overline{Y_i} \mid} \sum_{(y_j, y_k) \in Y_i \times \overline{Y_i}} \xi_{ijk}$$
$$\text{subject to:} \quad \langle w_j - w_k, x \rangle + b_j - b_k \geq 1 - \xi_{ijk}, \ (\xi_{ijk} \geq 0, 1 \leq i \leq m, \ (y_j, y_k) \in Y_i \times \overline{Y_i}) \tag{2.29}$$

This algorithm allows incorporating kernels to solve non-linear separable problems [84], which can be achieved by solving the dual form of the objective function.

For an unseen instance x, Rank-SVM needs to derive the set of relevant labels from the real-value ranks $(f(x, y_1), \ldots, f(x, y_q))$. It uses a stacking procedure which assumes a linear model for $t(\cdot)$, i.e., $t(x) = \langle w^*, f^*(x) \rangle + b^*$, where $f^*(x) = (f(x, y_1), \ldots, f(x, y_q)) \in \mathbb{R}^q$ is a q-dimensional vector stacking the real-valued outputs of each label. To find the optimal w^* and b^*, it solves the linear least squares problem on the training set $\mathcal{D}$:

$$\min_{w^*, b^*} \sum_{i=1}^{m} \big(\langle w^*, f^*(x_i) \rangle + b^* - s(x_i) \big)^2$$
$$s(x_i) = \arg \min_{t \in \mathbb{R}} \big(\mid \{ y_j \mid y_j \in Y_i, f(x_i, y_j) \leq t \} \mid + \mid \{ y_k \mid y_k \in \overline{Y}_i, f(x_i, y_k) \geq t \} \mid \big) \tag{2.30}$$

Here, $s(x_i)$ represents the outputs of the stacking model which partitions the labels into relevant and irrelevant sets with minimum miss-classifications.

The prediction for each label y_j is computed according to the trained parameters w^* and $f^*(x)$. Therefore, the predicted set of relevant labels for an unseen instance x would be:

$$Y = \{y_j \mid \langle w_j, x\rangle + b_j > \langle w^*, f^*(x)\rangle + b*,\ 1 \leq j \leq q\} \tag{2.31}$$

This method is considered a *second-order* approach which adapts the margin strategy from traditional SVM to multi-label problems. The algorithm defines the margin over hyperplanes that bipartition the relevant and irrelevant label pairs. Although Rank-SVM presents promising results, it suffers from the same problems than the pair-wise methods, i.e., it posses a quadratic complexity with respect to the number of labels.

A number of extensions have been proposed based on the foundations of this method. For example, in [85] they propose a formulation that involves features extracted jointly from inputs and outputs. Additionally, in [86] they present a cutting plane algorithm that solves the optimization problem in polynomial time.

Instance-based algorithms

Multi-label *k* Nearest Neighbors (ML-KNN) [87] adapts the *k-nearest neighbors* techniques to the multi-label problem, by using the *Maximum A Posteriori* (MAP) principle to consider the labels of the neighboring instances in the prediction process.

For an unseen instance x, let $\mathcal{N}(x)$ be the set of k nearest neighbors within the training instances in $\mathcal{D}$. Unless directly specified, the nearest neighbors algorithm always assumes the Euclidean distance as the similarity metric employed. Therefore, for the j-th label, the algorithm defines the *membership counting* as:

$$C_j = \sum_{(x^*, Y^*) \in \mathcal{N}(x)} [\![y_i \in \mathcal{Y}^*]\!] \tag{2.32}$$

Let H_j be the event that x has label y_j, $P(H_j|C_j)$ represents the probability that H_j holds under the condition that x has exactly C_j neighbors with label y_j. Consequently, $P(\neg H_j|C_j)$ represents the probability that H_j does not hold under the same condition. Following the MAP principle, ML-KNN can assign the value of Y by determining for each y_j whether $P(H_j|C_j)$ is greater than $P(\neg H_j|C_j)$ or not:

$$Y = \{y_j | \frac{P(H_j|C_j)}{P(\neg H_j|C_j)} > 1,\ 1 \leq j \leq q\} \tag{2.33}$$

Using the Bayesian rule, the prediction formulation can be reformulated as:

$$Y = \{y_j \mid \frac{P(H_j) \times P(C_j|H_j)}{P(\neg H_j) \times P(C_j|\neg H_j)} > 1,\ 1 \leq j \leq q\} \tag{2.34}$$

where $P(H_j)$ represents the *prior probability* that H_j holds, thus $P(\neg H_j)$ represents the probability that H_j does not hold. Next, $P(C_j|H_j)$ represents the *posterior probability* that x has exactly C_j neighbors with label y_j and that H_j holds. Hence, $P(C_j|\neg H_j)$ represents the probability of the same event but when H_j does not hold.

All the information needed to predict the label vector of an instance is the prior probabilities $P(H_j)$ and $P(\neg H_j)$, and the posterior probabilities $P(C_j|H_j)$ and $(P(C_j|\neg H_j)$, which can be estimated from the training data using frequency counting.

Firstly, the prior probabilities are calculated by counting the number of training instances associated with each label:

$$\begin{gathered} P(H_j) = \frac{s + \sum_{i=1}^{m} [\![y_j \in Y_i]\!]}{s \times 2 + m},\ 1 \leq j \leq q \\ P(\neg H_j) = 1 - P(H_j) \end{gathered} \tag{2.35}$$

Here, s is a smoothing parameter that is used to control the strength of the uniform prior (by default s is set to 1 which yields the Laplace smoothing).

Secondly, the posterior probabilities use the nearest neighbors of the training instances. For the j-th label, it computes two frequency arrays each containing $k+1$ elements:

$$\begin{aligned}
\kappa_j[r] &= \sum_{i=1}^{m} [[y_j \in Y_i]] \times [[\delta_j(x_i) = r]],\ (0 \leq r \leq k) \\
\tilde{\kappa}_j[r] &= \sum_{i=1}^{m} [[y_j \notin Y_i]] \times [[\delta_j(x_i) = r]],\ (0 \leq r \leq k) \\
\delta_j(x_i) &= \sum_{(x*,Y*)\in N(x_i)} [[y_j \in Y*]]
\end{aligned} \tag{2.36}$$

where $\delta_j(x_i)$ finds the number of neighbors for x_i that have label y_j. Thus, $k_j[r]$ stores the number of training instances that have label y_j and have exactly r neighbors with label y_j. Therefore, $\tilde{k}_j[r]$ counts the number of training instances which do not have y_j and have exactly r neighbors with label y_j. Using these frequency arrays, the posterior probabilities can be estimated:

$$\begin{aligned}
P(C_j|H_j) &= \frac{s + \kappa_j[C_j]}{s \times (k+1) + \sum_{r=0}^{k} \kappa_j[r]},\ (1 \leq j \leq q,\ 0 \leq C_j \leq k) \\
P(C_j|\neg H_j) &= \frac{s + \tilde{\kappa}_j[C_j]}{s \times (k+1) + \sum_{r=0}^{k} \tilde{\kappa}_j[r]},\ (1 \leq j \leq q,\ 0 \leq C_j \leq k)
\end{aligned} \tag{2.37}$$

The final set of relevant labels of an unseen instance can be determined by substituting the prior probabilities (Eq. 2.35) and the posterior probabilities (Eq. 2.37) into Eq. 2.34.

ML-KNN is a *first-order* approach since it considers each label independently. This algorithm has the advantage of inheriting the advantages of both lazy learning and Bayesian reasoning [52]. The decision boundary can be adjusted by using a varying number of neighbors in each query instance. Additionally, the class imbalance, which is a common problem in multi-label data [88], is mitigated due to the prior probabilities estimated for each label. The computational complexity has been also addressed using Graphic Processing Units (GPUs) [89], [90] to speed up ML-KNN in streaming scenarios [91].

Some extensions of ML-KNN have been proposed to smooth the *first-order* approach or the negative impact of the number of labels. DML-KNN [92] which instead of using only the statistical information from positive instances, it also considers the negative instances.

BR-KNN [93] combined multiple KNN, one per label, in a binary relevance manner. They use the count of labels in the set of neighbors as the confidence score for the predictions. MLCW-KNN [94] improves the previous method by assigning weights to each of the instances according to their distances to the query sample. In a similar manner, the labels can be ranked according to the probabilities of the label association using the neighboring samples around a query sample [95]. Finally, IBLR-KNN [60] combines the linear regression and KNN algorithms, having one classifier per label as in BR methods.

Neural Networks

Backpropagation for Multi-label Learning (BP-MLL) [23] adapts the backpropagation algorithm from neural networks. This adaptation is done by replacing its error function with an adaptation of the ranking loss, where the relevant labels of a given instance should be ranked higher than its irrelevant labels.

Given a multi-label training set $\mathcal{D}$ with q labels, BP-MLL defines a neural network with three layers in its topology. The first layer corresponds to the input layer which has d input units, each corresponding to the d-dimensional feature space. The second layer is known as the hidden layer and has r hidden units. Finally, the third layer corresponds to the output layer, with q units, each corresponding to one of the labels. The input and the hidden layer are fully connected with weights v_{hs}, $(1 \leq s \leq r, 1 \leq j \leq q)$ and the hidden layer is also fully connected to the output layer with weights $wsj(1 \leq s \leq r, 1 \leq j \leq q)$. The bias of the hidden units γ_s, $(1 \leq s \leq r)$ are shown as weights from an input unit a_0 with a fixed value of 1. Similarly, the bias of the output units $\zeta_j(1 \leq jleqq)$ are represented as weights from a hidden unit b_0 with a fixed value of 1.

A traditional neural network would be trained by updating the weights of its network based on the error function (usually the least mean square), computed by the difference between the expected result and the actual output. However, in a multi-label problem the error metric needs to consider multiple labels simultaneously, therefore BP-MLL addresses

the characteristics of multi-label problems by re-writing the general error function as follows:

$$E = \frac{1}{\mid Y_i \mid\mid \overline{Y_i} \mid} \sum_{(y_j, y_k) \in Y_i \times \overline{Y_i}} \exp(-(c_j^i - c_k^i)) \tag{2.38}$$

Here, $c_j^i - c_k^i$ measures the difference between the outputs of the network for one label $y_j \in Y_i$ that belongs to the instance x_i, and another label $y_k \in \overline{Y_i}$ that belongs to the complementary set of Y_i, i.e., a label that does not belong to the set of relevant labels. Therefore, the larger the difference between both labels, the better the overall performance of the network. This error is negated and used in an exponential function in order to substantially increase the penalty of the error term.

Assuming that *tanh* is the activation function for the units in the network:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.39}$$

Therefore, the output b_s and input net_{b_s} values associated with the s-th unit in the hidden layer can be defined as follows:

$$\begin{aligned} b_s &= f(net_{b_s} + \gamma_s) \\ net_{b_s} &= \sum_{h=1}^{d} a_h \upsilon_{hs} \end{aligned} \tag{2.40}$$

Here, a_h is the h-th feature of x_i and υ_{hs} is the weight connecting the h-th input unit to the s-th hidden unit.

Similarly, the output c_j and input net_{c_j} values associated with the j-th unit in the output layer are:

$$\begin{aligned} c_j &= f(net_{c_j} + \zeta_j) \\ net_{c_j} &= \sum_{s=1}^{r} b_s w_{sj} \end{aligned} \tag{2.41}$$

Here, b_s refers to the previously defined output of the s-th hidden unit and w_{sj} is the weight connecting the s-th hidden unit and the j-th output unit.

Since the *tahn* function is differentiable, the general error of any given unit is:

$$-\frac{\partial E_i}{\partial net_i} \tag{2.42}$$

Therefore, the general error for the j-th output unit can be defined as:

$$d_j = -\frac{\partial E_i}{\partial net_{c_j}} = -\frac{\partial E_i}{\partial c_j}\frac{\partial c_j}{\partial net_{c_j}} = -\frac{\partial E_i}{\partial c_j} f'(net_{c_j} + \zeta_j) \tag{2.43}$$

Then, considering the following simplifications:

$$\begin{aligned} \frac{\partial E_i}{\partial c_j} &= \frac{\partial \Big[\frac{1}{|Y_i||\overline{Y_i}|}\sum_{(y_j,y_k)\in Y_i\times\overline{Y_i}} \exp(-(c_j^i - c_k^i))\Big]}{\partial c_j} \\ \frac{\partial E_i}{\partial c_j} &= \begin{cases} -\frac{1}{|Y_i||\overline{Y_i}|}\sum_{y_l\in\overline{Y}_i} \exp(-(c_j - c_l)), & \text{if } y_j \in Y_i \\ \frac{1}{|Y_i||\overline{Y_i}|}\sum_{y_k\in Y_i} \exp(-(c_k - c_j)), & \text{if } y_j \in \overline{Y}_i \end{cases} \end{aligned} \tag{2.44}$$

$$f'(net_{c_j} + \zeta_j) = (1 + c_j)(1 - c_j) \tag{2.45}$$

Finally, the general error for the j-th output unit is defined as:

$$d_j = \begin{cases} \Big(\frac{1}{|Y_i||\overline{Y_i}|}\sum_{y_l\in\overline{Y}_i} \exp(-(c_j - c_l))\Big)(1 + c_j)(1 - c_j), & \text{if } y_j \in Y_i \\ \Big(-\frac{1}{|Y_i||\overline{Y_i}|}\sum_{y_k\in Y_i} \exp(-(c_k - c_j))\Big)(1 + c_j)(1 - c_j), & \text{if } y_j \in \overline{Y}_i \end{cases} \tag{2.46}$$

In a similar manner, the general error for the s-th hidden unit is defined as follows:

$$e_s = -\frac{\partial E_i}{\partial net_{b_s}} = -\frac{\partial E_i}{\partial b_s}\frac{\partial b_s}{\partial net_{b_s}} = -\Big(\sum_{j=1}^{q} \frac{\partial E_i}{\partial net_{c_j}}\frac{\partial net_{c_j}}{\partial b_s}\Big) f'(net_{c_j} + \gamma_j) \tag{2.47}$$

Then, considering the following simplifications:

$$d_j = -\frac{\partial E_i}{\partial net_{c_j}} \tag{2.48}$$

$$net_{c_j} = \sum_{s=1}^{r} b_s w_s j \tag{2.49}$$

$$f'(net_{b_s} + \gamma_s) = (1 + b_s)(1 - b_s) \tag{2.50}$$

Finally, the general error for the s-th hidden unit is defined as:

$$
\begin{aligned}
e_s &= \Big(\sum_{j=1}^{q} d_j \frac{\partial \big[\sum_{s=1}^{r} b_s w_{sj} \big]}{\partial b_s} \Big) f'(net_{b_s} + \gamma_s) \\
e_s &= \big(\sum_{j=1}^{q} d_j w_{sj} \big) f'(net_{b_s} + \gamma_s) \\
e_s &= \big(\sum_{j=1}^{q} d_j w_{sj} \big)(1 + b_s)(1 - b_s)
\end{aligned}
\tag{2.51}
$$

Using all the previous definitions, it is possible to define the change of the weights by using the *gradient descent* strategy, i.e., updating the weights proportionally to the negative gradient:

$$
\Delta w_{sj} = -\alpha \frac{\partial E_i}{\partial w_{sj}} = -\alpha \frac{\partial E_i}{\partial net_{c_j}} \frac{\partial net_{c_j}}{\partial w_{sj}} = \alpha d_j \left[\frac{\partial \big(\sum_{s=1}^{r} b_s w_{sj} \big)}{\partial w_{sj}} \right] = \alpha d_j b_s \tag{2.52}
$$

$$
\Delta \upsilon_{hs} = -\alpha \frac{\partial E_i}{\partial \upsilon_{hs}} = -\alpha \frac{\partial E_i}{\partial net_{b_s}} \frac{\partial net_{b_s}}{\partial \upsilon_{hs}} = \alpha e_s \left[\frac{\partial \big(\sum_{h=1}^{d} a_h \upsilon_{hs} \big)}{\partial \upsilon_{hs}} \right] = \alpha e_s a_h \tag{2.53}
$$

$$
\begin{aligned}
\Delta \zeta_j &= \alpha d_j \\
\Delta \gamma_s &= \alpha e_s
\end{aligned}
\tag{2.54}
$$

where α denotes the *learning rate* whose value is in the range of $(0, 1)$.

For an unseen instance x, BP-MLL uses the results from the output units $c_j (1 \leq j \leq q)$ to produce a label ranking. Therefore, the relevant set of labels depends on a threshold function. This presents the same problem as in Rank-SVM since both of them define the ranking loss as their error criterion. In this case, BP-MLL decides to follow the same approach, which is to model $t(x)$ by a linear function, as it can be seen in Eq. 2.30. Therefore, once the parameters of the linear function (w^*, b^*) have been learned, the set of relevant labels can be set as follows:

$$
Y = \{ y_j \mid c_j > \langle w^*, f^*(x) \rangle + b*, \, 1 \leq j \leq q \} \tag{2.55}
$$

BP-MLL is considered a *second-order* approach which adapts the traditional back-propagation error to multi-label problems. This approach compares the relevant and irrelevant label pairs in the error function, in order to adjust the weights of the network. This method achieves competitive performance but at a high computational cost, which is a common characteristic of the neural networks.

Some extensions to the concepts presented by BP-MLL have been proposed in other works. For example, in [96] a deep neural network for multi-label data, where they used up to five hidden layers each composed of up to 1000 units. Another popular method is *Multi Label Radial Basis Function (ML-RBF)* [97] which adapts the radial basis function to multi-label problems. ML-RBF consists of two hidden layers: the first layer is formed by conducting clustering analysis on instances of each possible label, where the centroid of each group is regarded as the prototype vector of a basis function. The second layer learns a series of weights by minimizing the sum of the square errors function. Here, the information encoded in the prototype vectors corresponding to all classes is fully exploited to optimize the weights corresponding to each specific label.

Probabilistic models

Multi-label Naive Bayes (MLNB) [98] adapts the naive Bayes classifier to multi-label data by using the Bayesian rule. Therefore, it assumes independence among the features.

For an unseen instance x, let H_j be the event that x considers y_j as a relevant label, $P(x|H_j)$ represents the label conditional probability that H_j holds on x with label y_j. Consequently, $\neg H_j$ represents the event that x consider y_j as an irrelevant label. Therefore, $P(x|\neg H_j)$ is the conditional probability that H_j does not hold conditioned to x. Following that notation, MLNB determines the relevant set of labels by using the following *maximum a posteriori* (MAP) principle:

$$Y = \{y_j | \frac{P(H_j|x)}{P(\neg H_j|x)} > 1,\ 1 \leq j \leq q\} \tag{2.56}$$

Using the *Bayesian rule* under the assumption of label conditional independence among the features, as it is assumed in the traditional Naive Bayes, the conditional probabilities are rewritten as:

$$\begin{aligned} P(H_j|x) &= \frac{P(\neg H_j)P(x|\neg H_j)}{P(x)} = P(\neg H_j)\prod_{k=1}^{d} P(x_k|\neg H_j) \\ P(\neg H_j|x) &= \frac{P(H_j)P(x|H_j)}{P(x)} = P(H_j)\prod_{k=1}^{d} P(x_k|H_j) \end{aligned} \tag{2.57}$$

Assuming that the density of the k-th feature conditioned on the label follow the *Gaussian probability density function* $g(\cdot, \mu_{jk}, \sigma_{jk})$, the conditioned probabilities are calculated as follows:

$$\begin{aligned} P(H_j|x) &= P(\neg H_j)\exp(\phi_j) \\ \text{where } \phi_j &= -\sum_{k=1}^{d}\frac{(x_k-\mu_{jk})^2}{2\sigma_{jk}^2} - \sum_{k=1}^{d}\ln\phi_{jk} \\ P(\neg H_j|x) &= P(H_j)\exp(\neg\sigma_j) \\ \text{where } \neg\phi_j &= -\sum_{k=1}^{d}\frac{(x_k-\neg\mu_{jk})^2}{2\neg\sigma_{jk}^2} - \sum_{k=1}^{d}\ln\neg\sigma_{jk} \end{aligned} \tag{2.58}$$

Here, μ_{jk} refers to the mean, while σ_{jk} refers to the standard deviation, of the k-th feature conditioned to H_j.

However, it is noticed that for a high number of features the term ϕ may be too negatively large, therefore making the computation of $\exp(\phi)$ exceed the floating precision. In order to avoid that problem, it is proposed to compute the probability of $P(H_j|x)$ as:

$$\begin{aligned} P(H_j|x) &= \frac{P(H_j)P(x|H_j)}{P(H_j)P(x|H_j)+P(\neg H_j)P(x|\neg H_j)} = \frac{P(H_j)}{P(H_j)+P(\neg H_j)\frac{P(x|\neg H_j)}{P(x|H_j)}} \\ &= \frac{P(H_j)}{P(H_j)+P(\neg H_j)\exp(\neg\phi_j-\phi_j)} \\ P(\neg H_j|x) &= 1-P(H_j|x) \end{aligned} \tag{2.59}$$

In this case, the exponent of the difference $(\neg\phi_j - \phi_j)$ it is computationally feasible. In order to predict the final set of relevant labels, these conditional probabilities can be substituted into Eq. 2.56.

MLNB is considered a *first-order* approach since it evaluates the labels independently. Since the naive version of the algorithm assumes label independence, the algorithm was initially proposed to be combined with *principal component analysis* (PCA) and *genetic algorithms* (GA) to mitigate the effect of that assumption. Furthermore, it used a specific fitness function in the GA to address the label correlations explicitly. Although they report competitive performance, this might be the consequence of the PCA and GA combination [98].

2.1.3 Evaluation metrics

Evaluation metrics for multi-label, in contrast to traditional learning, need to verify whether a prediction is correct (all the labels have been predicted correctly), wrong (all the labels predicted are wrong), or partially correct (some of the labels predicted are correct). As a result, there are a number of evaluation metrics which capture different aspects of the predictions. These metrics will be summarized following the taxonomy proposed in [52], [65], [69], [77] which categorize them into *example-based* metrics and *label-based* metrics.

The metrics are defined according to the notation in Table 2.1, therefore given a test set $\mathcal{S}$ with p test instances, Y stands for the true label set and Z for the predicted label set.

2.1.3.1 Example-based metrics

Example-based metrics evaluate the predictions in each of the test instances and then averages across all the test set. These metrics give the same weight to all the instances, thus ignoring any type of imbalance in the number of labels present in each instance, i.e., it gives the same weight to an instance with only one label present than those with many.

Hamming loss computes the symmetric difference (Δ) between the predicted set of labels and the true labels. The *Hamming loss* is useful when the application wants to consider errors of all types equally important, i.e., incorrect prediction of negative labels and missing positive labels [99].

$$Hamming\ loss = \frac{1}{p}\sum_{i=1}^{p}|Z_i\ \Delta\ Y_i| \tag{2.60}$$

Subset accuracy requires the predicted set of labels to exactly match the real labels. This is an overly strict evaluation measure, especially for high-dimensional label spaces, which penalizes the subsets that are almost correct. This measure is very important in applications where the prediction of values is one step in a chain, and if the values are predicted incorrectly the rest of the process may fail [99].

$$Subset\ accuracy = \frac{1}{p}\sum_{i=1}^{p}[\![Z_i = Y_i]\!] \tag{2.61}$$

In order to be able to measure partially correct predicted label sets, [61] proposed the following definitions for the traditional *accuracy*, *precision*, *recall*, and F_1.

Example-based accuracy is the proportion of correct predicted labels to the total number of labels of that instance (true and predicted).

$$Accuracy_{example} = \frac{1}{p}\sum_{i=1}^{p}\frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \cup Z_i \mid} \tag{2.62}$$

Example-based precision is the proportion of correct predicted labels to the total number of predicted labels.

$$Precision_{example} = \frac{1}{p}\sum_{i=1}^{p}\frac{\mid Y_i \cap Z_i \mid}{\mid Z_i \mid} \tag{2.63}$$

Example-based recall is the proportion of correct predicted labels to the total number of true labels.

$$Recall_{example} = \frac{1}{p}\sum_{i=1}^{p}\frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \mid} \tag{2.64}$$

Example-based F1 is the harmonic mean between the example-based precision and example-based recall, hence it takes into account both false positives and false negatives into account.

$$F_{1\text{-}example} = \frac{1}{p}\sum_{i=1}^{p}\frac{2 \mid Y_i \cap Z_i \mid}{\mid Y_i \mid + \mid Z_i \mid} \tag{2.65}$$

2.1.3.2 Label-based metrics

Label-based metrics evaluate the predictions per label and differ from the example-based metrics in the way they are averaged. As mentioned previously in the notation, the labels are usually represented as a binary vector, therefore any evaluation metric from binary classification could be used. The label metrics can be categorized based on how they average the binary evaluation metric, into either *macro-averaged* metrics and *micro-averaged* metrics.

Macro-averaged metrics: They are computed on individual instances first and then averaged over all labels. These measures give equal weight to every label, regardless of its frequency, thus is heavily influenced by the predictions of the under-represented labels.

$$Macro - average = \frac{1}{q}\sum_{j=1}^{q} B(Y^j, Z^j) \tag{2.66}$$

$$Accuracy_{macro} = \frac{1}{q}\sum_{j=1}^{q}\left[\frac{\sum_{i=1}^{p} \mid Y_i^j \cap Z_i^j \mid}{\sum_{i=1}^{p} \mid Y_i^j \cup Z_i^j \mid}\right] \tag{2.67}$$

$$Precision_{macro} = \frac{1}{q}\sum_{j=1}^{q}\left[\frac{\sum_{i=1}^{p} \mid Y_i^j \cap Z_i^j \mid}{\sum_{i=1}^{p} \mid Z_i^j \mid}\right] \tag{2.68}$$

$$Recall_{macro} = \frac{1}{q}\sum_{j=1}^{q}\left[\frac{\sum_{i=1}^{p} \mid Y_i^j \cap Z_i^j \mid}{\sum_{i=1}^{p} \mid Y_i^j \mid}\right] \tag{2.69}$$

$$F_{1-macro} = \frac{1}{q}\sum_{j=1}^{q}\left[\frac{\sum_{i=1}^{p} 2 \mid Y_i^j \cap Z_i^j \mid}{\sum_{i=1}^{p} \mid Y_i^j \mid + \mid Z_i^j \mid}\right] \tag{2.70}$$

Micro-averaged metrics: They are computed globally over all instances and labels. These measures give equal weight to every instance, therefore it tends to be more influenced by the predictions of the most common labels.

$$Micro - average = B\Big(\sum_{j=1}^{q} Y^j, Z^j\Big) \tag{2.71}$$

$$Accuracy_{micro} = \frac{\sum_{j=1}^{q}\sum_{i=1}^{p} \mid Y_i^j \cap Z_i^j \mid}{\sum_{j=1}^{q}\sum_{i=1}^{p} \mid Y_i^j \cup Z_i^j \mid} \tag{2.72}$$

$$Precision_{micro} = \frac{\sum_{j=1}^{q}\sum_{i=1}^{p} \mid Y_i^j \cap Z_i^j \mid}{\sum_{j=1}^{q}\sum_{i=1}^{p} \mid Z_i^j \mid} \tag{2.73}$$

$$Recall_{micro} = \frac{\sum_{j=1}^{q}\sum_{i=1}^{p} \mid Y_i^j \cap Z_i^j \mid}{\sum_{j=1}^{q}\sum_{i=1}^{p} \mid Y_i^j \mid} \tag{2.74}$$

$$F_{1-micro} = \frac{\sum_{j=1}^{q}\sum_{i=1}^{p} 2 \mid Y_i^j \cap Z_i^j \mid}{\sum_{j=1}^{q}\sum_{i=1}^{p} \mid Y_i^j \mid + \mid Z_i^j \mid} \tag{2.75}$$

Although there is no agreement about when to use macro-averaged or micro-averaged metrics, [100] stated that macro-averaged metrics should be used when the predictions need to be consistent across all labels regardless of their frequency, while micro-averaged metrics should be used when the distribution of the labels is important.

2.1.3.3 Multi-label data statistics

Multi-label data can use traditional indicators such as the number of instances, features, and labels. However, there are other measures specific to describe the characteristics of a label set [49], [52].

Cardinality is the average number of labels associated with each instance.

$$Cardinality(\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}|Y_i| \tag{2.76}$$

Density is the normalized version of cardinality, i.e., it is the cardinality divided by the number of labels.

$$Density(\mathcal{D}) = \frac{1}{q}\ Cardinality(\mathcal{D}) \tag{2.77}$$

Distinct is the number of distinct label sets present across all the instances.

$$Distinct(\mathcal{D}) = |\{Y \mid \exists x : (x, Y) \in \mathcal{D}\}| \tag{2.78}$$

Furthermore, there are many other indicators regarding the imbalance of the labels and label sets, including the maximum imbalance ratio, the mean imbalance ratio, or the coefficient of variation of IRLbl [88], [101], [102].

2.1.4 Benchmark datasets

Tables 2.7, 2.8, and 2.9, summarize the characteristics of the public multi-label benchmark datasets. The datasets are ordered alphabetically, however, whenever they are used in the experiments the order in which they are presented might change to facilitate its readability with respect to the size of the problem to solve.

The considerable growth of the multi-label popularity in recent years has to lead to many publicly available datasets from many different domains. These datasets have been collected from the *Knowledge Discovery and Intelligent Systems* (KDIS) repository[2], although originally they could be found on the MULAN[3], and MEKA[4] repositories websites. Each of the datasets in the tables has associated its domain, number of instances, total number of features and its breakdown by type, number of labels, number of unique subsets, cardinality and density.

[2]KDIS: `http://www.uco.es/kdis/mllresources`

[3]MULAN: `http://mulan.sf.net`

[4]MEKA: `http://meka.sf.net`

Table 2.7.: Summary of multi-label datasets and associated statistics (I)

Dataset	Domain	Instances	Attributes	Unary	Binary	Nominal	Numeric	Labels	Distinct	Cardinality	Density
20NG	Text	19,300	1,006	0	0	1,006	0	20	55	1.0289	0.0514
3Sources (BBC)	Text	352	1,000	0	0	0	1,000	6	15	1.1250	0.1875
3Sources (Guardian)	Text	302	1,000	0	0	0	1,000	6	14	1.1258	0.1876
3Sources (inter)	Text	169	3,000	0	61	263	2,676	6	11	1.1420	0.1903
3Sources (Reuters)	Text	294	1,000	0	0	0	1,000	6	14	1.1259	0.1876
Bibtex	Text	7,395	1,836	0	0	1,836	0	159	2,856	2.4019	0.0151
Birds	Audio	645	260	156	0	1	103	19	133	1.0140	0.0534
Bookmarks	Text	87,856	2,150	0	0	2,150	0	208	18,716	2.0281	0.0098
CAL500	Music	502	68	68	0	0	0	174	502	26.0438	0.1497
CHD	Medicine	555	49	0	0	42	7	6	34	2.5802	0.4300
Corel16k (1)	Image	13,766	500	0	0	500	0	153	4,803	2.8587	0.0187
Corel16k (2)	Image	13,761	500	0	0	500	0	164	4,868	2.8824	0.0176
Corel16k (3)	Image	13,760	500	0	0	500	0	154	4,812	2.8286	0.0184
Corel16k (4)	Image	13,837	500	0	0	500	0	162	4,860	2.8420	0.0175
Corel16k (5)	Image	13,847	500	0	0	500	0	160	5,034	2.8577	0.0179
Corel16k (6)	Image	13,859	500	0	0	500	0	162	5,009	2.8849	0.0178
Corel16k (7)	Image	13,915	500	0	0	500	0	174	5,158	2.8859	0.0166
Corel16k (8)	Image	13,864	500	0	0	500	0	168	4,956	2.8830	0.0172
Corel16k (9)	Image	13,884	500	0	0	500	0	173	5,175	2.9301	0.0169
Corel16k (10)	Image	13,618	500	0	0	500	0	144	4,692	2.8153	0.0196
Corel5k	Image	5,000	499	0	0	499	0	374	3,175	3.5220	0.0094
Delicious	Text	16,105	500	0	0	500	0	983	15,806	19.0200	0.0193
Emotions	Music	593	72	69	0	0	3	6	27	1.8685	0.3114
Enron	Text	1,702	1,001	0	0	1,001	0	53	753	3.3784	0.0637
Eukaryote (GO)	Biology	7,766	12,689	0	0	12,689	0	22	112	1.1456	0.0521
Eukaryote (PseAAC)	Biology	7,766	440	440	0	0	0	22	112	1.1456	0.0521

Table 2.8.: Summary of multi-label datasets and associated statistics (II)

Dataset	Domain	Instances	Attributes	Unary	Binary	Nominal	Numeric	Labels	Distinct	Cardinality	Density
EUR-lex (DC)	Text	19,348	5,000	116	0	25	4,859	412	1,615	1.2923	0.0031
EUR-lex (EV)	Text	19,348	5,000	116	0	25	4,859	3,993	16,467	5.3102	0.0013
EUR-lex (SM)	Text	19,348	5,000	116	0	25	4,859	201	2,504	2.2133	0.0110
Flags	Image	194	19	1	0	6	12	7	54	3.3918	0.4845
Genbase	Biology	662	1,186	1	1,185	0	0	27	1	27.0000	1.0000
Gnegative (GO)	Biology	1,392	1,717	0	0	1,717	0	8	19	1.0460	0.1307
Gnegative (PseAAC)	Biology	1,392	440	433	0	0	7	8	19	1.0460	0.1307
Gpositive (GO)	Biology	519	912	0	0	912	0	4	7	1.0077	0.2519
Gpositive (PseAAC)	Biology	519	440	366	0	0	74	4	7	1.0077	0.2519
Human	Biology	3,106	440	440	0	0	0	14	85	1.1851	0.0847
Human (GO)	Biology	3,106	9,844	0	0	9,844	0	14	85	1.1851	0.0847
Human (PseAAC)	Biology	3,106	440	440	0	0	0	14	85	1.1851	0.0847
IMDB	Text	120,919	1,001	0	0	1,001	0	28	4,503	1.9997	0.0714
LangLog	Text	1,460	1,004	0	6	998	0	75	304	1.1801	0.0157
Mediamill	Video	43,907	120	120	0	0	0	101	6,555	4.3756	0.0433
Medical	Text	978	1,449	0	0	1,449	0	45	94	1.2454	0.0277
Nus-Wide (128D cVLAD+)	Image	269,648	129	129	0	0	0	81	18,430	1.8685	0.0231
Nus-Wide (500D)	Image	269,648	501	501	0	0	0	81	18,430	1.8685	0.0231
Ohsumed	Text	13,929	1,002	0	0	1,002	0	23	1,147	1.6631	0.0723
Plant	Biology	978	440	428	0	0	12	12	32	1.0787	0.0899
Plant (GO)	Biology	978	3,091	0	0	3,091	0	12	32	1.0787	0.0899
Plant (PseAAC)	Biology	978	440	428	0	0	12	12	32	1.0787	0.0899
Reuters-K500	Text	6,000	500	98	0	36	366	103	811	1.4622	0.0142
Reuters-RCV1 (1)	Text	6,000	47,236	27,686	6,385	12,184	981	101	1,028	2.8797	0.0285
Reuters-RCV1 (2)	Text	6,000	47,236	27,631	6,422	12,197	986	101	954	2.6342	0.0261
Reuters-RCV1 (3)	Text	6,000	47,236	27,694	6,476	12,076	990	101	939	2.6142	0.0259
Reuters-RCV1 (4)	Text	6000	47229	27876	6444	11929	980	101	816	2.4837	0.0246
Reuters-RCV1 (5)	Text	6000	47235	27863	6460	11923	989	101	946	2.6415	0.0262

Table 2.9.: Summary of multi-label datasets and associated statistics (III)

Dataset	Domain	Instances	Attributes	Unary	Binary	Nominal	Numeric	Labels	Distinct	Cardinality	Density
Scene	Image	2,407	294	0	0	0	294	6	15	1.0740	0.1790
Slashdot	Text	3,782	1,079	0	1,079	0	0	22	156	1.1809	0.0537
Stackex (Chemistry)	Text	6,961	540	0	0	540	0	175	3,032	2.1093	0.0121
Stackex (Chess)	Text	1,675	585	0	18	567	0	227	1,078	2.4113	0.0106
Stackex (Coffee)	Text	225	1,763	0	1,482	281	0	123	174	1.9867	0.0162
Stackex (Cooking)	Text	10,491	577	0	1	576	0	400	6,386	2.2248	0.0056
Stackex (CS)	Text	9,270	635	0	0	635	0	274	4,749	2.5562	0.0093
Stackex (Philosophy)	Text	3,971	842	0	2	840	0	233	2,249	2.2720	0.0098
TMC2007	Text	28,596	49,060	130	48,930	0	0	22	1,341	2.1579	0.0981
TMC2007 (500)	Text	28,596	500	0	500	0	0	22	1,172	2.2196	0.1009
Virus (GO)	Biology	207	749	0	749	0	0	6	17	1.2174	0.2029
Virus (PseAAC)	Biology	207	440	0	0	264	176	6	17	1.2174	0.2029
Water quality	Chemistry	1,060	16	0	0	4	12	14	825	5.0726	0.3623
Yahoo (Arts)	Text	7,484	23,146	4	5,188	17,954	0	26	599	1.6539	0.0636
Yahoo (Bussiness)	Text	11,214	21,924	5	4,139	17,780	0	30	233	1.5990	0.0533
Yahoo (Computers)	Text	12,444	34,096	30	8,027	26,039	0	33	428	1.5072	0.0457
Yahoo (Education)	Text	12,030	27,534	6	6,131	21,397	0	33	511	1.4632	0.0443
Yahoo (Entertainment)	Text	12,730	32,001	15	5,393	26,592	1	21	337	1.4137	0.0673
Yahoo (Health)	Text	9,205	30,605	18	11,051	19,536	0	32	335	1.6441	0.0514
Yahoo (Recreation)	Text	12,828	30,324	6	5,481	24,837	0	22	530	1.4289	0.065
Yahoo (Reference)	Text	8,027	39,679	15	11,658	28,006	0	33	275	1.1744	0.0356
Yahoo (Science)	Text	6,428	37,187	9	11,176	26,002	0	40	457	1.4498	0.0362
Yahoo (Social)	Text	12,111	52,350	30	14,684	37,635	1	39	361	1.2793	0.0328
Yahoo (Society)	Text	14,512	31,802	21	3,487	28,293	1	27	1,054	1.6704	0.0619
Yeast	Biology	2,417	103	0	0	0	103	14	198	4.2371	0.3026
Yelp	Text	10,806	671	0	671	0	0	5	32	1.6383	0.3277

2.1.5 Open source multi-label libraries

There are a series of advantages to incorporating the open source philosophy to the machine learning research [103], such as reproducibility of the experiments, the fair comparison of methods, the detection of errors, the combination of advances, the emergence of standards, and the faster adoption of new methods, among others.

