University of Macedonia

School of Information Sciences, Department of Applied Informatics

Graduate Program in Artificial Intelligence and Data Analytics (AIDA)

# CLASSIFICATION ON MULTI-LABEL DATA WITH ORDERED LABELS

Master Thesis

of

**ANTONIOS KAGIAS**

**Supervisor:** Georgios Evangelidis
Professor

Thessaloniki, October 2024

University of Macedonia
School of Information Sciences, Department of Applied Informatics
Graduate Program in Artificial Intelligence and Data Analytics (AIDA)

# CLASSIFICATION ON MULTI-LABEL DATA WITH ORDERED LABELS

Master Thesis

of

**ANTONIOS KAGIAS**

**Supervisor:** Georgios Evangelidis
Professor

Approved by the examination committee on dd/mm/2024.

| | | |
|---|---|---|
| *(Signature)* | *(Signature)* | *(Signature)* |
| ............ | ........... | .............. |
| Georgios Evangelidis | Georgia Koloniari | Eftychios Protopapadakis |
| Professor | Assistant Professor | Assistant Professor |

Thessaloniki, October 2024

University of Macedonia
School of Information Sciences, Department of Applied Informatics
Graduate Program in Artificial Intelligence and Data Analytics (AIDA)

**DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS**

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

*(Signature)*

.............................
Antonios Kagias

dd/mm/2024

# Abstract

Over the last few decades, an increasing amount of data that have multiple labels - such as songs and images - has been generated and utilized in many aspects of everyday life. While there is a number of past researches that address the classification of this type of data, the aspect of taking the order of the labels into account is a field that needs further investigation. This thesis examines the problem of multi-label data classification where the labels of the instances are ordered based on importance. By developing and deploying four different classifiers, each with a distinct approach of classifying data, multiple experiments were conducted in order to evaluate whether our algorithms can produce satisfactory classifications while at the same time taking into consideration the order of the labels. Using a variety of metrics to measure the performance of the classifiers, our analysis demonstrated that, while none of the classifiers significantly outperformed the others, two of them provided somewhat superior results in certain areas. This marginal prevalence of the two classifiers leads us to conclude that their use is probably preferable in classifications problems under specific circumstances.

## Keywords

multi-label data, data mining, data classification, multi-label classification, ordered labels

*to my family*

# Acknowledgements

This thesis is a result of research conducted within the "MSc in Artificial Intelligence and Data Analytics" of the Department of Applied Informatics of the University of Macedonia.

*in case of paper publishing:*

In the context of this thesis, the paper titled [**paper title**] was published at [**conference name**]. We thank University of Macedonia Research Committee for the funding.

Thessaloniki, October 2024

Antonios Kagias

# Table of Contents

# List of Figures

# List of Tables

# Preface

This thesis, entitled "Classification on multi-label data with ordered labels", is the product of my dedication over the last two years. My interest and love for data processing, analysis and visualization led me to choose the master's degree on Artificial Intelligence and Data Analytics to develop any existing knowledge as well as to learn more about the field. While the journey up to this point has not been easy, I am proud to have reached the end of it filled with new skills, experiences and resources for the future. Seeing that difficulties and challenges are part of the process taught me invaluable life and career lessons.

I would like to thank my supervisor, Georgios Evangelidis, for his guidance and support throughout the process of this thesis and the postgraduate programme as a whole. His suggestion to work on a particularly interesting topic as well as his comments, guidance and motivation were among the main reasons why this research took shape.

I would also like to thank all the other professors in the master's program for their interest in helping me learn new concepts on data analysis and for providing all kinds of resources during these two years.

Finally, I would like to thank my family for their endless support and motivation to become the best version of myself. Also, my friends for their support and company whenever I needed them the most.

I deeply hope that the reader will appreciate my work and find it interesting and enjoyable.

Sincerely,

Antonios Kagias, 2024

Chapter 1

# Introduction

## 1.1 Problem - Importance of the topic

Multi-class classification is one of the most popular classification tasks where, given a set of unique labels, an instance can be assigned only one label. For example, if we have a set of images of animals where the possible classes are dog, cat and frog, an unlabeled image can only be classified as image of a dog, a cat or a frog. Multi-label classification and multi-label ranking are concepts closely related to the above. According to the analysis of [1], multi-label classification involves associating an instance with a subset of the possible labels instead of a single one. It is frequently used in order to classify diverse forms of data, such as songs or images [2]. For example, a song that is assigned "jazz, blues" belongs to both genres at the same time. Moreover, multi-label ranking not only labels each sample with a number of labels but these are also ranked in an order based on their relevance to the sample. An example to this is the previous song which in the context of multi-label ranking is primarily a "jazz" song and secondly a "blues" song.

## 1.2 Aim - Objectives

This thesis revolves around the classification on multi-label datasets where the labels of the instances are ranked in an order based on their importance. The main goal is to significantly enhance the classification accuracy of this type of data. Additionally, by achieving better classification results, this thesis attempts to advance the state-of-the-art in multi-label data classification and provide useful insights for academics and practitioners working with such datasets. Moreover, an ambition of this work is also to close the gap between the practical needs of applications in real life and the complexity of multi-label data.

## 1.3 Structure of the study

The rest of the thesis is structured as follows: Studies related to the topic are presented in Section 2. Section 3 provides a quick review of the BRkNN algorithm (Binary Relevance kNN). In Section 4 we present the algorithms we created for our experiments. These experiments and their results are described in Section 5. Ultimately, Section 6 provides a summary of our work, outlines limitations and discusses future research opportunities.

Chapter 2

# Literature review

Discussing multi-label ranking, [3] described it as the task of building a model that associates to an instance a bipartite split of the label set into relevant and irrelevant labels, as well as a ranking of the entire label set. Consequently, it is possible to think of multi-label ranking as a generalization of multi-label classification and ranking. Their suggested model includes a calibrated scale with a built-in zero-point, $\lambda_0$. This zero-point offers a way to separate the labels that are relevant from those that are irrelevant. Positive labels are rated above the zero-point, while negative labels are ranked below it.

Moreover, a case-based approach is presented by [4]. According to the authors, model-based multi-label ranking techniques have a high level of computational complexity even for a moderate number of class labels because of the increased complexity of the target space in multi-label ranking compared to binary classification. They present an alternative multi-label ranking framework using a case-based methodology (by adapting the kNN algorithm) that is conceptually and computationally simpler. They also demonstrate that their approach compares favorably with model-based alternatives in terms of both complexity and predictive accuracy.

Multi-label ranking has also had applications on the field of computer vision. Given a test image, [5] propose an efficient algorithm for multi-label ranking aiming to order the object classes in a way that relevant classes are ranked higher than irrelevant classes. They are also able to record relationships between class labels without assuming anything regarding them. Compared to methods based on binary classification, their methodology appears to be more efficient both in terms of object recognition and computationally.

