UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE POSTGRADO Y EDUCACIÓN CONTINUA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DIVIDE AND CONQUER: AN EXTREME MULTI-LABEL CLASSIFICATION APPROACH FOR CODING DISEASES AND PROCEDURES IN SPANISH

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIA DE DATOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

JOSÉ MIGUEL BARROS SANFUENTES

PROFESORA GUÍA:
JOCELYN DUNSTAN ESCUDERO

PROFESOR CO-GUÍA:
ANDRÉS ABELIUK KIMELMAN

MIEMBROS DE LA COMISIÓN:
BENJAMÍN BUSTOS CÁRDENAS
DENIS PARRA SANTANDER
MATÍAS TORO IPINZA

SANTIAGO DE CHILE
2023

Resumen

Divide y conquista: Un enfoque basado en extreme multi-label classification para codificar procedimientos y enfermedades en Español

La codificación clínica es la tarea de transformar documentos médicos en códigos estructurados siguiendo una ontología estándar. Dado que estas terminologías están compuestas por miles de códigos, este problema puede considerarse una tarea de clasificación extrema de etiquetas múltiples. Esta tesis propone una nueva arquitectura basada en redes neuronales para la codificación clínica.

Primero, aprovechamos al máximo la naturaleza jerárquica de las ontologías para crear clústeres basados en relaciones semánticas. Luego, usamos un módulo *Matcher* para asignar la probabilidad de que los documentos pertenezcan a cada grupo. Finalmente, el *Ranker* calcula la probabilidad de cada código considerando solo los documentos en el clúster. Esta división permite una diferenciación detallada dentro del grupo, que no puede abordarse utilizando un único clasificador.

Además, dado que la mayor parte del trabajo anterior se ha centrado en resolver esta tarea en inglés, realizamos nuestros experimentos en cuatro corpus de codificación clínica en español. Los resultados experimentales demuestran la efectividad de nuestro modelo, logrando resultados de vanguardia en tres de los cuatro conjuntos de datos. Específicamente, superamos a los modelos anteriores en dos subtareas de la tarea compartida CodiEsp: CodiEsp-D y CodiEsp-P. También superamos a los modelos anteriores en el corpus FALP.

Abstract

Clinical coding is the task of transforming medical documents into structured codes following a standard ontology. Since these terminologies are composed of thousands of codes, this problem can be considered an Extreme Multi-label Classification task. This thesis proposes a novel neural network-based architecture for clinical coding.

First, we take full advantage of the hierarchical nature of ontologies to create clusters based on semantic relations. Then, we use a *Matcher* module to assign the probability of documents belonging to each cluster. Finally, the *Ranker* calculates the probability of each code considering only the documents within the cluster. This division allows a fine-grained differentiation within the cluster, which cannot be addressed using a single classifier.

In addition, since most of the previous work has focused on solving this task in English, we conducted our experiments on four clinical coding corpora in Spanish. The experimental results demonstrate the effectiveness of our model, achieving state-of-the-art results on three of the four datasets. Specifically, we outperformed previous models on two subtasks of the CodiEsp shared task: CodiEsp-D and CodiEsp-P. Also we obtained state-of-the-art results in the FALP corpus.

Dedicado especialmente a mi familia y amigos. En segundo plano también a toda persona que sienta que me ayudó para lograr hacer esto posible.

Acknowledgments

Primero que nada, le agradezco a mi familia que me han apoyado durante toda mi vida sin pedir nada a cambio y me aguantaron a lo largo de los vaivenes que significaron 6 años de estudiar ingeniería. Sin la fuerza que me dan, las risas y los descargos nada de esto habría sido posible.

En segundo lugar, me gustaría agradecer especialmente a mi profesora guía, Jocelyn, por todo el apoyo y ayuda que me ha brindado a lo largo del tiempo que estuve haciendo la tesis e incluso después de este. Incluso teniendo en cuenta de mis errores continuos y tardanzas en temas administrativos estuvo ahí siempre con paciencia empujándome, sin ella es imposible que hubiera conseguido la motivación para terminar esta etapa. Además su guía en todo lo que fue mi iniciación en la academia fue impecable y me ayudó a aprender mucho más de lo que imaginaba.

También me gustaría agradecer a todos mis amigos y amigas, los de la infancia, los del colegio, los de la U, los del trabajo, por brindarme apoyo y descargo emocional continuo, por sacarme a carretear, por acompañarme en infinitas aventuras a lo largo del tiempo. Una mención especial para mis roomies que aguantaron la música y el humo de cigarro de las intensas jornadas de programación y escritura.

Doy las gracias también a los integrantes de mi comisión, los profesores Benjamín Bustos, Dennis Parra y Matías Toro, por todos sus comentarios y observaciones propuestas para mejorar esta tesis.

Finalmente, quiero dar las gracias al grupo de investigación PLN CMM, y en especial a Matías Rojas, por ayudarme a definir el tema de tesis, proveerme ideas de datasets, dar observaciones del documento de tesis y acompañarme en el diseño de las soluciones desde las distintas perspectivas interdisciplinarias de cada uno de ellos.

Table of Content

1 Introduction **1**

1.1 Problem Statement 2

1.2 Hypothesis 2

1.3 Objectives 3

1.3.1 General Objective 3

1.3.2 Specific Objectives 3

1.4 Methodology 4

1.5 Thesis Structure 5

2 Background and Related Work **6**

2.1 Scientific Disciplines 6

2.1.1 Artificial Intelligence (AI) 6

2.1.2 Machine Learning 7

2.1.3 Natural Language Processing (NLP) 10

2.2 Text Classification 12

2.2.1 Task Formalization 12

2.2.2 Multi Label Text Classification 13

2.2.3 Extreme Multi Label Text Classification (XMC) 14

2.3 Clinical coding 15

2.3.1 Clinical coding in languages other than English 16

2.3.2 Codiesp 16

2.3.3 Cantemist 17
2.3.4 FALP 17

3 Divide and Conquer - DaC 18

3.1 DaC Corpus Preprocessing 18
3.2 Matcher 20
3.2.1 Transformers 20
3.2.2 Settings Matcher 22
3.3 Ranker 23
3.3.1 TF-IDF 24
3.3.2 One Vs Rest 25
3.3.3 Gradient Boosting Trees 26
3.3.4 Settings XGBoost 27
3.4 Combining results of the Matcher and Ranker 28
3.5 Ensemble 29
3.6 Data Augmentation using Named Entities 30
3.7 Library 31
3.7.1 Library important classes 31
3.7.2 Library files 37

4 DaC on multiple medical corpora 39

4.1 Corpora 39
4.1.1 CodiEsp 40
4.1.2 Cantemist 40
4.1.3 FALP 41
4.2 Ontologies and Cluster Choice 42
4.2.1 Ontology - ICD-10-CM 42
4.2.2 Ontology - ICD-10-PCS 43

4.2.3 Ontology - ICD-O-3 44
4.3 Metrics 46
4.4 Evaluation 47
4.5 Results 48
4.6 Module Analysis 50

5 Conclusions and Future Work **55**
5.1 Conclusions 55
5.2 Future Work 55
5.3 Contributions 56

Bibliography **66**

List of Tables

4.1 Statistics of the corpora. 39

4.2 Clusters defined for ICD-10-CM Ontology. 42

4.3 Clusters defined for ICD-10-PCS Ontology. 44

4.4 Clusters defined for ICD-O-3 Ontology. 44

4.5 Overall results on three clinical coding datasets. Results of Clinical Transformers are taken from the author's paper. FlatNER + Search engine results were calculated by us using predictions provided to us by the authors. All the other results are from the competitions overview. Some results are missing because those approaches were not implemented for the corresponding tasks. 48

4.6 Report of metrics for each module and model trained in CodiEsp Diagnostics. The F_1 scores of both the DaC model and the Ranker use only the first three characters of the code as the label, following the procedures of how to evaluate the models created by the competition. The bolded results indicate the best metric score for each module and the underline mark the worst performance. 51

4.7 Report of metrics for every module and model trained in CodiEsp Procedures. The F_1 scores of both the DaC model and the Ranker use only the first four characters of the code as the label, following the procedures of how to evaluate the models created by the competition. The bolded results point to the best metric score for each module, and the underline marks the worst performance. 52

4.8 Report of metrics for every module and model trained in Cantemist. The bolded results point to the best metric score for each module, and the underline marks the worst performance. 53

4.9 Report of metrics for every module and model trained in FALP. The bolded results point to the best metric score for each module, and the underline marks the worst performance. 54

List of Figures

1.1 CodiEsp Electronic Health Record annotated, every diagnostic and procedure mention has a unique code. Every code from this Electronic Health Record is aggregated and the document is labeled with all the codes present in the document. Each entity mention and its span is later used in the different data augmentation techniques explained in section 3.6. Figure extracted from [77]. 3

2.1 Diagram with the main disciplines belonging to the Artificial Intelligence field. Figure extracted from [5]. 7

2.2 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit. Figure extracted from [100]. . . . 9

2.3 Artificial Neural Network. Figure extracted from [2]. 10

2.4 Basic Decision Tree that decides whether to surf or not to surf. Figure extracted from [7]. 11

3.1 Overview of the DaC Architecture. 19

3.2 Overview of the Matcher module. $P(C_i)$: probability of document having a code in cluster i. 20

3.3 Overview of the Transformers architecture [109]. 21

3.4 Overview of the Ranker module. $P(L_{ij})$: probability of document having a mention of code i in cluster j. 23

3.5 Example of CART Tree. Figure extracted from [4] 26

3.6 Overview of the DaC Library structure. 38

Chapter 1

Introduction

Mapping electronic health records into alphanumeric codes allow rapid summarization of information, which is necessary to calculate costs, support clinical decisions, and conduct epidemiological studies. However, manual coding is time-consuming, resource-intensive, and error-prone, even for specialists. For this reason, developing tools to support this task is precious. The International Classification of Diseases (ICD) is a medical glossary (or ontology) published by the World Health Organization, which establishes specific coding rules for healthcare procedures and diseases.