There have been some efforts towards the goal of creating open source libraries for multi-label learning. These libraries usually differ in the methods that have implemented, especially for the algorithm adaptation methods since they are more complicated to develop. Nevertheless, multi-label learning is still a relatively new adopted paradigm by the machine learning community. The most popular open source libraries for multi-label learning are:

- **Mulan** [104]: It can be considered the first multi-label learning library, developed by Tsoumakas et al. in 2010. Mulan is a Java programmatic library built on top of Weka [105] and available under the GNU GPL license. Mulan was the pioneer library in multi-label learning, therefore it includes most of the first algorithm adaptations methods that cannot be found in newer libraries.

- **Meka** [106]: It is a very popular library which was built to address the performance issues from Mulan. Meka is also a Java library built on top of Weka and available under the GNU GPL license. Meka specializes in the problem transformation methods, in particular, the classifier chains paradigm, implementing at least 10 variations; and also meta-learners for combining them together.

- **Scikit-Multilearn** [107]: It is a multi-label learning library that was built on top of Scikit-learn using Python and under the BSD license. They report performance on par with Meka, and superior to Mulan using the RAkEL method. Scikit-Multilearn specializes in label space division, which uses a graph with co-occurrences of labels to divide the output space using community detection methods from the igraph library.

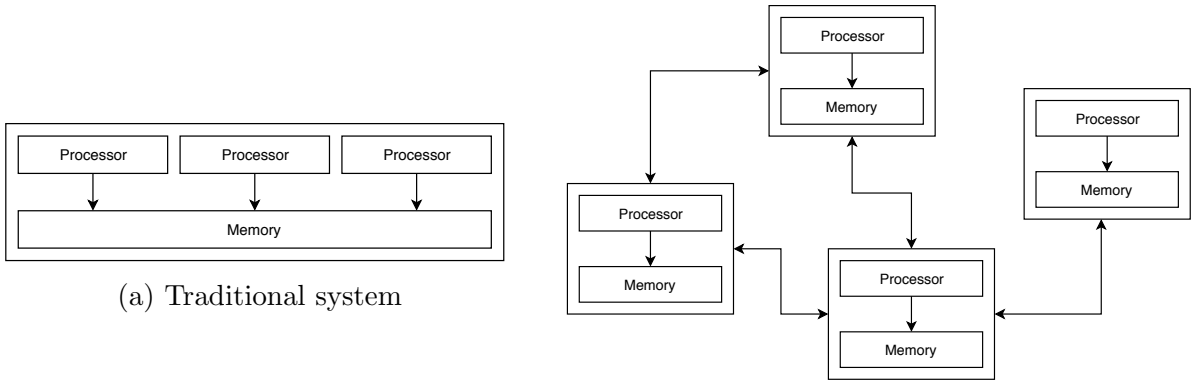
2.2 Distributed systems

The computational throughput of current systems can only be improved by either increasing the clock speed of the processors or by increasing the number of processors. Increasing the number of processors in a system would lead to better performance and higher bandwidth. There are many definitions of distributed systems in literature [108], however, the most appropriate for currents systems (and for this work) is: "a collection of independent computers that appears to its users as a single coherent system" [109] and "a collection of autonomous computers linked by a computer network with distributed system software" [40]. As indicated by their definitions, these systems do not have a shared memory or a shared clock and communicate with each other by passing information over the network. This collection of computers appears to the user as a single system, despite their possible differences in hardware and software.

Figure 2.5 presents a simple diagram of traditional and distributed systems. A distributed system has a series of advantages over traditional systems, such as: sharing computational resources and data, logical simplicity since each data in a remote machine can be considered an object, more reliable because the failure of a machine does not imply the failure of the whole system, modular because it allows adding resources easily, and lower cost than a single bigger system. However, they possess a series of challenges related to the communication and synchronization over the network which can lead to potential delays in the computation.

If a traditional system would see its workload drastically increased, the only way to increase the throughput of the system is by upgrading its hardware. This concept is called *scaling vertically*, also known as scaling up. This type of scaling requires to shut the system down while the new resources are being added. It is also limited by the latest hardware capabilities, however, in most of the cases, these capabilities are insufficient for moderate to big workloads.

On the other hand, whenever a distributed system sees its workload increase it just needs to add more resources rather than upgrading the hardware of a single machine. This concept



(a) Traditional system

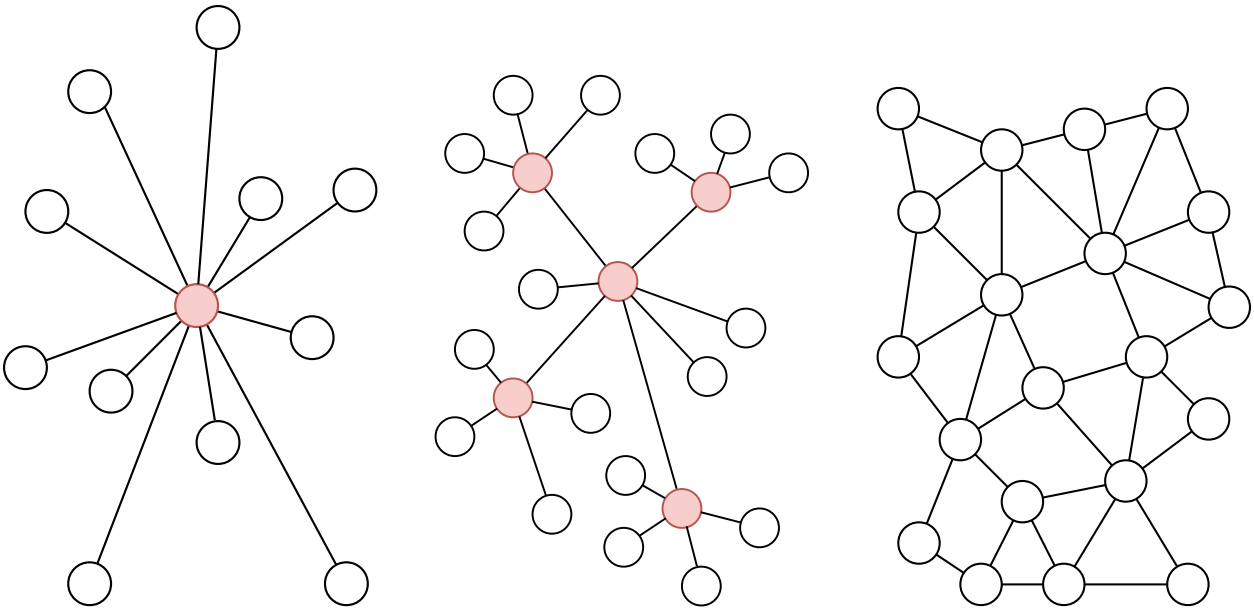
(b) Distributed system

Fig. 2.5.: A traditional system with three processors sharing memory space and a distributed system with four processors, with independent memory spaces, connected through a network

is known as *scaling horizontally*, also known as scaling out, and it is significantly cheaper than vertical scaling after a small threshold. The distributed systems are characterized by having no cap in how much it can scale, whenever there is a need for more performance, more machines can be added to the system.

Figure 2.6 shows the types of networks according to [110]. Here, a network is considered a collection of interlinked nodes that exchange information, where a node is simply a user or a machine. These networks can be categorized into three categories that would be applicable to the architecture of computer systems.

In a **centralized** network, a series of terminals are connected to a single machine. The main machine is in charge of performing the main computation and provide the results to the terminals. This type of system is easy to maintain since there is only a single point of failure, however, this makes the system very unreliable since a failure on the main machine could take the whole system down. This system can only scale vertically by upgrading the main machine on the network. However, this approach can be deployed straightforward by applying the desired framework to the machine. This type of system was used in old libraries, where there were a series of terminals which would connect to the main catalog and allow the user to query the database.



(a) Centralized (b) Decentralized (c) Distributed

Fig. 2.6.: Types of network processing

A **decentralized** system considers that each terminal is connected to the network or subnetwork. The computation can be either be performed at a terminal or by any other node, and the resulting system behavior is the aggregation of all the responses. Conceptually, it is a network of centralized networks, where the terminals could interact with the system through many entry points. These systems have better stability and fault tolerance, but if you kill one of the main nodes in a subnetwork many of the terminals would experience issues. It also posses better scalability since it allows for both vertical and horizontal scalability.

In a **distributed** network there are no central machines and each terminal is connected to various other terminals, the data simply travels through whichever terminal allows the most convenient route to the recipient. In a distributed system the computation is done in each machine, still, in some cases there are *leaders* which other terminals must follow. These systems are the most difficult to maintain, however, they are very stable and a single failure barely harms the system. Their scalability is completely horizontal, the more terminals connected the larger the network. These systems are very hard to deploy as it needs to handle some details such as information sharing and communication.

The systems considered in this thesis can be considered centralized distributed systems in the technical sense. Although most of them were conceived as a decentralized system, in practice there is an owner of the system which provides its entry point. This entry point takes care of most of the issues in a distributed environment such as synchronization, resource allocation, naming, configuration, etc.

2.2.1 Characteristics of distributed systems

Although it is possible to build a distributed system in many cases, it is not recommendable to distribute all the problems. There are some problems which are not worthy of distribution, therefore doing so would be pointless. A distributed system is characterized by solving the following challenges:

- **Heterogeneity**: A system is heterogeneous if it is composed of dissimilar hardware and software. Distributed systems allow to scatter the information among a heterogeneous collection of machines by implementing common standards, otherwise, the representation of primitive data and message structure could differ between machines. Whenever there is not a standard agreed and adopted, the distributed systems define a middleware layer which masks the underlying differences between the systems. This layer ensures to translate the messages to the appropriate format by considering the unique characteristics of each device.

- **Openness**: It is the characteristic that enables systems to be extended to meet new application requirements and user needs. This is achieved by specifying and documenting the key software interfaces of a system and making them available to software developers, i.e., the interfaces are published [40].

- **Security**: The resources in a distributed system need to be secure by achieving three goals [40]: *confidentiality* that guarantees protection against disclosure to unauthorized individuals, *integrity* which provides protection against alteration or corruption, and *availability* that grants protection against interference with the means to access the

resources. These challenges are addressed by two main parts of the security system: *authentication* and *authorization*. The first considers guaranteeing that an entity is what it claims to be [111], and the former determines the user privileges permitting only those privileges to be available.

- **Scalability**: It is the ability of a distributed system to grow without users or applications being affected. A system is defined as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users [40]. Scalability is a major factor in distributed systems and it should be considered while designing the components. It is hard to scale systems that have not been designed to do so, thus it is required that applications and platforms were conceived for that purpose.

- **Failure handling**: A distributed system is considered to be fault tolerant if it is able to continue processing when one or more components of the system fail. Each component of the distributed system needs to be aware of the possible ways in which the components it depends on may fail or be designed to deal with each of those failures appropriately [40]. There are a series of techniques for dealing with failures in a distributed environment:

  ⋆ *Detecting failures*: The detection of failures can be done actively by sending a ping type request, or passively by waiting for the components to send a communication to the monitor. Another type of errors considers corrupted data, which can be detected by applying checksum.

  ⋆ *Masking failures*: Some errors can be hidden to the user, or in the worst case at least subdue their impact. For example, whenever there is an error with transmitted data it can be resent again, or stored data can be duplicated in case it gets corrupted.

⋆ *Tolerate failures*: It is not feasible to attempt to detect and mask all the failures that might happen in a distributed system. This technique involves the user tolerating the errors.

⋆ *Recovery from failures*: If the nature of the error can be completely and accurately acceded, then it is possible to remove those errors and enable the system to continue. This technique is known as forwarding error recovery. On the other hand, if the nature of the error cannot be accessed, the only way to remove those errors from the system state is by returning to a previous error-free stable state. This technique is known as a backward error recovery.

⋆ *Redundancy*: When an error is produced at some point during the execution, the redundant component can serve two purposes: first, to provide backup service while the main component is down, and second, to restore the failed component to an error-free state.

- **Concurrency**: It refers to the possibility that multiple processes interact with the same resource simultaneously. In order to guarantee the correct behavior of the system, while maintaining its output, the system ensures the use of a shared resource is synchronized among different processes.

- **Transparency**: It defines the concealment from the user and from the application programmer of the separation of components in a distributed system so that the system is perceived as a whole rather than as a collection of independent components [40], [112].

2.2.2 Categories of distributed systems

A distributed system is usually a conglomerate of complex components or even different distributed systems. According to their goal, and fulfilling the previous characteristics, the distributed systems can be categorized into the following: *Databases*, *Computing*, *File Systems*, *Messaging*, *Applications*, and *Ledgers*.

Distributed Databases

A distributed database is a database in which not all storage devices are attached to a common processor. These type of database systems follow the CAP theorem, which states that only two of the following three properties can be met:

- **Consistency**: Any read operation that begins after a write operation completes must return the updated value [113]. This definition aligns better with linearizability [114], which is a very specific (and very strong) notion of consistency.
- **Availability**: Every request received by a non-failing node in the system must result in a response [113]. It also considers the case in which a sibling node was updated, the current node could queue requests until it is able to reflect the most recent changes.
- **Partition tolerance**: It refers to the possibility to lose arbitrarily any message sent through the network [113]. If this property is not met, it considers that a single node going down would take the whole system with it.

When a distributed database gets partitioned it can only guarantee consistency or availability, not both. The only way to meet both properties would be to have a traditional relational database, which drops the partition property.

Given a distributed database with two nodes, assume that the connection network between both nodes fails and each of them is isolated. The distributed database has two clear choices: to keep both nodes fully running, however, the changes in each node will not appear in the other (it violates consistency), or to make sure only one of them accepts requests until the connection with the other node can be reestablished (it violates availability).

The systems that prioritize the availability settle for *eventual consistency* which means that if no new updates are made to a given item, eventually all accesses to that item will return the latest updated value. Some distributed databases which prioritize availability are *Cassandra*, *Riak*, *Voldemort*, and *CouchDB*. While other systems focus on having a strong consistency, such as *Neo4j*, *Google Bigtable*, *MongoDB*, *HBase*, *Hypertable*, *Redis*.

Distributed Computing

Distributing computing is the methodology that splits extremely large tasks, which cannot be executed in a single machine, into smaller independent tasks that can be performed on many machines in parallel. This description could fit easily in traditional parallel computing, where many tasks are sent to different processors in a single machine. However, all these processors share their memory space allowing all the task to access the same data. On the other hand, a distributed computing system does not have a global address space across all the processors, consequently, the information is exchanged through a network.

Some examples of distributed computing frameworks are *PVM*, *MPI*, *Hadoop*, *Spark*, *Scalding*, *Pig*. The choice of the distributed computing framework was a key issue during our research. This category of distributed systems is studied in depth in the Section 2.2.3, focusing on the evolution of the systems and comparing the most important features.

Distributed File Systems

A distributed file system is a file system that has its components spread across multiple machines with proper authorization rights. Just like in a traditional operative system the files are organized in a hierarchical file management system, the distributed file system uses a uniform naming convention and a mapping scheme to keep track of where files are located. When a process requests a file, it is sent to the local machine where it can be read and modified, after it is no longer needed it is sent back over the network to update its state.

The difference between a distributed file system and a distributed data store is that a distributed file system allows files to be accessed using the same interfaces and semantics as local files. For example, mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model. Distributed data stores, by contrast, require using a different API or library and have different semantics (most often those of a database). Some examples of distributed file systems are *HDFS*, *GFS*, *DFS*, *GlusterFS*, *Ceph*, *HekaFS*.

Distributed Messaging

Distributed messaging is based on the concepts of reliable message queuing. These messages are queued asynchronously between users and the messaging system. This allows decoupling your application logic or your users from directly talking with your other systems.

Most of these systems follow the publish-subscribe model where the senders of the messages are known as publishers and the receivers are called consumers. Once a message has been published by the sender, the consumers can subscribe to one or more messages using some filtering options. These filters are usually topic-based or content-based, and help to select the desired messages. The most popular distributed message system are *Kafka*, *RabbitMQ*, and *Amazon SQS*.

Distributed Applications

A distributed application is software that is executed on multiple machines within a network to achieve specific goals or tasks. It is very important to emphasize that we do not consider a traditional application using a distributed database to be a distributed application. Although technically, it uses multiple machines to execute, they do not collaborate towards the task.

Distributed applications usually follow the peer-to-peer (P2P) computing model, where peers are equally privileged and equipotent participants in the application. Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts. Some examples of distributed applications are *Bittorrent*, *Bitcoin*, *SETI*, *FAROO*, *Tor*.

2.2.3 Distributed computing

Distributed computing is accomplished by using parallel processing but in a pool of loosely coupled computers which collaborate towards a common goal. This environment contrast to a tightly coupled system where all the processors belonging to the same device. A task can

be parallelized by splitting it into multiple chunks and compute those independently. If the task cannot be divided, e.g., a form of a linear sequence of repeated steps where each step is needed to perform the subsequent step, the task is not suitable for distribution. However, if none of the tasks need to execute in a specific sequence, e.g., many forms of search where each worker can take a particular area and do their job independently, the job is appropriate for distribution.

Parallel Virtual Machine (PVM) [115] is the ancestor of distributed computing. This system uses a message-passing model to allow programmers to use distributed computing across a wide variety of systems. The key to PVM is that it makes a collection of heterogeneous computers appear as one large virtual machine. Message-Passing Interface (MPI) [116] is a standardized and portable message-passing standard designed for distributed computation. MPI describes the communications between computational nodes to coordinate calculations.

Both are specifications for libraries that can be used for distributed computing based on message-passing, though, they were designed with different goals [117]. PVM was aimed at providing a portable, heterogeneous environment for using clusters of machines communicated using TCP/IP sockets. While MPI focused on proving an interface to write distributed applications capable of delivering high performance on processors. Therefore MPI performs better respect to execution times, while PVM focuses on network performance [118].

2.2.3.1 MapReduce programming model

The MapReduce [41] programming model was originally designed by Google and it was proposed for distributed processing and generating large data over several machines. This model is suitable for processing large data because of its low infra-cluster communication and its fault-tolerant mechanism, which are highly recommendable for long time executions over large data.

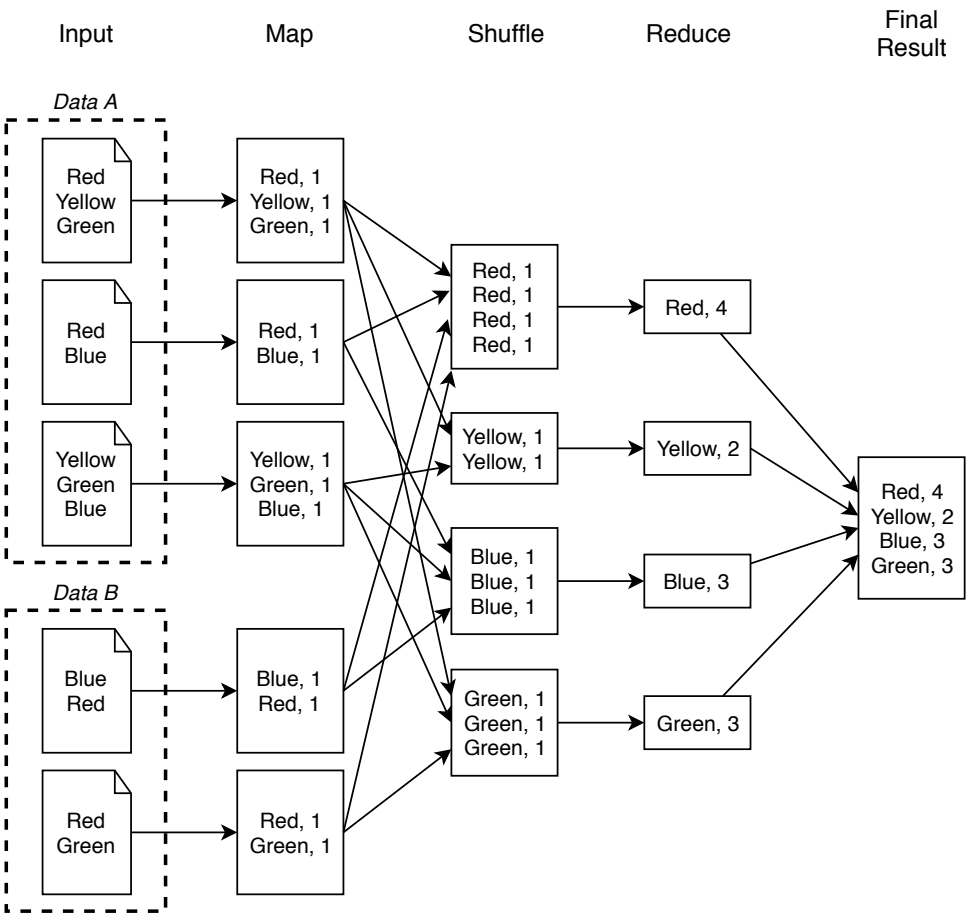


Fig. 2.7.: Workflow of word counting on MapReduce

MapReduce is a completely different paradigm than message-passing models. MapReduce partitions and distributes the data in order to perform the computations locally in each of the workers. Thus, MapReduce takes advantage of local storage to avoid the network bottleneck, which is especially relevant on large volumes of data. On the other hand, message-passing models are best suited for efficient inter-process communication, especially if the application requires asynchronous communication. As a consequence, MapReduce should be the choice for data-oriented scenarios, such as machine learning [119].

The MapReduce framework takes care of the details of partitioning the input data, scheduling the program‘s execution across a set of machines, handling machine failures, and managing the required inter-machine communication [41]. This model takes its name from the two primitives inspired by the functional languages: *Map* and *Reduce*. The Map phase

maps the data into a collection of $< key, value >$ pairs, each of these collections can later be read and transformed into another set of pairs (intermediate results). The Reduce phase combines the key coincident pairs in the same node and merges them into the final result.

Figure 2.7 presents the canonical MapReduce use case, where the frequencies of each word are counted for multiple documents. First, the input data is read from as many documents as needed. Second, the Map function takes a set of words and transforms it into $< word, counter >$ where *word* represents the *key* and *counter* represents the *value* (starting with 1). Third, the data is shuffled in order to have all the tuples with the same key together. Fourth, the Reduce function receives for each key all the values present in the tuples and produces a new tuple with the result, in this case, it just sums the counters of each word. Finally, the results from all the Reduce functions are combined into a set of pairs, where each word has associated its frequency across all the documents.

2.2.3.2 Apache Hadoop

Apache Hadoop [42] is a collection of open source utilities that facilitate using distributed systems. The core of Apache Hadoop consists of a storage part, known as *Hadoop Distributed File System* (HDFS), and a processing part which is a MapReduce programming model. HDFS is a distributed file system that provides scalable, fault-tolerant, cost-efficient storage. By distributing the storage across multiple machines, the combine storage resource can grow horizontally. The MapReduce framework allows writing distributed applications that process large amounts of structured and unstructured data in HDFS.

Hadoop splits the input data into blocks and distributes them among the nodes in the cluster. Then, each of the nodes receives a packaged code which will be used to process the data in parallel. This approach uses the data locally since the nodes only process the data that they have access to. This allows processing data faster and more efficiently than it would be in message-passing systems or traditional super-computers where the data and computation are sent over the network.

Hadoop is built around an acyclic data flow model from stable storage to stable storage. This model has the advantages that it can decide in execution time were to run the tasks and can automatically recover from failures. However, it is not suitable for applications that reuse a set of data across multiple operations, as well as interactive data analysis tools. Therefore, Hadoop is considered a poor fit for low-latency applications and iterative computations, such as machine learning and graph algorithms.

2.2.3.3 Apache Spark

Apache Spark [120] is a distributed computing platform that has become one of the most powerful tools for the big data scenario. Spark was designed to overcome the limitations of Hadoop by generalizing the MapReduce computation model, while dramatically improving performance and ease of use.

The generalization of Spark comes from including a wide range of workloads that previously would have been covered by separate frameworks, including batch applications, iterative algorithms, interactive queries, and streaming. Spark allows to seamlessly combine different processing types while reducing the burden of maintaining separate tools. Moreover, Spark introduces a considerable performance improvement by offering the possibility to maintain data in memory across multiple computations, which has been shown to outperform Hadoop by up to 100x times [47]. The system is also more efficient for complex computations running on disk, which has also shown a performance improvement of up to 10x.

Spark contains multiple closely integrated libraries. Figure 2.8 presents the components of the ecosystem. The framework is built around *Spark Core*, which performs the scheduling, optimizations, data abstractions, as well as connection to the correct filesystem (HDFS, S3, RDBMs, or Elasticsearch). There are multiple libraries that operate on top of the Spark Core, such as *Spark SQL*, which allows handling data in a SQL manner, *MLlib* for machine learning, *GraphX* for graph computation and *Streaming* for processing of live streams of data.

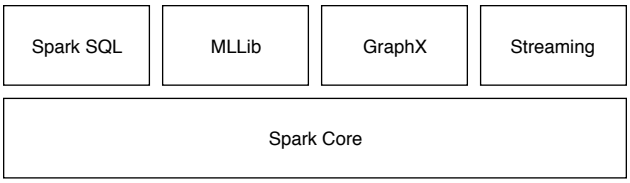


Fig. 2.8.: Apache Spark framework overview

Apache Spark architecture

Figure 2.9 presents the diagram of an application running in a cluster with two nodes. The architecture and workflow of Spark are built around the following concepts:

- *Driver*: Separate process to execute user applications. It creates the *SparkContext* to schedule jobs executions and negotiates with the *Cluster Manager*.
- *Cluster Manager*: An external service for acquiring resources on the cluster, e.g., standalone manager, Mesos, YARN.
- *Node* (or *Worker*): Any node that can run application code in the cluster.
- *Executors*: A process launched on a node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
- *Task*: A unit of work that will be sent to one executor.
- *Job*: A parallel computation consisting of multiple tasks that get spawned in response to a Spark action, e.g., save, collect.
- *Stage*: Each job gets divided into smaller sets called stages that depend on each other.

The steps an application follows on a cluster are: the application starts and instantiates the *SparkContext* to communicate with the *Master* in the *Cluster Manager*. Once connected, the application acquires Executors on nodes in the cluster. Next, it sends your application code to the executors. Finally, the *SparkContext* can request jobs to be run in the cluster, by splitting them into multiple tasks that will be distributed to the *executors*.

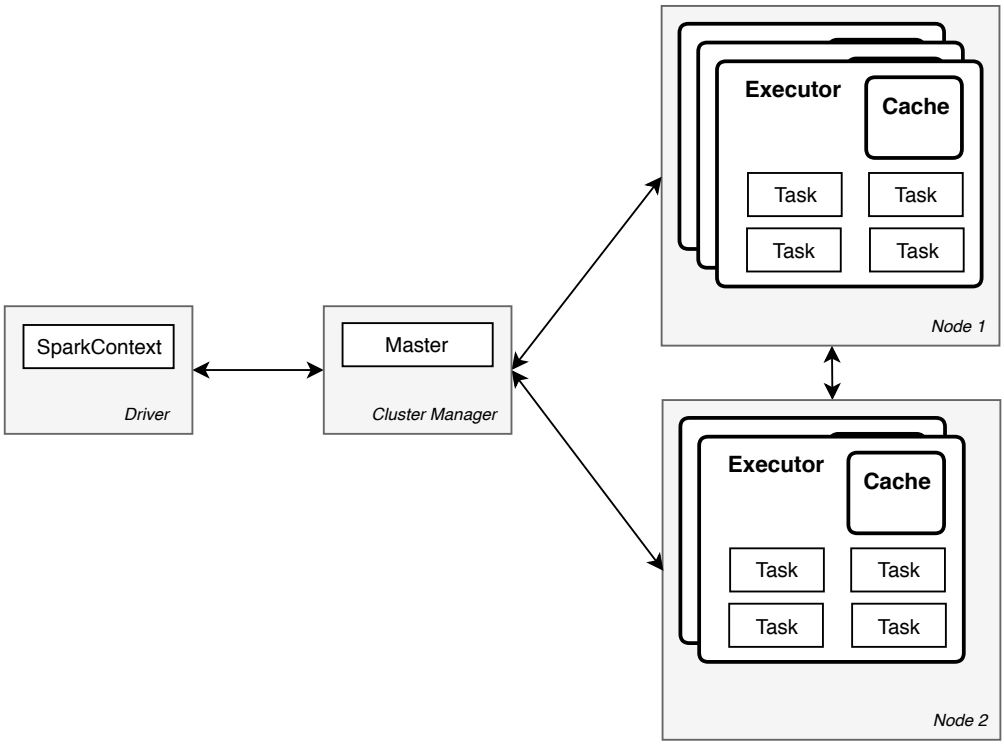


Fig. 2.9.: Apache Spark components diagram

Data abstraction

The main abstraction Spark provides is a *Resilient Distributed dataset* (RDD) [47], which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are initially created with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the *driver*, and convert it. RDDs can be persisted in memory, allowing to reuse them efficiently across parallel operations. Additionally, RDDs can recover automatically from node failures.

RDD defines a series of operations which can be categorized into *transformations* and *actions*. The transformations are lazy operations on an RDD that define a new RDD, while actions launch a computation that would execute the queued transformations and return a value or write the data to external storage.

RDDs do not need to be materialized at all times. Instead, an RDD keeps the information about the set of operations (lineage) that led to its current state. This allows them to

efficiently provide fault tolerance by logging the lineage rather than the actual data. In case of a partition is lost, or there is a failure, the RDD has enough information to recompute just that partition. Therefore, data can be recovered quickly, without the need for replication.

Apache Spark introduced another data abstraction which is built on top of RDD, known as *DataFrames*. A DataFrame is an immutable distributed collection of structured data. Unlike an RDD, data is organized into named columns, like a table in a relational database. DataFrames are able to optimize its query plan (Catalyst Optimizer) and efficiently serialize its object as well as generate compact bytecode (Project Tungsten) [121].

Execution workflow

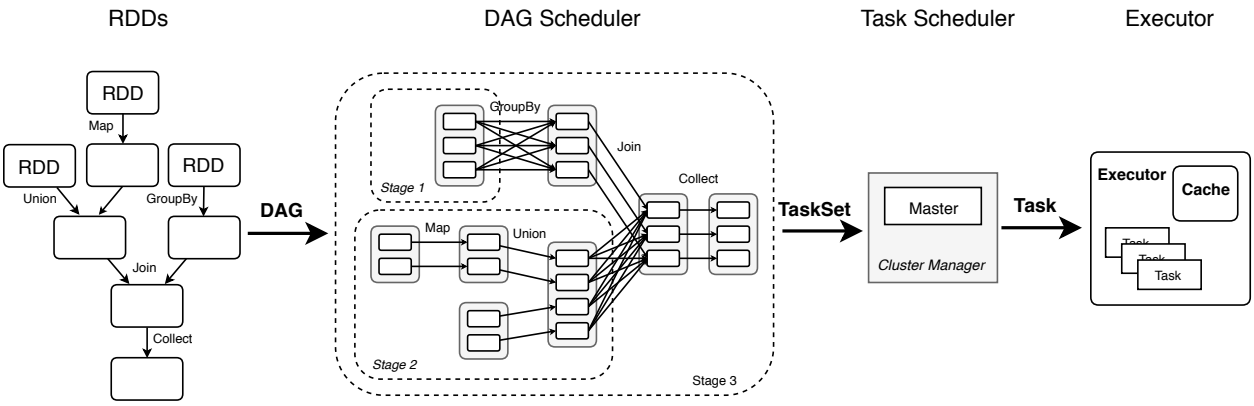


Fig. 2.10.: Apache Spark workflow

Figure 2.10 represents the workflow of a series of RDD transformations, forming a lineage, which is then executed by requesting an action. First, our application specifies a series of transformations using three RDDs this lineage is then executed when the *collect* action is called. Then, the *DAG Scheduler* (part of the SparkContext) creates a directed acyclic computation graph (DAG) describing the distributed operations in a coarse-grained way. Each of the stages is defined as a wide dependency, while pipelines inside the stage are formed by narrow dependencies (it combines tasks that do not require a shuffle operation). Next, the *Task Scheduler* communicates with the *Cluster Manager* to submit tasks that will be performed by the *executors*.

Machine learning applications

MLlib [122] is one of the main components of Spark and the largest machine learning library available for Spark. This library targets large-scale learning that benefits from parallelism to store and operate on data and models. MLlib consists of efficient and scalable implementations of standard learning algorithms and techniques including classification, regression, collaborative filtering, clustering, and dimensionality reduction. There is also an ongoing effort into adapting the machine learning methods to the Stream module [123], allowing to execute on dynamic datasets.

Additionally, Spark has been successfully applied by a number of researchers and industry experts to other machine learning techniques that are not covered by the official MLlib library. Some of the most relevant are:

Deep learning: Although MLlib supports Multilayer Perceptron classifier, which is a feedforward neural network, most of the features are still in the development stage. Therefore, a number of third-party frameworks have emerged [124], which aim to scale deep learning in a distributed environment. Here, we list the most relevant ones that are open source and public available to use: *DeepLearning4j*[5], *H20 Deep Water*[6] [125], *CaffeOnSpark*[7], *TensorFlow on Spark*[8], and *SparkNet*[9] [126] among others.

Natural language processing: The original MLlib library from Spark provides some feature extraction methods that can be used in natural language processing. However, there is not a dedicated part of the library for natural language processing. As a solution, developers can decide to combine Spark with a Java-based library such as *OpenNLP*, which is open source, or *Stanford NLP*, which requires licensing in order to use in a commercial

[5]https://deeplearning4j.org/

[6]https://www.h2o.ai/deep-water/

[7]https://github.com/yahoo/CaffeOnSpark

[8]https://github.com/yahoo/TensorFlowOnSpark

[9]https://github.com/amplab/SparkNet

product. Another option is to incorporate *spaCy*, a Python-based library that has become very popular for its trade-offs between analytical accuracy and performance. However, this requires to transform the data back and forth, which can have a considerable impact on the final performance. Finally, the most efficient and practical option is to use a library which extends the MLlib source code and can work seamlessly with Spark. *John Snow Labs' NLP*[10] extends the MLlib library to include techniques like a tokenizer, a lemmatizer, sentence boundary detection, and paragraph boundary detection among others.

Others: There are various machine learning methods that have been developed for Spark. Some of them are adaptations of widely known algorithms and others are novelty algorithms, some of the most popular are: a distributed Newton method for solving logistic regression as well linear SVM [127], a one-vs-one SVM for multi-label classification [128], an iterative k nearest neighbors classifier [129], a parallel genetic algorithm for pairwise test suite generation [130], an improved implementation of random forest [131], a simulated annealing method for solving unconstrained optimization problems [132], an extreme learning machine implementation [133], among others.

[10]https://nlp.johnsnowlabs.com/

CHAPTER 3

ARCHITECTURES FOR PARALLEL AND DISTRIBUTED MULTI-LABEL LEARNING

Many algorithms have been specifically designed to tackle multi-label problems, however, most of them have a considerable computational complexity. This issue can be addressed by introducing a parallel computation. However, there are many approaches to parallelize the computation in order to increase the scalability of the system, e.g., local parallelization using same memory space, distributed parallelization using independent memory spaces, or distributed parallelization using shared memory.

In order to study the different approaches, it is required to select a suitable problem that would evaluate equally each of the approaches. A popular method to address the additional complexity of predicting multiple labels is to use ensemble techniques. Ensemble techniques have become increasingly popular as they have demonstrated the ability to improve the results of individual classifiers [134]. Ensembles are built using a combination of base classifiers, thus suffering from high computational complexity.

Traditional implementations of multi-label learning methods focus on performing each task sequentially, even when there is not any kind of dependency between them. There are many scenarios in which a multi-label ensemble classifier needs to build each of their components sequentially, such as *Ensemble of Classifier Chains*. However, we will focus on the scenario where each of the components can be built independently since they do not incur into any data dependencies. A popular multi-label ensemble falling in this category would be *RAndom k-labELsets* (RAkEL), described in Section 2.1.2.1. RAkEL builds a multi-label

ensemble by splitting the label space $\mathcal{Y}$ into random subsets of k labels. The components of RAkEL are multi-label classifiers which target each of these random subsets.

RAkEL meets the requirements for this study: it is widely used and complex enough to seek parallelization. Additionally, it can be easily parallelized using multiple approaches, thus covering all the possible strategies that should be considered to propose a high-performance multi-label framework.

3.1 Proposed parallel and distributed architectures

1. Mulan

The first implementation is the original RAkEL from Mulan [104], which is built on top of Weka [105]. The *Mulan* implementation is used as a reference to study the performance of the traditional ensemble methods. This method only supports sequential single-threaded execution, hence it is limited by the resources of a single node. Therefore, it is not scalable to large datasets [135]. The data is read once and loaded into memory, but since the construction of the models is sequential, they do not compete for memory.

2. Mulan threading

This implementation parallelizes the construction of the ensemble components, e.g., multi-label classifier. Each of these classifiers works in a different label subset, therefore a thread can be assigned to each of them to work independently. It is expected to have a speedup bounded by the number of cores available in the machine being executed. All the threads execute locally, therefore they are able to share the available memory avoiding unnecessary copies of the data. However, this implementation is limited by the computational and memory resources available in a single machine. Building all the base classifiers in parallel will impact the total memory consumption, then being severely limited to small dataset sizes.

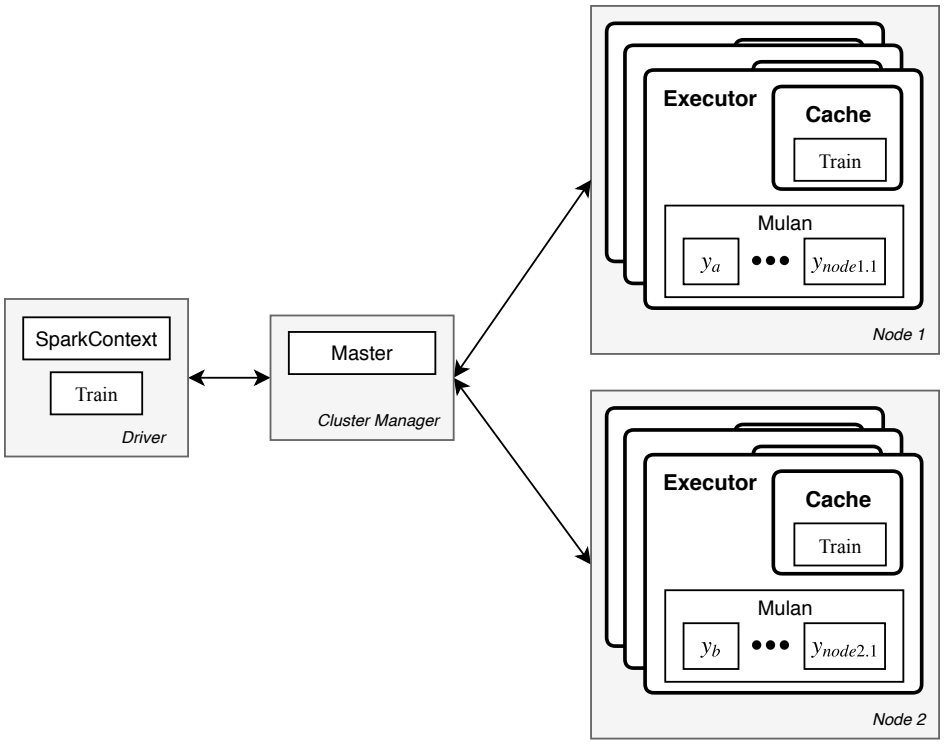


Fig. 3.1.: Mulan distributed implementation. Each node has multiple executors, where each of them will handle a subset of labels using Mulan

3. Mulan distributed

This method is based on wrapping the Mulan functionality and distribute the workload using Spark. This method uses the out of the box classifiers provided by Mulan. Therefore, it requires to serialize the Mulan library to deploy it in each of the *executors*. Figure 3.1 presents how the computation and data are distributed. The *driver* extracts the label information from the training dataset. Then each *executor* gets assigned a series of labels, thus using a local copy of the whole training dataset to build a Mulan model.