Rather than examining labels equally, [6] concentrate on the special problem of multi-label ranking in cases when the real label set's order is known. They utilize the ranking information in the input and offer a unique dedicated loss function to optimize models by adding penalties for pairings that are rated wrongly. Operating with both ranked and unranked datasets, a method that only works with unranked datasets is to maximize the cross entropy between the predicted probability and the ground truth. The Log-Sum-Exp Pairwise (LSEP) loss [7] is an additional solution to the label ranking issue. LSEP attempts to train a scoring function for an unranked dataset such that a positive label should have a greater score than a negative one. One issue is that while the scores exist, it is unclear which are labeled as positive and which as negative. To solve this issue, they employ the dynamic multi-threshold approach [7], which predicts thresholds for each label in order to determine if it is positive or negative depending on its score. After that, they learn a scoring function f that yields a label-score vector in order to perform multi-label ranking. The relationship of the respective labels is represented by the relationship between the items in this vector. They propose the Ranked Log-Sum-Exp Pairwise (RLSEP) loss to learn how to sort the labels and they use the threshold technique to generate the ranking results.

Finally, [8] have provided an overall overview of multi-label classification. According to them, multi-label classification methods are split into two categories: a) problem transformation methods which transform the multi-label classification problem into one or more single-label classification or regression problems and b) algorithm adaptation methods which extend certain learning algorithms to directly handle multi-label data. Furthermore, they discuss these two classification categories by giving examples for each one, they explain basic terminology surrounding multi-label datasets characteristics and multi-label classification metrics and lastly they compare some of the aforementioned methods by running experiments.

Chapter 3

# BRkNN: a review

This thesis was based on the idea of the BRkNN algorithm to develop the classifiers that were used in the multi-label data classification task. Therefore, it is deemed appropriate to provide a brief presentation of this algorithm.

BRkNN (Binary Relevance kNN) is essentially a multilabel classification adaption of the kNN algorithm ($k$-nearest neighbor) that is theoretically similar to utilizing the Binary Relevance (BR) problem transformation approach in combination with the kNN algorithm. It was introduced by [9] who in their research developed two versions of the classifier, namely BRkNN-a and BRkNN-b.

Using BR, a multi-label classification problem with $L$ labels is transformed into $L$ single-label independent binary classification problems. Instead of implementing BR together with kNN, [9] developed two versions of BRkNN in order to extend the kNN algorithm so that separate predictions are produced for every label after a single search of the $k$ nearest neighbors rather than calculating the $k$ nearest neighbors $|L|$ times, saving important computation time. Version BRkNN-a, assigns to the test instance the labels that are assigned to at least half of its neighbors. Version BRkNN-b, considers the average number of labels $s$ assigned to the neighbors, and assigns to the test instance the $s$ labels with the highest confidence.

Based on these two versions of the BRkNN algorithm, we came up with four different classifiers to use in multi-label classification tasks. Our methodology involved executing the kNN algorithm on each instance and obtaining its n neighbors. Next, we deploy our classifiers for these neighbors and a classification is made according to the approach of each classifier.

Chapter 4

# The proposed algorithms

Before we present the algorithms we created, a few clarifications have to be made. The most common form of multi-label data, as well as those we used in our experiments, is the representation of the presence of a label with 1 and the non-existence of a label with 0. For example, if for a set of songs the possible labels are “rock”, “pop”, “jazz” and “blues”, then a representation in the form of 0011 means that the song has the labels “jazz” and “blues”. Similarly, a representation in the form of 0101 means that it has the labels “pop” and “blues”. For reasons that will be explained in section 5.2, the data we worked with was processed in such a way that, instead of having representations with 0 and 1, we have representations with characters (A, B, C, etc.). Thus, in the two previous examples the available labels will be A, B, C, D and the representations will be CD (for the 1st song) and BD (for the 2nd song).

While BRkNN does not consider the order of labels and assigns labels to unlabelled instances based only on the presence or absence of labels in their neighbors, our algorithms do consider the order of labels in the neighboring instances of unlabelled instances.

Before presenting the algorithms, we discuss the approach we took for representing ordered labels. Remember that in single-label classification problems (binary or multi-class), data object $i$ is represented by $x_i \in \mathbb{R}^d$, a feature vector of size $d$, followed by $y_i \in \{1, 2, \ldots, C\}$, the label, where $C$ is the total number of possible classes. In multi-label classification problems, data object $i$ has the same feature vector as before, followed by $y_i \in \{0, 1\}^C$, a binary vector of length $C$, where each element $y_{ij} = 1$ if the $i$-th instance belongs to the $j$-th class, and $y_{ij} = 0$ otherwise. Here, $C$ is the total number of possible classes.

Obviously, one needs a different approach when labels are ordered. One possible approach is to replace the ones and zeros that denote the presence or absence of a label, with an integer denoting the order of the label and omit the integer when the label is not present. Thus, $y = (4, -, 1, 3, -, -, 2, -)$ denotes that the instance is labelled with the 3rd, 7th, 4th and 1st labels in that order. The specific multi-label dataset has 8 labels and syntactically correct instances with $l <= 8$ assigned labels must have present in their $y$ vector integers 1 through $l$, with the remaining labels marked as unused.

Another approach is to assign every one of the $|L|$ labels to a single UTF-8 (or UTF-16) character and represent $y$ as a string of unique characters of maximum length $|L|$. Thus, in the previous example, $y$ = “*CGDA*”, assuming that we use characters A through H for the 8 labels. In this scenario, we deal with labelstrings instead of labelsets.

In the following, we use these two notations interchangeably in order to help the reader understand the involved concepts.

## 4.1 Majority vote per position

The first classifier (*clf1*) we developed was based on the core idea of BRkNN-a. Given a set of neighbor instances to an unlabelled instance, (*clf1*) will examine the labelstrings of the neighbors in each position and assign the label that is in majority. To give an example, suppose we have the following neighbor instance labelstrings:

- ABD
- BCA
- ACD

Based on the principle explained above, *clf1* will examine the first position of all labelstrings. There exist 2 occurrences of 'A' and 1 occurrence of 'B', therefore, '**A**' will be assigned as the first position label. Continuing at the second position, there are 2 occurrences of 'C' and 1 occurrence of 'B', so *clf1* will assign label '**C**'. Lastly, there are 2 occurrences of 'D' and 1 occurrence of 'A' at the third position, so *clf1* will assign label '**D**'. The final classification of *clf1* in this case will be labelstring **ACD**.

Of course, the example provided was a simple one with small sized labelstrings, instances with the same number of labels and labels that were not the majority in any position more than once. Most of the time, these requirements will not be met. In order to handle more complex cases as well as tiebreakers, we list some rules that we followed during the development of all classifiers:

**Rule 1** If two or more labels are in the majority at the same position, the label that is present in instances with the smallest sum of labelstring lengths (SLL) will be the one selected. This rule was implemented with the idea of "rewarding" labels that are more influential to the instances they are present.

**Rule 2** If there are still ties, a label is chosen randomly.

**Rule 3** Labels that have been assigned are not considered again in later positions.

**Rule 4** The length of the final labelstring is equal to the label cardinality of the neighboring instances that are examined, unless we run out of labels. Label cardinality is the average number of labels the instances have, rounded to the closest integer. In our example above, label cardinality is 3.