Natural Language Processing (NLP) is the area in computer science that deals with the interaction between humans and machines through language, either by machine recognition of text or speech, or the machine-production of text or voice responses [50]. Some contemporary uses of NLP are machine translation, dialogue systems (voice assistants and chatbots), extracting information from social media, or machine reading (for example, to organize emails). Like many areas of knowledge, NLP has shown tremendous advances from the use of representation learning neural networks along with access to computing power, as well as the availability of large bodies of texts (known as corpora, plural of the Latin word *corpus*) [50].

Clinical coding is an important NLP task that seeks to automatically assign codes to medical documents following a standard terminology, such as the ICD ontology. Since each document can be labeled with more than one code from an extensive list, this task can be considered an Extreme Multi-label Classification task [65].

Despite the availability of clinical resources in Spanish, such as Cantemist [76] or CodiEsp [77], the available resources are not yet comparable to those available in English. For example, CodiEsp has 1.000 documents, while the MIMIC-III dataset [54] has 52.726 discharge summaries. This scarcity of data forces models in other languages to have a different architecture than those correctly working on the English datasets.

We introduce a novel architecture for solving clinical coding on four Spanish clinical corpora to fill this gap. Our model is composed of two modules: the Matcher and the Ranker. The first module calculates the probability of a document belonging to a cluster, while the second performs the code classification task. Each cluster is created previously by analyzing the ontology of the labels of each corpus.

Finally, we evaluate the architecture on four different corpora that exist for clinical coding in Spanish; CodiEsp Diagnostics (CodiEsp-D) [77], CodiEsp Procedures (CodiEsp-P) [77], Cantemist [76] and FALP [110], achieving state-of-the-art performance on CodiEsp-D, CodiEsp-P, and FALP corpus according to the MAP and F_1 score. Also, we obtain competitive results on the Cantemist corpus.

1.1 Problem Statement

Identifying which codes are mentioned in a medical document is challenging due to the highly imbalanced distribution of disease codes and the vast number of codes present in the medical documents. For example, the Codiesp-D corpus has 2,557 distinct disease codes in only 1,000 documents, and some of these codes are not present in the training and validation examples. This enormous amount of classification labels characterizes this problem as an Extreme Multi-Label Classification (XMC) task. XMC as a field of study is characterized for various difficulties regarding the training resources needed, the amount of time that it consumes to train a classifier, and the scarcity of examples for each label [65].

These difficulties and requirements force the creation of different architectures than those used for more straightforward classification tasks. One characteristic that helps with the design of architectures for this task is that because there are a vast amount of labels, these labels have to be related in some way. For example, the labels on these corpora are paired to their dictionary definition, so some definitions are semantically close to others while others are more apart. Another example would be if the labels are part of categories of labels, so labels in the same category are closer to each other than labels in another category. This relation can be easily obtained for the clinical domain by using the code structure. Each code is part of an ontology that was created for grouping similar entities with each other, so the use of these ontologies can help us group semantically similar labels.

In Figure 1.1 we can see an example of an annotated Electronic Health Record with diagnostic and procedure mentions. Each one of these mentions is associated with a unique disease or procedure code. For the clinical coding task, the goal is to predict the list of codes in each electronic health record.

1.2 Hypothesis

The work hypothesizes that it is possible to leverage semantic relations systematized by health organizations to build an extreme multi-label text classifier that can obtain state-of-the-art performance in Spanish clinical coding corpora. We believe that a successful clusterization of medical entity codes, recent advances in representation learning algorithms, and gradient boosting trees will give us an edge compared to other approaches to the same task.

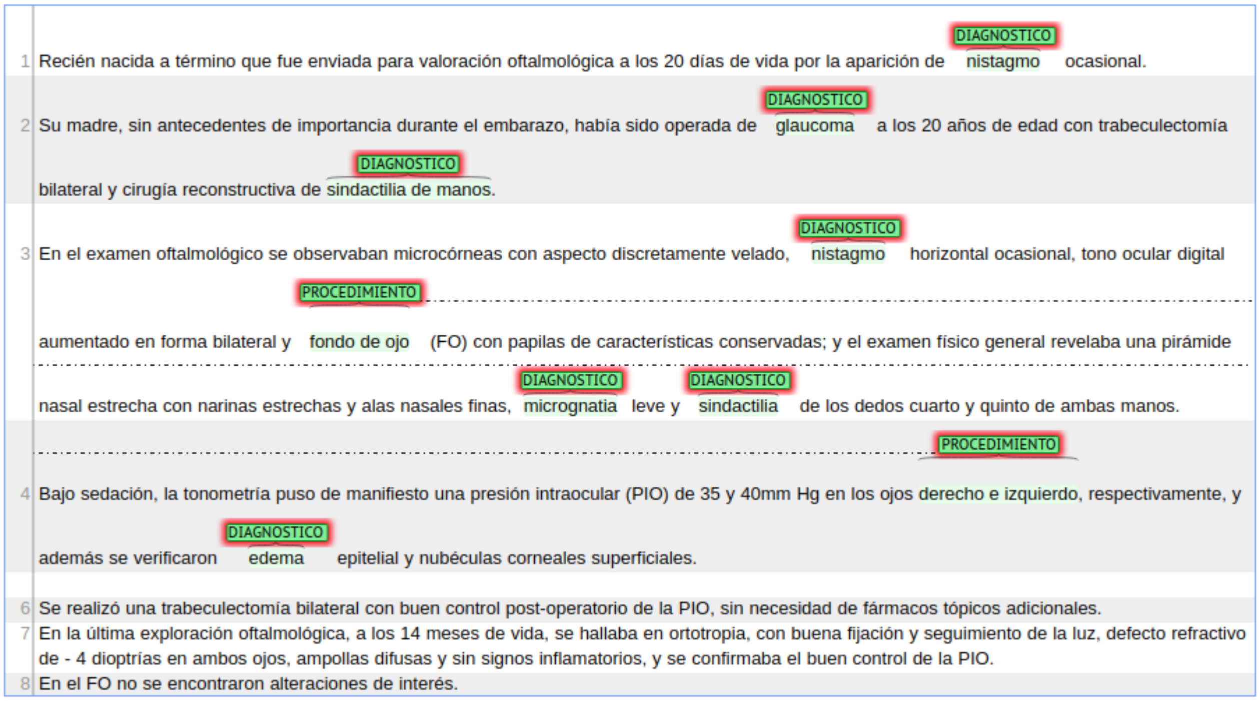
Recién nacida a término que fue enviada para valoración oftalmológica a los 20 días de vida por la aparición de nistagmo ocasional.
Su madre, sin antecedentes de importancia durante el embarazo, había sido operada de glaucoma a los 20 años de edad con trabeculectomía bilateral y cirugía reconstructiva de sindactilia de manos.
En el examen oftalmológico se observaban microcórneas con aspecto discretamente velado, nistagmo horizontal ocasional, tono ocular digital aumentado en forma bilateral y fondo de ojo (FO) con papilas de características conservadas; y el examen físico general revelaba una pirámide nasal estrecha con narinas estrechas y alas nasales finas, micrognatia leve y sindactilia de los dedos cuarto y quinto de ambas manos.
Bajo sedación, la tonometría puso de manifiesto una presión intraocular (PIO) de 35 y 40mm Hg en los ojos derecho e izquierdo, respectivamente, y además se verificaron edema epitelial y nubéculas corneales superficiales.
Se realizó una trabeculectomía bilateral con buen control post-operatorio de la PIO, sin necesidad de fármacos tópicos adicionales.
En la última exploración oftalmológica, a los 14 meses de vida, se hallaba en ortotropia, con buena fijación y seguimiento de la luz, defecto refractivo de - 4 dioptrías en ambos ojos, ampollas difusas y sin signos inflamatorios, y se confirmaba el buen control de la PIO.
En el FO no se encontraron alteraciones de interés.

Figure 1.1: CodiEsp Electronic Health Record annotated, every diagnostic and procedure mention has a unique code. Every code from this Electronic Health Record is aggregated and the document is labeled with all the codes present in the document. Each entity mention and its span is later used in the different data augmentation techniques explained in section 3.6. Figure extracted from [77].

1.3 Objectives

1.3.1 General Objective

The main objective of this thesis is to develop an extreme multi-label classifier that can obtain state-of-the-art performance in one of the most relevant shared tasks in the Spanish language for clinical coding, Codiesp-D. We expect to prove that our architecture approaches the task in a way that optimizes the Mean Average Precision (MAP), surpassing previous work in the same evaluation conditions. Applying this architecture to other corpora and establishing competitive performance is also a relevant part of this work.

1.3.2 Specific Objectives

1. Design and implement an architecture that can leverage the semantic relations between clinical codes and obtain state-of-the-art performance on clinical coding corpora in Spanish.
2. Evaluate this architecture on relevant and available clinical coding corpora in Spanish. Choose only corpora with prior work that can be compared to the architecture performance in a straightforward manner.

3. Create a library that allows the architecture to be easily extended to other extreme multi-label corpora.

1.4 Methodology

In order to accomplish the specific objectives described above, this section presents the methodology proposed for our research. Precisely, our work mainly consists of the following steps:

1. Clusterize ICD-10-CM (diseases), ICD-10-PCS (procedures), and ICD-O-3 (oncology diseases) codes in a way that maximizes the semantic relation of codes in the same cluster. The ontologies and the choice of clusters are detailed in section 4.2.
2. Build a module, the Matcher, that can correctly predict to what clusters each document belongs to. This module is built using transformers [109], and we test different pre-trained transformers alternatives. The module description can be found in section 3.2.
3. Build a module, the Ranker, that can correctly predict a document's labels if we already know to what cluster it belongs. This module is built by creating one classifier for each label following a one-vs-rest (OvR) approach. The medical documents are encoded using TF-IDF, and every classifier is an XGBoost model [30]. The module description can be found in section 3.3.
4. Combine the predictions of both modules, giving light to the architecture proposed, the Divide and Conquer (DaC) model. The approach taken to combine the output of the two models is explained in section 3.4.
5. Design a method to combine the output of different instances of the architecture, thus creating an ensemble of strong learners. The ensemble voting system is described in section 3.5.
6. Provide methods for doing data augmentation on the corpora, given the scarce resources available for each corpus. This data augmentation methods come in two forms: one using the named entities of the corpus and another one using the dictionary definitions of the ontology codes. The data augmentation techniques are described in section 3.6.
7. Create a library that simplifies the reproducibility of experiments performed using the architecture and allows for implementing the model on other corpora in a straightforward manner. The library implementation is explained in section 3.7
8. Evaluate this architecture on Codiesp-D, Codiesp-P, Cantemist, and FALP corpora on the standard metrics used by the Codiesp and Cantemist shared tasks, which are F_1 score and Mean Average Precision (MAP) of the ranked codes. The results are reported in section 4.5.
9. Provide general insights as to when the DaC architecture thrives and do a brief analysis of the performance of each module. The analysis of the modules performance is executed in section 4.6.