The principal advantage of this method is delegating the parallelism to Spark. Spark takes care of the transparency and fault-tolerance mechanisms, thus allocating the resources properly for parallelization. Also, this approach avoids any communication between *executors*. On the other hand, this requires each *executor* to read the full training data, which leads to multiple reads of the same data in the same machines.

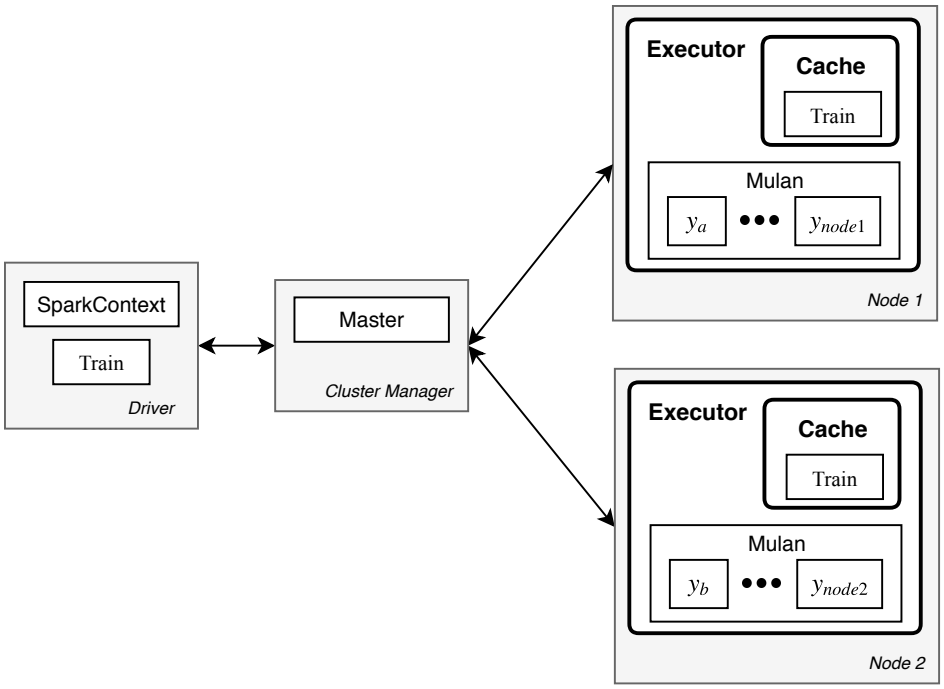


Fig. 3.2.: Mulan distributed threading implementation. Each node has a single *executor* that will handle a subset of labels using multiple threadings and Mulan

4. Mulan distributed threading

This implementation takes the previous approach one step forward and uses Spark only to control the distribution of the tasks, not the parallel execution of those. Figure 3.2 summarizes the execution of this method. As mentioned previously, the *driver* extracts the label information and distributes the workload among the *executors*. However, in this case, the *executors* declare a thread pool which handles the concurrency of the tasks. In order to avoid multiple unnecessary reads of the data, which would take unnecessary time and duplicated memory allocations, and having multiple *executors* competing for the shared resources, this method uses a single *executor* per node. Despite avoiding successfully the multiple reads of the data, the same data is still fully read in each node.

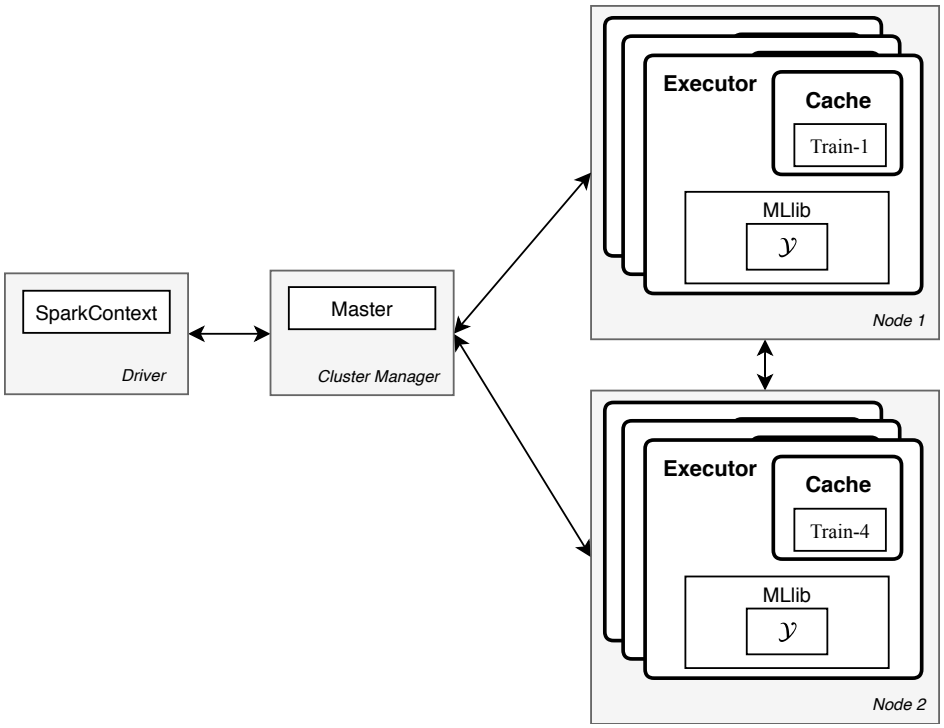


Fig. 3.3.: Spark implementation. The nodes have multiple executors, each of them with a local partition of the training data with the full label set to build collaboratively a model

5. Spark

The *Spark* native implementation is built on top of the native machine learning library *MLlib* [122]. Figure 3.3 presents a diagram with the implementation. This method reads the dataset which can be in a Distributed File System (e.g. HDFS), therefore the partitions are already allocated in the nodes. Then, each *executor* handles their local partition in order to perform intermediate tasks that will lead to the construction of the full model.

This method reads the data a single time, and if the partitions are correctly allocated in the system, no information needs to be initially sent over the network. Therefore, this approach fully shares the resources in the system allowing to handle large-scale datasets. Additionally, this approach performs intermediate tasks based on the partitions of the data, which are subsets of instances, thus allowing to use all the available cores simultaneously especially for datasets with a large number of instances.

Table 3.1.: Implementation summary

Implementation	Distributed	Redundant data reads	Shared memory	Scalability
Mulan	✗	✗	✗	✗
Mulan Threading	✗	✗	✓	Num. cores
Mulan Distributed	✓	✓	✗	Num. executors
Mulan Distributed Threading	✓	✓	✓	Num. cluster cores
Spark	✓	✗	✓	Num. cluster cores

Architectures summary

Table 3.1 summarizes the key aspects of the five approaches. It presents the following series of characteristics: distribution of the computation, multiple reads of the dataset, construction of various models in the same memory space, and the resources that influence the scalability.

3.2 ARFF data source for Apache Spark

The data source API at a high level is an API for turning data from various sources into Spark DataFrames and facilitates to manage the structured data in any format. Apache Spark has some built-in data sources such as Parquet, LibSVM, Parquet, JSON, JDBC, etc. Out of those, LibSVM is the only data source specifically designed for machine learning applications. However, this format is unable to specify information about the attributes, thus limiting its application to real-valued attributes and traditional learning paradigms.

Here we present a native data source to support the Attribute-Relation File Format (ARFF) on Apache Spark[1]. This data source infers additional information about the attributes and relationships among them, allowing to define new learning paradigms such as multi-label learning. The implementation extends seamlessly the Apache Spark source code, therefore the implementation uses the same syntax as with the official supported formats. Figure 3.4 shows a class diagram with the structure of the data source.

[1] `https://github.com/jorgeglezlopez/spark-arff-data-source/`

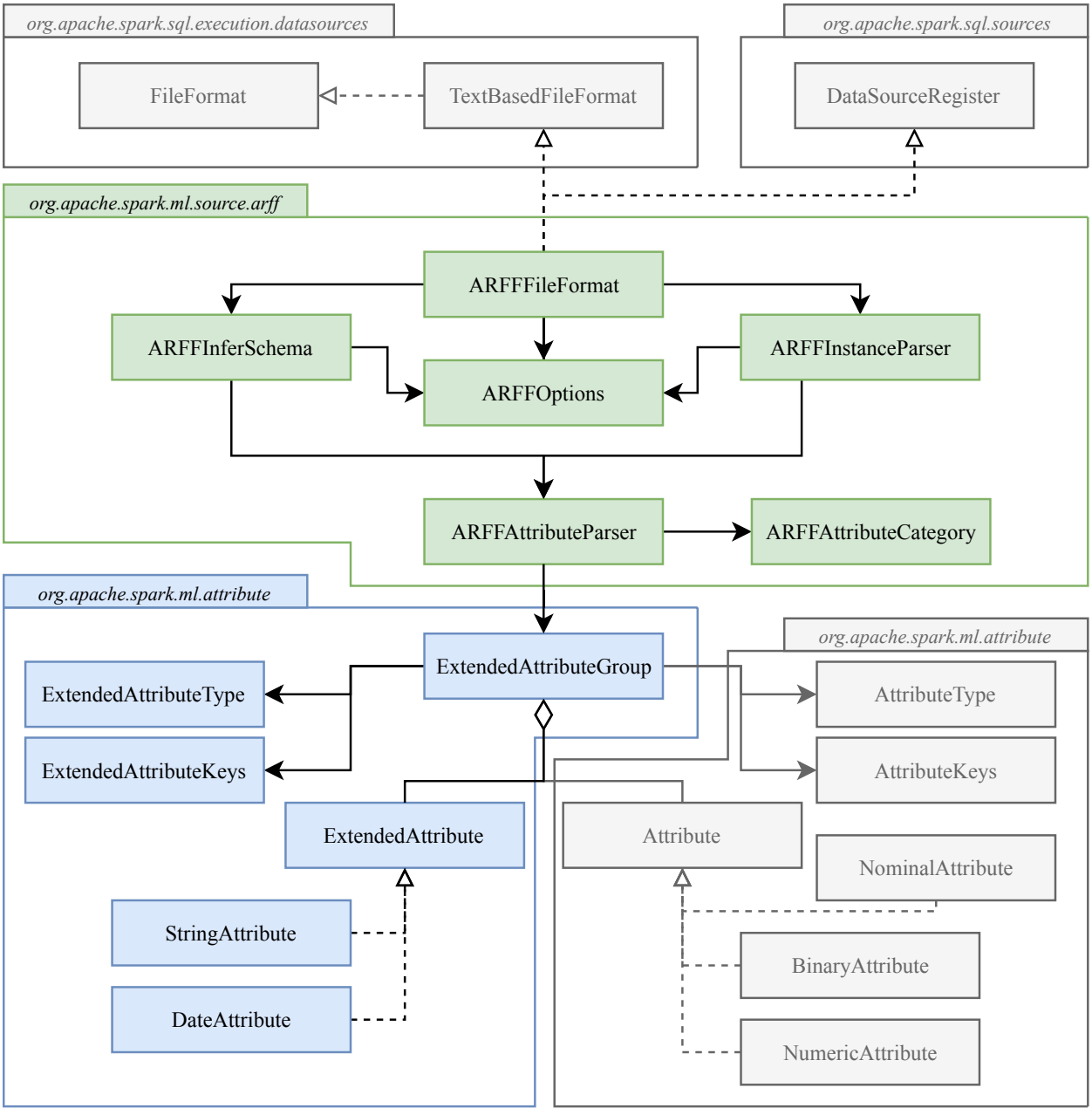


Fig. 3.4.: ARFF data source class diagram for Apache Spark.

- *ARFFFileFormat*: It is the entry point of the data source from the *DataFrameReader* interface. This class inherits from a series of interfaces in order to ensure the correct communication from the *DataFrameReader*. The first interface is the *DataSourceRegister* which can register the data source under an alias. The second set of interfaces is the *TextBasedFileFormat* and *FileFormat*, which define the methods that will be called from the *DataFrameReader* in order to create the proper DataFrame. The creation of the DataFrame is split into creating a suitable *schema* for the attributes and parsing all the instances. Both processes are isolated to each other because of the *FileFormat* interface.

- *ARFFInferSchema*: This class receives the header and the options defined by the user. The header comes either from the beginning of the file or from an independent file. The class uses the *ARFFAttributeParser* class to extract the information of each attribute using regular expressions. This information is used to create the required *schema* that matches the learning paradigm, as well as to store the information of the attributes in the *metadata*. Each column uses the *ARFFAttributeParser* and the *ExtendedAttributeGroup* to transform the corresponding information of the header into *metadata*. The *ExtendedAttributeGroup* adds support for new types of attributes, such as *String* and *Date*.

- *ARFFInstanceParser*: It parses each of the lines of data in the file into *Rows* for the DataFrame. It reconstructs the *ARFFAttributeParser* of each attribute from the *metadata* received in the *schema*. Once all the parsers have been constructed, it reads each line of data allowing to read both *dense* and *sparse* instances. In every instance, the original values are transformed into a *numeric* format and stored in the corresponding fields of a *Row* following the same order they present in the *header*.

- *ARFFOptions*: This class handles the options set by the user, which can only be set from there at the beginning and will be final during the execution of the data source.

This data source has a series of advantages over the *libSVM* data source, which is the data source used by the machine learning library. The main functionalities and advantages are:

- Support for different types of learning paradigms, each with a different *schema*.
- Automatic conversion of all the features to numeric types. It transforms types such as *Date*, *String*, or *Nominal* to a Double. This allows using the Dataframe directly with the machine learning methods.
- Storage of the information of each attribute in the metadata of the *schema*. This information can be used in different algorithms such as finding the best splits in decision trees over *nominal* data.
- Dynamic and automatic conversion to either *dense* or *sparse* instances, whichever uses less storage space.

This data source has been used with the *Spark* native implementation in order to correctly load the multi-label datasets in a distributed fashion. This data source ensures that the datasets are partitioned by instances and those partitions are assigned to the different executors.

3.3 Experimental setup

The experiments were executed in a cluster with 144 cores and 288 GB of memory. The system used Mulan 1.5 for the single-node classifiers and Spark 2.0.0 for the distributed computation. The method evaluated was RAkEL using BR as the base classifier which, in turn, used a decision tree whose maximum height is set to eight.

3.3.1 Datasets

Table 3.2 list the datasets, sorted by the number of instances, and their most important characteristics. More information about these datasets can be found in Section 2.1.4. These datasets were selected attending to the *label dimensionality* and the *number of instances*.

Table 3.2.: Multi-label datasets and their statistics

Dataset	Instances	Attributes	Labels	Cardinality	Density
Flags	194	26	7	3.39	0.48
Emotions	593	78	6	1.87	0.31
Birds	645	279	19	1.01	0.05
Genbase	662	1,213	27	1.25	0.05
Medical	978	1,494	45	1.25	0.03
Plant	978	452	12	1.08	0.09
Enron	1,702	1,054	53	3.38	0.06
Scene	2,407	300	6	1.07	0.18
Yeast	2,417	117	14	4.24	0.30
Human	3,106	454	14	1.19	0.08
Reuters-RCV1 (1)	6,000	601	101	2.88	0.03
Bibtex	7,395	659	159	2.40	0.02
Yahoo (Arts)	7,484	526	26	1.65	0.06
Yahoo (Health)	9,205	532	32	1.64	0.05
Yahoo (Business)	11,214	530	30	1.60	0.05
Yahoo (Social)	12,111	539	39	1.28	0.03
Yahoo (Entertainment)	12,730	521	21	1.41	0.07
Corel16k (1)	13,766	653	153	2.86	0.02
Yahoo (Society)	14,512	527	27	1.67	0.06
Delicious	16,105	1,483	983	19.02	0.02
20NG	19,300	1,026	20	1.03	0.05
EUR-lex (DC)	19,348	912	412	1.29	0.00
EUR-lex (EV)	19,348	4,493	3,993	5.31	0.00
EUR-lex (SM)	19,348	701	201	2.21	0.01
TMC2007	28,596	500	22	2.21	0.10
Mediamill	43,907	221	101	4.38	0.04
Bookmarks	87,856	708	208	2.03	0.01
IMDB	120,919	1,029	28	2.00	0.07
NUS-WIDE (128D cVLAD+)	269,648	129	81	1.87	0.02

The experiments were performed using 10-fold cross-validation in order to objectively evaluate the models' performances. The data is divided fairly into 10 equally sized folds where, at every iteration of the cross-validation evaluation, a fold is held out as the test instances, while the remainder of the data is used for train instances. These sets are stored and used by each algorithm, ensuring that the instances held in each of the fold are the same for all of them. This procedure ensures the model is not optimistically biased towards the full dataset and the algorithms are evaluated fairly over the same data in each fold.

3.4 Experimental results

This section presents the experimental study carried out to compare the performance impact of each implementation. The first part of the study compares the quality of the predictions, while the second part compares the execution performance.

3.4.1 Evaluation of predictions

In this experiment, we analyze the prediction results of the different implementations over all the datasets. To evaluate the predictions we use the multi-label classification metrics presented in Section 2.1.3 which show different perspectives of the same results.

Table 3.3 presents the averaged measures obtained for all the metrics. The measures are grouped attending to the type of averaging: example-based, micro-averaged, and macro-averaged. Additionally, it shows the results of the Wilcoxon rank sum test for subset accuracy, which allows us to identify whether there are significant differences in a pairwise comparison between two algorithms. A p-values < 0.01 indicates significant differences between the Spark method and the Mulan based methods.

Table 3.3.: Metrics averaged across all multi-label datasets and p-values comparison

Type	Metric	Mulan	Spark	p-value
Example-based	Hamming loss	**0.0699**	0.0670	1.04E-01
	Subset accuracy	0.1298	**0.2382**	1.70E-05
	Accuracy	0.2365	**0.3717**	1.49E-08
	Precision	0.2296	**0.4688**	1.87E-05
	Recall	0.2695	**0.4246**	1.49E-08
	F1	0.2746	**0.4221**	8.80E-06
	Specificity	**0.9654**	0.9597	8.86E-04
Micro-averaged	Precision	0.5417	**0.5721**	5.68E-02
	Recall	0.2447	**0.3919**	1.49E-08
	F1	0.3013	**0.4410**	1.49E-08
	Specificity	**0.9658**	0.9603	1.73E-03
Macro-averaged	Precision	0.2928	**0.3557**	2.09E-04
	Recall	0.2187	**0.2941**	1.35E-05
	F1	0.2263	**0.3001**	5.16E-05
	Specificity	0.9560	**0.9967**	1.20E-04

The p-values obtained by comparing the different measures support the hypothesis that there are statistical differences between the predictions produced Mulan-based methods and Spark. Therefore, we can conclude that each category of methods produces significantly different predictions.

The results highlight that Spark produces more competitive results than Mulan in terms of average values. These differences are introduced by the training process of the decision trees. Spark considers that any attribute with ten (or less) different values, is a nominal attribute and uses this information to find better split candidates, thus leading to a considerable improvement of the predictions. Additionally, Spark uses a sampled fraction of the data to find the best splits over continuous features. Finally, it is important to highlight that the results obtained by Spark are not influenced by the number of partitions used since this parameter only affects the parallelization level.

3.4.2 Evaluation of computational performance

In this experiment, we investigate how the parallelization and distribution of the computation affect execution times in the training process of the multi-label ensembles. Table 3.4 presents a comparison of the executions times and the resulting speedup using the full datasets. The left column shows the execution time of the original Mulan method as the reference. The right group of columns indicates the speedup of the proposed implementations with respect to the original Mulan.

The Mulan threading method outperforms the sequential version in every case, achieving a linear speedup of roughly four. This implementation has the best results for the smallest datasets. However, the execution times for larger datasets are still unacceptably long.

On the other hand, the performance of the distributed approaches for the Mulan-based and Spark-based implementations is significantly better for larger datasets. The distribution of the data, and hence the computation, comes with a small network overhead due to serialization, transfer, and synchronization. This overhead has a significant impact when the data size is small, and therefore it actually takes more time to distribute the data

Table 3.4.: Execution time (s) of Mulan and speedups of each proposed implementation

Dataset	Mulan - Execution time (s)	Speedup			
		Mulan thread.	Mulan distrib.	Mulan distrib. thread.	Spark
Flags	5.53E-01	**5.03**	0.12	0.09	0.05
Emotions	2.23E+00	**4.09**	0.29	0.25	0.16
Birds	9.83E+00	**4.14**	0.55	0.53	0.35
Genbase	3.94E+00	**2.44**	0.46	0.35	0.26
Medical	6.71E+01	4.1	**4.61**	4.01	1.66
Plant	9.04E+01	**4.27**	2.53	2.92	2.63
Enron	1.18E+03	3.9	5.15	5.17	**19.9**
Scene	3.47E+01	**4.07**	1.52	1.29	2.41
Yeast	3.64E+01	**3.84**	1.89	1.54	1.19
Human	4.40E+02	3.73	6.31	7.08	**8.38**
Reuters-RCV1 (1)	1.70E+03	4.83	16.67	**18.24**	13.47
Bibtex	4.24E+03	4.96	**38.97**	38.6	26.8
Yahoo (Arts)	7.48E+03	4.41	15.53	15.96	**100.5**
Yahoo (Health)	7.80E+03	4.47	15.05	15.13	**90.68**
Yahoo (Business)	1.22E+04	3.81	9.68	12.67	**112.37**
Yahoo (Social)	1.24E+04	3.83	16.87	21.87	**131.89**
Yahoo (Entertainment)	1.21E+04	3.65	10.92	13.89	**170.04**
Corel16k (1)	8.92E+03	3.71	13.3	13.11	**28.08**
Yahoo (Society)	2.90E+04	3.25	14.67	21.84	**369.42**
Delicious	1.95E+05	4.59	18.47	19.08	**100.91**
20NG	4.75E+04	4.05	15.19	31.96	**525.04**
EUR-lex (DC)	9.62E+04	4.06	70.46	75.17	**111.32**
EUR-lex (EV)	7.89E+05	4.03	**96.71**	54.88	48.93
EUR-lex (SM)	3.33E+04	1.97	81.46	77.78	**165.5**
TMC2007	2.66E+03	2.94	8.1	12.96	**28.83**
Mediamill	7.27E+03	3.43	32.72	34.23	**40.07**
Bookmarks	2.80E+05	4.32	74.04	76.17	**513.56**
IMDB	1.38E+06	3.73	24.07	27.7	**729.07**
NUS-WIDE (128D cVLAD+)	1.26E+05	4.12	28.47	30.11	**437.27**

than directly run the computation (achieving speedups smaller than one). However, this is compensated when the datasets are large enough in terms of instances, which is when the distribution of the data truly makes sense. Now, the bigger the dataset the better speedups achieved.

Interestingly, we noticed that the results for *Mulan distributed* and *Mulan distributed threading* are very similar, with a small advantage towards the threading version. The

difference between both methods is how the parallelism in each node of the distributed environment is handled. *Mulan distributed* delegates the parallelization to Spark and *Mulan distributed threading* creates the threads manually, avoiding multiple reads from the same data. The small difference indicates that the multiple reads from the data do not have a big impact on the performance, this is expected since the datasets take less than a few seconds of long execution times. However, this could change in a scenario of millions of instances. Additionally, bigger data takes more space in memory, which eventually can lead to running out of memory sooner in the *Mulan distributed* implementation. Furthermore, this small difference indicates that the overhead introduced by Spark to handle the parallelization is considerably small.

The Spark implementation outperforms the Mulan distributed approaches whenever the dataset has at least 7,000 instances (*arts*). This indicates that the overhead introduced to distribute the data among the workers and aggregate the results of the different tasks is only recommended for the large datasets. Again, this implementation achieves the best results for datasets with a large number of instances and/or labels, reducing the time to train the ensembles hundreds of times with respect to the original implementation.

Another important aspect is to consider the evolution of the execution time with regards to the size of the data. Figure 3.5 presents the speedup of the proposed implementations together for all the datasets sorted by increasing the number of instances.

First, the scalability of the Mulan threading implementation is linear, achieving speedup values limited to the number of cores available in a single node, which is relatively small. Second, the speedup of the distributed implementations scales better the more instances in the dataset. Third, the increase in the number of labels also affects the scalability of the models. Mulan distributed implementations use a single core on the distributed environment to train a given decision tree, which means that when there are more labels than cores in the cluster the behavior will be similar to the Mulan threading implementation. Hence, they are limited by the number of cores in the cluster.

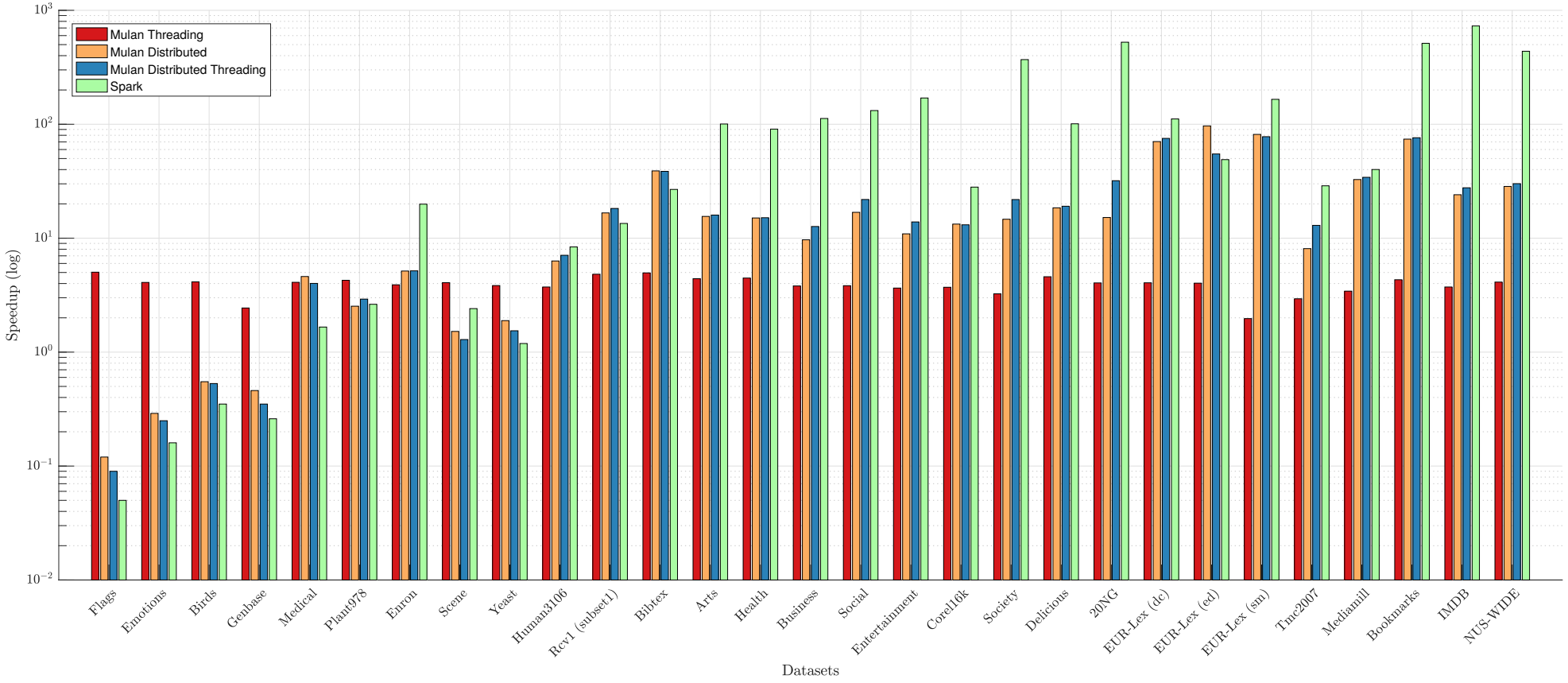


Fig. 3.5.: Speedup comparison on each proposed implementation

On the other hand, Spark achieves better speedup as soon as the data size grows enough to justify the distribution. Spark distributes the partitioned data and uses all the available cores to train for the partitioned data. This approach is more efficient and allows to handle larger datasets since the limit is set by the memory available, allowing eventually to execute using data from secondary storage. This sets a limit considerably larger than the ones established by the other implementations.

3.5 Conclusions

In this chapter, we have presented and evaluated five alternatives on the scalability of multi-label ensemble classification. RAkEL was selected as a reference classifier to evaluate the benefits of distributing the construction of the components of an ensemble. A series of parallel and distributed approaches were proposed and compared against their equivalent method in the Mulan framework.

The experimental study evaluated and compared the performance of the models with regards to the quality of the predictions and the execution times considering the data size as measured by the number of instances and labels. The results evaluating the quality of the predictions indicate that there are statistical differences between the predictions produced by the methods based on Mulan and Spark, having Spark produced better results. Regarding the scalability and overall performance, the distributed approaches significantly outperform the single-node version. The native Spark implementation that used the distributed construction of classifiers proved to be the most scalable, maximizing the use of all the available resources in the cluster, especially for large datasets. Spark performed hundreds of times faster than the Mulan implementations.

These results enhance the value of Spark as a solid framework to distribute the workload of large computational tasks, such as multi-label learning and present an open challenge demanding further research.

CHAPTER 4

DISTRIBUTED MULTI-LABEL K NEAREST NEIGHBORS

Multi-label instances have the information of all the labels, regardless of the relevance. Our research focus on this phenomenon, therefore, it is mandatory to define a way to evaluate it. The level of relevant information will be measured by looking at the performance improvement of a classifier that is greatly affected by the inclusion of irrelevant data.

There are two candidates that would be suitable to evaluate this problem: MLNB and Ml-knn. MLNB assumes that the features are independent given the set of labels. Therefore, the performance decreases with the number of features that are irrelevant to the labels. On the other hand, Ml-knn assigns the class according to a similarity metric, which in most of the cases is the Euclidean distance. This distance is linear respect the number of features, therefore its performance decreases by adding misleading features.

We decided to use ML-knn because it is computationally expensive since for each unseen instance it needs to find the nearest neighbors among the training instances. Therefore, it is a good candidate for distributed computing. Moreover, this algorithm is particularly challenging in a distributed environment since the data is spread among multiple nodes, and the communication between them is considered computationally slow.

We propose to address this issue and to adapt ML-knn to our distributed multi-label learning architecture on Spark. This algorithm will be used in future approaches, and it is expected to provide a scalable version of ML-knn that can handle large datasets.

4.1 Nearest Neighbors background

Nearest neighbors (NN) is a classic non-parametric and instance-based technique that has attracted the attention of the research community due to its simplicity and effectiveness. This technique has a wide range of applications such as density estimation [136], dimensional hashing [137], pattern recognition [138], data compression [139], and so on.

The nearest neighbors search is an optimization problem, whose goal is to find an instance that minimizes a certain distance or similarity function [140], [141]. The most popular distance used is the Euclidean distance, which measures the straight-line distance between two points in Euclidean space. A straightforward generalization of this problem is the k nearest neighbors search (KNN) which finds the k instances minimizing the distance.

Whenever the search method performs a pair-wise comparison using all the instances, it is called exact nearest neighbors search. This approach finds the exact nearest neighbors at high computational cost, however, there are many techniques that reduce this complexity by indexing the feature space.

4.1.0.1 Tree indexes

One of the first methods developed was the KD-*tree* [142]. This structure recursively subdivides the feature space by a hyperplane that is orthogonal to one of the axes and that partitions the data points as evenly as possible.

In [143] they propose to use simultaneously multiple KD-*tree* to increase the performance of nearest neighbors searches. They use rotations of the dataset that would force the tree to use features that otherwise would be discarded. They show that by rotating the dataset to align it with its principal axis direction using PCA, and then applying random Householder transformations that preserve the PCA subspace of appropriate dimension, the KD-*tree* performance can be significantly improved.

One of the weaknesses of KD-*tree* is that the indexed space can have a high aspect ratio, which makes it impossible to use volume bounds. Arya et al. [144] introduced a *Balanced*

Box-Decomposition tree (BBD-*tree*) which guarantees both a balanced aspect ratio and a logarithmic depth. This modification allows performing *error bound* approximate search by considering $(1+\epsilon)$-approximate nearest neighbors. This concept was also adapted by Duncan et al. [145] using the *Balanced Aspect Ratio tree* (BAR-*tree*), which was later extended to higher dimensions [146]. This tree does not exclusively use axis-orthogonal hyperplane cuts, which leads to good aspect ratio, balanced depth, and convex regions. Other variations of the KD-*tree* are: the PCA-*tree*[147], the RP-*tree*[148], and the trinary projection tree[149].

Another weakness of KD-*tree* is related to the curse of dimensionality. KD-*tree* is very effective at low dimensions: after traveling down the nodes of the tree, all the instances in one leaf tend to be much closer to each other than to instances in other leaves. However, this property disappears in high dimensional spaces. This has been solved in a later method called *Metric tree* [150] which subdivides the feature space by a hyperplane defined by the midpoint of two instances (pivots). This partitioning creates two disjoint sets with no information shared between them. The search process goes through the tree choosing the nearest node in each level, allowing to "backtrack" in case that some branches have remained unpruned. *Spill tree* [151] modifies the Metric tree avoiding the tedious backtracking process by allowing an overlap area between the nodes. This overlapped buffer allows that the same instance can be indexed by both pivots, which leads to an increased accuracy at the cost of redundancy.

In order to combine the advantages of both, Metric tree and Spill tree, [151] proposes a combination called *Hybrid tree*. This structure allows having both types of nodes, where a decision is made in each node whether to use an overlapping node or non-overlap node. In the search process, it only does backtracking for non-overlap nodes (as in conventional Metric tree), and defeatist search in overlapping nodes (Spill tree).

Some other approaches, which are based on completely different concepts, have been proposed. For example, [152] presents the product quantification approach in which they decompose the space into low dimensional subspaces and represent the instances by compact codes computed as the quantification indices in these subspaces. These compact codes can

be compared to the query points using an asymmetric approximate distance. A modification of the standard quantification process was introduced by [153] in which they use an inverted index with product quantification that produces denser subdivision of the search space.

4.1.0.2 Hashing indexes

The best-known hashing based nearest neighbors technique is *Locality Sensitive Hashing (LSH)* [154]. An LSH function maps the instances in the feature space to a space of reduced dimensionality in a way that similar instances map to the same hash entries. Then, a similarity search query can be answered by first hashing the query instance and then finding the close instances within the instances that have been mapped to the same entry. To guarantee both, good search quality and good search efficiency, one needs to use multiple LSH tables and combine their results. Unfortunately, LSH requires a large number of hash tables [155], [156]. There are some variants of LSH such as multi-probe LSH [157] in which the number of hash tables is reduced by searching other entries in the hash tables within a certain distance, and LSH Forest [158] in which they remove the data-dependent parameters achieving better adaptation for skewed data distributions.

The performance of LSH methods is highly dependent on the hash function. There is a large amount of research aimed at improving hashing methods by using data-dependent hash functions using various techniques: parameter sensitive hashing [159], spectral hashing [160], randomized LSH from learned metrics [161], kernelized LSH [162], learned binary embedding [162], shift-invariant kernel hashing [163], semi-supervised hashing [164], optimized kernel hashing [165], and complementary hashing [166].

4.1.0.3 Graph indexes

Nearest neighbors graph methods build a graph structure in which vertices represent the instances and edges connect nearest neighbors. There are two critical components in these methods: query strategy and graph construction.

There are multiple approaches that aim to minimize the impact of consulting the nearest neighbors graph. In [167] the authors consider to use a sample of well-separated instances as seeds and start the graph exploration using a best-first strategy. Similarly, [168] incorporate a hill-climbing strategy and pick the starting points at random.

The graph construction is the target of substantial research, however, these methods do not scale, or are specific to certain similarity measures. Paredes et al. [169] proposed two methods for graph construction using general metric spaces and low empirical complexity. However, both methods require a global data structure and are difficult to parallelize across machines. Chen et al. [141] propose to use divide and conquer methods to recursive data partitioning. In [170] authors presented a graph construction technique using Morton ordering and based on space-filling curves. Dong et al. [171] present a graph construction method based on local search. They consider that a neighbor of a neighbor is likely to be a neighbor too. Therefore, by initializing each vertex with a random set of neighbors the method iteratively improves the neighbors of each node.

Although there have been some methods which focused on efficiently building the nearest neighbors graph before, it is considered that they are not efficient enough in a distributed environment. Consequently, these techniques are not included in this study.

4.2 Distributed ML-KNN

We propose a Ml-knn implementation on Spark, which will focus on scaling the algorithm in a distributed environment. The distribution of the computation can be achieved in both, the *train* and *test* phases. The foundations of Ml-knn were described in Section 2.1.2.1.

Firstly, the *train* phase computes the statistical information of the training instances by finding the prior and posterior probabilities. The prior probabilities can be found by frequency counting of the labels. The posterior probabilities are more complex and need of nearest neighbors to gather statistical information.

Secondly, the *test* phase uses the previously computed prior and posterior probabilities. For each of the test instances, the probabilities are combined with the information gathered by the nearest neighbors on the training instances to produce a probability for each of the labels.

As can be seen, ML-KNN is a complex algorithm whose performance is limited by the two nearest neighbors searches. The first one between all the training instances, and the second one between the test and training instances. This introduces an increased complexity with respect to lazy methods in traditional classification problems. Additionally, some aspects such as broadcasting values or persisting the right data in-memory can have a large impact on the final performance. This section provides a detailed explanation of the implementation that maximizes the resources of a distributed environment.

4.2.1 Train phase: computing prior and posterior probabilities

The *train* phase is divided into computing the prior and posterior probabilities. The first to perform a frequency count of the labels, and the second to perform a frequency count subject to the neighbors.

Prior probabilities are defined as $P(H_j)$ and $P(\neg H_j)$ and represent the probabilities of a label y_j being found in the dataset before the arrival of new information. Figure 4.1 presents the process to compute the probabilities in a distributed manner.

A user-defined *reduce* is applied to the collection of labels of all the training instances. This operation adds the binary label vector Y to a vector $\mu_j = \sum_{i=1}^{m}[[y_j \in Y_i]]$, which counts the occurrence of each label. Next, the prior probabilities are computed averaging and smoothing the count vector $P(H_j) = (s + \mu_j)/(s * 2 + m)$. Additionally, we can compute $P(\neg H_j) = 1 - P(H_j)$ by finding the opposite probability.