To better understand these rules let's take a look in a more complex example. Suppose we have the following instances:

| | | | | |
|---|---|---|---|---|
| ABC | ACDE | CBA | C | BA |

**Table 4.1.** *Example instances for clf1*

| | | Labels | | | | |
|---|---|---|---|---|---|---|
| | | A | B | C | D | E |
| Occurrences in position | 1 | 2 | 1 | 2 | 0 | 0 |
| | 2 | 1 | 2 | 1 | 0 | 0 |
| | 3 | 1 | 0 | 1 | 1 | 0 |
| | 4 | 0 | 0 | 0 | 0 | 1 |

**Table 4.2.** *Occurrences of each label in every position*

Based on table 4.2, for the 1st position *clf1* will choose between 'A' and 'C' which both have 2 occurrences. To break this tie, Rule 1 examines the sum of labelstring lengths (SLL) of the instances each of these labels is present in. 'A' is present in ABC, ACDE, CBA and BA, hence SLL(A)=12. 'C' is present in ABC, ACDE, CBA and C, hence SLL(C)=11. Therefore, label '**C**' is chosen.

For the 2nd position, we see that there is no tie so label '**B**' is chosen'.

For the 3rd position, there is a tie between labels 'A', 'C' and 'D', all having 1 occurrence. According to Rule 3, Label 'C' is excluded because it has already been chosen. Also, since SLL(A)=12 and SLL(D)=4 ('D' is only present in ACDE), label '**D**' is chosen.

The final classification of *clf1* is labelstring **CBD** because the label cardinality of the instances is:

$$\frac{\text{sum of instances' labelstring lengths}}{\text{num of instances}} = \frac{13}{5} = 2.6 \approx 3$$

## 4.2 Relevant Neighbors Only

Our second classifier (*clf2*) uses the methodology of *clf1* regarding the determination of the labels, but differs on the set of neighbors that are taken into account at each step. More specifically, the first label is chosen using all the neighbors at hand. For the determination of the labels in the remaining positions, only the subset of neighbors that are labelled with the assigned label in the previous position are kept.

We show how (*clf2*) works using an example with eleven neighboring instances from table 4.3:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Initial state | ABC | ACDE | CBA | CE | BA | EDB | ABD | CEBD | BEA | ECA | DB |
| After 1st label | ~~ABC~~ | ~~ACDE~~ | CBA | CE | ~~BA~~ | ~~EDB~~ | ~~ABD~~ | CEBD | ~~BEA~~ | ~~ECA~~ | ~~DB~~ |
| After 2nd label | ~~ABC~~ | ~~ACDE~~ | ~~CBA~~ | CE | ~~BA~~ | ~~EDB~~ | ~~ABD~~ | CEBD | ~~BEA~~ | ~~ECA~~ | ~~DB~~ |

**Table 4.3.** *Example instances for clf2*

For the 1st position, *clf2* will choose between 'A' and 'C' (3 occurrences each), ultimately choosing label '**C**', because SLL(A)=21 and SLL(C)=19.

For the 2nd position, not all instances are taken into account. Instead, as shown in the second row of Table 4.3, *clf2* will examine labelstrings CBA, CE, and CEBD (shown underlined) because these are the instances that have label ‘C’ in the 1st position. Examining their 2nd positions, label ‘B’ has 1 occurrence and label ‘E’ has 2 occurrences, therefore *clf2* will choose ‘**E**’.

For the 3rd position, *clf2* will examine only labelstrings CE and CEBD (shown underlined in the third row of Table 4.3), because they have label ‘E’ in the 2nd position (third line of Table 4.3). In this occasion, only CEBD has label in the 3rd position, so *clf2* will assign label ‘**B**’. Had CEBD been CE instead, *clf2* would have terminated without assigning a label in the third position. This is running out of labels scenario of Rule 4.

The final classification of *clf2* is ‘**CEB**’ (label cardinality = 3).

## 4.3 Right Shifting

The previous two classifiers examine labels in a per position fashion. Such an approach penalizes labels that may have strong presence in the first positions of the labelstrings, but fail to be selected because other labels had stronger presence. Our third classifier (*clf3*) attempts to remedy this situation.

The first label is chosen in a similar way as in (*clf1*) and (*clf2*). In order to determine the second label, *clf3* examines the second positions of all labelstrings whose first position has the chosen label together with the first positions of the rest of the labelstrings. In other words, it is as if the rest of the labelstrings are right shifted by one position. This process is repeated for all remaining positions. An example is provided below with the instances of table 4.3 that are shown in a different representation so that it is easier to understand the procedure of *clf3*.

| | Positions | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| | A | B | C | | | |
| | A | C | D | E | | |
| | C | B | A | | | |
| | C | E | | | | |
| | B | A | | | | |
| Instances | E | D | B | | | |
| | A | B | D | | | |
| | C | E | B | D | | |
| | B | E | A | | | |
| | E | C | A | | | |
| | D | B | | | | |

**Table 4.4.** *Initial state of example instances before the start of clf3*

Table 4.4 above presents the labels each instance has in every position. As explained before for *clf1* and *clf2*, for the 1st position *clf3* will choose the label that has the most occurrences. Since we are using the same example instances as in the example of Table 4.3, *clf3* will also choose label ‘**C**’ that along label ‘A’ has the most occurrences in the first position (3 occurrences each, shown underlined) but has the smallest SLL (19 vs 21). The situation of the table then changes in the way shown below:

| | Positions | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| | | A | B | C | | |
| | | A | C | D | E | |
| | C | B | A | | | |
| | C | E | | | | |
| | | B | A | | | |
| Instances | | E | D | B | | |
| | | A | B | D | | |
| | C | E | B | D | | |
| | | B | E | A | | |
| | | E | C | A | | |
| | | D | B | | | |

**Table 4.5.** *State of example instances for clf3 after the 1st label is chosen*

As shown in Table 4.5, all labelstrings that did not have 'C' in their first position have been shifted one position to the right while the rest remained "frozen" (grey cells). After identifying which label is in the majority for the 2nd position, our *clf3* algorithm selects label '**E**' which has the most occurrences (4) among all labels (underlined).

Lastly, this process will be repeated to select the label for the 3rd place. This is how our table will look now:

| | Positions | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| | | | A | B | C | |
| | | | A | C | D | E |
| | C | B | A | | | |
| | C | E | | | | |
| | | | B | A | | |
| Instances | | E | D | B | | |
| | | | A | B | D | |
| | C | E | B | D | | |
| | | | B | E | A | |
| | | E | C | A | | |
| | | | D | B | | |

**Table 4.6.** *State of example instances for clf3 after the 2nd label is chosen*

All previously right shifted labelstrings that do not have 'E' as their first label, are right shifted again by one position. These are labelstrings ABC, ACDE, BA, ABD, BEA and DB, whereas, labelstrings EDB and ECA from the previous stage will "freeze" (grey cells). By examining the labels in the 3rd position (Table 4.6), *clf3* will choose label '**A**' that is in the majority with 4 occurrences (shown underlined). The final classification will be '**CEA**'.