1.5 Thesis Structure

The rest of the thesis is organized as follows: in Chapter 2, we give a brief overview of the theoretical background needed to understand our research and review the related work in clinical coding tasks. Chapter 3 explains the architecture and all its modules, giving theoretical insights and arguments for every design decision and parameter choice. Next, in Chapter 4 we evaluate the architecture and its modules in the corpora mentioned. Also, we describe the corpora, the ontologies of each corpus, and the clusterization decided for each ontology. Finally, the last chapter summarizes the conclusions of this work and discusses some of the future research lines for the project.

Chapter 2

Background and Related Work

This chapter starts by defining and reviewing the scientific disciplines in which our research was based. Then it describes the parent tasks that englobe our specific task, text classification, and its more difficult descendants: multi-label text classification and extreme multi-label text classification. It also presents some of the most known and used methods for these tasks.

Finally, this chapter describes the task at hand, disease coding, and some broadly used methods. It also extensively reviews the works for disease coding in the Spanish corpora in which we tested our architecture.

2.1 Scientific Disciplines

2.1.1 Artificial Intelligence (AI)

Historically, intelligence has been thought of as a unique capability of humans in which we can acquire, understand, and use knowledge. The ability to acquire knowledge is mainly associated with data mining. It has evolved fast due to the advent of the Internet, which has produced and continues to produce vast amounts of data.

The field of artificial intelligence attempts to understand and build intelligent entities that can use the collected data for specific tasks. To be more precise, the field of weak-AI tries to solve specific sets of tasks because there are other theoretical approaches known as strong-AI in which a machine would have intelligence equal or superior to humans not circumscribed to a specific task. This work is categorized in the weak-AI field given that it tries to solve a specific task [100]. As shown in Figure 2.1, there are two main fields in weak-AI that aim to solve different tasks: computer vision and natural language processing.

The base of an artificially intelligent entity, and the core concept behind computer programming, are algorithms. An algorithm is a sequence of instructions that should be carried out to transform the input into an output. Since computers were first built, we have been able to devise algorithms for many tasks, and as a consequence, nowadays, we use computers

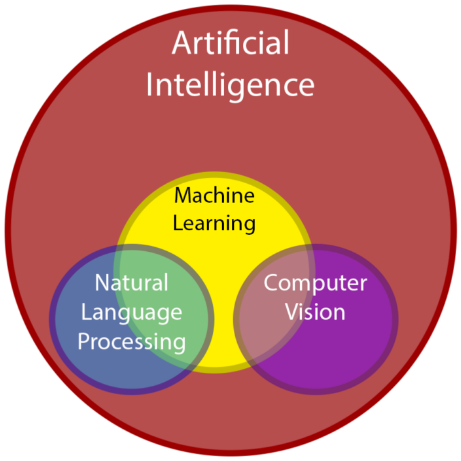

Figure 2.1: Diagram with the main disciplines belonging to the Artificial Intelligence field. Figure extracted from [5].

for all sorts of purposes. They have become an indispensable part of our everyday life, both professionally and socially, and digital technology has become the primary means to store, process, and transmit information [11].

2.1.2 Machine Learning

Although artificial intelligence is being able to create an artificial artifact that can solve a given task in various ways, it is different from machine learning on a fundamental level. Machine learning is the ability to solve a given task by *learning*. However, what is learning in this context?

Learning is applied to a broad range of processes, making it difficult to define precisely. A dictionary definition includes phrases such as "The act, process, or experience of gaining knowledge or skill" and "Behavioral modification, especially through experience or conditioning ." With respect to machines, we might broadly say that a machine learns whenever it changes its structure, program, or data based on its inputs or in response to external information, improving its expected future performance on a given task [84].

For example, an algorithm to accurately respond to a question from a user could be just randomly choosing from a manually predefined set of responses. However, this would not be machine learning because the program does not change, and the performance on the task, measured as correct responses, does not improve. Nevertheless, it may be categorized as artificial intelligence.

That is why machine learning tasks are usually tasks we do not have an algorithm to solve, despite decades of research in the task field. This is where we can see the power of machine learning because we train *models* on given data to learn an algorithm that can successfully solve a given task.

Most of these are tasks that we as human beings can do effortlessly without even being aware of how we do them. We can recognize a person from a photograph, move in a crowded room without hitting objects or people, play chess, drive a car, and hold conversations in a foreign language [11].

Machine learning algorithms are usually classified as unsupervised, supervised, or semi-supervised models. Unsupervised models seek to learn alternative data representations using the data points features. The most known unsupervised algorithms are the ones that attempt to do clusterization. Supervised models seek to learn an algorithm or mathematical function connecting input data to an output prediction. It requires a labeled dataset, and the labels are acquired by human annotation, thus being extremely costly to obtain in most cases. Finally, we have the semi-supervised models, which in their core are supervised models. However, the input and output pairs are extracted directly from the data, not by a human annotator.

In this work, we have used two algorithms that learn from the input data in a supervised manner: gradient boosting trees, described in 3.3.3, and transformers [109]. The base learners for transformers are artificial neural networks, and the base learners for gradient boosting trees are decision trees. We proceed to explain both base learners as examples of simple machine learning algorithms.

Artificial Neural Networks

Artificial neural networks are the base of recent advances in the machine learning community, mainly because they are the foundation of deep and representation learning. The base learner of an artificial neural network is an artificial neuron, which in the beginning was tightly based on how neurons work; however, nowadays, the evolution of artificial neural networks has diverged from biological replication. An artificial neuron is a mathematical abstraction that linearly scales different features based on the neuron's weights. After that, an activation function allows some of the information to pass to the next connected layer adding non-linear operations to the abstraction. Figure 2.2 shows how an artificial neuron works in a simplified manner.

The real power of neural networks is obtained by combining multiple artificial neurons in one layer, allowing for parallelization, and stacking various layers, allowing for more capacity. In Figure 2.3, we can see an artificial neural network with multiple layers, where the output of the last layer, which has the same amount of artificial neurons as classes in the dataset, can be interpreted as the probability of each one of the classes.

It is relevant to note that to learn from the dataset's training set, the artificial neural network compares the final layer probability for every class with the gold label of the example and calculates a loss function. Then, the derivate of the loss is evaluated in every neuron using back-propagation. Finally, we use gradient descent to update the weights of every neuron, subtracting the derivate, thus attempting to minimize the loss.

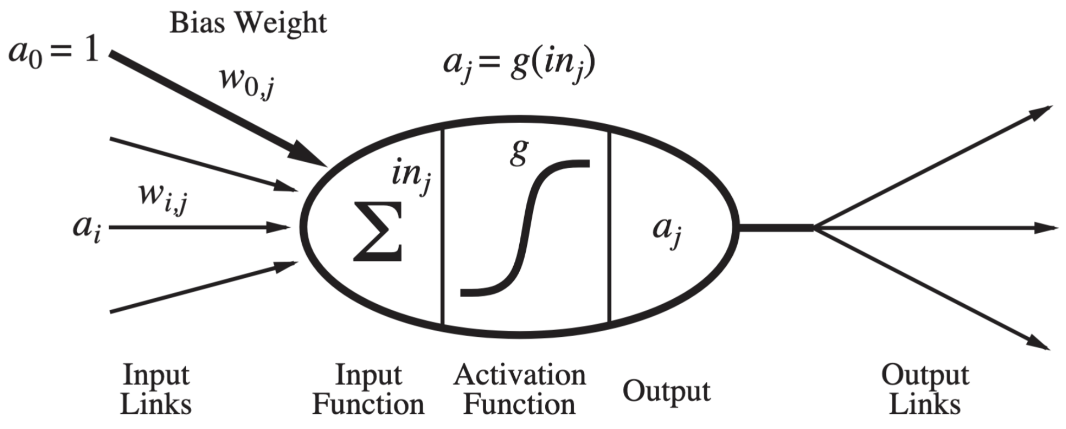

Figure 2.2: A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j}a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit. Figure extracted from [100].

Decision Trees

A decision tree is a simple and explainable structure to provide an output class for a given input example. It is built using a hierarchical structure in which we split every node based on some feature value of the input data point. Every example traverses through the tree based on its features until it reaches a leaf. Each leaf has an output class, so the example is labeled with the output class of the leaf. This basic structure with an easy-to-understand example can be seen in Figure 2.4.

The learning part of decision trees consists of conducting a greedy search to identify the optimal split points within a tree based on the training data provided. This means that the splitting process is repeated in a recursive top-down manner until most examples have been classified on one of the output classes. To obtain leaves in which there are instances of one class only, one would have to have a huge tree, leading to overfitting and incapability to generalize on new examples. That is why leaves have examples of different classes. When an unseen example belongs to a leaf, it has a probabilistic interpretation in which the output is the probabilities based on the relative weight of every class in the example's leaf.