Posterior probabilities are defined as $P(C_j|H_j)$ and $P(C_j|\neg H_j)$, it represents the probabilities of a label being present in exactly j instances among its k nearest neighbors, conditioned to the event of having the label present. Figure 4.2 presents the steps to compute the probabilities in a distributed manner.

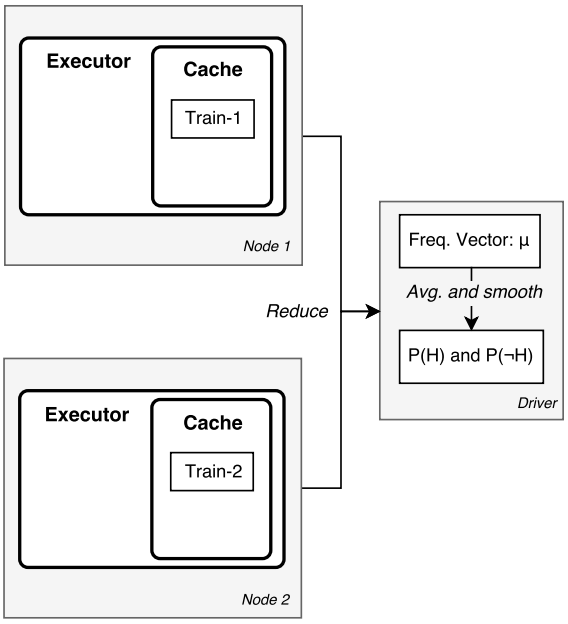


Fig. 4.1.: ML-KNN *train* phase: Computation of the prior probabilities $P(H)$ and $P(\neg H)$. Frequency count of the labels followed by averaging and smoothing the values

First, it finds the k nearest neighbors for every training instance, followed by a user-defined *map* applied to each instance. This operation computes the counts of each label among its neighbors C_j. Next, it creates two frequency matrices $\mathcal{K}_{rj} = \kappa_j[r]$ and $\tilde{\mathcal{K}}_{rj} = \tilde{\kappa}_j[r]$ for each instance $\boldsymbol{x}$. Each position (r, j) is initialized to 0, and it is updated with $\mathcal{K}(C_j, j) = 1$ for $y_j \in Y_i$ or $\tilde{\mathcal{K}}(C_j, j) = 1$ for $y_j \notin Y_i$.

Second, a user-defined *reduce* is applied to the collection of $\mathcal{K}$ and $\tilde{\mathcal{K}}$ adding the matrices of all the training instances. The final result stores the number of training instances that have exactly r neighbors with j-th label, both for the case $y_j \in Y_i$ and $y_j \notin Y_i$.

Finally, the posterior probabilities are computed averaging and smoothing the frequencies:

$P(C_j|H_j) = (s + \mathcal{K}_{C_j,j})/(s \times (k+1) \sum_{r=0}^{k} \mathcal{K}_{r,j})$

$P(C_j|\neg H_j) = (s + \tilde{\mathcal{K}}_{C_j,j})/(s \times (k+1) \sum_{r=0}^{k} \tilde{\mathcal{K}}_{r,j})$

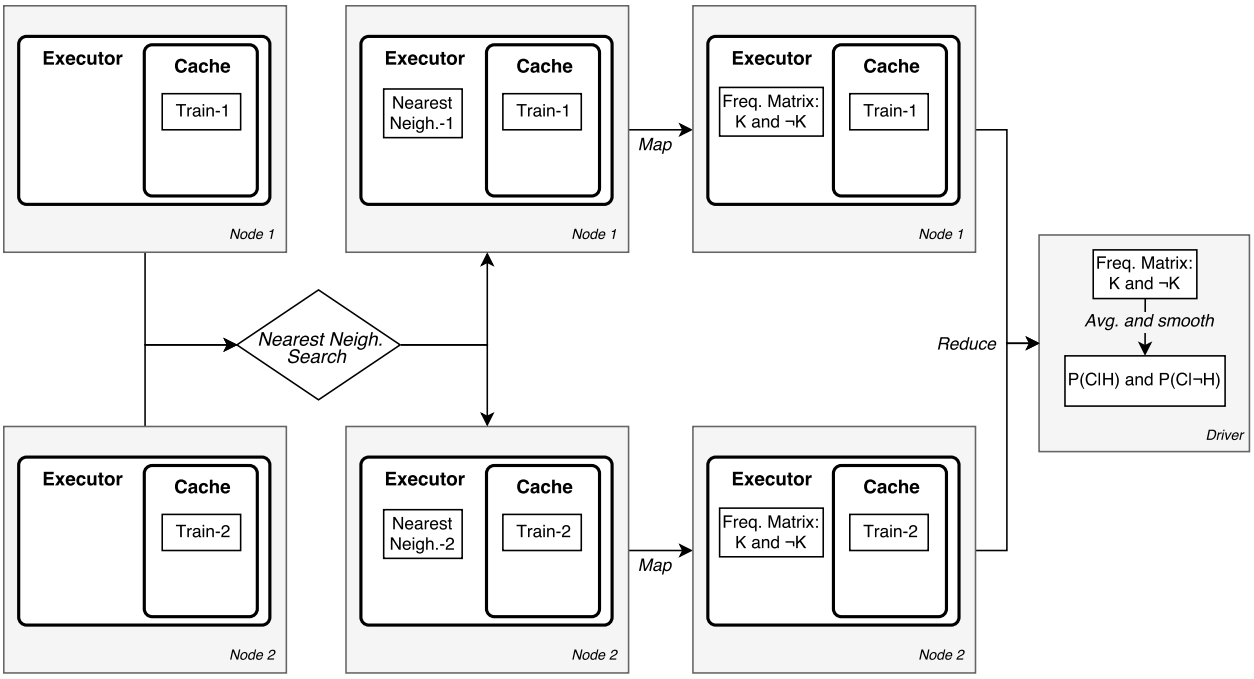


Fig. 4.2.: ML-KNN *train* phase: Computation of Posterior probabilities $P(C|H)$ and $P(C|\neg H)$. A frequency count of the labels among each of the neighbors is performed per instance, followed by averaging and smoothing the values

4.2.2 Test phase: prediction of label set

The *test* phase inducts the predicted label set for new unlabeled instances. This set relies on the prior and posterior probabilities, previously computed in the *train* phase. Figure 4.3 shows the predictions of the test phase labels.

First, it finds the k nearest neighbors for every test instance among the training instances. Next, a user-defined *map* is performed on each test instance to compute the counts of each label C_j among its neighbors. Then, each test instance finds the corresponding value in the posterior probabilities and assigns each label by determining for each y_j whether $P(H_j) \times P(C_j|H_j)$ is greater than $P(\neg H_j) \times P(C_j|\neg H_j)$.

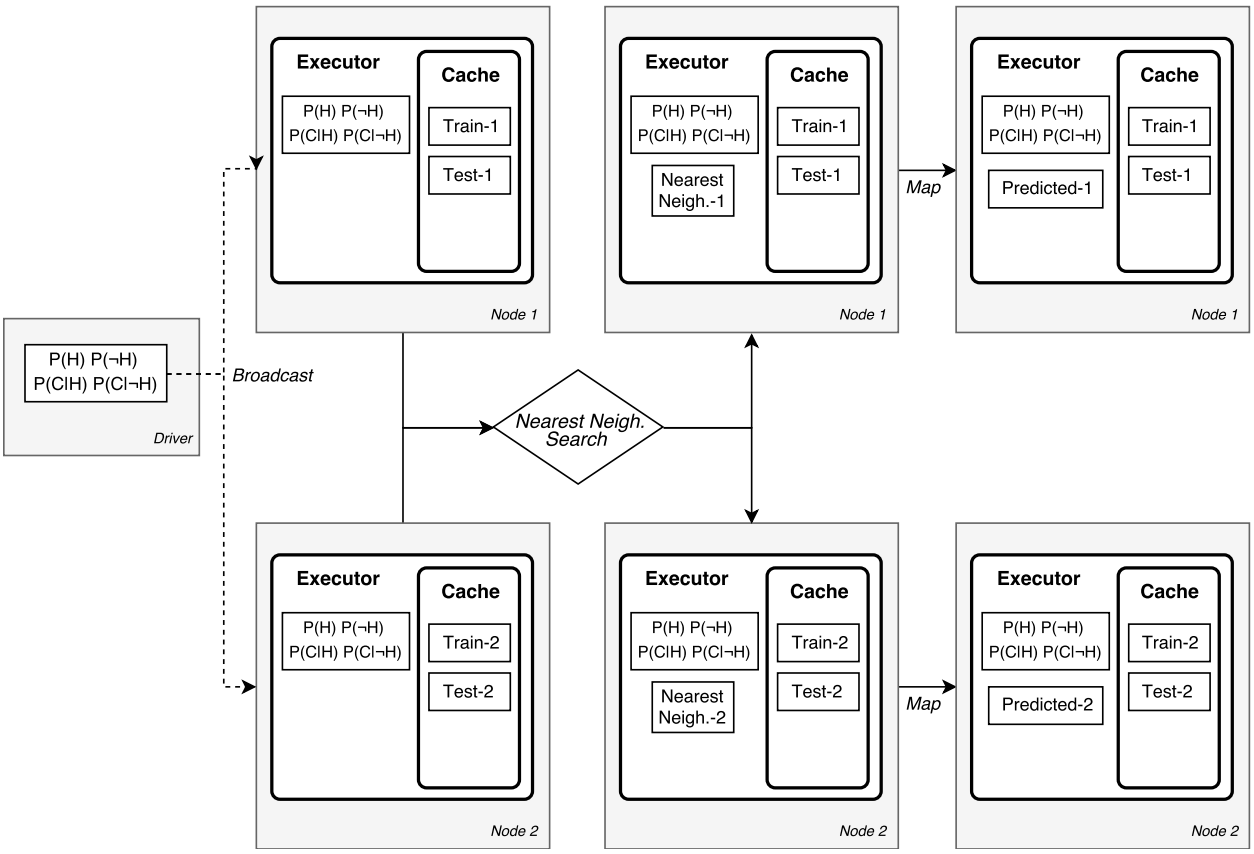


Fig. 4.3.: Ml-knn *test* phase. First, the prior and posterior probabilities are broadcasted. Then, each label set is predicted by combining the probabilities with the information collected by the nearest neighbors

4.3 Distributed Nearest Neighbors methods

The Ml-knn performance is going to be bound by the k nearest neighbors search, both in the *train* and *test* phases. The main issue of distributing the search process over a cluster of nodes is that every time a node needs to access the information of another node it triggers a *shuffle* operation, which sends information over the network and is considered to be a slow process. Although the most naive strategy would be to use a *cross product* and exchange the information of all the nodes with each other, in practice this is unfeasible because of memory and network limitations.

There are multiple strategies that aim to minimize this impact, from distributed index structures to hashing matching. We propose three versions of ML-KNN, to study their impact on the overall performance of the algorithm. Each implementation represents one of the strategies studied in Section 4.1, namely ML-KNN-IT, ML-KNN-HT, and ML-KNN-LSH.

4.3.1 Iterative Multi-label k Nearest Neighbors (ML-KNN-IT)

Our first approach was to incorporate an iterative version of ML-KNN in Spark based on the principles presented by Maillo et. al [129], [172], where they adapted the *brute force* algorithm to a distributed environment. Despite being a naive approach, in which no index structure had been used, it showed promising results. However, they only compared to another algorithm that has been previously developed by themselves, hence it is difficult to appreciate the real performance of the algorithm. Additionally, their original implementation[1] suffers from some limitations, such as an excessive number of parameters, it is limited by an outdated version of Spark (not taking advantage of new functionality introduced in Spark 2.0+), it inefficiently iterates over the test instances on the *driver* by assigning a *partition id* and sorting the instances (triggering a shuffle operation), do not maintain the *row* structure which discard any extra information on the instances, among others.

We modified the original method, solving the previously mentioned issues and adding support to keep the label information. Our implementation performs an exact nearest neighbors search by iteratively broadcasting a buffer of test instances, instead of broadcasting full partitions of test data. However, both methods are comparable and if the buffer size is set to the partition size, they would be equivalent. The combination of this search method and our proposed ML-KNN is named ML-KNN-IT. Figure 4.4 shows the functionality of the *test* phase since this method does not require of any training. The diagram shows one iteration of the method, thus finding the nearest neighbors of the test instances stored in the buffer.

[1] KNN-IS: `https://github.com/JMailloH/kNN_IS`

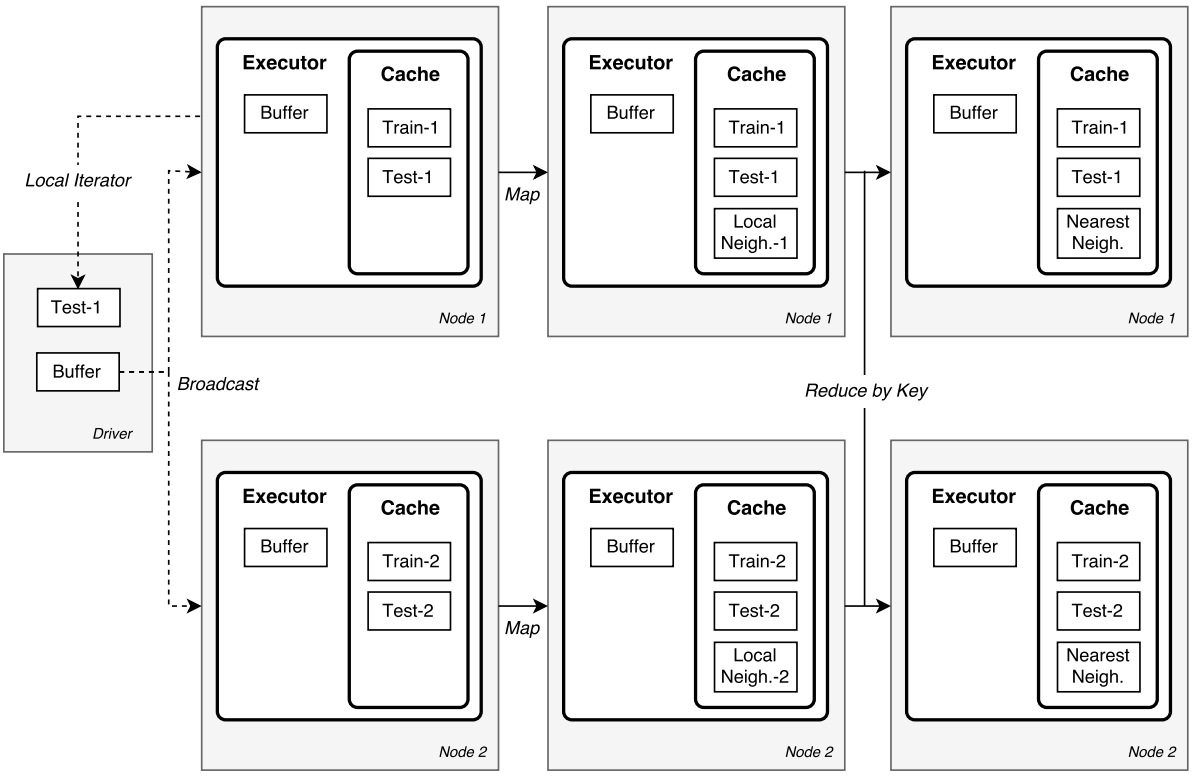


Fig. 4.4.: *Test* phase for ML-KNN-IT. In each iteration a buffer of test instances is broadcasted, which will be used to find the nearest neighbors among the training instances

First, it uses a *local iterator* of the test instances, which brings a partition at a time to the *driver* avoiding to overload the memory. This iterator allows iterating locally over the test instances while avoiding to filter the test instances by a *partition id* and collecting the results. Next, a buffer of fixed size is filled with the local instances and broadcasted to all the nodes. After that, a *map* operation over the train instances will find the k nearest neighbors of the broadcasted instances within the local partition. Finally, a *reduce by key* operation will combine all the partitions keeping the top k nearest neighbors of the buffer instances. The data should be in-memory for two reasons: it will be accessed multiple times, and this avoids undoing the transformations to restore the original state.

The main advantages of this method are that it performs an exact search and does not require any training. Furthermore, the *reduce by key* operation is more efficient than other operations which require a *shuffle* of data such as *join*.

On the other hand, this method suffers from the same problem than the traditional implementation: pair-wise distance computation. Therefore, each instance will be broadcasted to all the nodes once, no matter the size of the buffer or the number of iterations. Additionally, the result is created by combining the train partitions and the broadcasted test instances, hence it is not modifying the original test data. Instead, it is creating new test data with the neighbors in it. Consequently, it will iteratively duplicate the test data, until the *test* phase is over, and then the original test data can be discarded.

4.3.2 Hybrid Tree Multi-label k Nearest Neighbors (ML-KNN-HT)

This method was presented in [173] and aims to use a tree-based index structure to achieve high accuracy and search efficiency in a distributed environment, also there is a public implementation available[2]. The available implementation works seamlessly with the new versions of Spark, as well as supporting the specification of the columns to be preserved in the neighbors. Therefore, it can be easily incorporated into our ML-KNN algorithm, and it is named ML-KNN-HT.

This algorithm uses two structures: a top tree (*metric tree*) and multiple subtrees (*spill trees*) on the nodes, hence the combination of trees is named *hybrid tree*. This method requires a *train* phase, where all the trees are built, and a *test* phase, in which we can query the trees to find nearest neighbors. Figure 4.5 presents the process to build the trees.

First, it uses a randomized sample of the training instances to build the top tree (*metric tree*), whose leaf nodes correspond to specific partitions of the data. A copy of the top tree is broadcasted to all the nodes, so all the train instances can compute a value that identifies the index of the partition where they belong. Next, the training data is repartitioned by the index, hence it is sent to the partition indicated by the top tree. Then, each partition builds a subtree (*spill tree*) which will index the local training data. In the end, both types of trees (top tree and subtrees) are persisted in-memory since they can be consulted multiple times.

[2]KNN-HT: `https://github.com/saurfang/spark-knn`

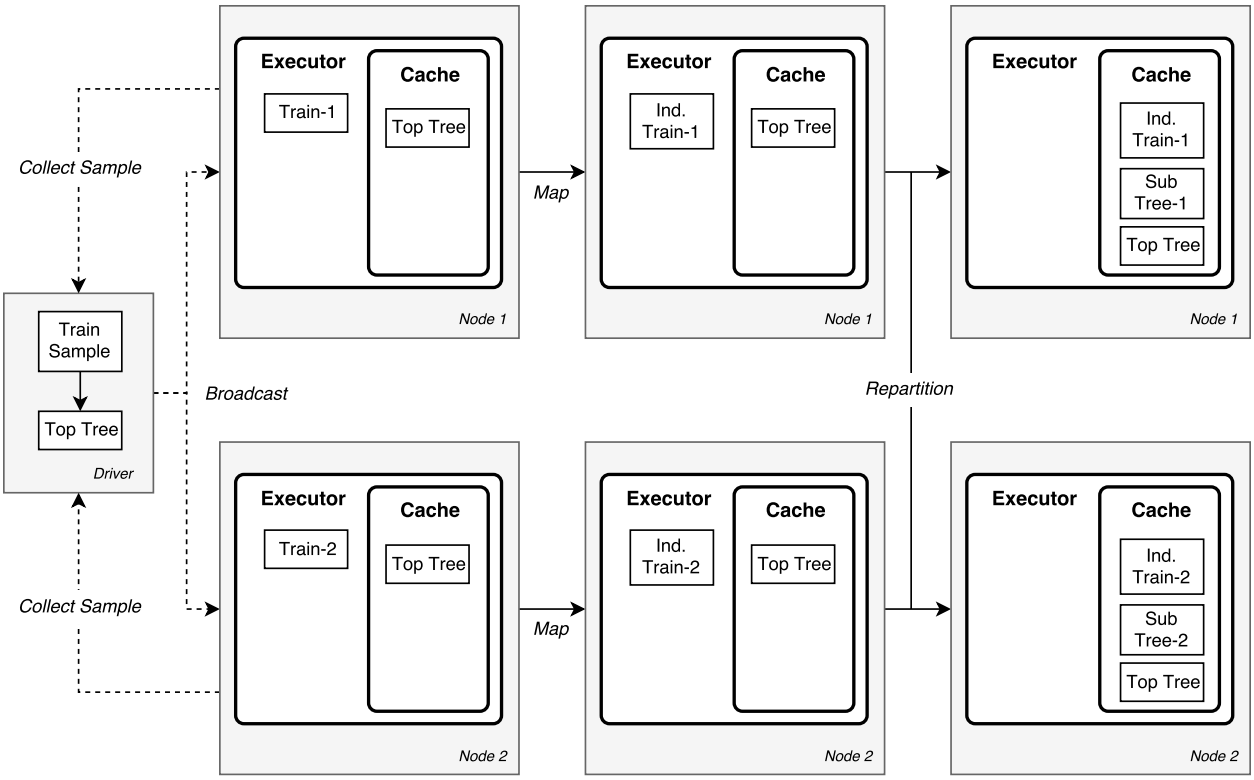


Fig. 4.5.: *Train* phase for ML-KNN-HT. First, a top tree (metric tree) is built locally on the *driver* using a sample of train instances. Next the instances are indexed and partitioned using the top tree. Then, each partition builds a local subtree (spill tree)

Once all the trees are computed and broadcasted, we can query the structure to find nearest neighbors among the training instances. Figure 4.6 shows the process to find nearest neighbors for the test instances. First, each partition is indexed using the top tree and the test instances are repartitioned by index. Next, each partition uses the local subtree to find the nearest neighbors among the training instances. Then, the distance to the farthest neighbors is used to evaluate if it is necessary to search for other partitions.

One of the parameters in this method is the overlap buffer width for the *spill nodes*. This buffer needs to be large enough to always include the k nearest neighbors, but not so large that it impacts negatively on the overall performance. The details of this parameter estimation can be found in [173].

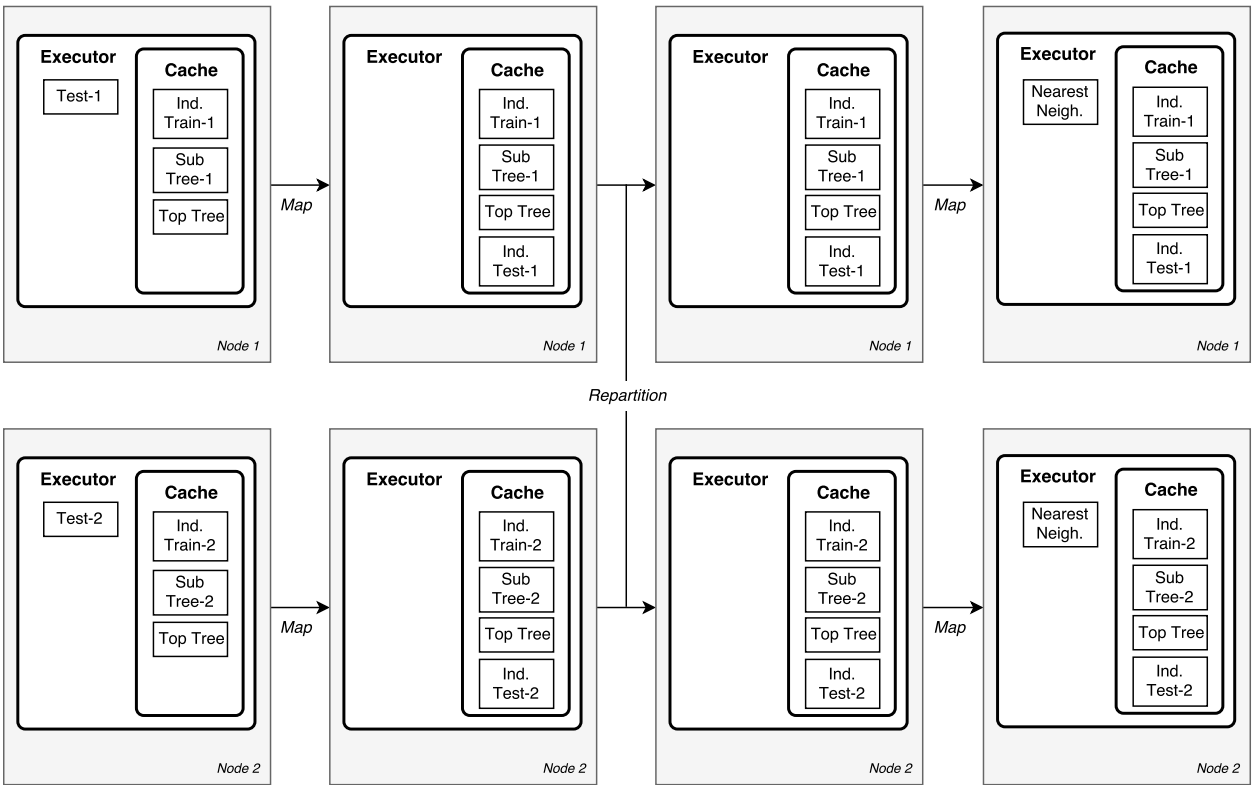


Fig. 4.6.: *Test* phase for ML-KNN-HT. First, the test instances are indexed and repartitioned using the top tree (metric tree). Then, each partition finds its nearest neighbors using their local subtree (spill tree)

The advantage of this algorithm is the speedup of the nearest neighbors searches using multiple index structures. First, by using the top tree to find the corresponding partition for each instance, and second by using the subtrees to find the nearest neighbors within each partition. Moreover, it only executes one *shuffle* operation to find the partition where each instance belongs.

However, these advantages come at the cost of building the indexes of training data. This cost is reflected both in computational time (find the splits) and memory (store the pivots). Moreover, to maximize the use of these structures it avoids using backtracking for both trees: in the top tree it might send duplicate test instances to several nodes instead, and in the subtrees, it uses an overlap buffer to consider instances near the decision boundary.

Additionally, the number of partitions used in this algorithm is the same as leaf nodes on the top tree. Therefore, the size of the partitions is decided by the splits on the tree, leading to unbalanced workloads. The only way to minimize this impact is to have more partitions that nodes since this ensures at least the full utilization of the cluster.

4.3.3 Locally Sensitive Hashing Multi-label k Nearest Neighbors (ML-KNN-LSH)

This method focuses on the application of *locally sensitive hashing* (LSH) functions that preserve the similarity of the original feature space. These functions map, with high probability, similar instances to the same hash entries. This method uses several hash tables to increase the probability of collision for similar instances.

The most popular LSH functions are: *MinHash* which finds the similarity between two sets defined by the ratio of the number of elements of their intersection and the number of elements of their union. *Bucketed Random Projection* that projects the feature vectors onto a random unit vector and portions the projected result into buckets. *Sign Random Projection* which creates a bit vector with the signs of the projection of the feature vector onto multiple random unit vectors.

In the nearest neighbors search problem, the data should be normalized to assign equal weight to all the features regardless of the scale. Our data is normalized in the range $[0, 1]$, consequently *MinHash* and *Bucketed Random Projection* cannot be applied here. For this reason, *Sign Random Projection* was the function selected.

The official machine learning library for Spark [122] (v.2.1.0) only offers an implementation of *MinHash* and *Bucketed Random Projection*. Therefore, the *Sign Random Projection* function was implemented by following the structure of the other functions. Furthermore, the original LSH method required to use a combination of *join* and *group by* operations to find the nearest neighbors of each instance, however, we found out that this produced performance issues that could be minimized by using the *co-group* operation instead.

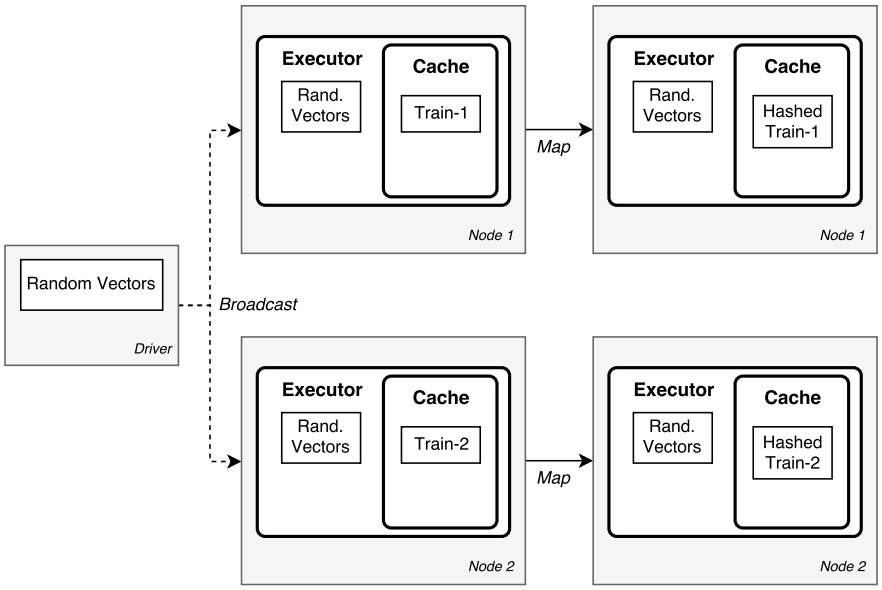


Fig. 4.7.: *Train* phase for ML-KNN-LSH. A set of random vectors is created and broadcasted. Then, the training instances will compute their sign projection using those vectors to find their keys for each of the hash tables

Figure 4.7 shows the *train* phase. First, for every hash table, it creates as many unit random vectors as the predefined signature length. Then, the random vectors are broadcasted to all the nodes where the training instances will compute their sign projection signature. Additionally, the hashed training instances are persisted in-memory since they can be consulted multiple times.

Figure 4.8 presents the *test* phase, where each test instance finds the approximate nearest neighbors. First, the test instances repeat the same steps of the *train* phase to find their hash values. Next, the training and test instances use a *explode* operation, which "flattens" the instances by the number of hash tables. Then, the instances are *co-grouped* by the tuple $(table\,position, signature)$, combining the test and training instances with the same entries of the hash tables. Finally, a *reduce by key* operation finds the top k nearest neighbors among the instances that were grouped together.

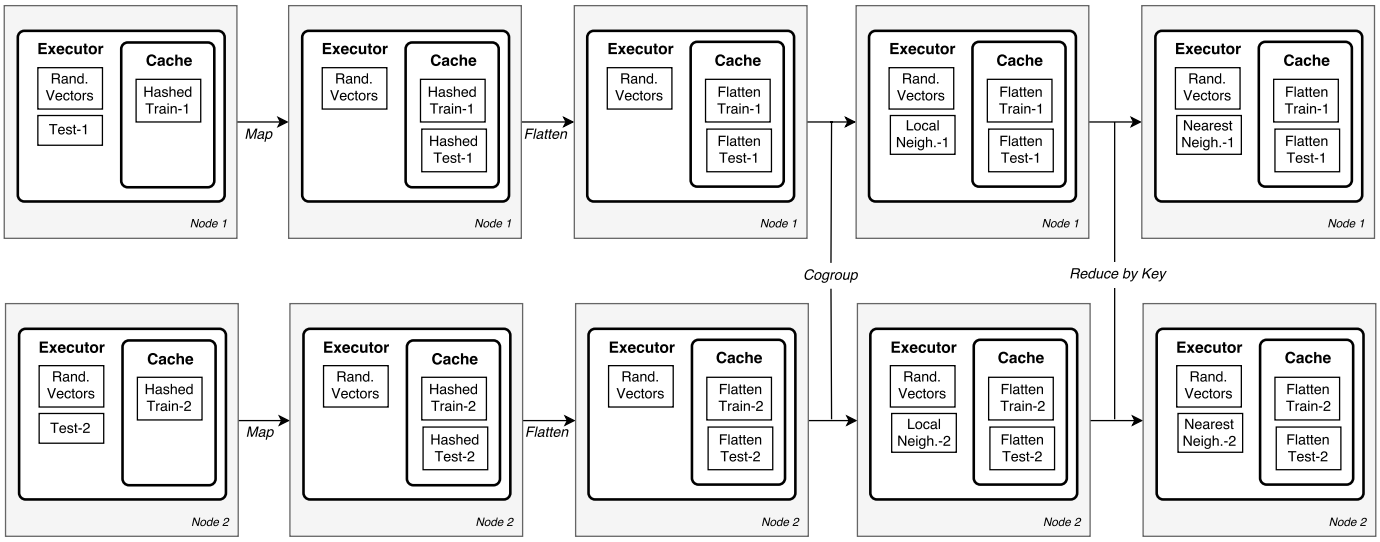


Fig. 4.8.: *Test* phase for Ml-knn-lsh. Each test instance computes their key using the sign projection over the random vector. Then, the data is "flattened" and co-grouped by keys. Finally, the nearest neighbors are searched among the instances in the same group

The main advantage of this method is the dimensionality reduction, the hashes should have a lower dimension than the features. Additionally, no data needs to be exchanged with the *driver*. However, this method has a high memory consumption since it will need to compare all the partitions to match the *hash entry* and *signature*, thus triggering a *cartesian product*.

4.4 Experimental setup

This section describes the experimental setup. Section 4.4.1 summarizes the characteristics of the benchmark datasets. Section 4.4.2 discusses the selection of the parameters. Finally, Section 4.4.3 specifies the hardware and software resources used in the experiments.

4.4.1 Datasets

Table 4.1 summarizes the characteristics of the 22 datasets for multi-label classification used in the experiments, along with the number of instances, number of features, number of labels, cardinality [49], and density [49].

Table 4.1.: Summary description of the datasets

Dataset	Instances	Features	Labels	Cardinality	Density
Flags	194	19	7	3.3918	0.4845
CAL500	502	68	174	26.0438	0.1497
CHD	555	49	6	2.5802	0.4300
Emotions	593	72	6	1.8685	0.3114
Birds	645	260	19	1.0140	0.0534
Medical	978	1,449	45	1.2454	0.0277
Plant	978	440	12	1.0787	0.0899
Water quality	1,060	16	14	5.0726	0.3623
Langlog	1,460	1,004	75	1.1801	0.0157
Enron	1,702	1,001	53	3.3784	0.0637
Scene	2,407	294	6	1.0740	0.1790
Yeast	2,417	103	14	4.2371	0.3026
Human	3,106	440	14	1.1851	0.0847
Slashdot	3,782	1,079	22	1.1809	0.0537
Corel5k	5,000	499	374	3.5220	0.0094
Bibtex	7,395	1,836	159	2.4019	0.0151
Yelp	10,806	671	5	1.6383	0.3277
20NG	19,300	1,006	20	1.0289	0.0514
TMC2007	28,596	500	22	2.2196	0.1009
Mediamill	43,907	120	101	4.3756	0.0433
Bookmarks	87,856	2150	208	2.0281	0.0097
IMDB	120,919	1001	28	1.9996	0.0714

Experiments were performed using 10-fold cross-validation to objectively evaluate the models' performances. The folds are built using a stratified division [174], where each unique subset of labels present in the data is considered as a fictitious label, and then the desired percentage of instances is extracted from each of those labels. This ensures that each of the folds has the same data distribution as the original file.

All our experiments are carried on using classifiers that rely on a metric distance, thus we decided to normalize (re-scale from 0 to 1) the datasets. Normalizing the data ensures that all the features have the same weight when computing the distances and that the distances for numeric and binary features are equal. The numeric features will produce a numeric distance, and the binary features will have distance 1 whenever the value is the same or 0 otherwise.

4.4.2 Methods and parameters

The methods to cover are the ones presented in Section 4.2, however, some of those methods depend on a series of parameters. The most relevant parameter for all the methods that attempt to use the nearest neighbors for classification is the number of neighbors to consider, in this case, we use $k = 3$. It is important to consider that although the final predictions would vary with the number of neighbors, we do not want to find the optimal number of neighbors, but to set the parameters in a way that all the methods are compared in equal conditions. The following parameters were used to facilitate the reproducibility of the experiments, and to provide further insight into the obtained results.

- Ml-knn-it depends on the number of iterations used to broadcast all the test instances and compute the pair-wise distances. This parameter should not be set manually, since the larger the data the more instance would need to be sent per iteration. Instead, we set the size to the buffer to send $1,000$ instance at a time from the *driver* to the rest of the *Nodes*.

- Ml-knn-ht requires a sample of train instances to build the top tree in the *driver*, to avoid collecting all the training instances. The sample size is $1,000$ training instances, to guarantee that the workload would be the same than in Ml-knn-it.

  Another critical parameter is the overlap buffer width for the *spill trees*. The details of this parameter estimation can be found in [173]. In short, assuming that the instances are uniformly distributed in the feature space, the width can be approximated to the average distance between instances. Specifically, the number of instances within a certain radius of a given point is proportional to the density of instances raised to the effective number of features (dimensions), of which manifold data exist on:

$$R_s = \frac{c}{N_s^{(1/d)}} \tag{4.1}$$

where R_s is the radius, N_s is the number of instances, d is the effective number of dimensions, and c is a constant. To estimate R_s for the entire data, we can take samples of different size N_s to compute R_s. We can estimate c and d using linear regression. Finally, we can calculate R_s using total number of instances.

- Ml-knn-lsh needs to set the number of hash tables and the signature length in each entry. The number of hash tables will have a considerable impact on memory since the instances will be duplicated by this value. On the other hand, the signature length will affect the number of training and test instances that are grouped together. We decided to study a wide range of values to evaluate their impact on the quality of the metrics and the execution times, the selected values are $\{1, 2, 4, 8, 16, 32\}$ for number of tables and $\{1, 2, 4, 8, 16, 32, 64\}$ for the signature length.

4.4.3 Hardware and software environment

All the experiments were executed on a local cluster composed of 2 Intel Xeon CPU E5-2690v4 with 28 cores (56 threads) in total and 128 GB of memory. Out of all the resources, 6 cores and 25 GB were reserved for the *driver* and the rest was assigned to the nodes. The experiments were executed using Spark 2.2.0 and Scala 2.11.

4.5 Experimental study

This section presents and discusses the experimental results. Section 4.5.1 compares the quality of the predictions based on the evaluation metrics presented in Section 2.1.3 and includes the study of any parameter that would have a significant impact on the predictions. Section 4.5.2 studies the execution times and considers whenever the performance gain introduced by the index methods surpasses the initial overhead. Section 4.5.3 evaluates whenever the methods scale-out correctly with respect to the increasing number of instances, features, and labels.

4.5.1 Prediction comparison: approximate versus exact

This experiment compares the quality of the predictions produced by the three approaches. The predictions of Ml-knn-it are not affected by any of its parameters, and it is considered an exact method. Ml-knn-ht can be affected by the overlap buffer width, however, the estimation of this value was explained in Section 4.4.2. On the other hand, Ml-knn-lsh is expected to be deeply affected by both the number of tables and the signature length of its hash entries. First, we are going to study the parameter configurations of Ml-knn-lsh, and then the overall best setting will be used in the final comparison to compare the three methods in equal conditions.