## 4.4 Labels like medals

The final algorithm with developed is *clf4*. It treats labels as Olympic medals, meaning that for each label, *clf4* counts its occurrences in every position in a set of labelstrings and then produces a classification medal table. In this table, ranking is decided by total number of 1st position occurrences first (total number of gold medals), then total number of 2nd position occurrences (total number of silver medals) and so on. If two or more labels have the same amount of total occurrences in any position, ranking is decided recursively by total number of occurrences in the remaining positions. In the case of labels having the same number of occurrences in every position the tiebreaker is, as stated before, the label that is present in labelstrings with the smallest SLL.

We present an example of how *clf4* works using the same example labelstrings from *clf2* and *clf3* (table 4.3). The following table 4.7 shows how many times a label is present in every position:

| | | Labels | | | | |
|---|---|---|---|---|---|---|
| | | A | B | C | D | E |
| Occurrences in position | 1 | 3 | 2 | 3 | 1 | 2 |
| | 2 | 1 | 4 | 2 | 1 | 3 |
| | 3 | 3 | 2 | 1 | 2 | 0 |
| | 4 | 0 | 0 | 0 | 1 | 1 |

**Table 4.7.** *Occurrences of each label in every position*

For the 1st position, *clf4* will decide between labels 'A' and 'C' that have 3 occurrences (gold medals). To break this tie, *clf4* will take a look at each label's 2nd positions occurrences. Label 'C' has 2 occurrences compared to the 1 occurrence of 'A' so the classifier's choice for the 1st position is label '**C**'.

Given that it is the only label left with three gold medals (occurrences in the 1st positions), *clf4* will undoubtedly select label '**A**' for the 2nd position.

For the 3rd position, we notice again that two labels are tied in gold medals, namely label 'B' and label 'E', each one having 2. Comparing their silver medals (occurrences in 2nd position), *clf4* will eventually assign label '**B**' because it has 4 silver medals while label 'E' has 3.

Thus, the final classification of *clf4* is '**CAB**'. The table below shows the classification medal table for the labelstrings that were examined.

| | Occurrences in position | | | |
|---|---|---|---|---|
| Label | 1st | 2nd | 3rd | 4th |
| C | 3 | 2 | 1 | 0 |
| A | 3 | 1 | 3 | 0 |
| B | 2 | 4 | 2 | 0 |
| E | 2 | 3 | 0 | 1 |
| D | 1 | 1 | 2 | 1 |

**Table 4.8.** *Classification medal table for clf4 algorithm*

With the help of table 4.8 it is now straightforward to understand why our classifier's classification was 'CAB'. Label 'C' is at the top of the table since it has 3 occurrences in 1st positions and 2 occurrences in 2nd positions, contrary to label 'A' which has also 3 occurrences in 1st positions but only 1 occurrence in 2nd positions. Similarly, label 'B' beats label 'E' because of more occurrences in 2nd positions (while both have 2 occurrences in 1st positions) and, lastly, label 'D' is at the bottom of the table having only 1 occurrence in 1st positions.

As a result, if *clf4* needed to assign one more label (meaning the label cardinality would be 4) it would choose label 'E' which is directly below 'B' since it has the same number of gold medals (1st position occurrences) but one less silver (2nd position occurrences).

Chapter 5

# Performance evaluation

## 5.1 Datasets

The datasets that were used in this research can be found on the website of the KDIS (Knowledge Discovery and Intelligent Systems) research group of the University of Cordoba [10]. It contains a variety of datasets that can be used in multi-label classification research along with their characteristics, description and source they were obtained from.

We ran our experiments in a total of 9 datasets from different fields. Details about the datasets are shown in the table below.

| **Dataset name** | **Instances** | **Labels** | **Cardinality** | **Density** |
|---|---|---|---|---|
| Birds | 645 | 19 | 1.014 | 0.053 |
| CAL500 | 502 | 174 | 26.044 | 0.150 |
| CHD49 | 555 | 6 | 2.580 | 0.430 |
| Emotions | 593 | 6 | 1.868 | 0.311 |
| Image | 2000 | 5 | 1.236 | 0.247 |
| Mediamill | 43,910 | 101 | 4.376 | 0.043 |
| Scene | 2407 | 6 | 1.074 | 0.179 |
| Water-quality | 1060 | 14 | 5.073 | 0.362 |
| Yeast | 2417 | 14 | 4.237 | 0.303 |

**Table 5.1.** *Datasets characteristics*

Apart from their names, table 5.1 includes for every dataset the number of instances it contains, the number of labels that are available, the cardinality of the instances and the density. The density of a dataset is its cardinality divided by the number of labels. All the datasets have standard formatting regarding their labelsets, i.e., a vector of ones and zeros, hence, their is no ordering recorded for the labels.

All our datasets have the same format. In an .arff file, each instance is in a separate line. At first, all the attributes are numeric and separated by comma followed by the labels this instance has. These labels are represented with 1 while the rest of the labels - which the instance does not have - are represented with 0. The order of the labels is the same for all instances.

## 5.2 Experimental setup

Depending on the number of labels each dataset has, we need the same number of characters to represent each label. Most of our datasets (7 out of the 9) have fewer labels than the number of characters of the English alphabet so we can use the capital letters from A to Z without a problem. For the remaining 2 datasets that have more possible labels (101 and 174), we can use capital and lowercase letters (A-Z and a-z) since the Jaro-Winkler distance differentiates them from each other as well as special characters (+, -, @, * etc.) and Unicode characters used with their codes.

As explained in the start of section 4, we transformed our data from sequences of ones and zeros to sequences of characters. At a next step, however, we had to deal with another limitation which was that, as stated in section 5.1, the label order for all instances is the same. For example, four instances with labels 01011 means that the string equivalents would always be four instances of 'BDE' and a label set of 'DEB' would never be possible. As a result, we decided that during the transformation to strings, the order of the labels would be randomized to ensure better representation of all possible combinations of labels. In the previous example, our approach ensures that instances with 01011 as their labels could each time be transformed to one of the possible 'BDE', 'BED', 'DBE', 'DEB', 'EBD' and 'EDB'.

After the transformation and randomization of the labels, we run a simple kNN classifier for every instance in our data and retrieve the *n* neighbors. The value of *n* ranges from 1 to 30. We then deploy our four classifiers which take as input the neighbors we found. Each classifier will return a classification for every instance based on the neighbors it examined and the process it follows to do so, as explained in section 4 for every classifier.

Our experiments were conducted in the environment of the server provided by the MSc in Artificial Intelligence and Data Analytics of the University of Macedonia. The hardware specifications are:

- CPU: Intel Core i9-10900K at 3.7-5.3GHz with 10 cores, 20 threads and 20MB cache
- RAM: 128GB DDR4
- Disk space: 4TB
- Graphics card: RTX3090

### 5.2.1 Jaro-Winkler distance

The reason we went with the approach of transforming our data from representations of ones and zeros to representations of characters is because of one of the metrics we decided to use in order to measure the performance of our classifiers. This metric is the **Jaro-Winkler distance** which is a string metric measuring an edit distance between two sequences. It was proposed by [11] and it extends the Jaro distance metric [12]. The degree of similarity between two strings is shown by their lower Jaro-Winkler distance. The score is standardized so that 1 denotes no similarity and 0 indicates an exact match.