Finally, all we have left to explain is how every individual split is performed to be able to learn from the training data. There are different methods to select an optimal split, but all of them have in common that they look in the feature space, which combination of feature and value would make child nodes that have a higher "purity" based on a purity metric. For example, one widely used approach is to minimize the Gini impurity of a split, which can be defined as,

$$\begin{aligned} \mathrm{GI}(\mathrm{split}) &= \frac{|N_1|}{|N_1| + |N_2|}\mathrm{GI}(N_1) + \frac{|N_2|}{|N_1| + |N_2|}\mathrm{GI}(N_2), \\ \mathrm{GI}(N_x) &= (1 - \sum_{c \in C} p_c(N_x)^2), \end{aligned} \tag{2.1}$$

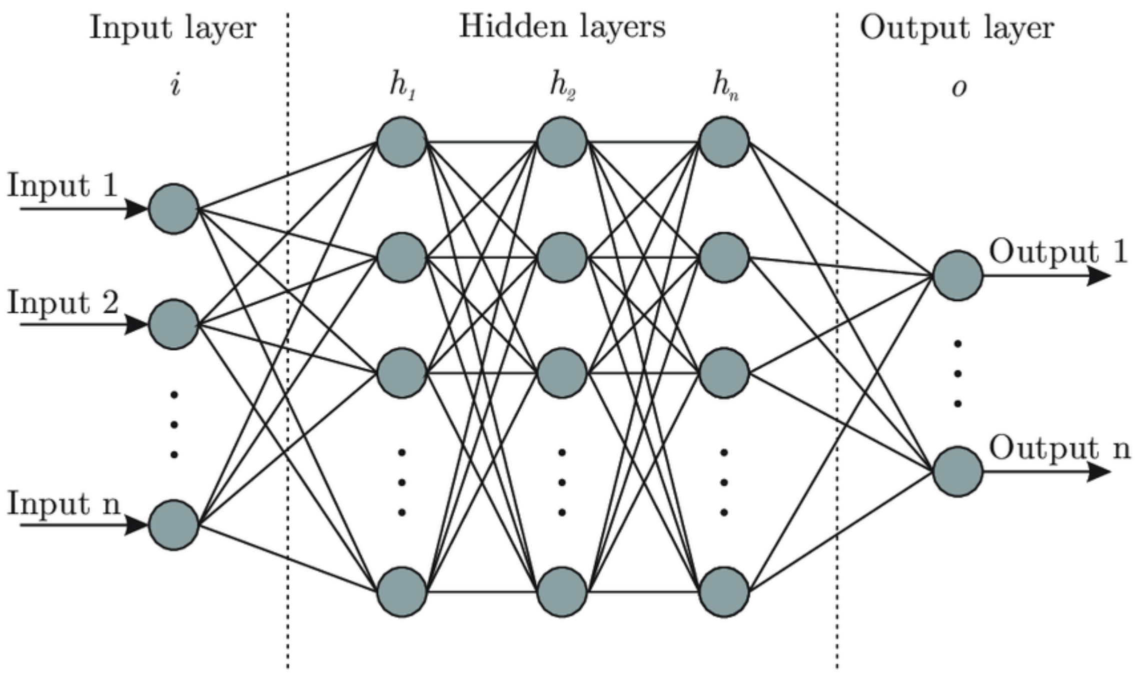

Figure 2.3: Artificial Neural Network. Figure extracted from [2].

where GI stands for Gini impurity, N_1 is the subnode 1 created, N_2 the subnode 2, N_x is the cardinality of node x, C are the classes in the dataset, and $p_c(N_x)$ is the probability of a class c in the node N_x.

2.1.3 Natural Language Processing (NLP)

Natural language processing is a collection of computational techniques for automatic analysis and representation of human languages and unstructured text. The automatic analysis of text, at par with humans, requires a deep understanding of natural language by machines, which is very difficult to obtain, taking into account the complexities and continuous evolution of human languages [33]. Also, the existence of domain-specific dialects in which the terminology is different from the general domain language used in day-to-day interactions makes it even more difficult. Most NLP tasks can be categorized in one of the following categories:

- **Text Classification:** This task's objective is to categorize documents into a set of classes. It can be used to organize, summarize, and process high volumes of data that would be unprocessable without a huge workforce doing annotation. Some of the most known tasks that can be categorized as text classification are sentiment analysis, spam filtering, hate speech recognition, and clinical coding.
- **Sequence Labeling:** This task's objective is to categorize each token of a sentence with a category. These labels can be used as features for other models, summarize information, and support human annotators. The most known examples of Sequence Labeling are part-of-speech tagging and named entity recognition (NER).

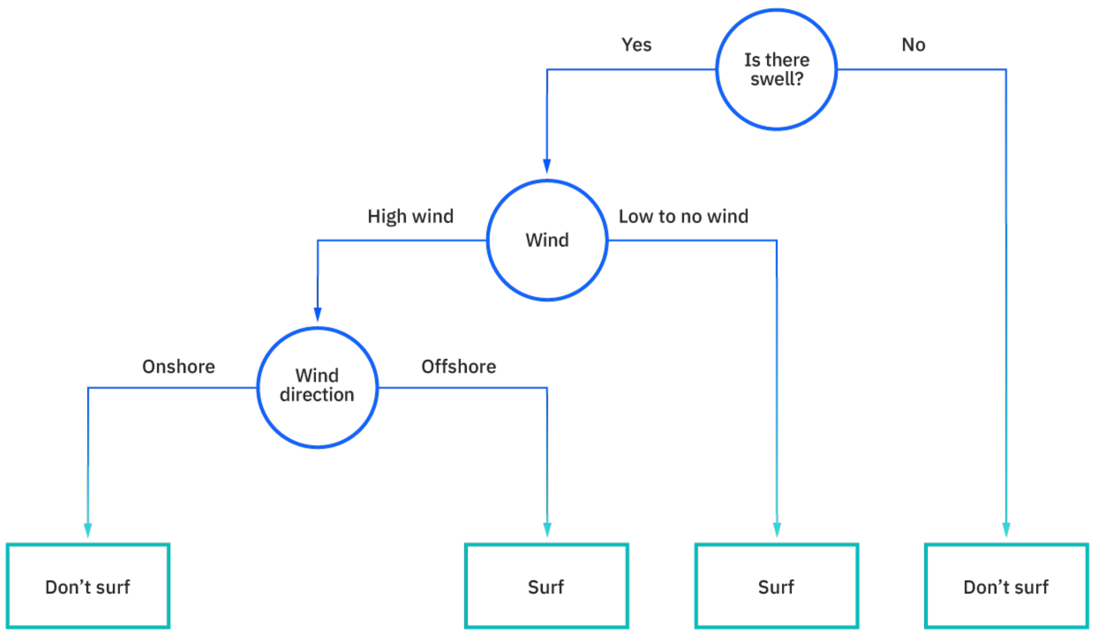

Figure 2.4: Basic Decision Tree that decides whether to surf or not to surf. Figure extracted from [7].

- **Sequence to Sequence:** This task's objective is to output a sequence of tokens, given as input another sequence of tokens. Both sequences may differ in length, so the models that tackle this task are usually built using an encoder of the input sequence and a decoder for the output sequence. They are very useful for direct interaction with humans through text. The most known tasks in which sequence to sequence is used are language translation, question answering, and summarization.

The text representation techniques used in NLP to process the text with machine learning models can be categorized into three different approaches. These three methods can be combined to solve a specific task.

- **Expert designed features:** This approach is the classical one, in which expert linguistics and developers create rule-based systems to extract features or even directly perform an NLP task. The most notable disadvantage is that the number of rules needed to correctly encapsulate all the information that a sentence or document contains is vast, so it does not scale. Also, it requires the work of experts in the matter, which yields higher costs than the following methods.
- **Bag of Words:** The text is represented as a count or a more complex weighted count, like tf-idf, of the word occurrences in a document. These have the convenience that they are effortless to implement, design, and yield decent text representations. However, its most notable drawback is that the order of the sentence is not encapsulated in this text representation technique; thus, it fails to accurately capture syntactic and semantic information. Most machine learning models can be used with this representation.

- **Unstructured text:** In this case, the order of the text is kept. This representation can only be used by models designed to capture sequential order in some manner. Recurrent Neural Networks and Transformers are the most known methods that use this representation. It has the advantage of preserving order so the model can capture syntactic and semantic information more accurately than the bag of words approach. Nowadays, most state-of-the-art methods use this kind of text representation.

We used the bag of words and the unstructured text approach for different architecture modules for this work. The following sections describe the NLP task addressed in our research: clinical coding. This problem belongs to the text classification category and can be more precisely defined as the most challenging form of text classification: extreme multi-label text classification.

2.2 Text Classification

Text Classification is an essential field of NLP in which the intention is to assign a category for a set of documents. Each one of the documents can be assigned none, one, or multiple categories. When machine learning is applied to this task, the objective is to learn classifiers that can assign these categories using training data to learn from a set of already categorized documents. Thus, it can be categorized as a supervised learning task.

Text classification is an essential and significant task in many NLP applications, such as sentiment analysis, topic labeling, question answering, and dialog act classification. Since the amount of information available to process spiked due to the advent of the internet, the required resources to process and classify vast amounts of text have made a manual approach much more time-consuming and challenging. This has forced machine learning methods to be used widely because of scalability and the ability to yield more reliable and less subjective results. Manually labeling documents can not process as many documents as machine learning algorithms and is prone to errors caused by various factors, such as fatigue and lack of expertise [64].

2.2.1 Task Formalization

In most NLP tasks, a formal definition is usually introduced to understand the problem better. This process involves identifying the input and output variables of the task under study. We present below a definition of text classification, specifically multi-label text classification, in which a document can have many different labels.

Definition 2.1 *(Multi-Label Text Classification) Given a set of documents $D = d_1, d_2, ..., d_n$, and an output set of labels $L = l_1, l_2, ..., l_m$, multi-label classification is a mapping function between the documents and the labels where for every document a variable number of labels is predicted: $f(d_j) = l_k, l_l, ..., l_x$. Text classification aims to minimize both the number of labels wrongly predicted and the number of correct labels that were not predicted.*

2.2.2 Multi Label Text Classification

Multi-Label Text Classification is a particular instance of text classification in which the amount of categories or labels that a text can be classified into is more than two, therefore it is text classification when we are not dealing with a binary classification task. Nowadays, there are three strategies to design and implement various discriminative multi-label classification methods: data decomposition, algorithm extension, and hybrid strategies [116]. It is relevant to note that extreme multi-label classification is distinct from multi-class classification, which aims to predict a single mutually exclusive label. In multi-label classification, the prediction is not only from a dataset that contains multiple labels but also a prediction of a variable number of labels.