Figure 4.9 illustrates the evolution of the predictions, and the impact on the execution time, produced by Ml-knn-lsh using up to 64 hash tables and signatures up to size 32. Since it would not be possible to show the results over all the datasets, the most representative datasets have been selected. These datasets cover a wide range of the number of instances, features, and labels. These datasets obtained a considerably high *subset accuracy* on the exact search, thus the metric which would be affected the most by the loss of performance. The plots on the left side present the *subset accuracy*, and on the right side show the execution times in minutes and on a logarithmic scale.

The most accurate predictions are obtained when more tables are used together with smaller signatures since the number of tables reflects the number of groups that will be created in the matching process and the signature length define inversely the size of those groups. Therefore, the more tables and smaller signatures, the more instances are compared to find neighbors. However, the trade-off is that the more instances are compared, the more distances need to be computed, and the method becomes slower. The results indicate that the execution times grow exponentially with the number of tables and scales logarithmically with the signature size. Therefore, the best compromise between prediction and execution performance is produced by using two tables and signature of size eight.

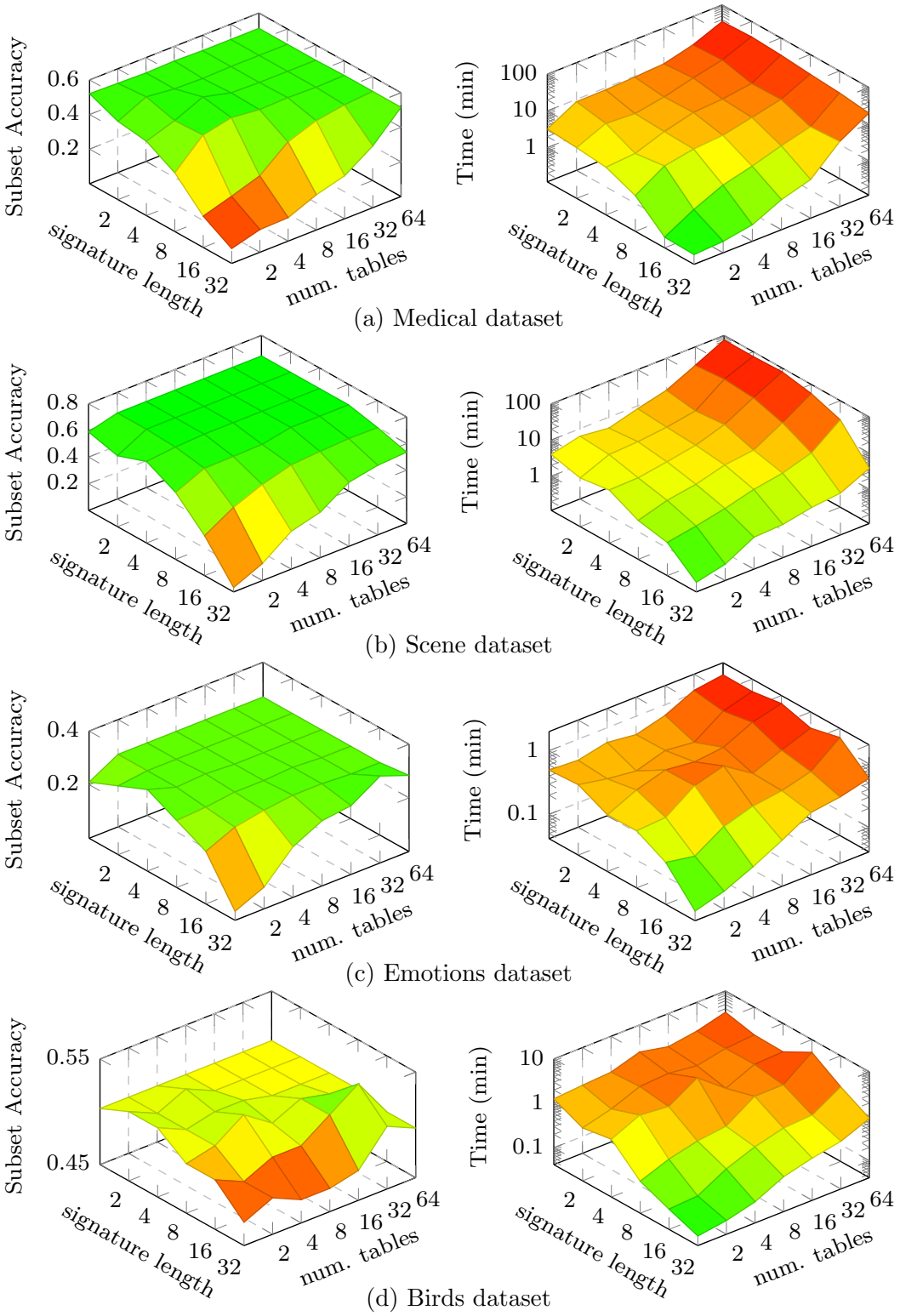


Fig. 4.9.: ML-KNN-LSH subset accuracy on Medical, Scene, Emotions, and Birds datasets

Table 4.2.: Hamming loss and subset accuracy results obtained by the three methods

Dataset	Hamming loss ↓			Subset accuracy ↑		
	ML-KNN-IT	ML-KNN-HT	ML-KNN-LSH	ML-KNN-IT	ML-KNN-HT	ML-KNN-LSH
Flags	**0.2411**	0.2440	0.2827	**0.1667**	0.1458	0.1042
CAL500	0.1360	**0.1357**	0.1370	**0.0000**	**0.0000**	**0.0000**
CHD	0.3031	0.3031	**0.3007**	0.1159	0.1159	**0.1522**
Emotions	0.2072	**0.2050**	0.2140	0.2703	**0.2770**	0.2568
Birds	**0.0487**	0.0497	0.0494	0.5031	**0.5155**	0.5031
Medical	0.0159	**0.0156**	0.0165	0.4751	**0.5339**	0.4932
Plant	**0.0879**	0.0890	0.0893	0.0422	**0.0844**	0.0042
Water quality	**0.3080**	0.3096	0.3248	0.0152	0.0152	**0.0190**
Langlog	**0.0155**	0.0158	0.0156	0.1399	**0.1433**	0.1365
Enron	**0.0498**	**0.0498**	0.0542	0.0669	**0.1297**	0.0209
Scene	0.0984	**0.0929**	0.1082	**0.6456**	0.6007	0.5973
Yeast	**0.2046**	0.2058	0.2060	**0.1493**	**0.1493**	**0.1493**
Human	0.0831	0.0831	**0.0829**	**0.0026**	**0.0026**	0.0000
Slashdot	0.0520	**0.0517**	0.0535	0.0538	**0.0802**	0.0033
Corel5k	**0.0094**	**0.0094**	**0.0094**	**0.0024**	0.0016	0.0000
Bibtex	**0.0088**	**0.0088**	0.0098	0.1075	**0.1110**	0.0043
Yelp	0.1916	**0.1890**	0.2096	0.4056	**0.4100**	0.3733
20NG	**0.0394**	0.0399	0.0507	**0.2832**	0.2830	0.0220
TMC2007	**0.0636**	0.0639	0.0714	**0.2683**	0.2555	0.2001
Mediamill	**0.0278**	**0.0278**	0.0313	**0.1656**	0.1655	0.0919
Bookmarks	**0.0055**	**0.0055**	–	0.2313	**0.2337**	–
IMDB	**0.0714**	**0.0714**	–	**0.0009**	**0.0009**	–

– Experiment could not execute due to computational/memory limitations.

Table 4.2 evaluates the performance of the methods using the selected parameters for the *subset accuracy* and *Hamming loss* metrics. ML-KNN-IT produces the best results in most cases since it is an exact method, it is surpassed by ML-KNN-HT on some occasions by the fact that the overlap buffer might not consider all the real neighbors, and this approximation conveniently considers more appropriate neighbors. Overall, the ML-KNN-IT and ML-KNN-HT performance can be considered equivalent. On the other hand, ML-KNN-LSH tends to have lower values of *subset accuracy*, however, there are some exceptions where the difference is small due to data distributions. Nevertheless, it is considered that ML-KNN-LSH is the most inaccurate of the three methods.

Table 4.3.: Micro-average F1 and macro-average F1 results obtained by the three methods

Dataset	Micro-average F1 ↑			Macro-average F1 ↑		
	ML-KNN-IT	ML-KNN-HT	ML-KNN-LSH	ML-KNN-IT	ML-KNN-HT	ML-KNN-LSH
Flags	**0.7362**	0.7338	0.6844	**0.5670**	0.5649	0.4870
CAL500	0.3171	**0.3315**	0.3305	0.0805	0.0822	**0.0832**
CHD	0.6144	0.6121	**0.6289**	0.3154	0.3095	**0.3410**
Emotions	0.6406	**0.6459**	0.6154	0.6205	**0.6286**	0.5876
Birds	**0.3318**	0.2475	0.1065	**0.1985**	0.1624	0.0546
Medical	0.6520	**0.6652**	0.6435	0.6374	**0.6553**	0.6326
Plant	0.0741	**0.1365**	0.0078	0.0272	**0.0413**	0.0038
Water quality	**0.5271**	0.5238	0.4468	**0.4307**	0.4281	0.3357
Langlog	0.0058	**0.0114**	0.0000	0.2963	**0.2982**	0.2933
Enron	0.4130	**0.4499**	0.3875	0.2385	**0.2432**	0.2070
Scene	**0.7126**	0.7089	0.6840	**0.7236**	0.7125	0.6932
Yeast	**0.6246**	0.6228	**0.6246**	0.3459	0.3448	**0.3472**
Human	**0.0045**	**0.0045**	0.0000	**0.0029**	**0.0029**	0.0000
Slashdot	0.1017	**0.1592**	0.0056	0.1907	**0.2089**	0.1401
Corel5k	**0.0210**	0.0095	0.0023	**0.1690**	0.1677	0.1642
Bibtex	0.2949	**0.3008**	0.0807	0.0911	**0.1010**	0.0289
Yelp	0.6556	**0.6607**	0.6423	0.5604	**0.5605**	0.5357
20NG	**0.4315**	0.4287	0.0464	**0.4218**	0.4162	0.0451
TMC2007	0.6362	**0.6573**	0.5782	0.4073	**0.4105**	0.3002
Mediamill	**0.6039**	0.6036	0.5282	**0.2125**	0.2123	0.0653
Bookmarks	**0.3551**	0.3591	–	0.1134	**0.1153**	–
IMDB	0.0017	**0.0016**	–	**0.0118**	0.0117	–

– Experiment could not execute due to computational/memory limitations.

Table 4.3 compares the predictions using two metrics which are more representative of the real quality of the predictions, *micro-average F1* and *macro-average F1*. ML-KNN-IT and ML-KNN-HT obtained equivalent results, just like for the previous metrics. On the other hand, ML-KNN-LSH performed considerably worse than the other methods. This difference can be attributed to the underlying data distribution, not having a uniform data distribution may lead to a poor approximation in LSH due to the random vectors not partitioning the space properly.

Table 4.4.: Execution times for the *train* and *test* phases in minutes for the three methods

Dataset	Train time			Test time		
	Ml-knn-it	Ml-knn-ht	Ml-knn-lsh	Ml-knn-it	Ml-knn-ht	Ml-knn-lsh
Flags	0.0470	0.0475	**0.0413**	0.0225	**0.0143**	0.0176
CAL500	**0.0630**	0.1524	1.5737	**0.0323**	0.0361	1.0140
CHD	**0.0501**	0.0597	0.0847	0.0243	**0.0214**	0.0651
Emotions	**0.0517**	0.0640	0.2412	0.0250	**0.0214**	0.3415
Birds	**0.0640**	0.0834	0.1530	0.0306	**0.0245**	0.2276
Medical	**0.1910**	0.2365	1.0820	0.0787	**0.0715**	2.3176
Plant	**0.0938**	0.1204	0.2549	0.0457	**0.0433**	0.4977
Water quality	**0.0575**	0.0880	0.4765	**0.0244**	0.0316	0.3426
Langlog	**0.1919**	0.3333	0.6817	**0.0891**	0.1058	0.5617
Enron	**0.2560**	0.3892	0.5233	**0.0718**	0.1027	0.2781
Scene	**0.1397**	0.1861	0.5484	**0.0552**	0.0689	1.9123
Yeast	**0.1256**	0.1776	1.3601	**0.0438**	0.0753	0.7848
Human	**0.2609**	0.2730	0.6156	**0.0978**	0.1231	1.3664
Slashdot	0.3831	**0.3632**	0.7124	**0.1286**	0.1335	1.0321
Corel5k	**0.8753**	3.6945	18.4798	**0.2816**	1.5695	4.7568
Bibtex	3.5429	**2.8774**	8.2819	**0.6209**	0.7865	6.8489
Yelp	3.1812	**0.3867**	0.4578	0.8685	**0.1460**	4.1907
20NG	14.4004	**1.0986**	3.7498	4.4056	**0.3758**	23.4893
TMC2007	44.9470	**1.2671**	9.1497	14.6589	**0.6059**	41.5393
Mediamill	168.2233	**3.8833**	691.1088	60.3439	**2.0957**	241.1892
Bookmarks	2170.5851	**27.9763**	–	543.7354	**8.6002**	–
IMDB	4844.1880	**17.6109**	–	1559.7201	**6.3888**	–

– Experiment could not execute due to computational/memory limitations.

4.5.2 Performance comparison: execution times for train and test phases

This experiment evaluates the execution times for the studied methods. The Ml-knn algorithm is affected by the number of instances, features, and labels. Consequently, we consider multiple datasets that represent a wide range of characteristics and analyze the impact of those factors. Table 4.4 shows the execution time, in minutes, for each dataset sorted by the number of instances. The left side of the table presents the results for the *train* phase, and the right side presents the results for the *test* phase.

Ml-knn-it has the best execution times for the small datasets, followed very closely by Ml-knn-ht. However, the performance declines with the number of instances since the complexity of the algorithm is tightly bounded by the number of instances.

Ml-knn-ht quickly surpasses Ml-knn-it once the dataset is big enough to require an execution time where the construction of an index represents a small fraction of the total time. The difference of performance is even greater for the *test* phase since the index has already been constructed and it only needs to query the structures.

Ml-knn-lsh has the longest execution times for most of the datasets. The gap in performance is especially big for the *test* phase since the *co-group* operation is less efficient between two different sets of instances. Despite computing pair-wise distances only within the same group, instead of all the instances like Ml-knn-it, the performance gain is dragged down by the duplication of instances and the computation of the hash entries for all the hash tables. This could lead to memory issues, as it can be appreciated for the largest datasets where it was not possible to finish the executions due to hardware limitations.

4.5.3 Scalability analysis on the number of instances, features, and labels

Another important aspect is to consider the evolution of the execution times with regards to the size of the data. The size can increase by multiple factors: number of instances, the number of features, and the number of labels. This experiment studies the scalability of the three methods by observing the total execution times (*train* and *test* phases together) over different samplings of the *20NG* dataset. Since this experiment only studies the computational performance, it is irrelevant which instances, features or labels are selected.

Figure 4.10 presents the execution times on a range of 1,000 up to 16,000 instances, with a fixed number of features and labels. The execution times of Ml-knn-it and Ml-knn-lsh increase exponentially with the number of instances. Ml-knn-it needs to execute a pair-wise comparison between instances, hence this exponential growth is expected. Ml-knn-lsh executes a pair-wise comparison only within grouped instances, however, the *explode* operation duplicates the instances by the number of tables. This leads to an extra overload of memory and computational time that eventually drags down the performance of the method. Finally, Ml-knn-ht presents the best scalability, with considerably reduced and linear execution times.

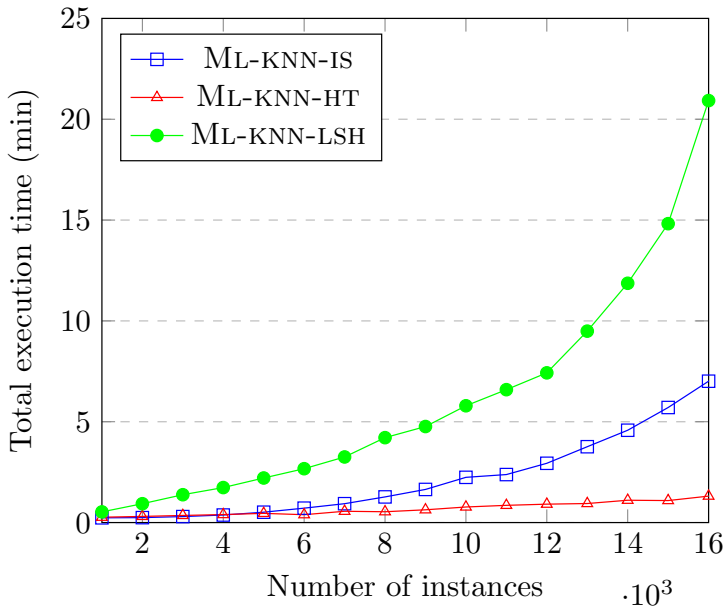


Fig. 4.10.: Execution times according to the number of instances

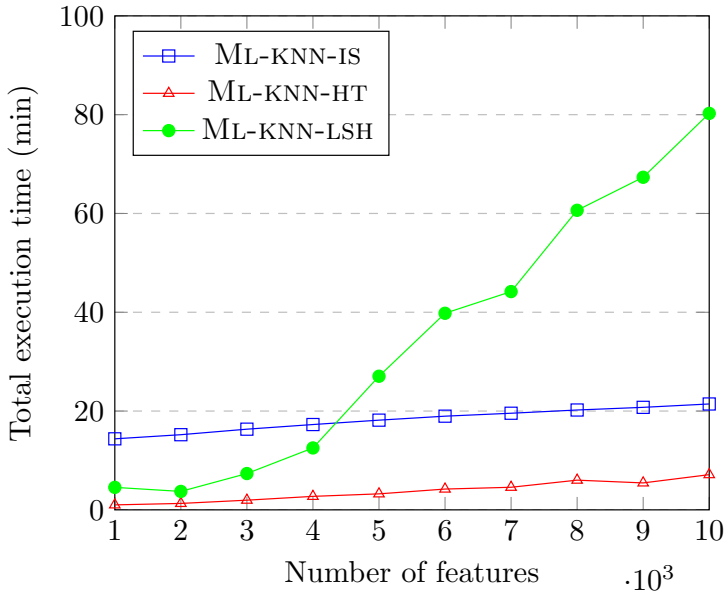


Fig. 4.11.: Execution times according to the number of features

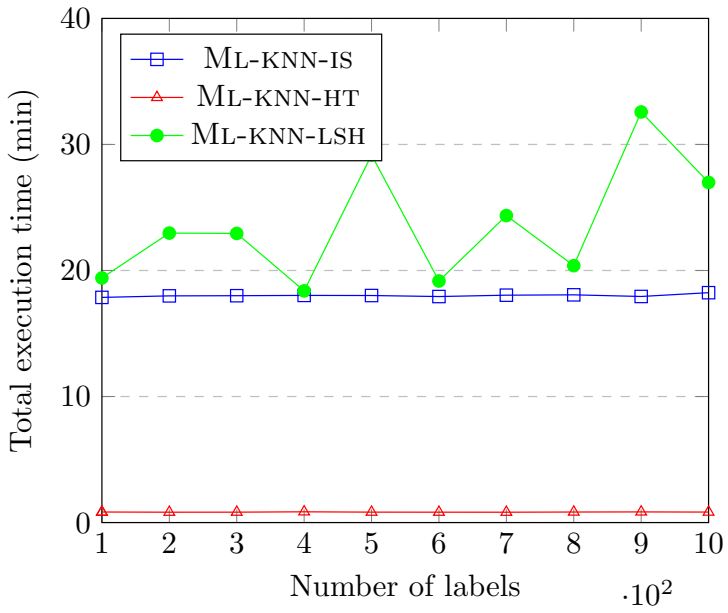


Fig. 4.12.: Execution times according to the number of labels

Figure 4.11 shows the performance of the methods on a range of features from 1,000 up to 10,000, with a fixed number of instances and labels. ML-KNN-IT and ML-KNN-HT scale linearly with the number of features, where ML-KNN-HT has executions times orders of magnitude smaller since a reduced number of instances will compute their distances. On the other hand, ML-KNN-LSH increases exponentially with the number of features. This is produced by the computation of the entries in the hash tables since they depend directly on the number of features. Moreover, this method depends on exchanging a lot of information between nodes, hence the high-dimensionality increases the network traffic.

Figure 4.12 illustrates the execution times varying the number of labels from 100 up to 1,000, with a fixed number of instances and features. The three methods present a constant execution times despite the number of labels used. This indicates that the execution times of ML-KNN are not deeply affected by the number of labels, becoming a good alternative for datasets with high-dimensional label spaces. Despite not having a significant impact on the performance, the number of labels could eventually lead to memory problems, especially for those methods that duplicate the instances.

4.6 Conclusions

In this chapter, we have presented and evaluated three strategies to distribute Ml-knn over Spark. Each of the three approaches incorporates a different strategy for the distributed nearest neighbors search: brute force, tree-based index, and locally sensitive hashing. The impact of these strategies in Ml-knn has been studied into detail considering multiple metrics, regarding the quality of the predictions, execution times, and scalability factor.

The experimental study carried out has shown that Ml-knn can handle large datasets over the Spark framework, obtaining competitive results, both in prediction and computational performance. Considering each of the three methods, Ml-knn-it obtained the baseline accuracy, since it is an exact method and the lowest execution times for the smaller datasets. However, the experiments show that it scales poorly, especially with the number of instances. Ml-knn-ht produced an accuracy equivalent to an exact method, besides having the fastest execution times for most of the datasets. Additionally, it scaled-out more efficiently than the other methods, being able to handle even the largest datasets. Ml-knn-lsh produced the most inconsistent results, while it produced larger differences over the strictest metrics, it is considered that final accuracy was acceptable for an approximate method. However, it scaled-out poorly and it had problems to execute the largest datasets.

These results indicate that by incorporating the right strategy for nearest neighbors searches, Spark enables Ml-knn to execute over large datasets that would not be feasible to consider in a single machine. As future work, Ml-knn-it could possibly be improved by exchanging the information using cross-joins with a specific test partition, that way we would avoid using the *driver* as an intermediary to exchange information. On the other hand, the algorithm with the largest capacity for improvement is the Ml-knn-lsh. At the moment, the available implementation relies on flattening rows and co-grouping them which has a high computational cost. We could use a hash table on the *driver* to indicate the partitions that store specific buckets of the hash tables. Eventually, a proper implementation of Ml-knn-lsh could even surpass the Ml-knn-ht performance for high-dimensional data.

CHAPTER 5

DISTRIBUTED FEATURE SELECTION OF MULTI-LABEL DATA

Multi-label learning is characterized by the added difficulty of handling multiple labels using data which is distinctive by their large dimensionality. The multi-label classification models tend to suffer from the curse of dimensionality since the predictions produced by a model are deeply influenced by the quality of the input features. It has been shown that discarding redundant and irrelevant features leads to increased accuracy of the model. There are two main approaches that modify the input space: feature transformations (e.g. principal component analysis, polynomial kernel, etc) and feature selection (e.g. mutual information maximization, χ^2 test, etc). The former modifies the original input space, thus, making it impossible to extract useful information, while the latter preserves the original data.

Feature selection algorithms are divided into three main categories [175], according to how they assess the importance of candidate feature subsets. Namely, these are *wrappers*, *embedded methods*, and *filters*. Wrappers select a subset of features that maximize the performance of a classification algorithm [176]. They are expected to achieve good results at a high computational cost. Embedded methods [177] perform simultaneously feature selection and prediction using the specific structure of the classifier being used, e.g. bounds on the leave-one-out error of SVMs [178]. Filters find the subset of features that maximizes some criteria. They are particularly effective in computation time and robust to over-fitting. These methods are the fastest approach, and unlike the other categories, they are independent of the learning algorithm. In the literature, several criteria have been proposed to evaluate

the quality of the subset of features. The most frequently employed are the correlation coefficient [179] and the mutual information [180]–[182].

Most of the works in multi-label feature selection agree on the advantages of adopting a criterion to handle multiple labels, however, there is much debate over how to address this issue. Here, we present a comprehensive and detailed analysis of multi-label feature selection strategies, where we discuss the main approaches.

We introduce two algorithm adaptation methods based on mutual information, which do not require any type of data transformation nor discretization of continuous features. The first method maximizes the mutual information between the selected subset of features and the labels. The second method minimizes the redundancy of the selected subset while maximizing the relevance between the features and labels. These are compared with three traditional multi-label feature selection methods.

Finally, we propose two methods which take opposite approaches to combine the best of the previous methods: the redundancy minimization and the constant runtime. These methods study the best approach to aggregate the MI of multiple labels. The first method selects the features with the largest L^2-norm whereas the second selects the features with the largest geometric mean. These proposed methods are compared with all five multi-label feature selection methods using large-scale data with discrete features.

5.1 Multi-label feature selection background

Let F be the feature set such that $F = \{f_1, \ldots, f_d\}$, the feature selection task aims to identify a subset of features $S \in F$ such that it has the highest relevance on the label set $\mathcal{Y}$. *Tsoumakas et al.* [49] divide the approaches to handle multi-label problems into *problem transformation* and *algorithm adaptation*. This categorization can also be used by feature selection methods, in order to distinguish between methods that attempt to handle label correlations directly or by modifying the original data.

Table 5.1.: Summary of problem transformation methods for multi-label feature selection

Name	Method description
Feature ranking	
[183] BR-χ^2	BR + χ^2
[184] BR-RF	BR + ReliefF
[184] BR-IG	BR + Information Gain
[39] LP-χ^2	LP + χ^2
[184] LP-RF	LP + ReliefF
[184] LP-IG	LP + Information Gain
[185] PPT-χ^2	PPT + χ^2
[186] PPT-RF	PPT + ReliefF
[185] PPT-IG	PPT + Information Gain
[78] ELA-χ^2	Weighted BR + χ^2
[78] ELA-IG	Weighted BR + Information Gain
[78] ELA-OCFS	Weighted BR + Orthogonal Centroid Feature Selection
Subset search	
[187] PPT-MI-SFS	PPT + Mutual Information + Sequential Forward Selection

5.1.1 Problem transformation methods

The problem transformation techniques are the most straight forward approaches. The feature selection methods based on this technique usually perform the following steps: transform the label space into one or many subsets, rank the feature using a defined score, combine the results of all the subsets, and finally select the feature which optimizes a given criterion. Table 5.1 presents the most relevant methods in the literature based on problem transformation.

The Binary Relevance (BR) decompose the multi-label dataset into q binary datasets, where the j-th dataset contains the binary representation of label y_j (i.e. instances with the label present will be labeled positively, otherwise they will be zero). This method has been integrated with χ^2 [183], ReliefF [184], and information gain [184]. However, by considering each label independently it might ignore the possible dependencies and correlations between the labels.

Label Powerset (LP) transforms the multi-label dataset to a multi-class dataset by defining a function that maps each unique label subset to a single class. This approach

has been used with χ^2 [188], ReliefF [184], and information gain [184]. Additionally, it has been used in combination with mutual information and forward feature selection, in order to minimize the redundancy and maximize the relevance [187]. LP has been shown to outperform BR in the cases where the labels are highly correlated. However, LP might transform the data to a large number of classes, specifically $min(n, 2^{|\mathcal{Y}|})$ where n is the number of instances. It also suffers from an extreme class imbalance, since the number of instances belonging to a unique label subset is expected to be small.

Pruned Problem Transformation (PPT) [64] modifies the previous method by removing the instances belonging to a label subset whose frequency is below a certain threshold. This method defines a way to reinsert this information into the original data by splitting the label subset into subsets with a higher frequency. After this step, it applies the LP method which is expected to have a reduced number of classes. This method has been used with χ^2 [185], ReliefF [186], and information gain [185], [187].

Entropy-based Label Assignment (ELA) extracts the labels present in each instance and assigns them a weight equivalent to the inverse of the original number of labels. Therefore, the instances annotated with a large number of labels will be associated with a lower weight than with fewer labels. This method assumes that the information originated from smaller label subsets is more focused. This method has been integrated with χ^2 and with the Orthogonal Centroid Feature Selection (OCFS) [78].

5.1.2 Algorithm adaptation methods

The algorithm adaptation methods modify the traditional feature selection methods to use multi-label data. These approaches can be categorized into: *ranking methods* and *optimization methods*. The former ranks the features by using a multivariate version of a score metric, or any other adaptation that allows handling multiple labels. The latter defines the multi-label feature selection process as an optimization problem which considers in various ways the relevance and redundancy of the features and then determines globally the best subset. Table 5.2 presents the most relevant methods based on algorithm adaptation.

Table 5.2.: Summary of algorithm adaptation methods for multi-label feature selection

Name	Method description
Feature ranking	
[186] ReliefF-ML	ReliefF
[184] RF-ML	ReliefF
[189] RReliefF-ML	Regression RliefF
Subset search	
[190] mRMR	One-by-one + Mutual Information + Sequential Forward Selection
[191] MDMR	One-by-one + Mutual Information + Sequential Forward Selection
[192] SCLS	One-by-one + Mutual Information (improved) + Sequential Forward Selection
[193] MFNMI	Local Mutual Information + Sequential Forward Selection
[194] GFS-ML	Label Granulation + One-by-one + Mutual Information + Sequential Forward Selection
[195] FIMF	Label Combination + Mutual Information (limited interaction) + Sequential Forward Selection
[185] PMU	Label Combination of Second-Order + Mutual Information + Sequential Forward Selection
[196] MAMFS	Label Combination of High-Order + Mutual Information + Sequential Forward Selection
Optimization	
[197] QPFS-ML	Quad. Programming Feature Selection
[198] MIFS	Alternating Optimization
[199] QPFS-ML-NYSTROM	Quad. Programming Feature Selection with Nystrom
[200] R-QPFS-ML-FWM	PPT-Regularized Quad + Programming Feature Selection Frank-Wolfe
[201] MMI-PSO	Particle Swarm Optimization

Relief feature scoring is based on the identification of feature value differences between nearest neighbor instance pairs. If a feature value difference is observed in a neighboring instance pair with the same class (a hit), the feature score decreases. Alternatively, if a feature value difference is observed in a neighboring instance pair with different class values (a miss), the feature score increases. There are various adaptations of Relief for multi-label classification. A multi-label ReliefF built via modifying prior probability estimation was proposed in [186], [189]. Another modification, that considers nearest instances belonging to a different set of labels have different feature values, was presented in [189]. This method was modified to use a dissimilarity function based on Hamming distance in [184].

Mutual information is the metric that has been adapted more frequently. There are two strategies to measure the information provided by a feature and multiple labels: One-by-one (OBO) and Label Combination (CM). OBO considers each label sequentially, therefore it sums the individual scores between the feature and each label. OBO has been applied in multiple occasions in combination with sequential forward selection in [190]–[192]. CM considers all the labels simultaneously by handling each label combination, therefore the score will consider all the labels interactions (or alternatively a limited number). This approach has been combined with sequential forward selection in [185], [191], [195].

Optimization methods aim to formulate the multi-label feature selection task as a constrained optimization problem. These methods usually aim to balance the trade-off between redundancy and relevance of the selected feature subset. The advantage is that they do not require to specify the number of required features in the selection process, whereas the other filter methods need to specify the size of the subset manually. However, they usually consider complex algorithms with high computational complexities. The reference algorithm for traditional classification is the Quadratic Programming Feature Selection (QPFS) [202]. This method has been adapted to multi-label classification by considering each label independently in [197]. The optimization procedure has been improved by using the Nystrom low-rank approximation in [199]. A modification to this method which uses a regularizer to achieve sub-linear converge rate was also proposed in [200]. Additionally, other methods have tried to explore different approaches such as an alternating optimization algorithm [198] or using a particle swarm optimization method [203].

5.2 Preliminaries

This section introduces the theoretical background and reviews the definitions related to feature selection. Section 5.2.1 introduces the entropy and mutual information (MI) concepts.

5.2.1 Basic definitions

The Shannon entropy is a measure of the uncertainty of a random variable. The level of uncertainty is related to the probability of the elements composing the variable. The concept of uncertainty can be seen as the measure of how much information is needed to describe the item. Intuitively, a high entropy indicates that the elements in the variable have about the same probability of occurrence, while a low entropy means larger differences in the probabilities of occurrences. Thus, entropy is related to the probabilities of the variable rather than the actual values.

Let A be a discrete random variable, entropy is defined as:

$$H(A) = -\sum_{a \in A} p(a) \log_2 p(a) \tag{5.1}$$

Let A and B be two random discrete variables, the joint entropy is the sum of the uncertainty contained by the two variables. Formally, joint entropy is defined as follows:

$$H(A, B) = -\sum_{a \in A} \sum_{b \in B} p(a, b) \log_2 p(a, b) \tag{5.2}$$

The maximum value on joint entropy happens when A and B are completely independent, hence, the minimum value occurs when A and B are completely dependent. Closely related is the conditional entropy, which measures the remaining uncertainty of B when A is known. The conditional entropy is defined as:

$$\begin{aligned} H(B|A) &= -\sum_{a \in A} p(a) \sum_{b \in B} p(b|a) \log_2 p(b|a) \\ &= -\sum_{a \in A} \sum_{b \in B} p(a, b) \log_2 p(b|a) \end{aligned} \tag{5.3}$$

Entropy can be used to measure the information that one variable contains about another by measuring the decrease in the uncertainty of one variable due to the knowledge of another. This term stands for mutual information (MI), and formally is defined as follows:

$$
\begin{aligned}
MI(A,B) &= \sum_{a\in A}\sum_{b\in B} p(a,b)\log_2\frac{p(a,b)}{p(a)p(b)} \\
&= \sum_{a\in A}\sum_{b\in B} p(a,b)\log_2\frac{p(b|a)}{p(b)} \\
&= -\sum_{a\in A}\sum_{b\in B} p(a,b)\log_2 p(b) + \sum_{a\in A}\sum_{b\in B} p(a,b)\log_2 p(b|a) \\
&= -\sum_{b\in B} p(b)\log_2 p(b) - \Big(-\sum_{a\in A}\sum_{b\in B} p(a,b)\log_2 p(b|a)\Big) \\
&= H(B) - H(B|A)
\end{aligned}
\tag{5.4}
$$

MI is a symmetrical measure, the amount of information gained about B after observing A is equal to the amount of information gained about A after observing B:

$$
\begin{aligned}
MI(A,B) &= H(B) - H(B|A) \\
&= H(A) - H(A|B) \\
&= H(A) + H(B) - H(A,B)
\end{aligned}
\tag{5.5}
$$

Figure 5.1 presents the relationships between these information measures associated with two correlated variables A and B. The area contained by both circles is the join entropy $H(A,B)$ (Eq. 5.2). The perimeter of each circle is the individual entropy $H(A)$ and $H(B)$ (Eq. 5.1), being the filled area the conditional entropy $H(A|B)$ and $H(B|A)$ (Eq. 5.3), which does not consider the intersected area. Finally, the intersected area represents the MI (Eq. 5.5).

Both, entropy and MI rely on discrete variables. However, they can be extended to continuous spaces in terms of probability density functions by turning the previous sums into integrals, therefore the differential entropy is defined as:

$$
H(A) = -\int_A \mu(a)\log_2\mu(a) \tag{5.6}
$$

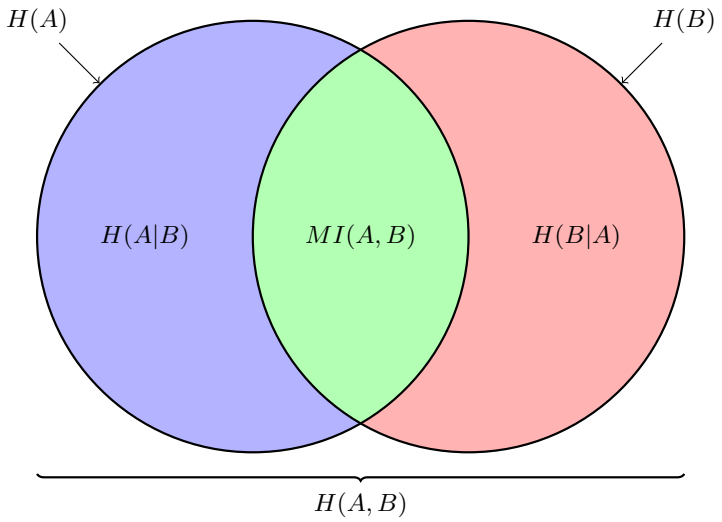


Fig. 5.1.: Venn diagram showing the relationships for entropy and MI associated with two correlated variables A and B

The MI for a continuous variable can be expressed as:

$$MI(A,B) = \int_A \int_B \mu(a,b) \log_2 \frac{\mu(a,b)}{\mu(a)\mu(b)} \tag{5.7}$$

In practice, none of the probability density functions μ are known in a real-world problem, therefore it is necessary to use an estimation from the data.

5.2.2 Estimators

The most simple and widespread estimation approach consists in partitioning the space into bins of finite size, hence approximating $\int_i \mu_a(a)da = p_a(i)$, $\int_j \mu_b(b)db = p_b(j)$, and $\int_i \int_j \mu(a,b)dadb = p(i,j)$ where $\int_i$ denotes the integral over bin i. If $n_a(i)$ and $n_b(j)$ are the number of instances falling into the i-th bin for A and B respectively, and $n(i,j)$ is the number of instances in their intersection, then we can approximate $p_a(i) \approx n_a(i)/N$, $p_b(j) \approx n_b(j)/N$, and $p(i,j) \approx n(i,j)/N$, where N is the number of instances. By applying such an approximation to the differential entropy in Eq. 5.6 we can obtain Eq. 5.1. Similarly, by applying it to the differential MI in Eq. 5.7 we can obtain Eq. 5.4.

However, such estimators incur into systematic errors from approximating the (logarithms of) probabilities by (logarithms of) frequency ratios. This error is reduced when $N \to \inf$ and $Bin_{size} \to 0$, however, there is not such a case in a real-world problem. It has been shown that this error can be minimized by using estimators with different techniques such as adaptive bin sizes which are geared to have the same number of $n(i,j)$ for all pairs (i,j) [204].

Another approach is to estimate the continuous distributions evaluated at given data examples. This method, originally developed by *Kozachenko and Leonenko* [205], uses the nearest neighbors technique to estimate the entropy by considering the density function μ constant throughout the neighborhood of an instance. This technique has been extensively studied for estimating differential entropies [187], [206]–[208].

This estimator uses the concept that Eq. 5.6 can be approximated (up to the minus sign) as the average of $\log \mu(a)$.

$$\hat{H}(A) = -N^{-1} \sum_{i=1}^{N} \widehat{log\mu(a_i)} \tag{5.8}$$

Thus, if we had an unbiased estimator of $\widehat{log\mu(a_i)}$, we could estimate the entropy at each given instance. The estimator can be obtained using ϵ-balls centered at a_i whose radius is the distance from a_i to its k-th neighbor. By assuming that $\mu(a)$ is constant in the entire ϵ-ball, we obtain:

$$\widehat{log\mu(a_i)} \approx \psi(k) - \psi(N) - \log(V_d) - dE(\log(\epsilon)) \tag{5.9}$$

where ψ stands for the digamma function, k is the number of neighbors, and V_d is the volume of the d-dimensional unit ball. Therefore, using Eq. 5.8 and Eq. 5.9 one obtains:

$$\hat{H}(A) = -\psi(k) + \psi(N) + \log(V_d) + \frac{d}{N} \sum_{i=1}^{N} \log(\epsilon_i) \tag{5.10}$$

where ϵ_i is twice the distance from a_i to its k-th neighbor.