Another characteristic of this metric is that it rewards strings with common prefixes. In other words, using the Jaro-Winkler distance, we can determine how similar two text strings are based on how similar their prefixes are.

In order to calculate the Jaro-Winkler distance, we first have to calculate the Jaro similarity and then the Jaro-Winkler similarity as depicted below:

**Jaro similarity** of two strings, $s_1$ and $s_2$, is

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \tag{5.1}$$

where:

- $|s_i|$ is the length of the string $s_i$
- $m$ is the number of the common characters between the strings
- $t$ is the half the number of the transpositions needed for all the common characters to be in the same order

**Jaro-Winkler similarity** of two strings, $s_1$ and $s_2$, is

$$sim_{jw} = sim_j + lp(1 - sim_j) \tag{5.2}$$

where:

- $sim_j$ is the Jaro similarity of strings $s_1$ and $s_2$
- $l$ is the number of the common prefix at the start of the strings with the maximum value being 4 characters
- $p$ is a constant scaling factor to give better scores for common prefixes and the standard value is $p = 0.1$

Obviously, the **Jaro-Winkler distance** of two strings, $s_1$ and $s_2$, is

$$dist_{jw} = 1 - sim_{jw} \tag{5.3}$$

The Jaro-Winkler distance was calculated easily using a python package, however a simple example is provided to understand how this metric is calculated. Let 'TRATE' and 'TRACE' be the strings we will compare. First, we calculate the Jaro similarity.

| | | | | |
|---|---|---|---|---|
| **T** | **R** | **A** | T | **E** |
| **T** | **R** | **A** | C | **E** |

**Table 5.2.** *The two strings and their common characters*

As seen in table 5.2, we can count a total of four common characters between the two strings, so $m = 4$. Continuing, to find the number of transpositions we look for the number of common characters that are not in the right order.

|  |  |  |  |
|---|---|---|---|
| T | R | A | E |
| T | R | A | E |

**Table 5.3.** *Order of the common characters*

Since all characters are in the same order, the number of transpositions that have to be made is zero, so half that number is $t = 0$. Using the Jaro similarity formula:

$$sim_j = \frac{1}{3} \times \left(\frac{4}{5} + \frac{4}{5} + \frac{4-0}{4}\right) = \frac{1}{3} \times \frac{13}{5} = 0.867 \tag{5.4}$$

Similarly, using the Jaro-Winkler similarity formula:

$$sim_{jw} = 0.867 + 3 \times 0.1 \times (1 - 0.867) = 0.867 + 0.0399 = 0.907 \tag{5.5}$$

Finally, the Jaro-Winkler distance of 'TRATE' and 'TRACE' is:

$$dist_{jw} = 1 - 0.907 = 0.093 \tag{5.6}$$

The metric confirms what was initially apparent: these two strings are really close to one another and very similar.

### 5.2.2 Hamming loss

Another metric we used in our experiments is the **Hamming loss** metric, a widely known and one of the most used metrics in classification problems. In multi-label classification, Hamming loss treats an entire predicted set of labels as a wrong classification if it does not exactly match the true set of labels. For example, if the true labels are 'ABCDE' and our classifier predicts 'ABCD', then Hamming loss is 1 (the worst possible value, the best being 0).

### 5.2.3 Ranked Hamming loss

The third metric we employed in our research is the **ranked Hamming loss**. In order to calculate it for two labelstrings, we first need to create two arrays for each labelstring. The first array, A1, is the representation of the labelstring as an ordered labelset. A1 has a size equal to the number of the possible labels. Each position in the array corresponds to a label and contains the order this label has in the instance examined. For example, if the possible labels are A, B, C, D, E and F and our labelstring is 'CAB', A1 is shown in Table 5.4.

| **A** | **B** | **C** | **D** | **E** | **F** |
|---|---|---|---|---|---|
| 2 | 3 | 1 | - | - | - |

**Table 5.4.** *Ranked Hamming loss: array A1*

Number 2 in the start of the table means that label 'A' is in the 2nd position in 'CAB', number 3 means that label 'B' is in the 3rd position and number 1 means that label 'C' is in the 1st position of the labelstring. Dashes (-) are used for labels that are not present in an labelstring.

The second array, A2, we create is based on the first one. Starting from the first position, we compare each label's rank with all the rest. We assign 1 if the rank $i$ we compare is superior to the rank $j$ (better rank implies lower numeric value); otherwise, we assign 0. In the case of comparing a rank with a dash, the rank value always wins and we assign 1 and if we compare two dashes we will assign a dash. All ranks will be compared with each other only one time. That means that the size of array A2 is equal to $\binom{n}{k}$:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

where $n$ is the number of all possible labels and $k$ is the number of labels we compare each time. In our example, $n$=6 and $k$=2 so the size of array A2 will be:

$$\frac{6!}{2!(6-2)!} = \frac{6!}{2! \times 4!} = 15$$

The array will be filled this way:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | - | - | - |

**Table 5.5.** *Ranked Hamming loss: array A2*

For the first position of array A2 we compare positions 1 and 2 from the first array, meaning values 2 and 3. Since 2<3, 2 is a better rank from 3 and we assign 1. In other words, in 'CAB' label's 'A' position (2) is better than label's 'B' position (3).

For the second position of array A2 we compare positions 1 and 3 from the first array, meaning values 2 and 1. Since 2>1, 2 is a worse rank from 1 and we assign 0. In other words, in 'CAB' label's 'A' position (2) is worse than label's 'C' position (1). We will then compare positions 1 and 4 from the first array and this process continues until array A2 is fully completed.

Ranked Hamming loss between two labelstrings is calculated by comparing their A2 arrays and calculating the standard Hamming loss. For example, if we have the following arrays for two labelstrings, one for the true (A2) and one for the predicted (A2'):

| | | | | | |
|---|---|---|---|---|---|
| A2 | 1 | 0 | 1 | - | - |
| A2' | 1 | 0 | 0 | 1 | - |

**Table 5.6.** *Example arrays for ranked Hamming loss*

We notice that the predicted labelstring has 2 misclassifications so the final ranked Hamming loss would be $\frac{2}{5}$ = 0.4.

### 5.2.4 Precision, recall, accuracy, F1 score

Finally, we also calculate the following metrics: **precision, recall, accuracy and F1 score**. After creating arrays A2 and A2', we calculate the confusion matrix for the assignments of 1, 0 and - we make. We compute the aforementioned metrics using this confusion matrix.