Data decomposition strategy divides a multi-label data set into either one or more single-label subsets, constructs a sub-classifier for each subset using an existing binary classification technique, and then ensembles all sub-classifiers into an entire multi-label classifier [116]. Data decomposition is the most flexible technique because it is classifier agnostic; every classifier that supports binary classification can be used when a data decomposition strategy is utilized. One disadvantage of data decomposition is that it does not directly capture relations between labels, involves training multiple classifiers, and thus has a longer training time.

An algorithm extension strategy uses a single multi-label classifier that uses all training data set points with all their labels in a single instance all at once. This strategy can induce complicated optimization problems, such as large-scale quadratic programming in multi-label support vector machine [43], and unconstrained optimization in multi-label neural networks [118]. The most known method that uses the algorithm extension technique is neural networks, in which the last layer is the size of the classification labels and has a sigmoid activation function to categorize in a multi-label environment. One advantage of algorithm extension is that it can successfully reflect label correlation of different labels.

Finally, a hybrid strategy aims to mix the previously mentioned methods by modifying a single label classifier to be able to implicitly, using the classifier characteristics, divide the dataset into different subsets. These implementations are often the most complicated ones and require expertise on the classifier that will be modified.

The first model used for the text classification task was NB [74]. Since then many different models have been proposed for multi-label text classification such as KNN [38], SVM [53], and Random Forest [25]. Recently, the eXtreme Gradient Boosting [30], and the Light Gradient Boosting Machine [57] have been the go-to choice for tabular and generic classification, and the best choice from the traditional methods for text classification.

Apart from the traditional machine learning methods, we have the deep learning methods, which are now the most widely used ones for text classification. From these, it is crucial to mention convolutional neural networks, which were initially designed for computer vision tasks and obtained stated of the art results in that field. TextCNN [120] was the first adaption of convolutional neural networks to the field of text classification, which yielded excellent results. Finally, the transformers architecture [120] succeeded in obtaining the standard of being the best approach for text-related tasks and thus for text classification.

2.2.3 Extreme Multi Label Text Classification (XMC)

Extreme multi-label classification is a subset of multi-label classification in which the objective is to learn feature architectures and classifiers that can automatically tag a data point with the most relevant subset of labels from an extremely large label set [20].

The amount of labels that can be categorized as vast is one that most of the time exceeds or is relatively equal to the number of documents that are available to train a given corpus. In the extreme multi-label repository [20] we have various English corpora that are considered an XMC task. It is relevant to note that we have a corpus that has from a hundred labels to one million labels, so even in this particular subfield, there are many problems with their difficulties regarding optimization of time and choosing classifiers. This makes this subfield one where there are no standard methods that achieve state-of-the-art or similar to state-of-the-art results in a straightforward manner.

The vast amount of labels makes traditional and deep learning approaches inapplicable because of time complexity issues and the incapability of those methods to easily adapt to an ample label space. The baseline approach for XMC is to classify using one-vs-rest linear classifiers, which is convenient because it does not need an extended period of time to train and can be easily parallelized.

Extreme classification is a significant research problem not just because modern-day applications have many categories but also because it allows the reformulation of core learning problems such as recommendation and ranking [94]. For example, the ranking of which user one would be most acquainted with to add as a friend on social media or which video one would like to see next can be formulated as an XMC task that aims to rank the different users or videos.

Most XMC algorithms can be classified into two categories: tree-based methods and embedding-based methods. In tree-based methods, the objective is to learn a hierarchy over the label space quickly and accurately using an ensemble of trees over the feature space and minimizing a specific purity metric. Then for each leaf node, a single classifier of choosing is trained. For prediction, the new data point traverses the tree until it reaches its leaf node, and then the output is the output of the leaf classifier. Most notably, from the algorithms that use this approach, we have [94], [32], and [93].

Embedding methods exploit label correlations and sparsity to compress the number of labels. A low-dimensional label space embedding is typically found through a linear projection. Then a one-vs-rest approach can be applied over the compressed label space minimizing the number of classifiers that need to be trained. The results of the one-vs-rest classification over the compressed label space are then decompressed to the original label space to predict a new unseen data point. The algorithms that use this approach mainly differ in the choice of compression and decompression techniques such as compressed sensing, Bloom filters, SVD, landmark labels, and output codes. The most important disadvantage of embeddings methods is the loss of prediction accuracy due to the loss of information during the compression phase. Most notably, from the algorithms that use this approach, we have [21], [107], and [16].

One work that heavily inspired this work is X-Transformers [29], which proposes creating a clusterization of labels using the distance between the label descriptions encoded using transformers' contextualized embeddings. Then they predict the clusters using a transformers classifier, and finally, they predict the labels over the subset of predicted clusters using one-vs-rest linear classifiers. This is thought to handle corpora much larger than the ones we have studied in this work, thus prioritizing time efficiency much more.

2.3 Clinical coding

Clinical coding is an important NLP task that seeks to automatically assign codes to medical documents following standard terminology, such as the ICD glossary.

The most widely used corpus for clinical coding is Medical Information Mart for Intensive Care (MIMIC-III) [54], a publicly available database from MIT with de-identified medical text data of more than 50,000 patients. More than 9,000 unique ICD-9 codes are associated with the hospital admissions in this corpus, and each Electronic Health Record has more than one code. Another relevant corpus is the Electronic Intensive Care Unit (eICU) [92], formed by the Philips eICU program and contains de-identified data for more than 200,000 patients admitted to intensive care units. eICU contains 883 unique ICD-9 codes.

There has been a growing interest in clinical coding from the NLP research community in recent years. Early work focuses on creating machine learning-based classifiers with heavy feature engineering [60, 47]. However, as mentioned in [56] and [108], recent advancements in deep learning have greatly improved the performance of clinical coding models for all languages and domains.

Some of the most successful solutions to the clinical coding task come from the deep learning field and have used different approaches, including CNNs, RNNs, and Hierarchical Attention Networks [78]. Convolutional Attention for Multi-Label Classification (CAML) [79] is considered the state-of-the-art method for automatically predicting medical codes from EHRs by the most recent survey of deep learning methods for ICD coding [78]. CAML trains a neural network that passes text through a convolutional layer to compute a base representation of the text of each document, making one binary decision for each code. They use an attention mechanism instead of a pooling operation to select the parts of the document that are most relevant for each possible code.

Although transformers have revolutionized the bio-medical field by obtaining state-of-the-art on different NLP tasks without needing to use an approach more complicated than straightforward transfer learning [49], these results have not been extended to the clinical coding task. Much of this has to do with the incapability of transformers to generalize well in an XMC environment due to data scarcity and a vast amount of labels [117]. Also, another disadvantage of the transformers approach is that a great number of documents are longer than 512 tokens, which is the maximum length of a document for most transformers. Longformer [19] can address this issue by applying an attention mechanism that scales linearly to the sequence length, making it easy to process documents of thousands of tokens or longer. This attention mechanism is a replacement for the standard self-attention that combines local attention with task-motivated global attention.

2.3.1 Clinical coding in languages other than English

The ability to analyze clinical text in languages other than English opens access to critical medical data concerning cohorts of patients treated in countries where English is not the official language or in generating global cohorts, especially for rare diseases [83].

English is by far the most resource-rich language, not only in amount of public corpora available for clinical tasks, but also because of the development of advanced tools dedicated to the biomedical domain such as part-of-speech taggers [104], parsers [39], and biomedical concept extractors [14].

Despite the availability of clinical resources in Spanish, such as CANTEMIST [76] or CodiEsp [77], the available resources are not yet comparable to those available in English. For example, Codiesp has 1.000 documents, while the MIMIC-III dataset [54] has 52.726 discharge summaries.

One widely used method for dealing with fewer resources is adapting systems that work well for English to another language. In practice, this approach has been carried out with varying levels of success depending on the task, language, and system design [83]. This forces architectures in other languages to have a singular and different approach or a language adaption of those architectures correctly working on the English datasets.

Another option to address the resource scarcity in languages other than English is to use machine translation over the vast amounts of resources available in English. Nowadays, given the recent developments in machine translation, automatic translators, e.g. Google translate, can potentially reduce language bias in clinical NLP tasks.

In the following sections, we briefly describe the most notable architectures that have been implemented for three clinical coding corpora in Spanish (Codiesp, Cantemist, FALP). The descriptions of these corpora and their results can be found in Chapter 4.

2.3.2 Codiesp

For the Spanish language, one of the most popular datasets used for clinical coding is CodiEsp [77]. In Codiesp, most participants used different approaches to the task. Some identified the task as a straightforward text classification issue, while others used a NER approach and joined the mentions identified and their codes for the final classification.

Most of the work proposed formulated the problem as text classification. For example, on [71] they used a transformer-based model to classify the sentences of the documents. Then, they joined each sentence set of codes with the codes of other sentences in the same document to get the final set of codes.

Among the approaches that solved the problem as a Named Entity Recognition task, we have [36]. They created a dictionary based on entity mentions and code definitions. Then, they matched spans of documents with the code definitions in the dictionary using a tree-based algorithm.

Finally, others used an ensemble of text classification models and NER. For example, [23] implemented a model that used two different string-matching algorithms and a text classifier using one-vs-rest. The two string-encoders identified spans of texts using as target the entity mentions of codes in the documents and predicted the code with a lower distance to the entity mention. The first algorithm measured the distance between the target and every span in the document using the Levenshtein distance. The second one used the distance between the transformer contextual embeddings of the spans in the text and the target. Finally, the text classifier used a one-vs-rest approach with XGBoost as every single classifier. This model obtained the best results in the competition.