5.2.2.1 Mutual information estimator between continuous features

A MI estimator based on *Kozachenko-Leonenko* entropy estimator is presented in [209]. In order to obtain $MI(A, B)$, we have to subtract $H(A, B)$ from estimates for $H(A)$ and $H(B)$, as stated in Eq. 5.5. The estimator for the marginal entropies has been shown previously, thus, we only need to estimate the joint entropy. In this case they consider the joint random variable $J = (A, B)$, where the radius of the ϵ-ball would be the distance from i-th to its k-th neighbor in the J space. By replacing in Eq. 5.10, d with $d_j = d_a + d_b$, and V_d with $V_j = V_{d_a} * V_{d_b}$, the joint entropy estimator is:

$$\hat{H}(A, B) = -\psi(k) + \psi(N) + \log(V_j) + \frac{d_j}{N} \sum_{i=1}^{N} \log(\epsilon_i) \tag{5.11}$$

By combining Eq. 5.10 and Eq. 5.11, using the same k number of neighbors, we could calculate $MI(A, B)$. However, this would mean that there are different distance scales for the estimators. The distances will be larger in the join space than the distances in the marginal spaces. Since the bias in Eq. 5.10 depends on the distances, the biases for $H(A)$, $H(B)$ and $H(A, B)$ would not cancel.

However, since Eq. 5.10 holds for any value of k, there is no need to use a fixed k when estimating the marginal entropies. Thus, using the ϵ-ball equal to the distance from the i-th instance to its k-th neighbor in the J space, in $H(A)$, $H(B)$ and $H(A, B)$; produces a good approximation. Now, k can be replaced by n_a which is the number of neighbors within each ϵ-ball at every i-th instance plus one (the instance itself). Therefore, it leads to:

$$\hat{H}(A) = -\frac{1}{N} \sum_{i=1}^{N} \psi(n_a + 1) + \psi(N) + \log(V_{d_a}) + \frac{d_a}{N} \sum_{i=1}^{N} \log(\epsilon_i) \tag{5.12}$$

Replacing A by B in the right hand leads to $H(B)$. Then, the MI estimator can be found by subtracting the joint entropy to the marginal entropies:

$$\widehat{MI}(A, B) = \psi(k) - \langle \psi(n_a + 1) + \psi(n_b + 1) \rangle + \psi(N) \tag{5.13}$$

5.2.2.2 Mutual information estimator between continuous and discrete features

An estimator specific to classification problems was derived on [210] based on the estimator between continuous variables previously presented. They derive the entropy formula considering a continuous variable C and a discrete variable B, by using the continuous densities as $\mu(\cdot)$ and the discrete probability function as $p(\cdot)$: therefore, $p(B) = \int \mu(c,b)dx$ and $\mu(c) = \sum_b \mu(c,b)$.

$$\begin{aligned} MI(C,B) &= H(C) + H(B) - H(C,B) \\ &= -\int \mu(c)\log(\mu(c))\ dc - \sum_b p(b)\log(p(b)) \\ &+ \sum_b \int \mu(c,b)\log(\mu(c,b))\ dc \\ &= -\int \mu(c)\log\mu(c)\ dc + \sum_b \int \mu(c,b)\log(\mu(c|b)\ dc \\ &= -\langle\log\mu(c)\rangle + \langle\log\mu(c|b)\rangle \end{aligned} \tag{5.14}$$

The logarithms of continuous distributions can be approximated by using the *Kozachenko-Leonenko* method in Eq. 5.9. For each data example, we employ the estimator twice: once to estimate $\mu(c)$ by finding the neighbors from the full set of examples, and once to estimate $\mu(c|b)$ by finding the neighbors in the subset of data examples with the same value for the discrete variable B. The result is:

$$\begin{aligned} \widehat{MI}(C,B) &= \psi(N) - \psi(m) + \log(V_{m;c}) + dE(\log(\epsilon)) \\ &- \psi(N_b) + \psi(k) - \log(V_{k;c|b}) - dE(\log(\epsilon)) \end{aligned} \tag{5.15}$$

where N is the number of instances, N_b is the number of instances with the same class and k is the number of neighbors. We define d_i as the distance between the i-th instance and the k-th neighbor of the instances belonging to the same class of instance x_i. Then, m_i is the number of neighbors from the full set of instances that lie within distance d_i to the i-th instance (including the k-th neighbor).

There is an averaging error that can be minimized by using the same k and m, thus $V_{m;c} = V_{k;c|b}$ for each instance:

$$\widehat{MI}(C, B) = \psi(N) - \langle \psi(N_b) \rangle + \psi(k) - \langle \psi(m) \rangle \tag{5.16}$$

The cancellation is only partial, but the averaging error scales with the number of instance pairs as N^{-2} whereas the counting error scales as $N^{-1/2}$. Thus, the averaging error is insignificant except for very small data.

5.3 Mutual information estimator for multi-label data

There is much discussion regarding the best way to consider the label correlations in multi-label feature selection. Section 5.1 presented the most relevant works which can be divided into problem transformation or algorithm adaptation. We decided to discard problem transformations since they usually modify the data distribution during the transformation process, in addition to incurring into unnecessary computations. Instead, we focus on studying how MI has usually been adapted for multi-label data and present a detailed analysis of the different approaches.

Figure 5.2 illustrates information theoretic measures for three variables (one feature f_i and two labels $\{y_j, y_k\}$). The entropy of each variable $H(\cdot)$ is represented by each circle. The intersection of any two circles represents the mutual information for the two associated variables. However, when considering the three variables simultaneously it is necessary to distinguish between the true MI among three variables $MI(f_i, y_j, y_k)$, and the MI between the feature and both labels $MI(f_i, y_j) + MI(f_i, y_k) - MI(f_i, y_j, y_k)$. True MI is hard to interpret since it can be positive (*redundancy*) or negative (*synergy*). In this case, redundancy would refer to the case where the information provided by f_i about y_j also unveils information about y_k. On the other hand, synergy considers the case where f_i provides information about y_j and y_k, however, there is no shared information between both labels. Thus, the middle area of the diagram would be empty and it would produce a negative value.

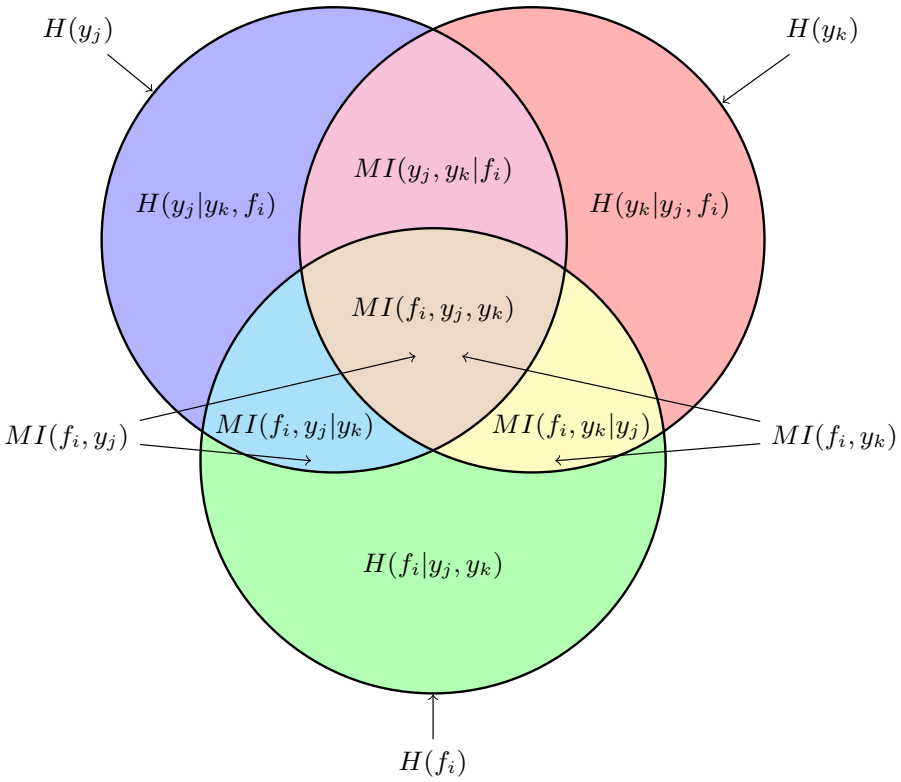


Fig. 5.2.: Venn diagram showing the relationships for entropy and MI associated with one feature f_i and two labels $\{y_j, y_k\}$.

Although synergy considers there is not shared information among the feature and the labels, this does not mean the feature does not provide information about the labels. A feature might reduce the remaining uncertainty of each label despite not offering information about their shared information. Therefore, we consider the multi-label MI as the information shared between the feature and the label set, defined as:

$$\begin{aligned} MI^1(f_i, \mathcal{Y}) &= H(f_i) + H(\mathcal{Y}) - H(f_i, \mathcal{Y}) \\ &= H(f_i) + H(y_1, \ldots, y_q) - H(f_i, y_1, \ldots, y_q) \end{aligned} \tag{5.17}$$

This formulation is considered the adaptation of MI using the Label Combination technique since it considers the real MI with a high-order of label correlations. This formulation

requires the computation of high-dimensional joint entropies, which can be expanded to:

$$H(y_1, ..., y_q) = -\sum_{y_1} \cdots \sum_{y_q} P(y_1, ..., y_q) \log P(y_1, ..., y_q) \tag{5.18}$$

Methods based on this multi-label MI usually rewrite the high-dimensional joint entropies terms as the cumulative sum over the entropies of the powerset with Möbius inversion. This leads to an exponential increase in the computation time with respect to the number of labels. By using approximations, e.g. only considering sets of a determined cardinality, the complexity may be reduced. Nevertheless, this method suffers from high complexity, despite using heuristics, with respect to the dimensionality of the label space.

There is a different strategy which addresses the exponential complexity by approximating the concept of multi-label MI. This method considers the multi-label MI as the sum of the independent measures of information between the feature and each label. This approach is the adaptation of MI using the One-by-one method, defined as:

$$MI^2(f_i, \mathcal{Y}) = \sum_{y \in \mathcal{Y}} MI(f_i, y) \tag{5.19}$$

This approximation scales linearly with the number of labels at the cost of overestimating the MI between the feature and the labels. Following the previous example with a feature f_i and two labels $\{y_j, y_k\}$, this approach would consider the multi-label MI as:

$$\begin{aligned} MI^2(f_i, \mathcal{Y}) &= MI(f_i, y_j) + MI(f_i, y_k) \\ &= MI(f_i, y_j|y_k) + MI(f_i, y_k|y_j) + 2\ MI(f_i, y_j, y_k) \end{aligned} \tag{5.20}$$

Consequently, it gives double weight to the true mutual information $MI(f_i, y_j, y_k)$. Following the graphical representation of our example in Figure 5.2, this formulation would consider the area in the middle an additional time. This grants extra weight to the label correlations; thus, the error directly depends on the amount of information that a feature has of the correlated labels. Therefore, this approach would assign larger values to features with information about highly correlated labels.

5.4 Proposed methods

We propose to adopt the most efficient MI estimator (Eq. 5.19) for multi-label feature selection in a distributed environment on Apache Spark. This estimator was adopted despite the overestimation of the information shared between each feature and the labels since this amount depends directly on the correlations between the labels. These correlations are constant across all the features; therefore it is expected to have a similar impact in the selection of every feature and consequently partially canceled each other out.

Additionally, we decided to use sequential forward selection (SFS), instead of a sequential backward elimination (SBE), to build the feature subset. Sequential forward selection is a bottom-up search, which starts with an empty set and adds a feature at a time. Formally, it adds the candidate feature f_i according to some criteria.

There are two straightforward methods that perform SFS using MI: *Mutual Information Maximization* (MIM) and minimum Redundancy and Maximum Relevance (mRMR).

MIM selects the subset of features S that has the maximum amount of MI respect to the label set $\mathcal{Y}$. This subset is built iteratively by adding in each iteration the feature with the largest value of MI:

$$S = S \cup \arg\max_{f_i \in F - S} \Big[MI(f_i, \mathcal{Y}) \Big] \tag{5.21}$$

mRMR selects the optimal subset of features by maximizing the *relevance* and minimizing the *redundancy*. It defines the relevance as the MI between the selected subset of features S and the label set $\mathcal{Y}$. Consequently, it defines the redundancy as the MI between the already selected features S. Therefore, the subset of selected features is built iteratively by adding the feature that satisfies:

$$S = S \cup \arg\max_{f_i \in F - S} \Big[MI(f_i, \mathcal{Y}) - \sum_{f_j \in S} MI(f_i, f_j) \Big] \tag{5.22}$$

mRMR is expected to improve the MIM performance by prioritizing features with a lower MI score that provide new information about the label set. However, this method has a considerable increased computational complexity with respect to MIM due to the redundancy minimization which requires to compare each candidate feature with those in S.

The previous methods consider the aggregated MI between each feature and the labels. Therefore, it is not possible to discern between features with small or big variations of information with respect to the labels. We address this problem by adopting a vectorized form of the MI for each feature:

$$MI^3(f_i, \mathcal{Y}) = \{MI(f_i, y) \mid y \in \mathcal{Y}\} \tag{5.23}$$

where the k-th element represents the MI between the feature and the k-th label.

The algebraic representation of the MI allows considering either features with a small dispersion of the measures or features with larger dispersion (and possibly larger scale). We propose two methods based on this estimator which will study the possibility to efficiently minimize the redundancy: *Eucidean Norm Maximization* (ENM) and the *Geometric Mean Maximization* (GMM). Each method considers the dispersion of the vectorized MI differently and selects the best features:

ENM selects the features with the largest L^2-norm. This norm considers the sum of the square of the measures, therefore it grants more weight to features with a lot of information about some labels independently of their possible low values about other labels.

$$\begin{gathered} S = S \cup \underset{f_i \in F - S}{\arg\max} \left[|MI^3(f_i, \mathcal{Y})| \right] \\ |MI^3(f_i, \mathcal{Y})| = \sqrt[2]{MI(f_i, y_j) + \ldots + MI(f_i, y_q)} \mid 1 \leq j \leq q \end{gathered} \tag{5.24}$$

GMM selects the features with the largest geometric mean. The geometric mean considers the product of the measures and then splits them with a root. The conceptual difference is seeing each measure as a scaling factor, which combines by increasing each other multiplicatively. This prioritizes the selection of features with the same amount of

Table 5.3.: Example of MI between four features and three labels. The bottom presents the MI of each label depending on which strategy is used to select two features.

	y_1	y_2	y_3	Eucl. norm	Geo. mean	Sum
f_1	0.95	0.25	0.05	**0.98**	0.23	**1.25**
f_2	0.20	0.10	0.95	**0.98**	0.27	**1.25**
f_3	0.45	0.35	0.45	0.73	**0.41**	**1.25**
f_4	0.40	0.60	0.25	0.76	**0.39**	**1.25**

information about each label independently of the scale of this amount.

$$\begin{aligned} S &= S \cup \arg\max_{f_i \in F-S} \Big[G(MI^3(f_i, \mathcal{Y})) \Big] \\ G(MI^3(f_i, \mathcal{Y})) &= \left\{ \prod_{j=1}^{q} MI(f_i, y_j) \right\}^{\frac{1}{q}} \mid MI(f_i, y_j) > 0,\ 1 \leq j \leq q \end{aligned} \tag{5.25}$$

Table 5.3 presents an example with four features and their mutual information scores with three labels. The right side of the table indicates the Euclidean norm, the geometric mean, and the sum of the measures respectively. Notice that the features have the same total amount of mutual information, therefore the traditional MIM method would consider them equally. Assuming the algorithm would require to select the best two features, ENM would give preference to the first and second feature, while GMM would select the third and fourth feature.

It can be seen that ENM gives priority to features with large measures while risking to have underrepresented labels. In our example, the selected subset from ENM would have a considerably smaller amount of information about the label y_2 than it would have about the other labels. GMM selects the features with smaller dispersion on the measures. Therefore, the selected subset would have a balanced amount of information among the labels but at the cost of having less information about y_1 and y_3.

Figure 5.3 illustrates the possible values of MI between a given feature f and three labels $\{y_1, y_2, y_3\}$ with a total sum of 1.0. The features whose MI values are located near the axis are preferred by ENM since they represent the cases where there are more differences

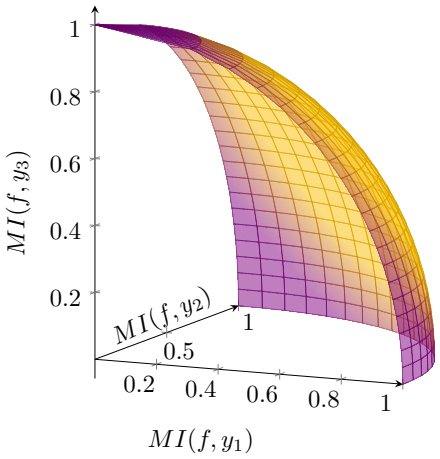


Fig. 5.3.: The surface represents the combinations of MI between a given feature and three labels with a total sum of 1.0. ENM and GMM would select the features with MI values in the purple and yellow areas respectively.

between the independent measures. While the features located towards the middle of the surface are selected by GMM and constitute features with smaller differences between their independent measures.

5.4.1 Distributed implementation for continuous features on Apache Spark

The computation of the MI is a computationally expensive operation, especially for the continuous variables. Both the MI between continuous variables, as well as the MI between a continuous and a discrete variable, rely on multiple searches of nearest neighbors as it was detailed in 5.2.2. First, to find the maximum distances among all the instances, and then to use that distance as the maximum range to find the neighbors under certain conditions.

The computation of nearest neighbor searches is an expensive and complex operation, especially in a distributed environment. Therefore we propose a distributed approach to compute the differential mutual information measures using Apache Spark. This measure relies on the computation of uni-dimensional (for $MI(f, \mathcal{Y})$) and bi-dimensional (for $MI(f, f_j)$) searches. We consider that these searches can be performed locally, thus speeding up the computation while minimizing the network computation.

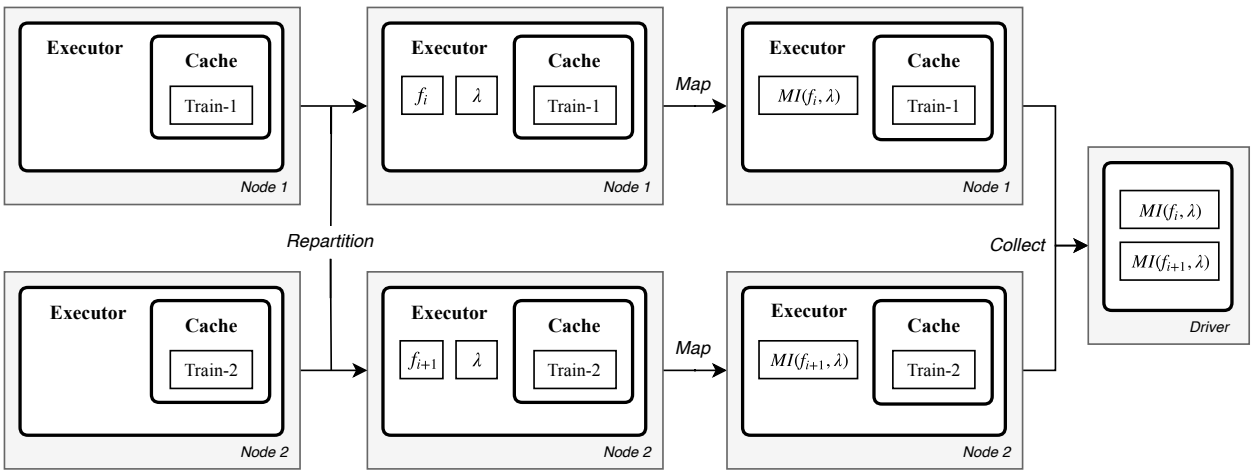


Fig. 5.4.: Distributed MI between a two features $\{f_i, f_{i+1}\}$ and a variable λ which would either be a label in $MI(f, \mathcal{Y})$ or a previously selected feature in $MI(f, f_j)$.

Figure 5.4 presents a schematic view of how the MI between two features $\{f_i, f_{i+1}\}$ and a variable λ is performed. This schema reflect the $MI(f, \mathcal{Y})$ if λ represents the set of labels $\mathcal{Y}$, while it would represent the $MI(f, f_j)$ whenever λ represents the previously selected feature. The figure shows how the values of each of the features are extracted from the local partitions and are being sent to different executors. Once all the information is present locally, each executor can perform the corresponding nearest neighbor searches with all the information avoiding any exchange of information over the network. Once each executor computes their local MI, the results will be sent over to the *Driver* which then will decide how to combine them.

5.4.2 Distributed implementation for discrete features on Apache Spark

The computation of MI, or any other score metric, using discrete features is less challenging than for continuous features, but still a complex operation, especially for a large number of features. Therefore, we proposed a distributed computation of any score for discrete features on Apache Spark in order to handle large-scale datasets.

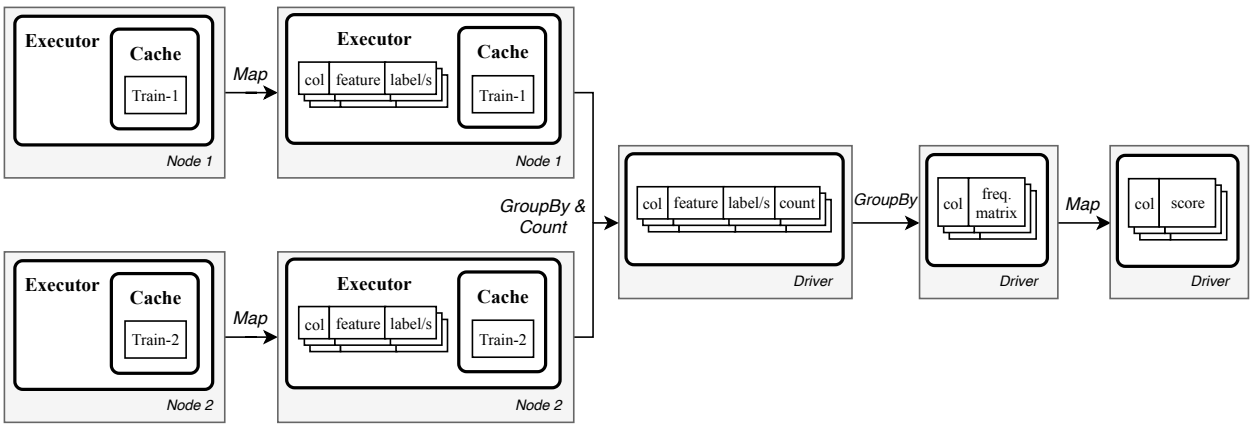


Fig. 5.5.: Distributed MI between a two features $\{f_i, f_{i+1}\}$ and a variable λ which would either be a label in $MI(f, \mathcal{Y})$ or a previously selected feature in $MI(f, f_j)$.

Figure 5.5 presents a schematic view of how the score between two discrete features $\{f_i, f_{i+1}\}$ and their corresponding output space is performed. This schema only uses two features in two nodes for simplicity, nevertheless, this approach is scalable to as many features and nodes as needed.

This approach starts with a series of instances partitioned over multiple executors, and possibly over multiple nodes as well. Each of those instances is then expanded to create triplets of $(column, feature, label/s)$, therefore it will create as many triplets as feature values. Then, all the triplets with the same values are grouped and counted. The results are then collected locally in the *Driver*, therefore all the consecutive operations are performed locally. The aggregated groups and their counts can then be additionally grouped by column, and then all the information available about the column can be used to create a frequency matrix where the columns represent the labels and the rows the different feature values. This frequency matrix is unique per column and it can be used to compute the corresponding score measure (MI or χ^2) for the feature.

5.5 Experimental setup

This section presents the experimental environment, datasets, and methods used. Experiments were executed on a cluster composed by 8 Intel Xeon CPU E7-8894v4 with 24 cores and 6 TB of memory. Experiments were run using Spark 2.4.0 on CentOS 7.4.

5.5.1 Datasets

Table 5.4 shows the multi-label datasets evaluated along with the number of instances, features, labels, cardinality, and density [49]. These datasets have been collected from the *Knowledge Discovery and Intelligent Systems* (KDIS) dataset repository[1], although originally, they could be found on the MULAN and MEKA repositories.

Table 5.4.: Summary description of the benchmark datasets.

Type	Dataset	Instances	Features	Labels	Cardinality	Density
Continuous	Emotions	593	72	6	1.8685	0.3114
	Birds	645	260	19	1.0140	0.0534
	Scene	2,407	294	6	1.0740	0.1790
	Yeast	2,417	103	14	4.2371	0.3026
	Human	3,106	440	14	1.1851	0.0847
	Eukaryote	7,766	440	22	1.1456	0.0520
	Mediamill	43,907	120	101	4.3756	0.0433
	Nus-wide	269,648	129	81	1.8685	0.0230
Discrete	Medical	978	1,449	45	1.2454	0.0277
	Slashdot	3,782	1,079	22	1.1809	0.0537
	Bibtex	7,395	1,836	159	2.4019	0.0151
	Yelp	10,806	671	5	1.6383	0.3277
	Corel16k	13,766	500	153	2.8587	0.0187
	Delicious	16,105	500	983	19.0200	0.0193
	20NG	19,300	1,006	20	1.0289	0.0514
	TMC2007	28,596	500	22	2.2196	0.1009
	Bookmarks	87,856	2,150	208	2.0281	0.0098
	IMDB	120,919	1,001	28	1.9997	0.0714
Synthetic	Hyperspheres	500,000	50	20	3.6665	0.1833
	Hypercubes	500,000	50	20	2.8760	0.1437

[1]KDIS: `http://www.uco.es/kdis/mllresources`

The *Hyperspheres* and *Hypercubes* datasets have been generated using a multi-label data generator [211][2]. Both datasets have been generated using 25 relevant features, 15 irrelevant features, and 10 redundant features, with a noise level of 0.05 and 0.2 for *Hyperspheres* and *Hypercubes*, respectively. The *Hyperspheres* generator randomly creates hyperspheres of a minimum and maximum radius of 0.05 and 0.2 respectively, while *Hypercubes* randomly generates hypercubes with a half-edge between 0.15 and 0.8 respectively (default parameters).

To appreciate the benefits of the proposed methods, there is a wide variety of datasets that cover the range of different characteristics in real scenarios. The continuous benchmark datasets have been normalized while the discrete benchmark datasets consist of binary features. The synthetic datasets consist of continuous features which have either been discretized or normalized, depending on the type of experiment performed.

The train and test splits are built using 3 equally size folds, where two the folds are used to train the algorithm and one fold is left out to test the model. The folds are built using a stratified division [174], where each unique subset of labels present in the instances is considered as a fictitious label, and then the desired percentage of instances is extracted from each of those labels. This ensures that each of the folds has the same data distribution as the full dataset.

5.5.2 Methods and parameters

We compare our methods to the state-of-the-art multi-label feature selection methods [185], [187], [195], [196], [200]: Entropy-based Label Assignment using χ^2 (ELA) [78], Label Powerset using χ^2 (LP) [188], Pruned Problem Transformation using χ^2 (PPT) [185], Maximum Mutual Information (MIM), and minimum Redundancy and Maximum Relevance (mRMR). The performance of the feature subset selected by each method was evaluated using the distributed Multi-label k Nearest Neighbors (ML-KNN) [87], [212] classifier with $k = 5$.

[2]MLdatagen: `http://sites.labic.icmc.usp.br/mldatagen/`

The discretization of the continuous features required by reference methods greatly affects the results of the selection process [213]. There is a wide range of discretization methods used in previous studies mentioned on Section 5.2. However, we decided to discard discretization methods that aim to maximize the accuracy in the discretized space since they would be adapting and biasing the distribution of the bins making unfair a subsequent feature selection comparison.

We decided to faithfully represent the prime continuous data distribution in a discrete space. For that purpose, we chose the Freedman-Diaconis rule [48], which is used to construct histograms using continuous data. The Freedman–Diaconis rule is designed to minimize the difference between the area under the empirical probability distribution and the area under the theoretical probability distribution.

5.5.3 Evaluation metrics

The evaluation metrics for multi-label learning differ from those in traditional classification. The evaluation metrics employed to compare the multi-label learning methods were introduced in Section 2.1.3. However, due to the limited space in the manuscript, we show performance plots for the most restrictive metric in multi-label (subset accuracy).

Summarized results and statistical analysis for 12 multi-label metrics are provided at the end of Sections 5.6.2 and 5.7.2, and detailed results on all datasets are available in [3] and [4] to facilitate the reproducibility and future comparisons.

5.6 Experimental results for continuous features

This section presents and discusses the experimental results produced by the methods which can be considered by two factors: *domain of the data* and *label correlations*. ELA, LP, and PPT require discrete features, while MIM and mRMR use the original continuous data.

[3] Detailed results for continuous features: `http://people.vcu.edu/~acano/MI/`

[4] Detailed results for discrete features: `http://people.vcu.edu/~acano/MI-ENM-GMM/`

Moreover, LP and PPT consider label correlations, while ELA, MIM, mRMR, ENM, and GMM handle the labels independently.

None the methods in the comparison offer the optimal size for a feature subset, therefore we tested the predictive power by varying the size of the feature subsets in the range 2 to 50. Figures 5.6 and 5.8 present the evolution of the subset accuracy over the increasing number of selected features. The bold dashed grid line represents the subset accuracy of the ML-KNN classifier as a reference baseline using all the features.

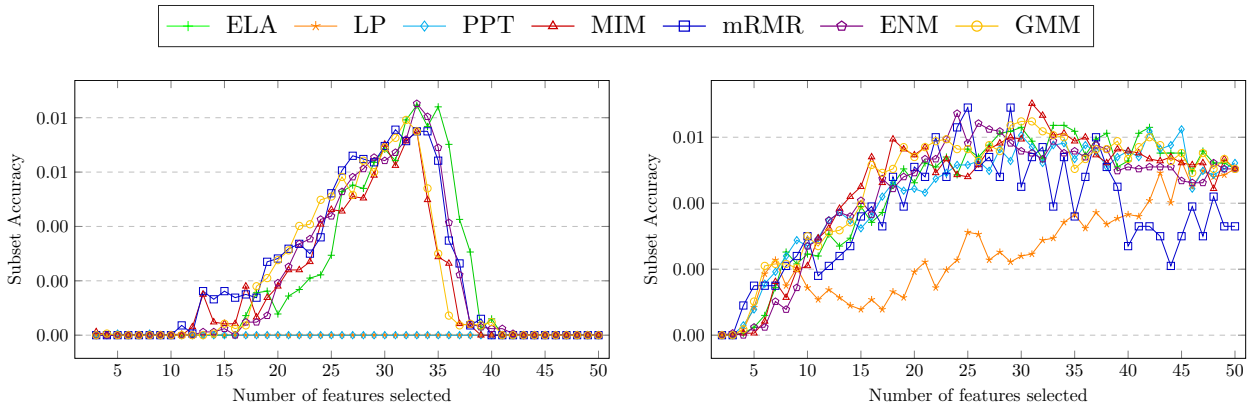
5.6.1 Synthetic datasets comparison

This experiment focuses on evaluating the subset accuracy of the synthetic datasets over the increasing number of features selected. This experiment allows us to determine whether there is a priority of the selection of relevant features over irrelevant or redundant.

Figure 5.6 presents the variations on the subset accuracy over the *Hyperspheres* and *Hypercubes* datasets.

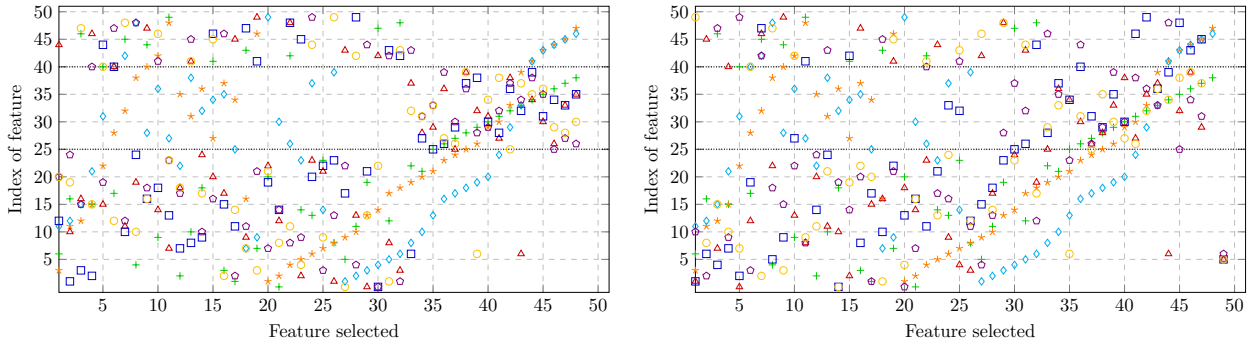
The subset accuracy obtained in *Hyperspheres* by all the methods, except PPT, is similar. However, it is important to notice which methods rise their accuracy earlier and drop it later, meaning that they correctly select the first features while delaying the insertion of irrelevant features. We can observe that mRMR and MIM are the first methods to improve their base accuracy, however, MIM fails to maintain it and its replaced by ENM and GMM. Regarding the insertion of irrelevant information, we can observe that ELA is the best method followed by ENM.

On the other hand, the subset accuracy obtained in *Hypercubes* by the different methods varies more. Here, the best performing algorithms are MIM and GMM which successfully maintain their performance above the other methods.



(a) Hyperspheres dataset (b) Hypercubes dataset

Fig. 5.6.: Subset accuracy obtained selecting up to 50 features on synthetic datasets.



(a) Hyperspheres dataset (b) Hypercubes dataset

Fig. 5.7.: Scatter plot that shows the order of feature selection. The indices $[0, 24]$, $[25, 39]$, and $[40, 49]$ indicate relevant, redundant, and irrelevant features, respectively.

Figure 5.7 indicates the order in which each feature has been selected for the *Hyperspheres* and *Hypercubes* datasets. The x-axis indicates the number of the last feature selected, while the y-axis indicates the index of the feature that was selected. The features whose indices range from 0 to 24 are relevant, from 24 to 39 are irrelevant, and from 40 to 49 are redundant.

LP and PPT are the worst performing methods since they insert irrelevant features earlier in the selection process. On both datasets, by the time of the selection of feature 25, there is already a considerable amount of irrelevant features selected. On the other hand, ELA succeeds in delaying the selection of irrelevant features towards the end of the process.

Table 5.5.: Subset accuracy comparison by dataset averaged across all feature subset sizes.

Dataset	ELA	LP	PPT	MIM	mRMR	ENM	GMM
Emotions	**0.1857**	0.1629	0.1640	0.1451	0.1659	0.1073	0.1504
Birds	0.4857	0.4857	0.4888	**0.4956**	0.4812	0.4900	0.4897
Scene	0.0370	0.0373	0.0343	0.2284	**0.2778**	0.1794	0.1229
Yeast	0.0878	0.0844	0.1082	0.1192	0.0994	0.1085	**0.1219**
Human	0.0366	0.0070	0.0115	0.0572	**0.0589**	0.0010	0.0542
Eukaryote	0.0235	0.0121	0.0207	0.0402	0.0395	0.0012	**0.0472**
Mediamill	0.0833	0.0827	0.0831	**0.0938**	0.0925	0.0800	0.0859
Nus-wide	**0.2182**	0.2165	0.2158	0.2140	0.2144	0.2171	0.2163
Hyperspheres	**0.0020**	0.0000	0.0000	0.0017	0.0017	**0.0020**	0.0019
Hypercubes	0.0044	0.0025	0.0043	0.0045	0.0037	0.0043	**0.0046**
Average	0.1164	0.1091	0.1131	0.1400	**0.1434**	0.1191	0.1295

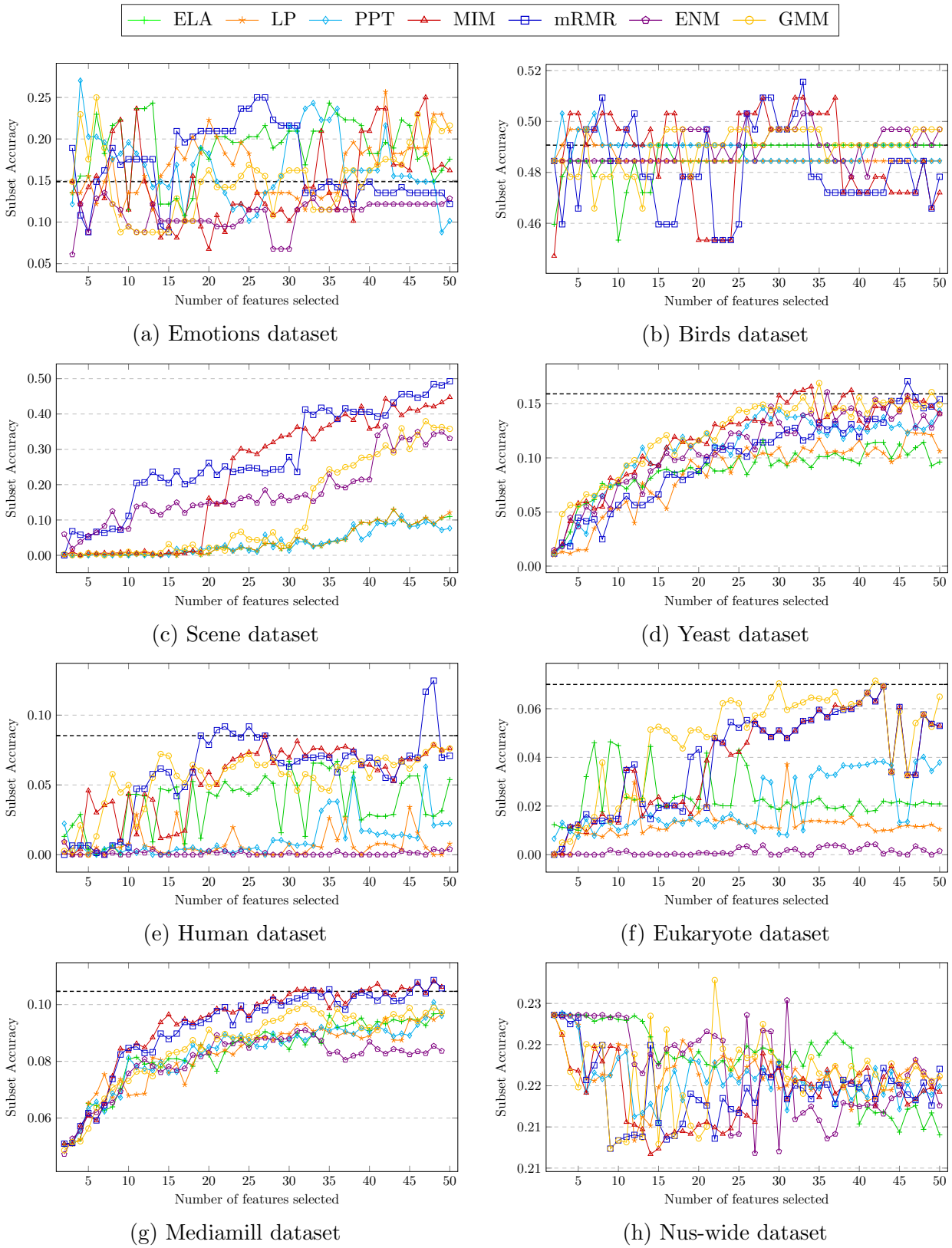
All the mutual information based methods manage to delay the selection of irrelevant features towards the end of the selection process. However, they differ in the order in which they introduce redundant features. Attending at each feature individually, by looking at the same value in the y-axis, we can see that in many cases ENM and GMM delay the insertion of a redundant feature already selected by MIM and mRMR.