Let's demonstrate what was said with an example. Assume the following format for the confusion matrix:

| | | Predicted label | | |
|---|---|---|---|---|
| | | **1** | **0** | **-** |
| True label | **1** | 6 | 2 | 2 |
| | **0** | 0 | 2 | 0 |
| | **-** | 2 | 0 | 1 |

**Table 5.7.** *Example confusion matrix*

For the assignments of 1:

- True positives (TP) are the instances that were predicted as 1 and were actually 1
- False positives (FP) are the instances that were predicted as 1 and were actually not 1
- True negatives (TN) are the instances that were not predicted as 1 and were actually not 1
- False negatives (FN) are the instances that were not predicted as 1 but were actually 1

The same applies for assignments of 0 and -. After finding the number of TP, FP, TN and FN for every assignment, we calculate precision, recall, accuracy and F1 score separately for each one based on the formulas below. We then compute an average final score for each metric.

$$precision = \frac{TP}{TP + FP}$$

Precision is the ratio of true positive predictions among all positive predictions made by the classifier, meaning the ability of the classifier not to label a negative sample as positive.

$$recall = \frac{TP}{TP + FN}$$

Recall (also known as *sensitivity*) measures the proportion of true positive predictions among all actual positive instances. In other words, it is the ability of the classifier to find all the positive samples.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

The accuracy of the classifier is determined by how many of its predictions are correct over the whole dataset.

$$F1score = \frac{2 \times precision \times recall}{precision + recall}$$

The harmonic mean of the precision and recall is used to compute the F1 score, which combines both of these metrics into a single metric.

## 5.3 Results

The results of the experiments we conducted on the nine datasets are shown in the summary tables below. The first table includes the metrics precision, recall, accuracy and F1 score. The second table presents the results for Jaro-Winkler distance, Hamming loss and ranked Hamming loss. The split of the results into two tables is intended to improve table comprehension. The values in the first table should be as **high** as they can get, while the values in the second table should be as **low** as they can get. The calculated metrics values for each of the four classifiers are displayed individually for each dataset. For every metric, the best value across all classifiers is shown in bold. This is not the case if the results of all four classifiers for a given metric are identical.

| Dataset | Classifier | Precision | Recall | Accuracy | F1 score |
|---|---|---|---|---|---|
| emotions | 1 | 0,49 | 0,50 | 0,68 | 0,47 |
| | 2 | 0,50 | 0,50 | **0,69** | 0,48 |
| | 3 | **0,51** | **0,51** | **0,69** | **0,49** |
| | 4 | 0,50 | **0,51** | **0,69** | 0,48 |
| scene | 1 | 0,70 | 0,70 | 0,88 | 0,69 |
| | 2 | 0,70 | 0,70 | 0,88 | 0,69 |
| | 3 | 0,70 | 0,70 | 0,88 | 0,69 |
| | 4 | 0,70 | 0,70 | 0,88 | **0,70** |
| yeast | 1 | 0,58 | 0,59 | **0,74** | 0,56 |
| | 2 | 0,57 | 0,57 | 0,73 | 0,55 |
| | 3 | **0,59** | 0,59 | **0,74** | 0,56 |
| | 4 | **0,59** | **0,60** | **0,74** | **0,57** |
| birds | 1 | **0,37** | 0,37 | 0,91 | 0,36 |
| | 2 | **0,37** | 0,37 | 0,91 | 0,36 |
| | 3 | 0,36 | 0,37 | 0,91 | 0,36 |
| | 4 | **0,37** | **0,38** | 0,91 | **0,37** |
| image | 1 | 0,54 | 0,54 | 0,78 | 0,53 |
| | 2 | 0,54 | 0,54 | 0,78 | 0,53 |
| | 3 | 0,54 | 0,54 | 0,78 | 0,53 |
| | 4 | **0,55** | **0,55** | 0,78 | **0,54** |
| water-quality | 1 | 0,46 | 0,47 | 0,65 | 0,44 |
| | 2 | 0,46 | 0,46 | 0,65 | 0,43 |
| | 3 | 0,47 | 0,47 | 0,65 | 0,44 |
| | 4 | **0,48** | **0,49** | **0,66** | **0,46** |
| CHD_49 | 1 | 0,49 | 0,49 | 0,66 | 0,45 |
| | 2 | **0,50** | **0,50** | **0,67** | **0,46** |
| | 3 | **0,50** | 0,49 | 0,66 | **0,46** |
| | 4 | 0,49 | 0,49 | 0,66 | 0,45 |
| CAL500 | 1 | 0,43 | 0,44 | 0,75 | 0,43 |
| | 2 | **0,47** | **0,46** | **0,77** | **0,46** |
| | 3 | 0,41 | 0,42 | 0,74 | 0,41 |
| | 4 | 0,46 | **0,46** | **0,77** | **0,46** |
| mediamill | 1 | 0,66 | 0,69 | 0,95 | 0,66 |
| | 2 | **0,69** | 0,67 | 0,95 | 0,66 |
| | 3 | 0,67 | 0,70 | 0,95 | 0,66 |
| | 4 | 0,68 | **0,71** | 0,95 | **0,67** |

**Table 5.8.** *Precision, recall, accuracy and F1 score values*

We can see that all classifiers have very similar results with each other in every dataset across all metrics. Let's examine each dataset seperately:

- **emotions**: *clf3* has the best performance across all metrics
- **scene**: all classifiers have the same performance, *clf4* has slightly better results only in F1 score
- **yeast**: *clf4* has the best overall performance
- **birds**: *clf4* has the best results in 3 out of the 4 metrics

- **image**: *clf4* has the best results in 3 out of the 4 metrics
- **water-quality**: *clf4* has the best performance across all metrics
- **CHD_49**: *clf2* has the best overall performance
- **CAL500**: *clf2* has the best overall performance with *clf4* being a very close second
- **mediamill**: there is no classifier that outperforms the others

| Dataset | Classifier | Jaro-Winkler distance | Hamming loss | Ranked Hamming loss |
|---|---|---|---|---|
| emotions | 1 | 0,69 | 0,30 | 0,48 |
| | 2 | 0,70 | 0,30 | **0,47** |
| | 3 | **0,68** | 0,30 | **0,47** |
| | 4 | **0,68** | 0,30 | **0,47** |
| scene | 1 | 0,30 | 0,10 | 0,18 |
| | 2 | 0,30 | 0,10 | 0,18 |
| | 3 | 0,30 | 0,10 | 0,18 |
| | 4 | 0,30 | 0,10 | 0,18 |
| yeast | 1 | **0,49** | 0,25 | **0,39** |
| | 2 | 0,53 | 0,25 | 0,40 |
| | 3 | **0,49** | **0,24** | **0,39** |
| | 4 | **0,49** | **0,24** | **0,39** |
| birds | 1 | 0,50 | 0,07 | 0,13 |
| | 2 | 0,50 | 0,07 | 0,13 |
| | 3 | 0,50 | 0,07 | 0,13 |
| | 4 | 0,50 | 0,07 | 0,13 |
| image | 1 | 0,47 | 0,20 | 0,34 |
| | 2 | 0,47 | 0,20 | 0,34 |
| | 3 | 0,47 | 0,20 | 0,34 |
| | 4 | **0,46** | 0,20 | **0,33** |
| water-quality | 1 | 0,54 | 0,36 | 0,53 |
| | 2 | 0,57 | 0,35 | 0,53 |
| | 3 | 0,54 | 0,35 | 0,52 |
| | 4 | **0,53** | **0,34** | **0,51** |
| CHD_49 | 1 | **0,65** | 0,34 | 0,51 |
| | 2 | 0,66 | 0,34 | **0,50** |
| | 3 | 0,66 | 0,34 | 0,51 |
| | 4 | 0,67 | 0,34 | 0,51 |
| CAL500 | 1 | 0,63 | 0,21 | 0,37 |
| | 2 | **0,61** | **0,19** | **0,34** |
| | 3 | 0,63 | 0,22 | 0,38 |
| | 4 | **0,61** | 0,20 | 0,35 |
| mediamill | 1 | 0,52 | 0,04 | 0,08 |
| | 2 | 0,57 | 0,04 | **0,07** |
| | 3 | 0,52 | 0,04 | 0,08 |
| | 4 | **0,51** | 0,04 | 0,08 |