2.3.3 Cantemist

Another important task of clinical coding in Spanish is Cantemist [76], which aims to identify codes present in cancer diagnoses.

This task had two winner systems obtaining the same MAP score. The first model proposed by [45] used different transformer-based models to predict specific parts of a code. These models were ensembled using a novel voting system. The second winner was [70], who reused their approach proposed in CodiEsp (classifying the sentences of the documents and then joining the results) but further pre-trained a language model with a private oncology corpus.

Recent work by [69] outperformed previous models in CodiEsp and CANTEMIST by a wide margin. First, they trained three multilingual language models using private oncology datasets and then fine-tuned these models for classifying documents into codes. To improve the performance of their models, they ensembled the results from five different instances of each trained transformer.

2.3.4 FALP

The FALP corpus has not been released publicly yet because it is intended to be used on a shared task. Thus, we only know one work that solved the clinical coding task in the FALP corpus [110]. This work creates a NER model applying transfer learning from state-of-the-art pre-trained language models. Then the mentions found with the NER model are subsequently coded using a search engine tailored to the ICD-O codes.

Chapter 3

Divide and Conquer - DaC

As described in 1 the task at hand is one of extreme multi-label classification. To be able to approach this task correctly, we have proposed the Divide and Conquer architecture. Our proposed architecture, described in Figure 3.1, comprises two main modules: the Matcher and the Ranker. The first module calculates the probability that a document belongs to some cluster of labels, while the second one calculates the probability of labels in the document. The results of both modules are used to perform the final prediction of labels. This process is carried out by multiplying the probability of labels obtained from the Ranker for each document with the label cluster probability obtained from the Matcher module. We refer to this approach as the Divide and Conquer model since dividing the original task into two simpler text classification subtasks allows us to improve the results considerably.

It is essential to clarify that this model is designed for extreme multi-label classification; thus, every document has many labels. Also, every label will be part of only one cluster, thus making the clusters a partition of the label space.

The source code to implement this model has been made available for future use by creating a library [1] which can quickly reproduce all the results obtained and can be straightforwardly extended for other corpora. We proceed to detail every component of the model and the library itself.

3.1 DaC Corpus Preprocessing

To explain how documents are preprocessed, first we must define what clusters are. Clustering is the process of grouping similar objects into different groups, or more precisely, partitioning a data set into subsets according to some defined distance measure [73].

Most clustering algorithms are based on two popular techniques known as hierarchical[44] and partitional[63] clustering. These methods try to minimize the distance between the points in the same cluster. We can achieve this minimization using a more straightforward approach

[1] `https://github.com/plncmm/dac-divide-and-conquer`

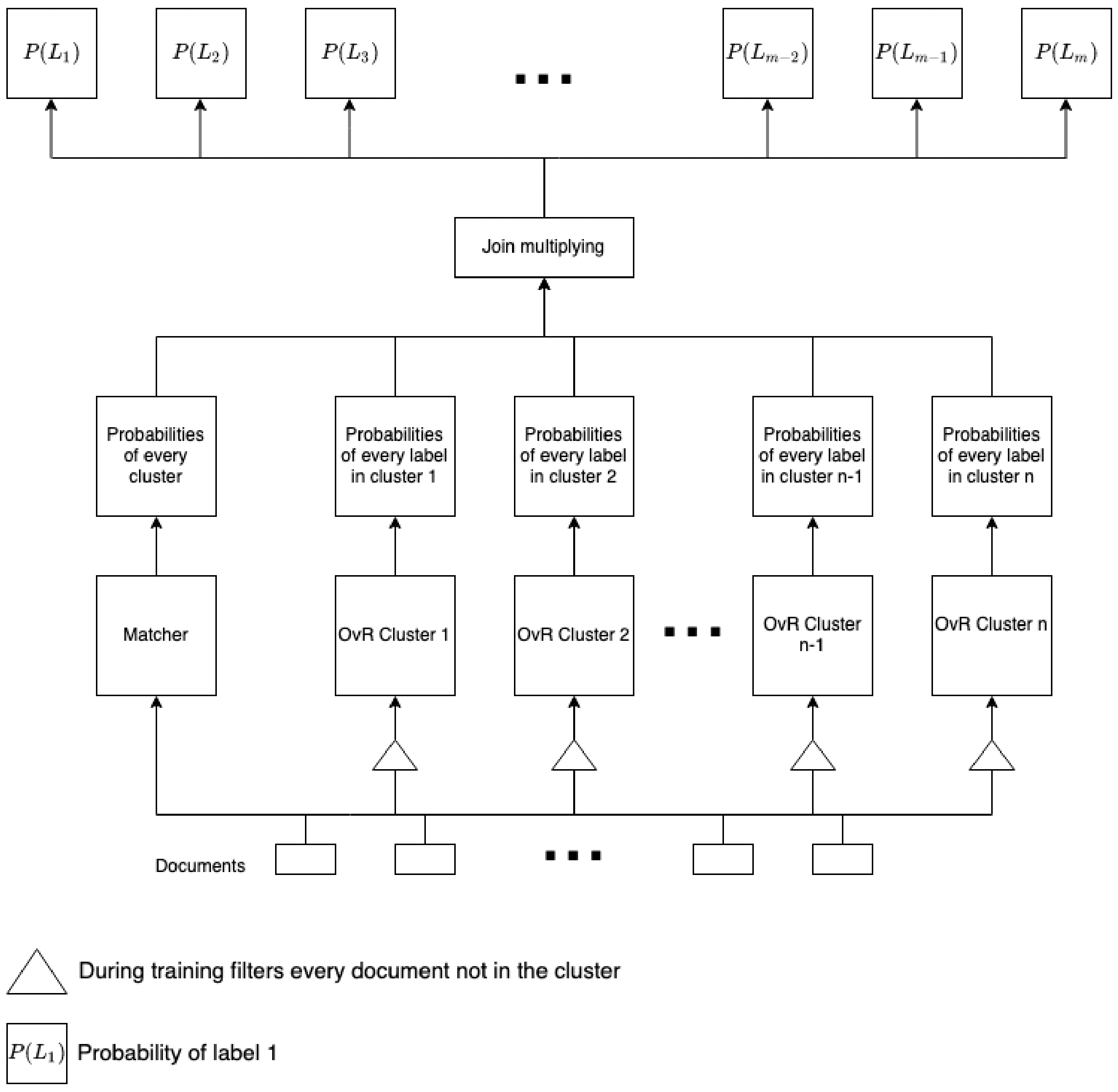
$P(L_1)$
$P(L_2)$
$P(L_3)$
$P(L_{m-2})$
$P(L_{m-1})$
$P(L_m)$
Join multiplying
Probabilities of every cluster
Probabilities of every label in cluster 1
Probabilities of every label in cluster 2
Probabilities of every label in cluster n-1
Probabilities of every label in cluster n
Matcher
OvR Cluster 1
OvR Cluster 2
OvR Cluster n-1
OvR Cluster n
Documents
During training filters every document not in the cluster
$P(L_1)$ Probability of label 1

Figure 3.1: Overview of the DaC Architecture.

leveraging the systematized ontologies in the clinical domain.

Our objective is to group semantically related labels, and for that, we use the clusters defined by the semantic clusterization of the ontologies which are divided into categories of labels. For example, for Codiesp Diagnostics, we use the first three letters of the label (that refer to a disease category) to obtain clusters of semantically related labels. We can see the defined clusters for each ontology in Tables 4.2, 4.3, and 4.4.

Then the process is divided into two modules. The Matcher for each document seen tries to predict the clusters of labels to which the document belongs. Secondly, the Ranker predicts the labels that each document contains. Then these two results are combined by multiplying the probabilities of the cluster and the labels or simply by filtering the labels predicted in clusters that were not predicted.

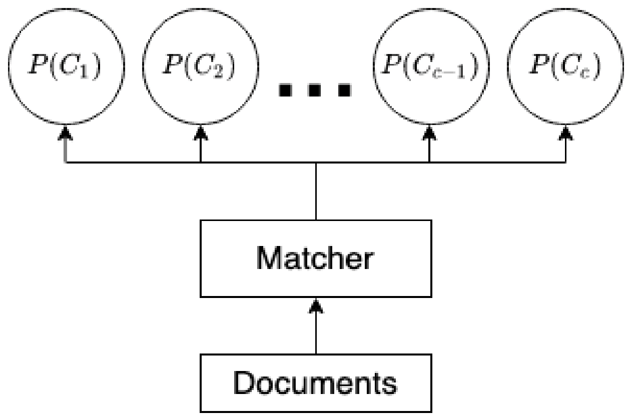

Figure 3.2: Overview of the Matcher module. $P(C_i)$: probability of document having a code in cluster i.

3.2 Matcher

As shown in Figure 3.2, the Matcher module assigns the probability of each document belonging to each cluster. Each document is part of all the clusters of the documents labels, where each label belongs to a single a cluster. This task can be formulated as multi-label text classification.

It is relevant to note that the number of clusters is significantly lower than the number of labels on the corpus. For example, in the Codiesp-Diagnostics subtask, the amount of different labels is 2.557, and the number of clusters is 21. This makes the task charged to the Matcher a more simple one, classifying in fewer classes using significantly more documents per class.

3.2.1 Transformers

To perform this classification, we decided to fine-tune a transformer-based architecture, as these models have boosted the performance of NLP architectures in several NLP tasks, including text classification.

Transformers models are based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention[109]. Figure 3.3 shows an overview of this architecture. This aims to draw global dependencies between input and output without the need for sequential computation of Recurrent Neural Networks (RNN)[99] or Long short-term memory (LSTM)[51].

The parallelization of computation that transformers achieved allowed for the pretraining of large representation learning models by speeding up the process of training them. This could not be accomplished by RNNs due to their linear nature and proved to be superior representations of words than those previously used, like Word2Vec [75], and Glove [89].