5.6.2 Subset accuracy comparison

This experiment illustrates the overall behavior and the quality of the predictions of each method over benchmark datasets that represent a wide range of multi-label scenarios.

Figure 5.8 presents the evolution of subset accuracy over the number of selected features, where the bold black dashed line indicates the subset accuracy of the ML-KNN classifier as a reference baseline using all the features. Table 5.5 presents the detailed subset accuracy results for each dataset averaged across all feature subset sizes.

Emotions and *Birds* datasets present inconclusive results. Nevertheless, we can highlight the overall good performance of ELA, and the competitive results obtained by mRMR especially for middle sizes of subsets on *Emotions*.



(a) Emotions dataset

(b) Birds dataset

(c) Scene dataset

(d) Yeast dataset

(e) Human dataset

(f) Eukaryote dataset

(g) Mediamill dataset

(h) Nus-wide dataset

Fig. 5.8.: Subset accuracy obtained selecting up to 50 continuous features on the datasets.

Yeast and *Mediamill* show similar trends for predictions of all the methods. Despite the small differences, we can observe a better performance of GMM on *Yeast* and of both, MIM and mRMR, on *Mediamill*.

Nus-wide presents a clear advantage to ELA, however, its performance declines after 40 features obtaining lower subset accuracy than the other methods in the comparisons. GMM presents the best results on average due to the quick improvement of its accuracy after selecting 12 features.

Scene, *Human* and *Eukaryote* highlight the superior performance of the methods based on mutual information. mRMR produced better predictions with a trend on its performance similar to MIM and GMM, even being surpassed by GMM on datasets such as *Eukaryote*.

We conclude that MIM, mRMR, and GMM achieve competitive results on subset accuracy, while also producing the most consistent results. This fact is surprising since they consider the labels independently, in a similar manner than ELA which achieves considerably worst performance. This difference validates our proposed mutual information score for multi-label problems and highlights the possibilities of considering the mutual information measure of each label individually.

Statistical analyses allow us to provide a more detailed comparison of the relative performance of the algorithms. Table 5.6 presents the results of the Bonferroni-Dunn test (multiple comparison statistical test) for each multi-label metric, as well as the rank of each metric (the lower the better), and the overall meta-rank (ranks of the ranks). This test assumes that two methods are significantly different if their ranks differ by at least some critical distance. In this case, the critical distance is 0.3640 for a statistical significance level of α of 0.05. The table includes the figures that highlight the critical distance (gray area) among algorithms in order to be considered statistically different. The methods that fall out of this area are claimed to perform statistically worse than the control method. The test indicates that MIM and GMM cannot be claimed as significantly different for all the quality metrics, while the others are statistically worse for all of some of the quality metrics.

Table 5.6.: Algorithm ranks for each of the multi-label performance metrics across all datasets and feature sizes.

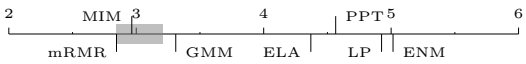
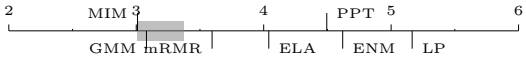
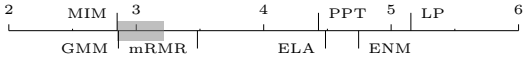
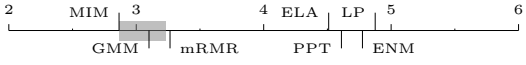
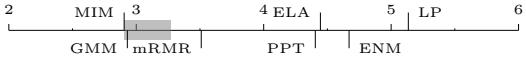
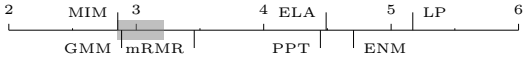
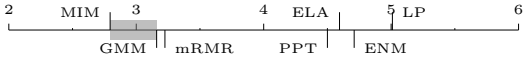
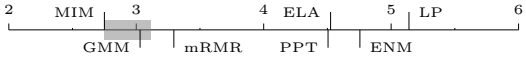
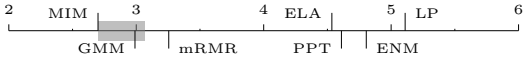
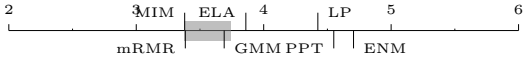
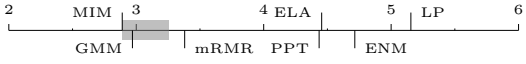
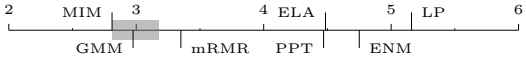
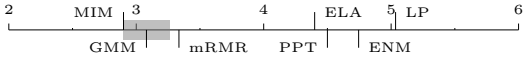
Metric	ELA	LP	PPT	MIM	mRMR	ENM	GMM	Bonferroni-Dunn test
Hamming Loss	4.37	4.93	4.57	2.97	**2.84**	5.02	3.31	
Subset accuracy	4.04	5.17	4.50	**3.01**	3.59	4.62	3.08	
Ex.-based accuracy	4.48	5.16	4.43	**2.85**	3.48	4.75	2.86	
Ex.-based precision	4.51	4.87	4.61	**2.86**	3.27	4.78	3.10	
Ex.-based recall	4.44	5.14	4.41	**2.91**	3.51	4.67	2.93	
Ex.-based F1	4.49	5.17	4.44	**2.85**	3.45	4.70	2.88	
Macro precision	4.59	5.01	4.50	**2.80**	3.23	4.71	3.16	
Macro recall	4.52	5.14	4.51	**2.75**	3.30	4.75	3.03	
Macro F1	4.53	5.11	4.61	**2.70**	3.25	4.80	2.99	
Micro precision	3.86	4.43	4.55	**3.38**	3.38	4.71	3.69	
Micro recall	4.45	5.15	4.44	**2.89**	3.38	4.71	2.97	
Micro F1	4.48	5.16	4.47	**2.81**	3.35	4.75	2.98	
Meta-rank	4.40	5.04	4.50	**2.90**	3.34	4.75	3.08	

Table 5.7.: Wilcoxon statistical test analysis for subset accuracy. MIM, mRMR, ENM and GMM vs reference methods (p-values < 0.01 indicate statistically significant differences).

Algorithm vs	MIM	mRMR	ENM	GMM
ELA	5.33E-17	2.45E-10	8.85E-03	6.69E-16
LP	2.00E-35	3.14E-30	8.40E-02	1.21E-34
PPT	2.20E-27	6.12E-14	5.62E-03	5.61E-27
MIM	0	6.10E-04	2.58E-32	1.53E-02
mRMR	6.10E-04	0	5.35E-22	3.41E-01
ENM	2.58E-32	5.35E-22	0	4.51E-19
GMM	1.53E-02	3.41E-01	4.51E-19	0

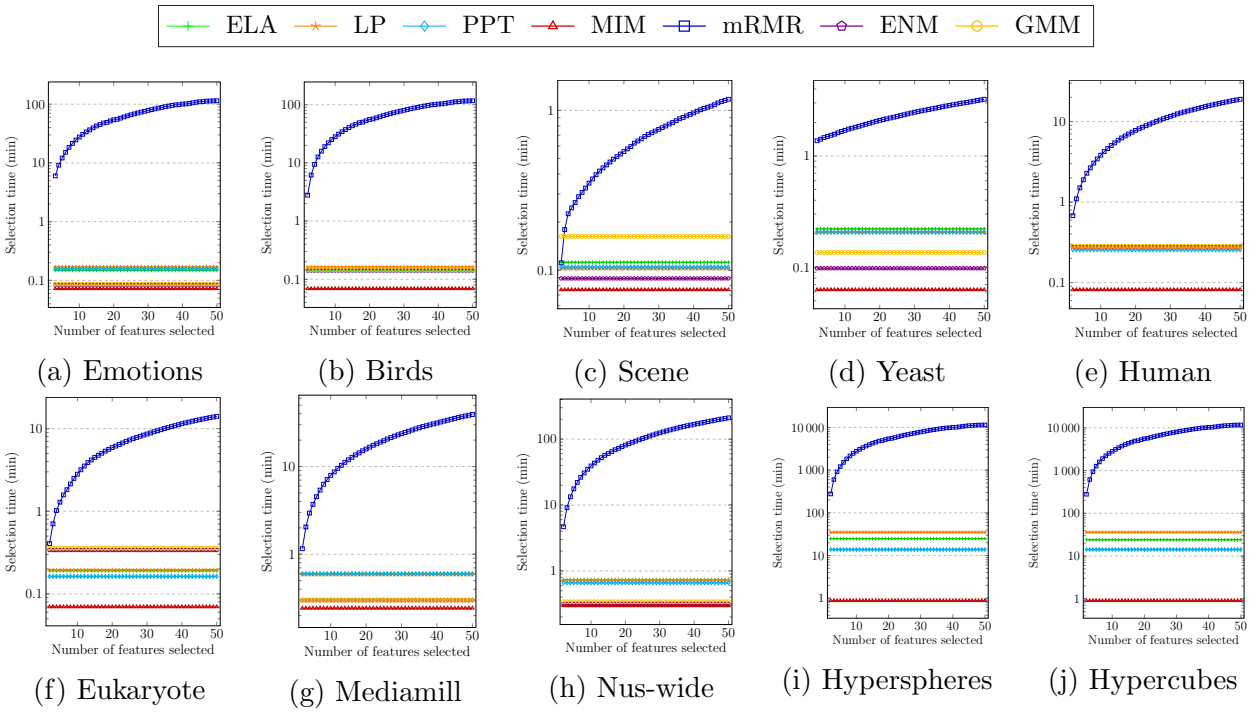


Fig. 5.9.: Runtime obtained selecting up to 50 features on all the datasets.

Similarly, Table 5.7 presents the results (p-values) of the Wilcoxon rank sum test for subset accuracy, which allows us to identify whether there are significant differences in a pairwise comparison between two algorithms. A p-values < 0.01 indicates significant differences between the two methods compared. According to this test, there are significant differences between all the methods except for GMM respect to MIM and mRMR.

5.6.3 Runtime comparison

This experiment evaluates the runtime required to select a specific number of features. The reported times for ELA, LP, and PPT methods include both the discretization and the selection phases. On the other hand, MIM, mRMR, GMM, and ENM only report the time to select the features since they do not require of a discretization step. Figure 5.9 presents the runtime in minutes and on a logarithmic scale, required by the methods to select the respective number of features.

ELA, LP, and PPT exhibit a constant runtime with respect to the number of features. The difference in performance between the methods is produced by the subset cardinality of the dataset. ELA usually has lower selection times since it considers the labels independently, followed by LP which considers each subset as a unique class, and followed by PPT which redistributes the information of the most infrequent subsets and then executes an LP. However, in some of the small datasets without a large number of labels PPT can outperform the other methods.

mRMR, MIM, GMM, and ENM show opposite results in terms of runtime. The methods do not rely on discretized data but they directly use the original continuous features. MIM achieves the fastest performance, not only compared to mRMR but also with respect to all the reference methods. MIM has a constant selection time which is followed closely by ENM and GMM, respectively. This small difference is due to the different processing of individual mutual information measures. Nevertheless, the three methods outpace mRMR by thousands of times specially selecting a large number of features in the biggest datasets. These results validate our proposed differential method as the best approach for continuous features. On the other hand, mRMR exhibits an extremely big difference compared to the rest of the methods. This is produced by the minimization of the redundancy, which is done by comparing each feature candidate to the previously selected feature. Overall, the mutual information methods showed to significantly improve the predictive power of multi-label feature selection in continuous attributes compared to reference methods.

5.7 Experimental results for discrete features

This section presents and discusses the experimental results produced by the methods over datasets with discrete features attending to multiple factors: the inclusion of label correlations, the preference of relevant features over irrelevant/redundant, and the runtime to select a determined feature subset.

None of the methods in the comparison offer the optimal size for a feature subset, therefore we tested their predictive power by varying the size of the selected subsets in the range 2 to 50. Figures 5.10 and 5.12 present the evolution of the subset accuracy over the increasing number of selected features. The bold dashed grid line represents the subset accuracy of the Ml-knn classifier as a reference baseline using all the features.

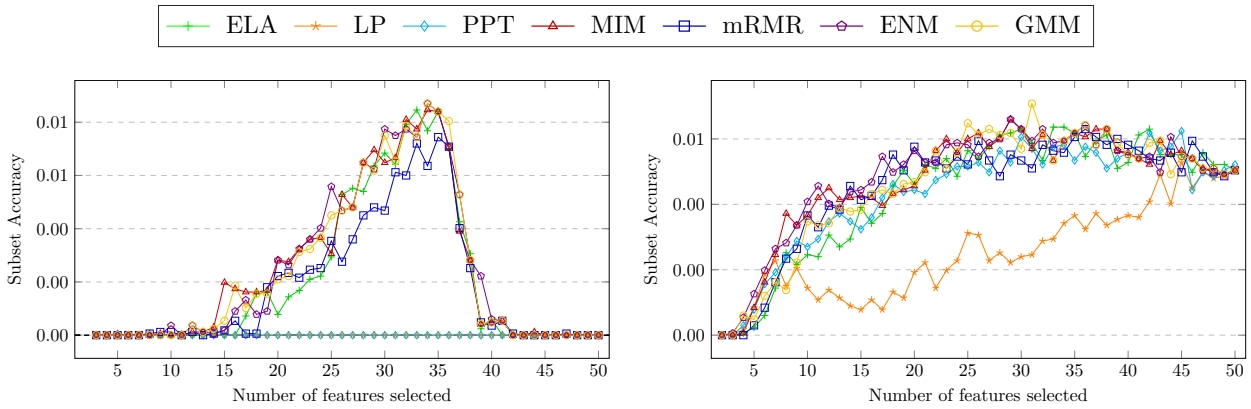
5.7.1 Synthetic datasets comparison

This experiment focuses on evaluating the subset accuracy of the synthetic datasets over the increasing number of features selected. This experiment allows us to determine whether there is a priority of the selection of relevant features over irrelevant or redundant.

Figure 5.10 presents the variations on the subset accuracy over the *Hyperspheres* and *Hypercubes* datasets. The subset accuracy obtained by the mutual information based methods is similar for both datasets, with perhaps the exception of mRMR for *Hyperspheres*. However, it is remarkable the low accuracy reported by LP and PPT for *Hyperspheres* and LP for *Hypercubes*. This is expected to be the result of selecting a series of irrelevant features, especially in the early process.

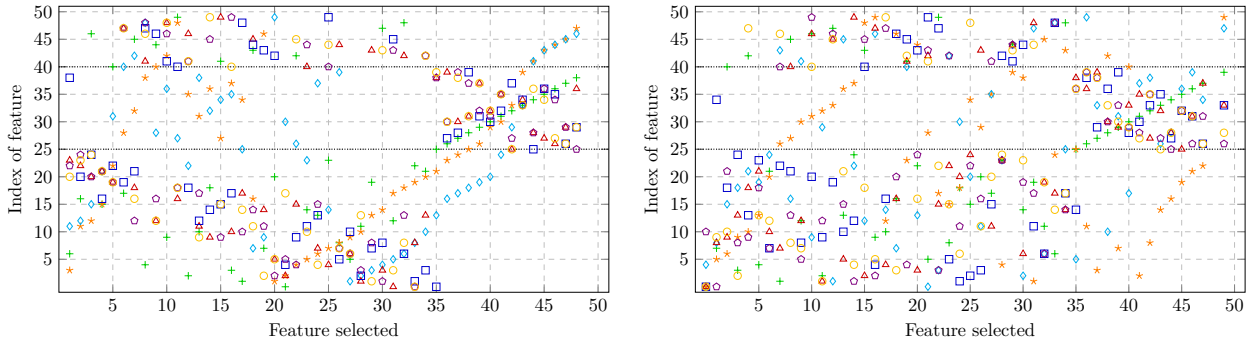
Figure 5.11 indicates the order in which each feature has been selected for the *Hyperspheres* and *Hypercubes* datasets. The x-axis indicates the number of the last feature selected, while the y-axis indicates the index of the feature that was selected. The features whose indices range from 0 to 24 are relevant, from 24 to 39 are irrelevant, and from 40 to 49 are redundant.

We can observe that LP and PPT are the first methods to select irrelevant features. The amount of irrelevant features is especially high for the *Hyperspheres* dataset, by the time of the selection of feature 25 there is already a considerable amount of irrelevant features selected. On the other hand, ELA succeeds in delaying the selection of the irrelevant features towards the end of the process, however, is the first method to incur into the selection of redundant features.



(a) Hyperspheres dataset (b) Hypercubes dataset

Fig. 5.10.: Subset accuracy obtained selecting up to 50 features on synthetic datasets.



(a) Hyperspheres dataset (b) Hypercubes dataset

Fig. 5.11.: Scatter plot that shows the order of feature selection. The indices $[0, 24]$, $[25, 39]$, and $[40, 49]$ indicate relevant, redundant, and irrelevant features, respectively.

All the mutual information based methods manage to delay the selection of irrelevant features towards the end of the selection process. However, they differ in the order in which they introduce redundant features. Attending at each feature individually, by looking at the same value in the y-axis, we can see that for *Hyperspheres* mRMR inserts redundant features earlier than the other methods, while for *Hypercubes* mRMR successfully delays the insertion of redundant features followed very closely by ENM.

Table 5.8.: Subset accuracy comparison by dataset averaged across all feature subset sizes.

Dataset	ELA	LP	PPT	MIM	mRMR	ENM	GMM
Medical	0.2211	0.0442	0.2161	0.5247	0.5247	**0.5832**	0.4396
Slashdot	0.0733	0.0262	0.0531	0.2035	0.2035	**0.2256**	0.2091
Bibtex	0.0251	0.0299	0.0145	0.0777	0.0777	**0.1178**	0.0596
Yelp	0.2866	0.2218	0.2218	0.3212	0.3212	0.3168	**0.3302**
Corel16k	0.0022	0.0000	0.0006	0.0006	0.0006	**0.0014**	0.0003
Delicious	0.0560	0.0226	0.0000	0.0631	0.0408	0.0557	**0.0764**
20NG	0.0161	0.0172	0.0180	0.2182	0.2182	**0.2515**	0.1580
TMC2007	0.0956	0.0963	0.0956	**0.1710**	**0.1710**	0.1698	0.1623
Bookmarks	0.0606	0.0606	0.0000	0.1553	0.1553	0.1496	**0.1607**
IMDB	0.0004	0.0001	0.0004	**0.0025**	**0.0025**	0.0020	0.0012
Hyperspheres	0.0020	0.0000	0.0000	0.0023	0.0018	**0.0023**	**0.0023**
Hypercubes	0.0044	0.0025	0.0043	0.0047	0.0045	**0.0048**	0.0046
Average	0.1164	0.0892	0.1019	0.1403	0.1385	**0.1507**	0.1286

5.7.2 Subset accuracy comparison

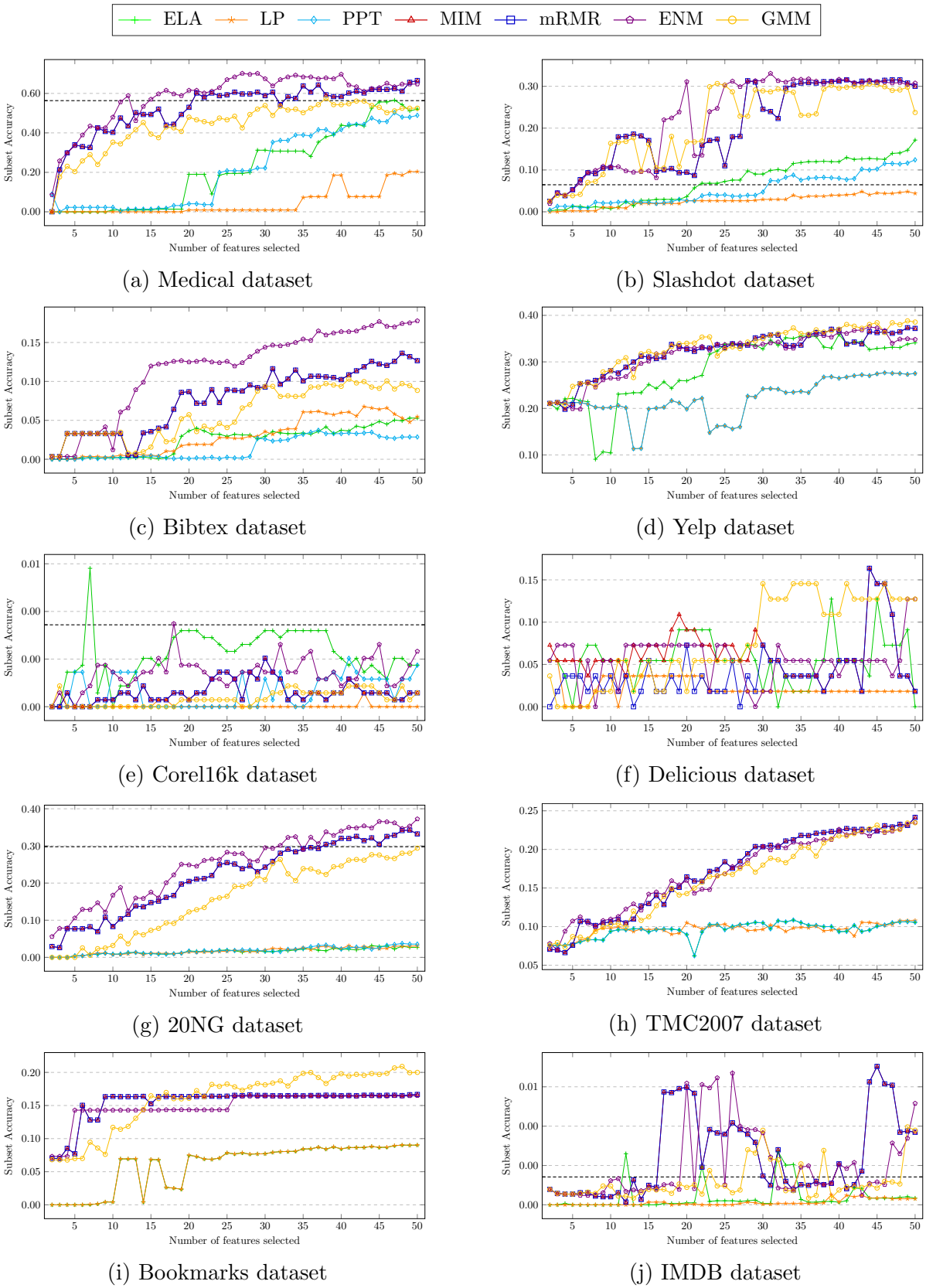
This experiment illustrates the overall behavior and the quality of the predictions of each method over 10 benchmark datasets that represent a wide range of multi-label scenarios.

Figure 5.12 presents the evolution of subset accuracy over the number of selected features, where the bold black dashed line indicates the subset accuracy of the ML-KNN classifier as a reference baseline using all the features. Table 5.8 presents the detailed subset accuracy results for each dataset averaged across all feature subset sizes.

The methods based on the χ^2 score report considerably worst results in comparison to those using mutual information. We can observe that ELA is the best χ^2 based method outperforming the other methods on *Corel16k* and presenting a rapid increment in the subset accuracy in other datasets such as *Medical*.

ENM presents the overall best results of all the methods. We can observe that ENM clearly outperforms the other methods on *Medical*, *Bibtex*, and *20NG*. While on other datasets it follows the trend of the other mutual information based methods such as *Slashdot*, *Yelp*, and *TMC2007*. Nevertheless, even on the datasets where ENM performs similarly, we can still observe a small improvement over them.

GMM obtains a lower performance than the other mutual information based methods



(a) Medical dataset

(b) Slashdot dataset

(c) Bibtex dataset

(d) Yelp dataset

(e) Corel16k dataset

(f) Delicious dataset

(g) 20NG dataset

(h) TMC2007 dataset

(i) Bookmarks dataset

(j) IMDB dataset

Fig. 5.12.: Subset accuracy obtained selecting up to 50 discrete features on the datasets.

for most of the datasets, with two exceptions:*Delicious* and *Bookmarks*. GMM presents a competitive subset accuracy, especially for the larger subset features. These datasets have the largest number of labels, 983 and 208 for *Delicious* and *Bookmarks*, respectively. Therefore the advantage over the other methods can be founded by the fact that this method focuses on having a balanced amount of information about all the labels, which might end up selecting redundant features, but also ensures that all the labels are represented.

We conclude that ENM presents the most competitive results, especially for a smaller number of labels. This is expected since this method might select feature subsets that under-represent some labels. On the other hand, GMM only outperforms ENM for the datasets with the largest number of labels. This is due to the binary nature of the features which make most of the individual mutual information measures in each feature to tend to zero.

Statistical analyses allow us to provide a more detailed comparison of the relative performance of the algorithms. Table 5.9 presents the results of the Bonferroni-Dunn test for each multi-label metric, as well as the rank of each metric, and the overall meta-rank. This test assumes that two methods are significantly different if their ranks differ by at least some critical distance. In this case, the critical distance is 0.3077 for a statistical significance level of α of 0.05. The results of the test indicate that ENM is the best performing algorithm since it ranks first for all the quality metrics. ENM is statistically different for the most strict metrics, such as subset accuracy, but it cannot be assured the same for the less restrictive metrics.

Similarly, Table 5.10 presents the results (p-values) of the Wilcoxon rank sum test for subset accuracy, which allows us to identify whether there are significant differences in a pairwise comparison between two algorithms. A p-values < 0.01 indicates significant differences between the two methods compared. According to this test both ENM and GMM show to have significant differences compared to reference methods, as well as with the other mutual information based methods

Table 5.9.: Algorithm ranks for each of the multi-label performance metrics across all datasets and feature sizes.

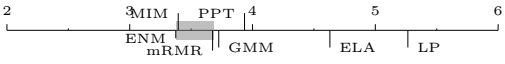
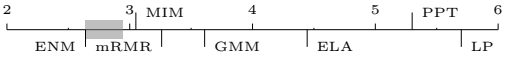
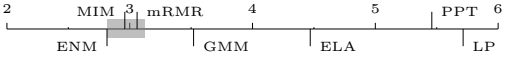
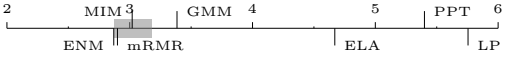
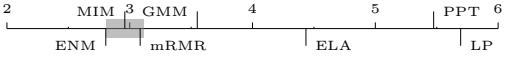
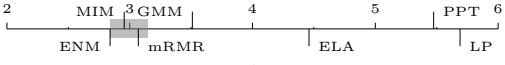
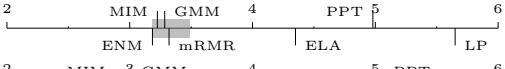
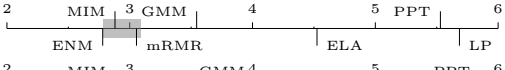
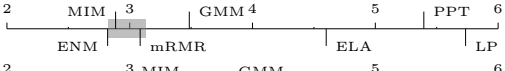
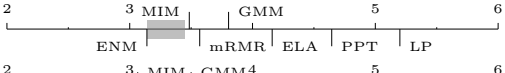
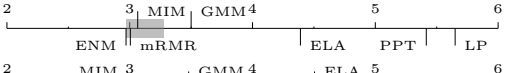
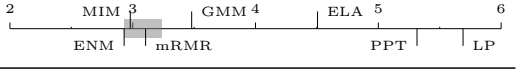
Metric	ELA	LP	PPT	MIM	mRMR	ENM	GMM	Bonferroni-Dunn test
Hamming Loss	4.63	5.26	3.94	3.39	3.68	**3.38**	3.73	
Subset accuracy	4.44	5.70	5.30	3.05	3.26	**2.64**	3.61	
Ex.-based accuracy	4.47	5.72	5.46	2.96	3.06	**2.81**	3.52	
Ex.-based precision	4.67	5.76	5.40	3.02	2.90	**2.87**	3.38	
Ex.-based recall	4.44	5.69	5.48	2.96	3.08	**2.80**	3.55	
Ex.-based F1	4.46	5.69	5.48	2.95	3.07	**2.84**	3.51	
Macro precision	4.35	5.65	4.98	3.22	3.32	**3.19**	3.29	
Macro recall	4.53	5.68	5.53	2.88	3.06	**2.78**	3.54	
Macro F1	4.60	5.74	5.39	2.88	3.08	**2.82**	3.48	
Micro precision	4.16	5.20	4.64	3.48	3.57	**3.14**	3.81	
Micro recall	4.39	5.65	5.42	3.06	3.01	**2.97**	3.50	
Micro F1	4.50	5.69	5.32	2.98	3.11	**2.92**	3.48	
Meta-rank	4.47	5.62	5.19	3.07	3.18	**2.93**	3.53	

Table 5.10.: Wilcoxon statistical test analysis for subset accuracy. MIM, mRMR, ENM and GMM vs reference methods (p-values < 0.01 indicate statistically significant differences).

Algorithm vs	MIM	mRMR	ENM	GMM
ELA	1.11E-32	2.08E-26	8.95E-37	3.14E-31
LP	1.95E-50	2.84E-50	3.00E-55	3.46E-49
PPT	2.57E-47	8.20E-47	1.53E-52	8.15E-47
MIM	0	2.95E-03	9.62E-14	2.81E-18
mRMR	2.95E-03	0	4.86E-19	5.99E-12
ENM	9.62E-14	4.86E-19	0	6.15E-23
GMM	2.81E-18	5.99E-12	6.15E-23	0

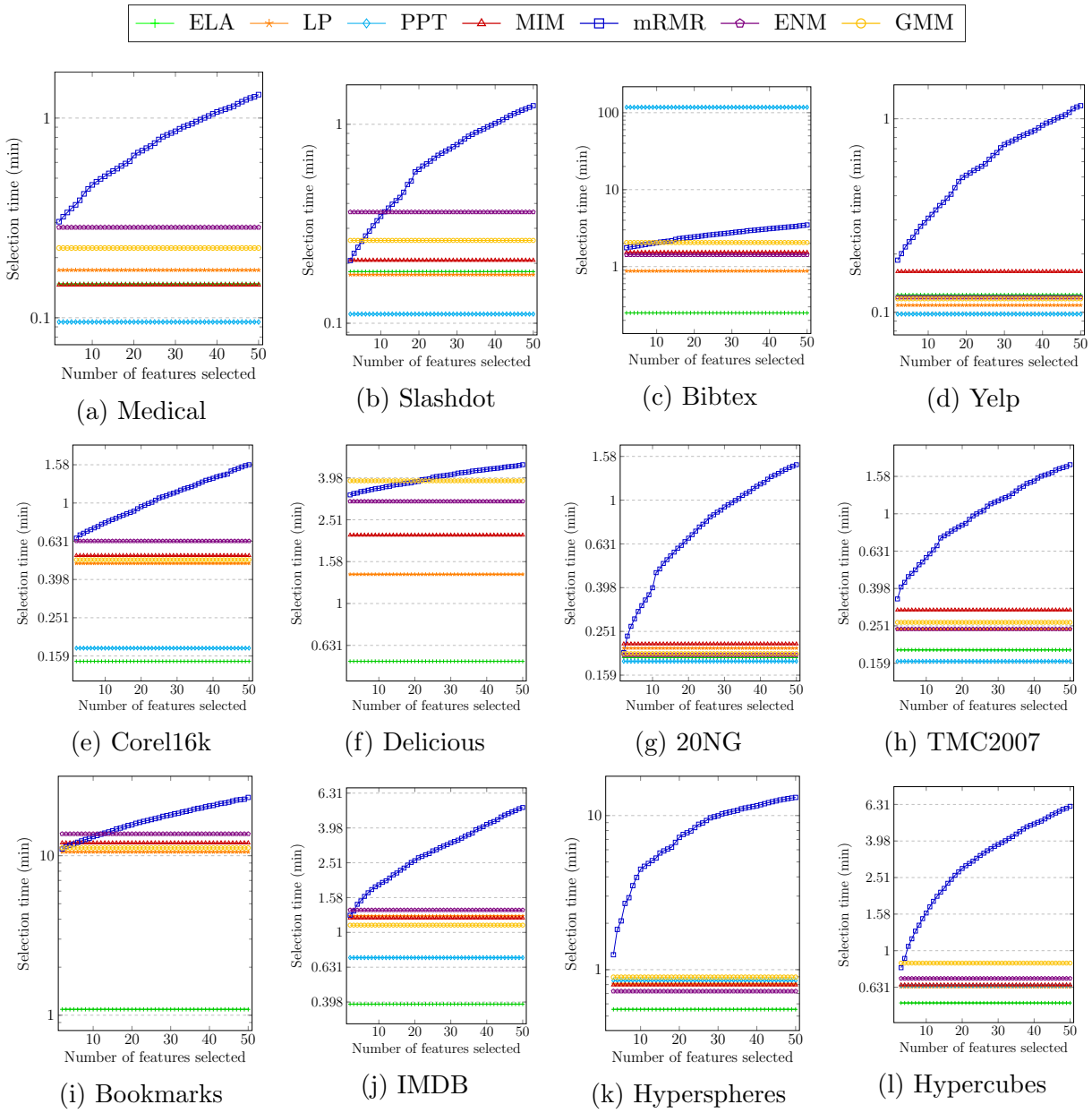


Fig. 5.13.: Runtime obtained selecting up to 50 features on all the datasets.

5.7.3 Runtime comparison

This experiment studies the runtime required by each method to select a feature subset regarding their size. The reported times include the discretization step for the synthetic datasets. Figure 5.13 presents the runtime in minutes and on a logarithmic scale for readability.

The reported execution times are very similar for all the methods that rank the features based on their relevance, i.e. excluding the mRMR method. We can observe that generally, the reported times for ELA are the lowest, especially for the largest datasets. MIM, ENM, and GMM report similar runtimes, where usually the lowest runtime corresponds to MIM since it only performs the sum of individual MI measures, followed by ENM which only needs to calculate the norm of the vector formed by the individual MI measures, and finally by GMM which performs the geometrical mean for each feature.

PPT is the method with the most variations among the runtime measures. This method performs the fastest for the smallest datasets with the lowest number of labels. However, its execution times quickly increase with the number of labels, as we can observer with *Bibtex*, to the point where the method was unable to execute for certain datasets such as *Delicious* and *Bookmarks*.

mRMR can be considered the slowest performing method in comparison. This is produced by the minimization of the redundancy, which is done by comparing each feature candidate to the previously selected feature. Nevertheless, in some cases, the minimization of the redundancy does not incur in execution times much higher than the other methods as the result of using binary features. This is expected to have a much higher impact for continuous features or nominal features with a large range of values.

5.8 Conclusions

In this chapter, we have evaluated the mutual information estimator that considers each label individually by using four feature selection methods on multi-label classification problems. The first method (MIM) selects the features that maximize the sum of the mutual information measures. The second method (mRMR) selects the features that maximize the sum of the mutual information measures while minimizing the information with previously selected features. The third method (ENM) selects the features that maximize the L^2-norm of the individual information measures. Finally, the fourth method (GMM) selects the features that

maximize the geometrical mean of the individual measures. The impact of these methods has been studied in detail by comparing them to three feature selection methods, regarding the subset accuracy and execution times.

The experimental study has proved that ENM efficiently improves the selection of discrete features, while GMM and MIM improve the selection of continuous features. Additionally, GMM also presents a remarkable performance of on discrete features associated with a large number of labels. These methods present a minimization of the redundancy on par with mRMR but with the advantage of constant selection time. Therefore, the present competitive selection times even for the largest datasets.

These results indicate that choosing the right method for feature selection can have a significant impact on the final results. We have presented two opposing ways to aggregate the information of individual measures for each label which attempt to balance the relevance and the redundancy of the selected features. Nevertheless, these methods could be extended or replaced by more complex methods which could attempt to find the optimal averaging strategy.

CHAPTER 6

CONCLUSIONS

This thesis introduced algorithms and a framework for distributed multi-label learning on Apache Spark. A series of methods have been implemented, ranging from newly proposed methods to traditional methods used in the comparisons of the experimental studies. The proposed methods were three distributed ML-KNN implementations which compared the different strategies for distributed nearest neighbors and two distributed multi-label feature selection method based on an adaptation of mutual information.

In order to study how to improve the computational performance of the multi-label algorithms a series of different strategies were considered: local parallelization using same memory space, distributed parallelization using independent memory spaces, and distributed parallelization using shared memory space. We defined five methods which considered each of those scenarios: the baseline Mulan implementation using a single thread on a single machine, a multi-threading version of Mulan on a single machine, a distributed version of Mulan where multiple instances of Mulan are deployed in each machine, a distributed version of Mulan where each machine has a multi-threading version of Mulan, and a Spark native implementation of the multi-label learning paradigm. The results highlighted that there were statistical differences between the predictions produced by the methods based on Mulan and the Spark native methods. Additionally, it proved that the distribution methods produce the best results once the data is large enough to overcome the distribution overhead. In particular, Spark proved to be the most scalable, maximizing the use of the shared resources in the cluster, executing hundreds of times faster than the Mulan implementations.

Ml-knn adapts the traditional knn algorithm to multi-label problems, therefore it inherits all the advantages and disadvantages from the original method. This method has been widely used in pre-processing scenarios, where the original data was transformed. However, it is computationally expensive since the prediction process needs to compute the distances against the training instances. Therefore, we considered it a good candidate for distribution due to its computational nature and its usefulness in future work. Three approaches for distributing the computation of Ml-knn were evaluated: an iterative pair-wise distance computation (Ml-knn-it), a tree-based method that index the instances across multiple nodes (Ml-knn-ht), and a local sensitive hashing method that builds multiple hash tables to index the data (Ml-knn-lsh). The results proved that the predictions of Ml-knn-ht are considered equivalent to those of an exact method. Additionally, Ml-knn-ht outperforms the execution times of all the other methods, regardless of the number of instances, features, and labels.