**Table 5.9.** *Jaro-Winkler distance, Hamming loss and ranked Hamming loss values*

Once more, we observe that the final results produced by each classifier are quite similar:

- **emotions**: *clf3* and *clf4* have the best overall performance in 2 out of the 3 metrics
- **scene**: all four classifiers have the same performance
- **yeast**: *clf3* and *clf4* have the best results, *clf1* very close to them
- **birds**: no classifier outperforms the others
- **image**: *clf4* has slightly better results that the rest of the classifiers
- **water-quality**: *clf4* has the best performance across all metrics
- **CHD_49**: overall same results from all classifiers
- **CAL500**: *clf2* has the best overall performance
- **mediamill**: generally same results across all classifiers

While it is clear that all four classifiers have very similar performances and not a single one of them stands out as clearly best or clearly worst of all, it is also important to mention that *clf4* slightly stands out in many cases, often delivering superior results compared to the other three classifiers. Similarly, *clf3* has shown instances where it either produces better outcomes than the others or is up there with *clf4*. These observations suggest that while the overall performance across classifiers is relatively balanced, *clf4* and *clf3* exhibit particular strengths in specific scenarios, highlighting their potential advantages in certain applications.

In order to better display the small difference in performance by the four classifiers, we present one graph for each dataset where Jaro-Winkler distance (in orange) and Hamming loss (blue) are illustrated for every $k$ we ran our experiments for (1 to 30).