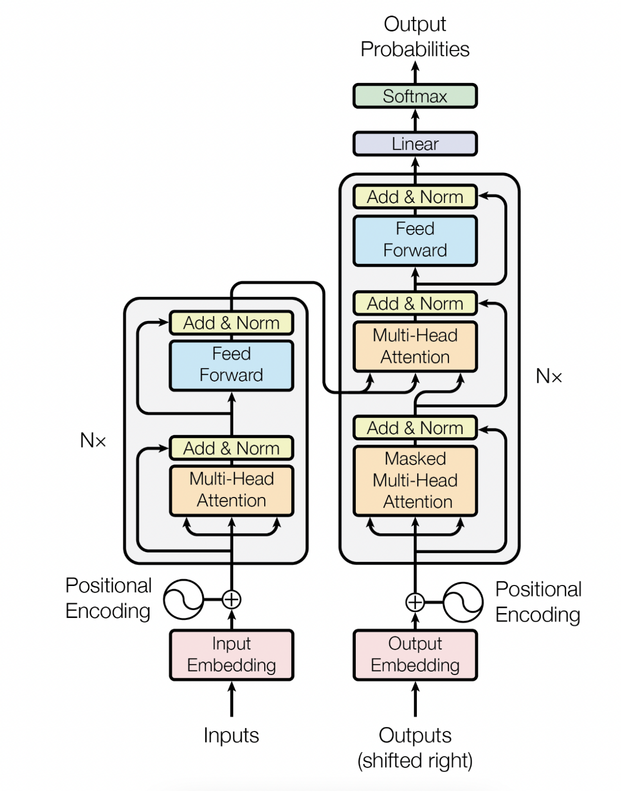

Figure 3.3: Overview of the Transformers architecture [109].

These representation learning models learn a contextual representation of the tokens training a transformer-based architecture on large corpora of unlabeled text. The models are trained for semi-supervised tasks like next word prediction, where the decoder predicts the next word of a sentence having the context of all the previous words (GPT[96], GPT-2[97]). Nowadays, the most common task to pre-train a transformer model is creating a masked language model where each sentence has a percentage of its tokens masked, and the decoder is asked to predict the words that should replace the masks (BERT[40], RoBERTa[68]).

Although these representation models were able to solve context issues, they are trained on general domain corpora such as Wikipedia, which limits their applications to specific domains like the medical domain. To enhance the performance in other domains, like the medical one, domain-specific transformer-based models like Sci-BERT[18], BioBERT[62], and BioAL-BERT[81] have been proposed [82].

One of the most important advantages of these representation models is that they learn contextual information and semantic relations and can be easily fine-tuned to more downstream tasks allowing outstanding performance on corpora with limited resources [95]. For example, for text classification, the last linear layer is discarded and replaced with a new linear layer with the number of labels in the corpus as the output size. This is later fine-tuned for a few epochs with the new corpora and task without needing the enormous amount of resources and data to train a transformer from scratch.

The fine-tuning of these models is done by freezing some of the layers of the transformer in training with a new corpus. The last layers are left to be trainable because the layers in the end are thought to have weights that relate more to the task for which they were initially trained. In contrast, the layers at the beginning are thought to capture universal features [61].

We have tested different transformers for this task (BioMedical RoBERTa [27], BioClinical RoBERTa [27] and BETO [26]) generally getting better results with BioClinical RoBERTa.

3.2.2 Settings Matcher

For the implementation, we have used the Flair framework [10] that has various tools that help simplify the implementation of these models. For instance, they implement the training cycle and metrics report. Also, they integrate with the HuggingFace hub for easily using pre-trained transformers models. It is worth noting that Flair uses torch [35] as its machine learning framework.

Because the task we are resolving is one of multi-label classification, we can not use softmax as the last layer activation function, considering that softmax keeps only one predicted class. This has led us to use the sigmoid as the last layer activation function. After having the results of the activation function, we apply a cross-entropy loss. This loss function is called Binary Cross Entropy with Logits Loss [3] and can be defined as,

$$l(x, y) = \text{mean}(l_1, ..., l_n), \quad l_i = -w_i * (y_i \log \sigma(x_i) + (1 - y_i) \log \sigma(1 - x_i)), \tag{3.1}$$

where x is the output of the linear layer prior to the activation function, y is the gold standard labels and n is the batch size.

As a scheduler, we use a linear scheduler with warmup, which linearly increases the learning from 0 to the max learning rate during warmup and then decreases the learning rate to 0.

The models are trained using the Adam with weight decay optimizer [72], which is an improved version of Adam [58]. Adam is an optimizer that uses momentum to calculate the gradient and tries to adapt the learning rate using past gradients. To accomplish this, it uses exponential windows of the gradients in each update, giving more importance to the last gradient when calculating the gradient and adapting the learning rate. This can be formalized as:

$$\begin{aligned}
P_{\delta W} &:= \beta_1 P_{\delta W} + (1 - \beta_1)\delta W & , \\
S_{\delta W} &:= \beta_2 S_{\delta W} + (1 - \beta_2)(\delta W * \delta W) & , \\
\overline{P_{\delta W}} &= \frac{P_{\delta W}}{(1 - \beta_1^t)} & , \\
\overline{S_{\delta W}} &= \frac{S_{\delta W}}{(1 - \beta_2^t)} & , \\
W &:= W - \frac{\lambda}{\sqrt{\overline{S_{\delta W}}}} * \overline{P_{\delta W}} & ,
\end{aligned} \tag{3.2}$$

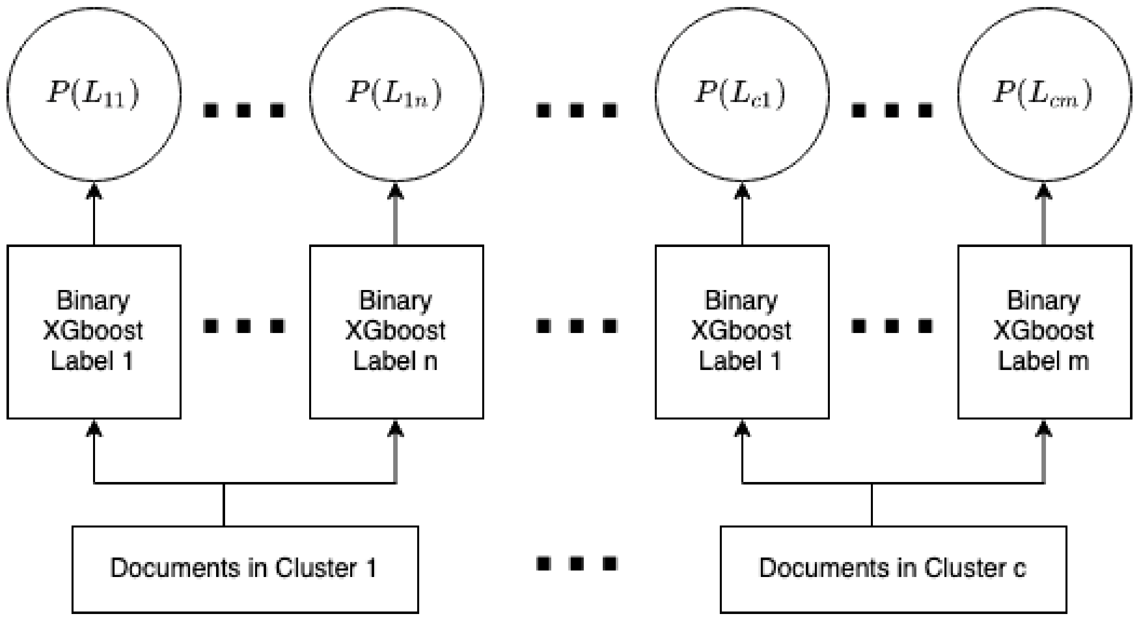

Figure 3.4: Overview of the Ranker module. $P(L_{ij})$: probability of document having a mention of code i in cluster j.

where W is the weights on each one of the parameters, $\overline{P_{\delta W}}$ and $\overline{S_{\delta W}}$ is a standard correction for exponential window means (to avoid a cold start), $P_{\delta W}$ is the gradient using momentum, $S_{\delta W}$ is the quadratic of the factor for which to adapt the learning rates, and β_1, β_2 are the weights of the past gradients for each one of $P_{\delta W}$ and $S_{\delta W}$.

As default settings, the model is trained for 15 epochs. This number was decided after finetuning the model multiple times and checking that learning stagnates in the tenth epoch. We have added five more epochs for security. It is relevant to note that the scheduler reduces the learning rate, so no damage is done by adding more epochs.

We have used different batch sizes depending on the machine we have had available to train the models but decided to keep 15 as a default batch size for evaluation, considering that it fits in the GPU memory of a Kaggle Notebook.

3.3 Ranker

The Ranker module (shown in Figure 3.4) calculates, for each possible code, the probability of belonging to the document. This process is carried out by training a single binary model per code, following a one-vs-rest approach. Each model is trained only with documents with codes belonging to the cluster. This way, the gold labels of this task are the codes in the document.

Since each document can contain many codes, this problem, like the Matcher, can be formulated as a multi-label text classification task. However, it is more challenging since possible codes are much larger than the number of possible clusters in the other task. For training the Ranker module, the input documents of the binary classifiers are encoded using the TF-IDF method, and the output is fed into an Extreme Gradient Boosting model [30].

3.3.1 TF-IDF

The objective we are seeking is to classify the documents with multiple labels from the cluster. One possible way of achieving this is by creating a ranking function for each one of the labels. The Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that reflects how important a word is to a document in the collection or corpus [101]. This method is often used in Information Retrieval to assign a score to a document according to a query and then use those scores to rank every document on the corpus and extract the most relevant.

TF-IDF is composed of two relevant term scores that support different notions and affect different metrics, which multiplied give light to a widely used term score. First, we have Term Frequency (TF) which is the number of occurrences of the query term t in document d.

Term Frequency as above, suffers from a critical problem: all terms are considered equally important when it comes to assessing relevancy of a query. In fact, certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document [102].