A comprehensible study of feature selection strategies for multi-label problems was presented, where we discussed the best adaptation of mutual information for multiple labels. Two distributed feature selection methods on continuous and discrete features for multi-label problems were proposed: Euclidean Norm Maximization (ENM), and Geometric Mean Maximization (GMM). The former selects the features with the largest $\boldsymbol{L^2}$-norm whereas the latter selects the features with the largest geometric mean. These methods handled the multi-label features directly, without relying on any type of transformation. The performance of the feature selection methods was measured by evaluating the predictions of Ml-knn-ht on the feature subset. GMM showed a similar performance as the maximization of the mutual information, which sums the information of the individual measures, on continuous features. ENM showed clearly superior performance for discrete features, improving the results in most scenarios. These methods produced constant execution times which enables them to use or large-scale datasets.

Multi-label data is usually characterized by a series of factors such as high dimensionality, a large number of instances, unbalanced data, and dependencies between the labels. This research aimed to show that these characteristics can be handled by a two-step strategy. First, by loading the data on the appropriate distributed architecture. This environment allows handling any size of data and considerably reduces the execution times in comparison to traditional systems. Second, by processing the original data and selecting the most relevant information. This processing step transforms the multi-label problem into one or more simpler problems that smoothed the difficulties of the original data.

CHAPTER 7

FUTURE WORK

The contributions introduced in this work highlight the advantages of distributing the multi-label learning paradigm using Apache Spark. Due to the novelty of this approach, there are numerous paths to be explored in future work.

- There are very few multi-label methods that have been implemented on the MapReduce programming model and even less on Apache Spark. This provides us with the opportunity to develop countless novelty methods, however, it also requires to develop from scratch many of the well-known algorithms out there in order to present a fair comparison of the results. We did a remarkable effort into implementing the multi-label evaluation metrics, loading of multi-label data, and many algorithms. Nevertheless, there are many methods in Mulan [104] and Meka [106] that could still be implemented.

- ML-KNN is a *first-order* approach since it considers independently the relevance of each label. Although this characteristic is not important for the scenarios where we applied it, such as evaluation of feature selection. It would be interesting to incorporate a high-order approach such as DML-KNN [92]. As a result, that method could be used to evaluate other scenarios besides the selection of features, such as the selection of correlated groups of labels.

- The proposed multi-label feature selection methods have shown promising results about aggregating the information of individual measures in different ways which can affect the relevance and redundancy of the selected features. Nevertheless, we focused on two

opposing approaches leaving open the possibility of including more complex approaches which would attempt to find the Pareto optimality.

- There is plenty of evidence that using correlated subsets of labels increases the quality of the predictions [53], [65], [79]. In [214] they proved that partitioning the label space using the frequency of occurrence of labels could lead to an improvement over the random selection of subsets. However, in all the studied approaches the same base classifier is used for every label subset. We believe that *first-order* approaches should be used in independent labels, while *high-order* approaches should be used in correlated label subsets. Additionally, most of the proposed methods define disjoint label spaces, while it has been proved that overlapping spaces produce better results [65].

- There has been some efforts into displaying graphically the characteristics of multi-label data [106], [215]. However, there are no precedents of plotting the multi-label classifiers on a 2D projection, showing the decision boundaries produced by the model. This could be achieved by plotting the overlapping contours of each of the labels. The visualization of multi-label data would be very useful insight into understanding how apparently simple datasets, with a low number of labels, are so difficult to learn for the current algorithms.

- Most of the approaches for multi-label learning are based on ensemble methods. This can be implemented on Apache Spark by building each of the base classifiers sequentially. However, it has been shown that it is possible to submit many tasks simultaneously [131], e.g. to construct a random forest by building all the trees in parallel. It is unknown if this technique could be applied for generic ensembles, or if it is necessary to modify the implementation of the base classifiers. Nevertheless, it is an area demanding further research.

Bibliography

[1] H. Guo, "Big Earth data: A new frontier in Earth and information sciences", *Big Earth Data*, vol. 1, no. 1-2, pp. 4–20, 2017.

[2] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The evolution of data to life-critical", *Dont Focus on Big Data*, 2017.

[3] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east", *IDC International Data Corporation*, vol. 2007, no. 2012, pp. 1–16, 2012.

[4] G. Sammut and M. Sartawi, "Perspective-Taking and the Attribution of Ignorance", *Journal for the Theory of Social Behaviour*, vol. 42, no. 2, pp. 181–200, 2012.

[5] S. Yu, M. Liu, W. Dou, S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou, *Networking for Big Data: A Survey*, 2018.

[6] T. Aho, B. Ženko, S. Džeroski, and T. Elomaa, "Multi-target regression with rule ensembles", *Journal of Machine Learning Research*, vol. 13, no. Aug, pp. 2367–2407, 2012.

[7] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, and I. Vlahavas, "Multi-target regression via input space expansion: Treating targets as inputs", *Machine Learning*, vol. 104, no. 1, pp. 55–98, 2016.

[8] G. Tsoumakas, E. Spyromitros-Xioufis, A. Vrekou, and I. Vlahavas, "Multi-target regression via random linear target combinations", in *Joint european conference on machine learning and knowledge discovery in databases*, Springer, 2014, pp. 225–240.

[9] O. Reyes, A. Cano, H. Fardoun, and S. Ventura, "A locally weighted learning method based on a data gravitation model for multi-target regression", *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, pp. 282–295, 2018.

[10] G. Melki, A. Cano, V. Kecman, and S. Ventura, "Multi-target support vector regression via correlation regressor chains", *Information Sciences*, vol. 415-416, pp. 53–69, 2017.

[11] L. Breiman and J. H. Friedman, "Predicting multivariate responses in multiple linear regression", *Journal of the Royal Statistical Society*, vol. 59, no. 1, pp. 3–54, 1997.

[12] P. J. Brown, J. V. Zidek, *et al.*, "Adaptive multivariate ridge regression", *The Annals of Statistics*, vol. 8, no. 1, pp. 64–74, 1980.

[13] Y. Haitovsky, "On multivariate ridge regression", *Biometrika*, vol. 74, no. 3, pp. 563–570, 1987.

[14] C. A. Micchelli and M. Pontil, "On learning vector-valued functions", *Neural computation*, vol. 17, no. 1, pp. 177–204, 2005.

[15] T. Similä and J. Tikka, "Input selection and shrinkage in multiresponse linear regression", *Computational Statistics & Data Analysis*, vol. 52, no. 1, pp. 406–422, 2007.

[16] K. Aas and L. Eikvil, *Text categorisation: A survey*, 1999.

[17] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification", *Pattern Recognition*, vol. 37, no. 9, pp. 1757–1771, 2004.

[18] G.-J. Qi, X.-S. Hua, Y. Rui, J. Tang, T. Mei, and H.-J. Zhang, "Correlative multi-label video annotation", in *International conference on Multimedia*, ACM, 2007, pp. 17–26.

[19] A. Dimou, G. Tsoumakas, V. Mezaris, I. Kompatsiaris, and I. Vlahavas, "An empirical study of multi-label learning methods for video annotation", in *International Workshop on Content-Based Multimedia Indexing*, IEEE, 2009, pp. 19–24.

[20] M. Wang, X. Zhou, and T.-S. Chua, "Automatic image annotation via local multi-label classification", in *International conference on Content-based image and video retrieval*, 2008.

[21] A. Clare and R. D. King, "Knowledge Discovery in Multi-label Phenotype Data", in *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 2001, pp. 42–53.

[22] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification", in *Advances in Neural Information Processing Systems*, 2002, pp. 681–687.

[23] M.-L. Zhang and Z.-H. Zhou, "Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization", *Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351, 2006.

[24] Z. Barutcuoglu, R. E. Schapire, and O. G. Troyanskaya, "Hierarchical multi-label prediction of gene function", *Bioinformatics*, vol. 22, no. 7, pp. 830–836, 2006.

[25] L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Džeroski, "Predicting gene function using hierarchical multi-label decision tree ensembles", *BMC Bioinformatics*, vol. 11, no. 1, p. 2, 2010.

[26] H. Kazawa, T. Izumitani, H. Taira, and E. Maeda, "Maximal margin labeling for multi-topic text categorization", in *Advances in Neural Information Processing Systems*, 2005, pp. 649–656.

[27] L. Tang, S. Rajan, and V. K. Narayanan, "Large scale multi-label classification via metalabeler", in *International conference on World wide web*, ACM, 2009, pp. 211–220.

[28] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma, "Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages", in *International conference on World Wide Web*, ACM, 2013, pp. 13–24.

[29] F. A. Thabtah, P. Cowling, and Y. Peng, "MMAC: A new multi-class, multi-label associative classification approach", in *International Conference on Data Mining*, IEEE, 2004, pp. 217–224.

[30] A. Veloso, W. Meira, M. Gonçalves, and M. Zaki, "Multi-label lazy associative classification", in *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 2007, pp. 605–612.

[31] A. Veloso and W. Meira, "Multi-label associative classification", in *Demand-Driven Associative Classification*, Springer, 2011, pp. 53–59.

[32] S. Gopal and Y. Yang, "Multilabel classification with meta-level features", in *International Conference on Research and development in information retrieval*, ACM, 2010, pp. 315–322.

[33] S. Zhu, X. Ji, W. Xu, and Y. Gong, "Multi-labelled classification using maximum entropy method", in *International Conference on Research and development in information retrieval*, ACM, 2005, pp. 274–281.

[34] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Multi-label text classification for automated tag suggestion", *ECML PKDD Discovery Challenge*, vol. 75, 2008.

[35] Y. Song, L. Zhang, and C. L. Giles, "A sparse gaussian processes classification framework for fast tag suggestions", in *Conference on Information and knowledge management*, ACM, 2008, pp. 93–102.

[36] S. M. Liu and J.-H. Chen, "A multi-label classification based approach for sentiment classification", *Expert Systems with Applications*, vol. 42, no. 3, pp. 1083–1093, 2015.

[37] A. Wieczorkowska, P. Synak, and Z. W. Raś, "Multi-label classification of emotions in music", in *Intelligent Information Processing and Web Mining*, Springer, 2006, pp. 307–315.

[38] C. Sanden and J. Z. Zhang, "Enhancing multi-label music genre classification through ensemble techniques", in *Conference on Research and development in Information Retrieval*, ACM, 2011, pp. 705–714.

[39] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas, "Multi-label classification of music into emotions", in *International Conference on Music Information Retrieval*, vol. 8, 2008, pp. 325–330.

[40] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. pearson education, 2005.

[41] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[42] T. White, *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[43] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce: a survey", *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2012.

[44] J. Lin, "MapReduce is good enough? if all you have is a hammer, throw away everything that's not a nail!", *Big Data*, vol. 1, no. 1, pp. 28–37, 2013.

[45] N. Spangenberg, M. Roth, and B. Franczyk, "Evaluating New Approaches of Big Data Analytics Frameworks", in *International Conference on Business Information Systems*, Springer, 2015, pp. 28–37.

[46] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets", *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[47] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing", *USENIX Conference on Networked Systems Design and Implementation*, p. 2, 2012.

[48] D. Freedman and P. Diaconis, "On the histogram as a density estimator:L2 theory", *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, vol. 57, no. 4, pp. 453–476, 1981.

[49] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining Multi-label Data", in *Data mining and knowledge discovery handbook*, Springer, 2009, pp. 667–685.

[50] Z.-H. Zhou, M.-L. Zhang, S.-J. Huang, and Y.-F. Li, "Multi-instance multi-label learning", *Artificial Intelligence*, vol. 176, no. 1, pp. 2291–2320, 2012.

[51] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel, "Decision trees for hierarchical multi-label classification", *Machine Learning*, vol. 73, no. 2, p. 185, 2008.

[52] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms", *IEEE transactions on knowledge and data engineering*, vol. 26, no. 8, pp. 1819–1837, 2014.

[53] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification", *Machine Learning*, vol. 85, no. 3, p. 333, 2011.

[54] J. Ramón Quevedo, O. Luaces, and A. Bahamonde, "Multilabel classifiers with a probabilistic thresholding strategy", *Pattern Recognition*, vol. 45, no. 2, pp. 876–883, 2012.

[55] M.-L. Zhang and K. Zhang, "Multi-label learning by exploiting label dependency", in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 999–1008.

[56] S.-J. Huang, Z.-H. Zhou, and Z Zhou, "Multi-label learning by exploiting label correlations locally.", in *AAAI Conference on Artificial Intelligence*, 2012, pp. 949–955.

[57] J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker, "Multilabel classification via calibrated label ranking", *Machine Learning*, vol. 73, no. 2, pp. 133–153, 2008.

[58] N. Ghamrawi and A. McCallum, "Collective multi-label classification", in *Iinternational conference on Information and knowledge management*, ACM, 2005, pp. 195–200.

[59] N. Ueda and K. Saito, "Parametric mixture models for multi-labeled text", in *Advances in Neural Information Processing Systems*, 2003, pp. 737–744.

[60] W. Cheng and E. Hüllermeier, "Combining instance-based learning and logistic regression for multi-label classification", *Machine Learning*, vol. 76, no. 2, pp. 211–225, 2009.

[61] S. Godbole and S. Sarawagi, "Discriminative methods for multi-labeled classification", in *Pacific-Asia conference on knowledge discovery and data mining*, Springer, 2004, pp. 22–30.

[62] S. Ji, L. Tang, S. Yu, and J. Ye, "Extracting shared subspace for multi-label classification", in *International conference on Knowledge discovery and data mining*, ACM, 2008, pp. 381–389.

[63] R. Yan, J. Tesic, and J. R. Smith, "Model-shared subspace boosting for multi-label classification", in *International conference on Knowledge discovery and data mining*, ACM, 2007, pp. 834–843.

[64] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets", in *International Conference on Data Mining*, IEEE, 2008, pp. 995–1000.

[65] G. Tsoumakas and I. Vlahavas, "Random k-Labelsets: An Ensemble Method for Multilabel Classification", in *European conference on machine learning*, Springer, 2007, pp. 406–417.

[66] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Effective and efficient multilabel classification in domains with large number of labels", in *Workshop on Mining Multidimensional Data*, vol. 21, 2008, pp. 53–59.

[67] K. Dembczynski, W. Cheng, and E. Hüllermeier, "Bayes optimal multilabel classification via probabilistic classifier chains.", in *International Conference on Machine Learning*, vol. 10, 2010, pp. 279–286.

[68] K. Dembszynski, W. Waegeman, W. Cheng, and E. Hüllermeier, "On label dependence in multilabel classification", in *International Workshop on learning from multi-label data*, Ghent University, KERMIT, Department of Applied Mathematics, Biometrics and Process Control, 2010.

[69] E. Gibaja and S. Ventura, "Multi-label learning: A review of the state of the art and ongoing research", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 6, pp. 411–444, 2014.

[70] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier, "On label dependence and loss minimization in multi-label classification", *Machine Learning*, vol. 88, no. 1-2, pp. 5–45, 2012.

[71] D. J. Hsu, S. M. Kakade, J. Langford, and T. Zhang, "Multi-label prediction via compressed sensing", in *Advances in Neural Information Processing Systems*, 2009, pp. 772–780.

[72] J. Read, A. Bifet, G. Holmes, and B. Pfahringer, "Scalable and efficient multi-label classification for evolving data streams", *Machine Learning*, vol. 88, no. 1-2, pp. 243–272, 2012.

[73] J. Read, "Scalable multi-label classification", PhD thesis, University of Waikato, 2010.

[74] S. Godbole, S. Sarawagi, and S. Chakrabarti, "Scaling multi-class support vector machines using inter-class confusion", in *International conference on Knowledge discovery and data mining*, ACM, 2002.

[75] G. Tsoumakas and I. Katakis, "Multi-Label Classification: An Overview", *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, pp. 1–13, 2007.

[76] E. Gibaja and S. Ventura, "A Tutorial on Multilabel Learning", *ACM Computing Surveys*, vol. 47, no. 3, pp. 1–38, 2015.

[77] M. S. Sorower, "A literature survey on algorithms for multi-label learning", *Oregon State University, Corvallis*, vol. 18, 2010.

[78] W. Chen, J. Yan, B. Zhang, Z. Chen, and Q. Yang, "Document transformation for multi-label feature selection in text categorization", in *IEEE International Conference on Data Mining*, 2007, pp. 451–456.

[79] J. Read, "A pruned problem transformation method for multi-label classification", in *New Zealand Computer Science Research Student Conference*, vol. 143150, 2008.

[80] H. Blockeel and L. De Raedt, "Top-down induction of first-order logical decision trees", *Artificial Intelligence*, vol. 101, no. 1, pp. 285–297, 1998.

[81] A. McCallum, "Multi-label text classification with a mixture model trained by em", in *AAAI workshop on Text Learning*, 1999, pp. 1–7.

[82] J. R. Quinlan, "C4.5: Programming for machine learning", *Morgan Kauffmann*, p. 38, 1993.

[83] D. Kocev, C. Vens, J. Struyf, and S. Džeroski, "Ensembles of Multi-Objective Decision Trees", in *European conference on machine learning*, 2007, pp. 624–631.

[84] T. Joachims, "Text categorization with Support Vector Machines: Learning with many relevant features", in *European conference on machine learning*, 1998, pp. 137–142.

[85] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces", in *International Conference on Machine learning*, ACM, 2004, p. 104.

[86] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables", *Journal of machine learning research*, vol. 6, no. Sep, pp. 1453–1484, 2005.

[87] M.-L. Zhang and Z.-H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning", *Pattern Recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[88] F. Charte, A. Rivera, M. del Jesus, and F. Herrera, "MLSMOTE: Approaching imbalanced multi-label learning through synthetic instance generation", *Knowledge-Based Systems*, vol. 89, pp. 385–397, 2015.

[89] A. Cano, "A survey on graphic processing unit computing for large-scale data mining", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 1, e1232, 2018.

[90] P. Skryjomski, B. Krawczyk, and A. Cano, "Speeding up k-Nearest Neighbors Classifier for Large-Scale Multi-Label Learning on GPUs", *Neurocomputing*, vol. In press, 2018.

[91] M. Roseberry and A. Cano, "Multi-label kNN Classifier with Self Adjusting Memory for Drifting Data Streams", in *LIDTA@PKDD/ECML*, 2018.

[92] Z. Younes, F. Abdallah, T. Denoeux, and H. Snoussi, "A dependent multi-label classification method derived from the k-nearest neighbor rule", *EURASIP Journal on Advances in Signal Processing*, pp. 1–14, 2011.

[93] E. Spyromitros, G. Tsoumakas, and I. Vlahavas, "An empirical study of lazy multi-label classification algorithms", in *Hellenic Conference on Artificial Intelligence*, 2008, pp. 401–406.

[94] J. Xu, "Multi-label weighted k-nearest neighbor classifier with adaptive weight estimation", in *International Conference on Neural Information Processing*, 2011, pp. 79–88.

[95] H. Zhang, S. Kiranyaz, and M. Gabbouj, "A k-nearest neighbor multilabel ranking algorithm with application to content-based image retrieval", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017, pp. 2587–2591.

[96] E. Cakir, T. Heittola, H. Huttunen, and T. Virtanen, "Polyphonic sound event detection using multi label deep neural networks", in *International Joint Conference on Neural Networks*, IEEE, 2015, pp. 1–7.

[97] M.-L. Zhang, "ML-RBF: RBF Neural Networks for Multi-Label Learning", *Neural Processing Letters*, vol. 29, no. 2, pp. 61–74, 2009.

[98] M.-L. Zhang, J. M. Peña, and V. Robles, "Feature selection for multi-label naive bayes classification", *Information Sciences*, vol. 179, no. 19, pp. 3218–3229, 2009.

[99] L. Rokach, A. Schclar, and E. Itach, "Ensemble methods for multi-label classification", *Expert Systems with Applications*, vol. 41, no. 16, pp. 7507–7523, 2014.

[100] J. P. Pestian, C. Brew, P. Matykiewicz, D. J. Hovermale, N. Johnson, K. B. Cohen, and W. Duch, "A shared task involving multi-label classification of clinical free text", *Association for Computational Linguistics*, pp. 97–104, 2007.

[101] F. Charte, A. Rivera, M. J. del Jesus, and F. Herrera, "A first approach to deal with imbalance in multi-label datasets", in *International Conference on Hybrid Artificial Intelligence Systems*, 2013, pp. 150–160.

[102] F. Charte, A. Rivera, M. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms", *Neurocomputing*, vol. 163, pp. 3–16, 2015.

[103] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. MÃžller, F. Pereira, C. E. Rasmussen, *et al.*, "The need for open source software in machine learning", *Journal of Machine Learning Research*, vol. 8, no. Oct, pp. 2443–2466, 2007.

[104] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning", *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2411–2414, 2011.

[105] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update", *SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[106] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes, "Meka: a multi-label/multi-target extension to weka", *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 667–671, 2016.

[107] P. Szymański and T. Kajdanowicz, "A scikit-based Python environment for performing multi-label classification", *ArXiv*, 2017. arXiv: `1702.01460 [cs.LG]`.

[108] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[109] A. S. Tanenbaum and R. Van Renesse, "Distributed operating systems", *ACM Computing Surveys*, vol. 17, no. 4, pp. 419–470, 1985.

[110] P. Baran, "On distributed communications networks", *IEEE transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.

[111] M. Singhal and N. G. Shivaratri, *Advaced concepts in operating systems. Distributed, database, and multiprocessor operating systems*. McGraw Hill, 1994.

[112] R. Chow and Y.-C. Chow, *Distributed operating systems and algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[113] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[114] M. Kleppmann, *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. " O'Reilly Media, Inc.", 2017.

[115] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam, *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press, 1994.

[116] J. Dongarra, R Hempel, A. Hey, and D. Walker, "MPI: A message-passing interface standard", *International Journal of Supercomputer Applications*, vol. 8, no. 3, p. 4, 1995.

[117] W. Gropp and E. Lusk, "Goals guiding design: PVM and MPI", in *International Conference on Cluster Computing*, IEEE, 2002, pp. 257–265.

[118] D. Guster, A. Al-Hamamah, P. Safonov, and E. Bachman, "Computing and network Performance of a distributed parallel processing environment using MPI and PVM communication methods", *Journal of Computing Sciences in Colleges*, vol. 18, no. 4, pp. 246–253, 2003.

[119] A. Cano, C. García-Martínez, and S. Ventura, "Extremely high-dimensional optimization with MapReduce: scaling functions and algorithm", *Information Sciences*, vol. 415, pp. 110–127, 2017.

[120] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache Spark: a unified engine for big data processing", *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[121] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, *et al.*, "Spark SQL: Relational data processing in Spark", in *ACM SIGMOD International Conference on Management of Data*, ACM, 2015, pp. 1383–1394.

[122] X. Meng, J. Bradley, B Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D Tsai, M. Amde, S. Owen, *et al.*, "Mllib: Machine learning in apache Spark", *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

[123] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized Streams: Fault-tolerant streaming computation at scale", in *ACM Symposium on Operating Systems Principles*, ACM, 2013, pp. 423–438.

[124] N. Johnsirani Venkatesan, C. Nam, and D. R. Shin, "Deep Learning Frameworks on Apache Spark: A Review", *IETE Technical Review*, pp. 1–14, 2018.

[125] W. P.M.S. M. Dymczyk and A. C. Q. Kou, "Deep Learning with Deep Water", 2017.

[126] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "SparkNet: Training deep networks in Spark", *ArXiv*, 2015.

[127] C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin, "Large-scale logistic regression and linear support vector machines using Spark", in *IEEE International Conference on Big Data*, IEEE, 2014, pp. 519–528.

[128] S. Daengduang and P. Vateekul, "Applying One-Versus-One SVMs to classify multi-label data with large labels using Spark", in *International Conference on Knowledge and Smart Technology*, IEEE, 2017, pp. 72–77.

[129] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera, "kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data", *Knowledge-Based Systems*, vol. 117, pp. 3–15, 2017.

[130] R.-Z. Qi, Z.-J. Wang, and S.-Y. Li, "A parallel genetic algorithm based on Spark for pairwise test suite generation", *Journal of Computer Science and Technology*, vol. 31, no. 2, pp. 417–427, 2016.

[131] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, "A parallel random forest algorithm for big data in a Spark cloud computing environment", *IEEE Transactions on Parallel & Distributed Systems*, no. 1, pp. 1–1, 2017.

[132] Z. Wang, Y. Zhao, Y. Liu, and C. Lv, "A speculative parallel simulated annealing algorithm based on Apache Spark", *Concurrency and Computation: Practice and Experience*, vol. 30, no. 14, e4429, 2018.

[133] M. Duan, K. Li, X. Liao, and K. Li, "A parallel multiclassification algorithm for big data using an extreme learning machine", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2337–2351, 2018.

[134] M. A. Tahir, J. Kittler, K. Mikolajczyk, and F. Yan, "Improving multilabel classification performance by using ensemble of multi-label classifiers", in *International Workshop on Multiple Classifier Systems*, 2010, pp. 11–21.

[135] D. Wegener, M. Mock, D. Adranale, and S. Wrobel, "Toolkit-based high-performance data mining of large data on MapReduce clusters", in *IEEE International Conference on Data Mining Workshops*, 2009, pp. 296–301.

[136] Q. Wang, S. R. Kulkarni, and S. Verdú, "Divergence estimation for multidimensional densities via k-nearest-neighbor distances", *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2392–2405, 2009.

[137] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions", in *Foundations of Computer Science*, 2006, pp. 459–468.

[138] Y. Song, J. Huang, D. Zhou, H. Zha, and C. L. Giles, "IKNN: Informative K-Nearest Neighbor Pattern Classification", in *European Conference on Principles of Data Mining and Knowledge Discovery*, 2007, pp. 248–264.

[139] Y. Iano, F. S. da Silva, and A. M. Cruz, "A fast and efficient hybrid fractal-wavelet image coder", *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 98–105, 2006.

[140] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification", in *Advances in Neural Information Processing Systems*, 2006, pp. 1473–1480.

[141] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti, "Similarity-based classification: Concepts and algorithms", *Journal of Machine Learning Research*, vol. 10, pp. 747–776, 2009.

[142] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time", *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[143] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[144] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions", *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[145] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov, "Balanced aspect ratio trees and their use for drawing very large graphs", in *International Symposium on Graph Drawing*, 1998, pp. 111–124.

[146] C. A. Duncan, M. T. Goodrich, and S. Kobourov, "Balanced aspect ratio trees: Combining the advantages of kd trees and octrees", *Journal of Algorithms*, vol. 38, no. 1, pp. 303–333, 2001.

[147] R. F. Sproull, "Refinements to nearest-neighbor searching ink-dimensional trees", *Algorithmica*, vol. 6, no. 1-6, pp. 579–589, 1991.

[148] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds", in *ACM symposium on Theory of computing*, 2008, pp. 537–546.

[149] Y. Jia, J. Wang, G. Zeng, H. Zha, and X.-S. Hua, "Optimizing kd-trees for scalable visual descriptor indexing", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 3392–3399.

[150] A. W. Moore, "The anchors hierarchy: Using the triangle inequality to survive high dimensional data", in *Conference on Uncertainty in Artificial Intelligence*, 2000, pp. 397–405.

[151] T. Liu, A. W. Moore, K. Yang, and A. G. Gray, "An investigation of practical approximate nearest neighbor algorithms", in *Advances in Neural Information Processing Systems*, 2005, pp. 825–832.

[152] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[153] A. Babenko and V. Lempitsky, "The inverted multi-index", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3069–3076.

[154] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality", in *ACM Symposium on Theory of Computing*, 1998, pp. 604–613.

[155] M. Covell and S. Baluja, "LSH banding for large-scale retrieval with memory and recall constraints", in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009, pp. 1865–1868.

[156] J. Buhler, "Efficient large-scale sequence comparison by locality-sensitive hashing", *Bioinformatics*, vol. 17, no. 5, pp. 419–428, 2001.

[157] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: efficient indexing for high-dimensional similarity search", in *International Conference on Very Large Data Bases*, 2007, pp. 950–961.

[158] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: self-tuning indexes for similarity search", in *Iinternational Conference on World Wide Web*, 2005, pp. 651–660.

[159] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing", in *IEEE International Conference on Computer Vision*, 2003, p. 750.

[160] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing", in *Advances in Neural Information Processing Systems*, 2009, pp. 1753–1760.

[161] P. Jain, B. Kulis, and K. Grauman, "Fast image search for learned metrics", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[162] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 2130–2137.

[163] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels", in *Advances in Neural Information Processing Systems*, 2009, pp. 1509–1517.

[164] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 3424–3431.

[165] J. He, W. Liu, and S.-F. Chang, "Scalable similarity search with optimized kernel hashing", in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 1129–1138.

[166] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu, "Complementary hashing for approximate nearest neighbor search", in *IEEE International Conference on Computer Vision*, 2011, pp. 1631–1638.

[167] T. B. Sebastian and B. B. Kimia, "Metric-based shape retrieval in large databases", in *International Conference on Pattern Recognition*, vol. 3, 2002, pp. 291–296.

[168] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph", in *International Joint Conference on Artificial Intelligence*, vol. 22, 2011, p. 1312.

[169] R. Paredes, E. Chávez, K. Figueroa, and G. Navarro, "Practical construction of k-nearest neighbor graphs in metric spaces", in *Workshop Ecosystem Architectures*, vol. 6, 2006, pp. 85–97.

[170] M. Connor and P. Kumar, "Fast construction of k-nearest neighbor graphs for point clouds", *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 4, pp. 599–608, 2010.

[171] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures", in *ACM International Conference on World Wide Web*, 2011, pp. 577–586.

[172] J. Maillo, I. Triguero, and F. Herrera, "A MapReduce-based k-nearest neighbor approach for big data classification", in *IEEE Trustcom/BigDataSE/ICESS*, vol. 2, 2015, pp. 167–172.

[173] T. Liu, C. Rosenberg, and H. A. Rowley, "Clustering billions of images with large scale nearest neighbor search", in *IEEE Workshop on Applications of Computer Vision*, 2007, pp. 28–28.

[174] K. Sechidis, G. Tsoumakas, and I. Vlahavas, "On the stratification of multi-label data", *Machine Learning and Knowledge Discovery in Databases*, pp. 145–158, 2011.

[175] G. Chandrashekar and F. Sahin, "A survey on feature selection methods", *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[176] R. Kohavi and G. H. John, "Wrappers for feature subset selection", *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[177] T. Lal, O Chapelle, J Weston, A Elisseeff, S Gunn, M Nikravesh, L. Zadeh, *et al.*, "Embedded methods", in *Feature Extraction: Foundations and Applications*, Springer, 2006, pp. 137–165.

[178] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik, "Feature selection for SVMs", in *Advances in Neural Information Processing Systems*, 2001, pp. 668–674.

[179] M. A. Hall, "Correlation-based feature selection for discrete and numeric class machine learning", in *International Conference on Machine Learning*, 2000, pp. 359–366.

[180] J. R. Vergara and P. A. Estévez, "A review of feature selection methods based on mutual information", *Neural computing and applications*, vol. 24, no. 1, pp. 175–186, 2014.

[181] A. Mariello and R. Battiti, "Feature selection based on the neighborhood entropy", *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2018.

[182] G. Ditzler, R. Polikar, and G. Rosen, "A sequential learning approach for scaling up filter-based feature subset selection", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2530–2544, 2018.

[183] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A New Benchmark Collection for Text Categorization Research", *Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[184] N. SpolaôR, E. A. Cherman, M. C. Monard, and H. D. Lee, "A comparison of multi-label feature selection methods using the problem transformation approach", *Electronic Notes in Theoretical Computer Science*, vol. 292, pp. 135–151, 2013.

[185] J. Lee and D.-W. Kim, "Feature selection for multi-label classification using multivariate mutual information", *Pattern Recognition Letters*, vol. 34, no. 3, pp. 349–357, 2013.

[186] O. Reyes, C. Morell, and S. V. Soto, "ReliefF-ML: An Extension of ReliefF Algorithm to Multi-label Learning", in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, 2013, pp. 528–535.

[187] G. Doquire and M. Verleysen, "Mutual information-based feature selection for multi-label classification", *Neurocomputing*, vol. 122, pp. 148–155, 2013.

[188] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas, "Multi-label classification of music into emotions", in *International Conference on Music Information Retrieval*, vol. 8, 2008, pp. 325–330.

[189] O. Reyes, C. Morell, and S. Ventura, "Scalable extensions of the ReliefF algorithm for weighting and selecting features on the multi-label learning context", *Neurocomputing*, vol. 161, pp. 168–182, 2015.

[190] J Lee, H Lim, and D.-W. Kim, "Approximating mutual information for multi-label feature selection", *Electronics letters*, vol. 48, no. 15, pp. 929–930, 2012.

[191] Y. Lin, Q. Hu, J. Liu, and J. Duan, "Multi-label feature selection based on max-dependency and min-redundancy", *Neurocomputing*, vol. 168, pp. 92–103, 2015.

[192] J. Lee and D.-W. Kim, "SCLS: Multi-label feature selection based on scalable criterion for large label set", *Pattern Recognition*, vol. 66, pp. 342–352, 2017.

[193] Y. Lin, Q. Hu, J. Liu, J. Chen, and J. Duan, "Multi-label feature selection based on neighborhood mutual information", *Applied Soft Computing*, vol. 38, pp. 244–256, 2016.

[194] F. Li, D. Miao, and W. Pedrycz, "Granular multi-label feature selection based on mutual information", *Pattern Recognition*, vol. 67, pp. 410–423, 2017.

[195] J. Lee and D.-W. Kim, "Mutual Information-based multi-label feature selection using interaction information", *Expert Systems with Applications*, vol. 42, no. 4, pp. 2013–2025, 2015.

[196] ——, "Fast multi-label feature selection based on information-theoretic feature ranking", *Pattern Recognition*, vol. 48, no. 9, pp. 2761–2771, 2015.

[197] H. Lim, J. Lee, and D.-W. Kim, "Multi-Label Learning Using Mathematical Programming", *IEICE Transactions on Information and Systems*, vol. 98, no. 1, pp. 197–200, 2015.

[198] L. Jian, J. Li, K. Shu, and H. Liu, "Multi-label informed feature selection", in *IJCAI*, 2016, pp. 1627–1633.

[199] H. Lim, J. Lee, and D.-W. Kim, "Low-rank approximation for multi-label feature selection", *International Journal of Machine Learning and Computing*, vol. 6, no. 1, p. 42, 2016.

[200] J. Xu and Q. Ma, "Multi-label regularized quadratic programming feature selection algorithm with Frank–Wolfe method", *Expert Systems with Applications*, vol. 95, pp. 14–31, 2018.

[201] C. Wang, Q. Hu, X. Wang, D. Chen, Y. Qian, and Z. Dong, "Feature selection based on neighborhood discrimination index", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 2986–2999, 2018.

[202] I. Rodriguez-Lujan, R. Huerta, C. Elkan, and C. S. Cruz, "Quadratic programming feature selection", *Journal of Machine Learning Research*, vol. 11, pp. 1491–1516, 2010.

[203] X. Wang, L. Zhao, and J. Xu, "Multi-label feature selection method based on multivariate mutual information and particle swarm optimization", in *International Conference on Neural Information Processing*, Springer, 2018, pp. 84–95.

[204] G. A. Darbellay and I. Vajda, "Estimation of the information by an adaptive partitioning of the observation space", *IEEE Transactions on Information Theory*, vol. 45, no. 4, pp. 1315–1321, 1999.

[205] L. Kozachenko and N. N. Leonenko, "Sample estimate of the entropy of a random vector", *Problemy Peredachi Informatsii*, vol. 23, no. 2, pp. 9–16, 1987.

[206] J. D. Victor, "Binless strategies for estimation of information from neural data", *Physical Review E*, vol. 66, no. 5, p. 051 903, 2002.

[207] G. A. Darbellay and I. Vajda, "Entropy expressions for multivariate continuous distributions", *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 709–712, 2000.

[208] F. Rossi, A. Lendasse, D. François, V. Wertz, and M. Verleysen, "Mutual information for the selection of relevant variables in spectrometric nonlinear modelling", *Chemometrics and intelligent laboratory systems*, vol. 80, no. 2, pp. 215–226, 2006.

[209] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information", *Physical review E*, vol. 69, no. 6, p. 066 138, 2004.

[210] B. C. Ross, "Mutual information between discrete and continuous data sets", *PloS one*, vol. 9, no. 2, e87357, 2014.

[211] J. T. Tomás, N. Spolaôr, E. A. Cherman, and M. C. Monard, "A framework to generate synthetic multi-label datasets", *Electronic Notes in Theoretical Computer Science*, vol. 302, pp. 155–176, 2014.

[212] J. Gonzalez-Lopez, S. Ventura, and A. Cano, "Distributed Nearest Neighbor Classification for Large-Scale Multi-label Data on Spark", *Future Generation Computer Systems*, vol. 87, pp. 66–82, 2018.

[213] A. Cano, J. M. Luna, E. L. Gibaja, and S. Ventura, "LAIM discretization for multi-label data", *Information Sciences*, vol. 330, pp. 370–384, 2016.

[214] P. Szymański, T. Kajdanowicz, and K. Kersting, "How is a data-driven approach better than random choice in label space division for multi-label classification?", *Entropy*, vol. 18, no. 8, p. 282, 2016.

[215] J. M. Moyano, E. L. Gibaja, and S. Ventura, "MLDA: A tool for analyzing multi-label datasets", *Knowledge-Based Systems*, vol. 121, pp. 1–3, 2017.

Vita

Jorge Gonzalez Lopez received his BSc. in Computer Science in 2011 and his MSc. in Computer Science in 2015, both from the Carlos III University of Madrid. As a full-time graduate student at the Ph.D. program at the Virginia Commonwealth University, his research is focused on distributed machine learning algorithms for large datasets and multi-label learning.

Publications:

J. Gonzalez-Lopez, A. Cano and S. Ventura, "Large-Scale Multi-label Ensemble Learning on Spark", *IEEE Trustcom/BigDataSE/ICESS* Sydney, pp. 893–900, 2017.

J. Gonzalez-Lopez, S. Ventura and A. Cano, "Distributed nearest neighbor classification for large-scale multi-label data on Spark", *Future Generation Computer Systems*, vol. 87, pp. 66–82, 2018.

J. Gonzalez-Lopez, S. Ventura and A. Cano, "ARFF data source library for distributed single/multiple instance, single/multiple output learning on Apache Spark", *International Conference on Computational Science*, 2019.

J. Gonzalez-Lopez, S. Ventura and A. Cano, "Distributed selection of continuous features in multi-label classification using mutual information", *IEEE Transactions on Neural Networks and Learning Systems*, under review, 2019.

J. Gonzalez-Lopez, S. Ventura and A. Cano, "Distributed multi-label feature selection using individual mutual information measures", *IEEE Transactions on Knowledge and Data Engineering*, under review, 2019.