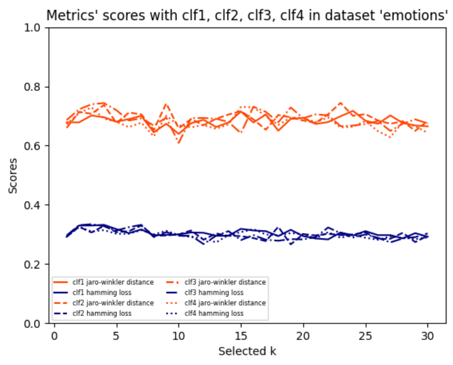

**Figure 5.1.** *Metrics for 'emotions' dataset*

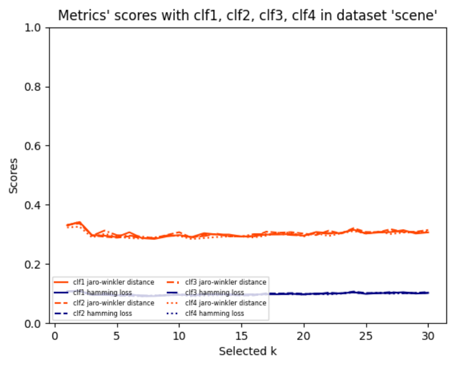

**Figure 5.2.** *Metrics for 'scene' dataset*

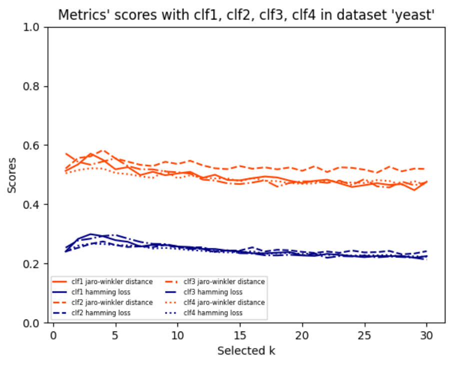

**Figure 5.3.** *Metrics for 'yeast' dataset*

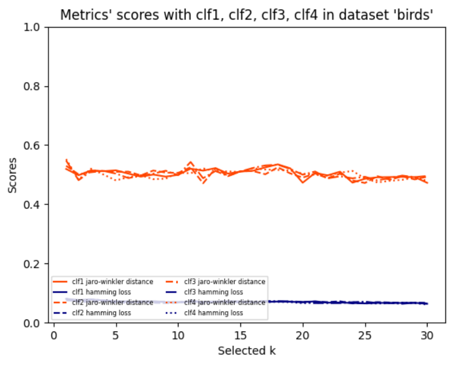

**Figure 5.4.** *Metrics for 'birds' dataset*

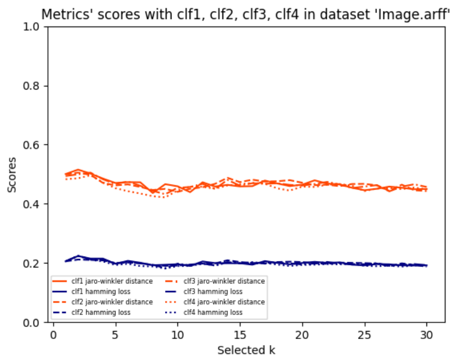

**Figure 5.5.** *Metrics for 'image' dataset*

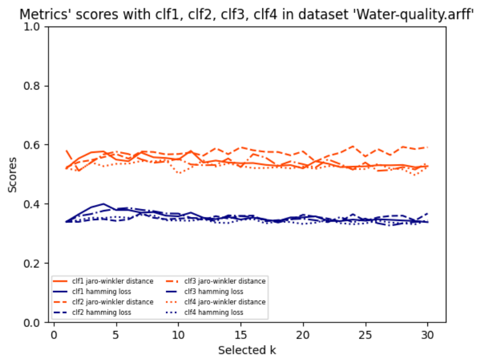

**Figure 5.6.** *Metrics for 'water-quality' dataset*

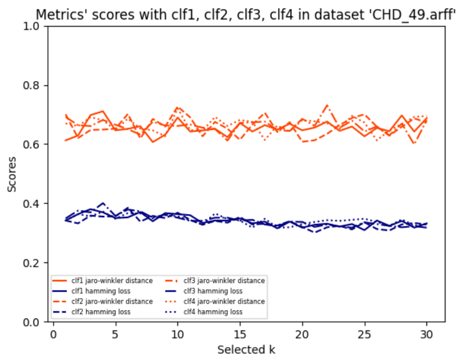

**Figure 5.7.** *Metrics for 'CHD49' dataset*

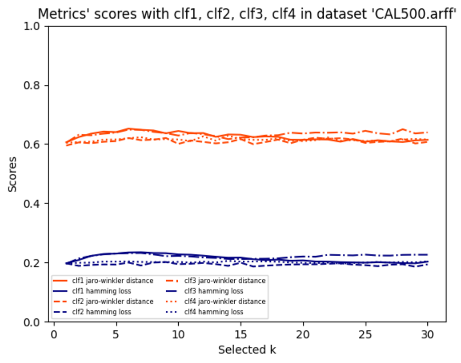

**Figure 5.8.** *Metrics for 'CAL500' dataset*

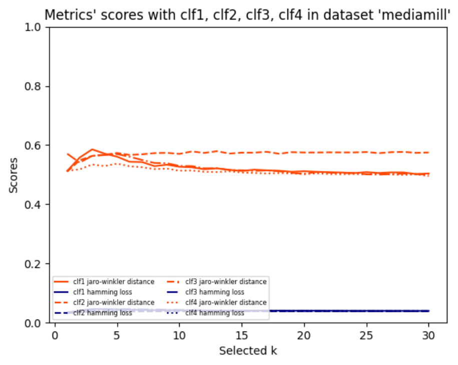

**Figure 5.9.** *Metrics for 'mediamill' dataset*

The results of the classifiers in the figures above demonstrate very clearly what was presented in table 5.9; that all four of them are very close in both metrics with little variations and spikes. In figure 5.1 we observe that all classifiers exhibit nearly identical behaviors for the '*emotions*' dataset with minor variations in their overall performance. On the other hand, figure 5.2 shows that the four classifiers perform almost the same on the '*scene*' dataset. For the '*yeast*' dataset (figure 5.3), although Hamming loss has no visible differences between the classifiers, it is worth mentioning that *clf2* has slightly worse Jaro-Winkler distance than the other three classifiers, an observation we previously reported in table 5.9. We also verify the results of the same table for datasets '*birds*' and '*image*' as figures 5.4 and 5.5 respectively show no apparent differences for all classifiers in either metric.

The graph for the metrics' results of the '*water-quality*' dataset (5.6) shows again a quite equal output of the classifiers in terms of Hamming loss values, however *clf2* stands out once more as a slightly inferior performer regarding Jaro-Winkler distance. Dataset '*CHD49*' results of figure 5.7 demonstrate a more balanced performance regarding Hamming loss. As for the Jaro-Winkler distance, there are a number of spikes visible along the range of *k*, but still no classifier shows clear advantage over the others. Moving to figure 5.8 and the '*CAL500*' dataset, we note that for a little bit over the first half of the *k* values *clf1* produces slightly worse results than the other three classifiers in both metrics, *clf3* does the same for the remaining *k* values and *clf2* is by far the top classifier across all *k* values in both metrics. Finally, in figure 5.9 of the '*mediamill*' dataset all classifiers have identical results regarding the Hamming loss metric while *clf2* stands out by having the worst results in terms of Jaro-Winkler distance.

Chapter 6

# Conclusion

## 6.1 Summary

The problem we intended to address with this research is the classification of multi-label data where the labels are ordered. Since all available for bench-marking datasets do not register order in their labelsets, we processed all datasets by introducing relevant labelstrings for each labelset. That was a necessary step in order to apply evaluation metrics that take order into consideration. Our methodology involved deploying the kNN algorithm and retrieving a number of neighbors for each instance to be classified. We developed four classification algorithms in the spirit of BRkNN with the added capability of considering label orders in their decisions.

Following the implementation of our classifiers, we recommended seven metrics to assess the performance of them: Jaro-Winkler distance, Hamming loss, ranked Hamming loss, precision, recall, accuracy and F1 score. It was clear that the four classifiers' performances were fairly comparable to one another. However, we would argue that classifier 4 (*labels like medals*) and classifier 3 (*right shifting*) performed somewhat better in several cases, which makes them worthy of a higher ranking than the other two. These observations suggest that while the overall performance across classifiers is relatively balanced, *clf4* and *clf3* exhibit particular strengths in specific scenarios, highlighting their potential advantages in certain applications.

## 6.2 Limitations

One of the most important limitations of our work was the difficulty to acquire real-world data. Multi-label datasets containing data with ordered labels can be challenging to find and obtain. Although we tried to remedy this deficiency by randomizing the produced labelstrings, unfortunately, this process of converting labelsets to labelstrings created unnatural datasets, since the order of the labels was decided in a random way. Normally, instances that are similar in terms of their features (neighbors in n-dimensional space, for n features), also share similar labelsets, and, should also share similar labelstrings. Unfortunately, this connection was probably ruined by the randomization procedure during the creation of the labelstrings.

## 6.3 Future work

One of the most important extensions to our work is the development of further algorithms that will follow different methodologies to classify multi-label data with ordered labels. New algorithms could be proven to provide better results and perform overall better than the four classifiers we suggested. Another opportunity that can be used in future research is the improvement of the time it takes for the algorithms to run. While at no point in our research were we limited by the computing time of our classifiers, we feel that handling datasets that are bigger in size is an aspect of this thesis that deserves to be explored more thoroughly. Lastly, it would be beneficial for this research field to have more real-world datasets with ordered labels available in order to conduct better testing of algorithms and obtain more precise findings.

# Bibliography

[1] Yangming Zhou, Yangguang Liu, Jiangang Yang, Xiaoqi He και Liangliang Liu. *A Taxonomy of Label Ranking Algorithms*. *Journal of Computers*, 9:558, 2014.

[2] Panagiotis Filippakis, Stefanos Ougiaroglou και Georgios Evangelidis. *Prototype Selection for Multilabel Instance-Based Learning*. *Information*, 14(10):1, 2023.

[3] Klaus Brinker, Johannes Fürnkranz και Eyke Hüllermeier. *A Unified Model for Multilabel Classification and Ranking*. *Proceedings of the 17th European Conference on Artificial Intelligence*, ECAI 2006, σελίδες 489–493. IOS Press, 2006.

[4] Klaus Brinker και Eyke Hüllermeier. *Case-based multilabel ranking*. *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, σελίδα 702–707, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[5] Serhat Bucak, Pavan Mallapragada, Rong Jin και A.K. Jain. *Efficient Multi-label Ranking for Multi-class Learning: Application to Object Recognition*. *Proceedings of the 12th IEEE International Conference on Computer Vision*, IEEE 2009, σελίδες 2098–2105, 2009.

[6] Emine Dari, Yesilkaynak V. Bugra, Alican Mertan και Gozde Unal. *RLSEP: Learning Label Ranks for Multi-label Classification*. *arXiv (Cornell University)*, 2022.

[7] Yuncheng Li, Yale Song και Jiebo Luo. *Improving Pairwise Ranking for Multi-label Image Classification*. *arXiv (Cornell University)*, 2017.

[8] Grigorios Tsoumakas και Ioannis Katakis. *Multi-Label Classification: An Overview*. *International Journal of Data Warehousing and Mining*, 3:1–13, 2009.

[9] Ioannis Vlahavas Eleftherios Spyromitros, Grigorios Tsoumakas. *An Empirical Study of Lazy Multilabel Classification Algorithms*. *Proc. 5th Hellenic Conference on Artificial Intelligence (SETN 2008)*, 2008.

[10] KDIS. *Multi-Label Classification Dataset Repository*, 2023.

[11] William Winkler. *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*. *Proceedings of the Section on Survey Research Methods*, 1990.

[12] Matthew A. Jaro. *Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida*. *Journal of the American Statistical Association*, 84:414–420, 1989.