To avoid giving relevance to a term that does not discriminate between documents, Inverse Document Frequency (IDF) is a good weight factor because it gives higher scores for terms that are present in fewer documents, thus effectively discriminating between different documents. IDF can be defined as

$$\mathrm{IDF}_t = \log\left(\frac{N}{df_t}\right), \tag{3.3}$$

where N is the number of documents in the corpus and df_t is the number of documents in the corpus where the term t is present.

One of the most simple ranking functions and a widely used one is to simply sum the TF-IDF of each term in a query for every document and use this as the document score,

$$S_{d,q} = \sum_{t \in q} \text{TF-IDF}_t, \tag{3.4}$$

where $S_{d,q}$ is the score of document d for query q and $t \in q$ are the terms in query q.

We want to give a score for every label in a document. For this, we could get the description of the label and treat this as the query and use the sum score function, but this approach is too simple to get competitive results and does not even use the training data provided.

Thus, one better way is to use a machine learning model as a scoring function and TF-IDF encoding to represent the document. TF-IDF encoding can be defined as the output vector when the complete vocabulary is the query. With this in mind, the score for each label in a document would be,

$$S_{d,l} = f \begin{pmatrix} \text{TF-IDF}_{t_1} \\ ... \\ \text{TF-IDF}_{t_n} \end{pmatrix}, \tag{3.5}$$

where $S_{d,l}$ is the score of label l for document d, f is a machine learning model and $t_1 ... t_n$ is the TF-IDF encoding of all terms in the vocabulary for document d.

3.3.2 One Vs Rest

As was discussed in 2.2.2 there are three strategies to design and implement various discriminative multi-label classification models: data decomposition, algorithm extension, and hybrid strategies. For the ranker we have decided to use data decomposition, which can be divided into four widely used decomposition techniques: one-versus-rest (OvR), one-versus-one (OVO), one-by-one (OBO), and label powerset (LP). [116].

We have chosen to use the OvR approach, also known in the multi-label scenario as Binary Relevance method. To justify this choice, it is essential to explain the method and compare it to the alternatives. OvR trains a single binary classifier for every label in the corpus. The label of the binary classifier is 1 if the example is categorized with the label and 0 otherwise. The final prediction is the ensemble of the results of every binary classifier,

$$\text{OvR}(x) = \begin{pmatrix} f_{l_1}(x) \\ ... \\ f_{l_n}(x) \end{pmatrix}, \quad f_{l_i}(x) \in [0, 1], \tag{3.6}$$

where f_{l_i} is the binary classifier for label i and x is the input document.

It is relevant to consider that this approach has certain advantages; it is possible to train any model because all models are capable of doing binary classification, and the complexity is $\theta(n)$ because there is only one model per label.

The one-vs-one approach consists of creating one subset for each combination of labels in the corpus; these subsets have documents labeled with one of the two labels only ($\frac{n(n-1)}{2}$ subsets). Then each one of these subsets is used to train a binary classifier that aims to separate the two classes. Finally, a voting system of the $\frac{n(n-1)}{2}$ classifiers is used to get the final predictions.

Although OVO has shown to have better results than OvR [91], the complexity is $\theta(n^2)$, which, compared to OvR's complexity of $\theta(n)$, is very costly for a large number of labels, thus inapplicable for XMC.

Another option would be to use LP, which takes every combination of labels present in the corpus as a new label, transforming the multi-label task into a simple multi-class problem. This approach has the disadvantage that, in an XMC setting, the number of unique labels explodes and the training examples for each one of the classes becomes scarce, thus making the task a much more complicated one [24].

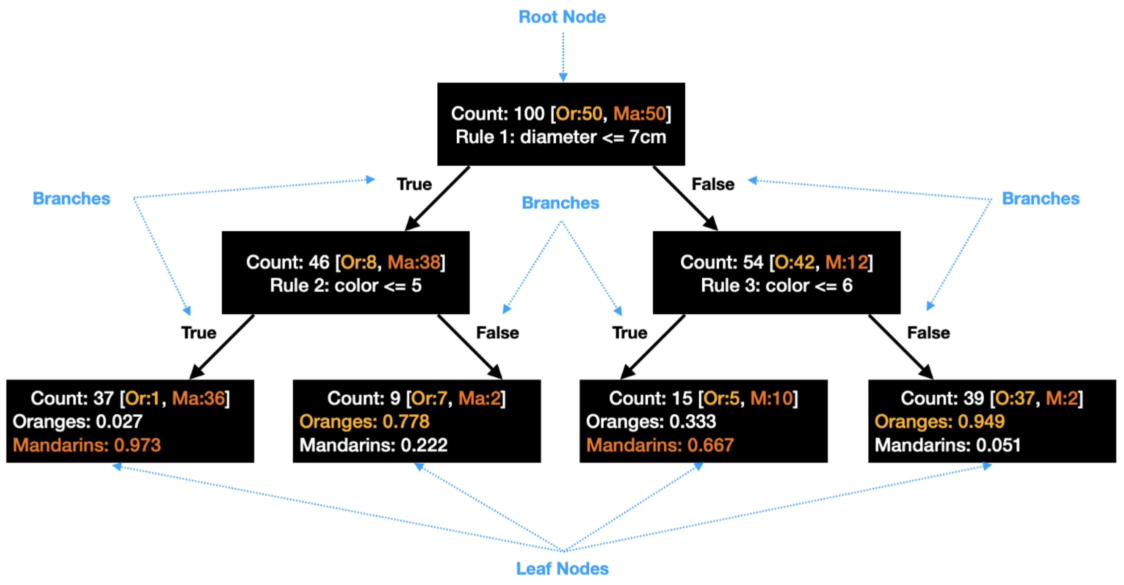

Figure 3.5: Example of CART Tree. Figure extracted from [4]

Finally, we could use OBO, which creates single-class subsets for each label, and support vector data description to build single-class classifiers. Linear ridge regression is utilized to integrate the single class classifiers into a multi-label classifier [115]. This approach has the disadvantage that not every kind of model can be trained as single-class classifiers, thus losing flexibility.

3.3.3 Gradient Boosting Trees

We have chosen to use Gradient Boosting Trees as a binary classifier for each one of the one-vs-rest models. This decision was taken, weighing the computational cost of training one model per label and the quality of the model's predictions. Although, as previously discussed, neural networks are the go-to choice when solving an NLP task, it is not feasible to train one neural network (specifically a Transformer or LSTM) per label due to the computational costs of this training in an XMC environment. Actually, due to the fact that each cluster has fewer examples than the entire corpus, even training one neural network model per cluster yields worse results because of the data scarcity issue. The other classical machine learning methods were disregarded, taking into account that the best one-vs-rest algorithm had been obtained using Gradient Boosting Trees by [23]. Also, because Gradient Boosting Trees has shown excellent results in competitions worldwide [30].

Gradient Boosting Trees is a Classification and Regression Trees (CART) ensemble. CARTs are Decision Trees that assign a score for each label in the leaf nodes using the distribution of the classes in that leaf. We can see an example of a CART that differentiates between oranges and mandarins in Figure 3.5.

The output of Gradient Boosted Trees is the sum of the results of all the composing trees.

This can be formalized as,

$$\hat{y_i} = \sum_{k=0}^{n} f_k(x_i), \tag{3.7}$$

where $\hat{y_i}$ is the predicted value for example x_i, f_k is the output of the tree k, and n is the number of trees.

It is relevant to note that Random Forests are the most commonly known ensemble of CARTs, but we are referring to these models as Gradient Boosting Trees because of how these trees are trained. While a Random Forest is trained in a parallelized way and trees are not related, Gradient Boosting Trees use a technique called tree boosting. This technique trains each tree sequentially and uses the previous trees' prediction so that the new tree minimizes the residual errors.

This optimization task is done by minimizing a regularized objective function,

$$L = \sum_{i} l(\hat{y_i}, y_i) + \sum_{k} \Omega(f_k), \tag{3.8}$$

where L is the objective function to minimize, l is any differentiable convex loss function, and $\Omega(f_k)$ is the regularization term of the tree k that penalizes the complexity of the model (i.e., the regression tree functions). The additional regularization term helps smooth the final learned weights to avoid over-fitting. Intuitively, the regularized objective will tend to select a model employing simple and predictive functions.

This optimization task includes parameters that can not be optimized using traditional optimization methods in Euclidean space, so the task is optimized using a greedy algorithm that minimizes the loss for each added tree, thus forcing tree boosting to have a sequential nature.

3.3.4 Settings XGBoost

As a tree boosting system, we have chosen to use XGBoost, which is a scalable end-to-end system that is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges. The developers of XGBoost proposed a novel sparsity-aware algorithm for sparse data (like tf-idf encoding of documents) and provided insights on cache access patterns, data compression and sharding to build a scalable tree boosting system. By combining these insights, XGBoost scales beyond billions of examples using far fewer resources than existing systems [30].

Because it is not computationally feasible to perform a grid search for hyperparameter tuning using a one-vs-rest approach, all hyperparameters were chosen using the XGBoost documentation's recommendations. First, we have chosen to use a column and row subsample of 0.6 because it helps to avoid overfitting. So each newly added tree is trained only with 60% of the documents and using only 60% of the document tf-idf features.

Also, we have set the weights of the loss function relevance of the positive class using XGBoost scale_pos_weight, which greatly helps when handling an imbalanced dataset. Since we train one XGBoost binary classifier for each label, the positive class is less than 5% of the training data in a cluster for almost all cases. The recommended value of scale_pos_weight is $\text{spw}_l = \frac{D_{\bar{l}}}{D_l}$ with $D_{\bar{l}}$ being the number of documents of the negative class and D_l being the number of documents of the positive class. We have calculated this for each label and used a constant scale_pos_weight per cluster of $\text{mean}(\text{spw}_l), l \in$ labels in cluster.

XGBoost mostly combines a huge number of CARTs with a small learning rate. In this situation, trees added early are significant, and trees added late are unimportant. We have chosen to use the DART booster, which addresses this issue by adding dropout techniques from the deep neural net community into boosted trees and reported better results in some situations [111].

Using DART, in each iteration, the loss function is calculated using a predicted value that loses the prediction leaves of randomly selected trees. Thus forcing each newly added tree to be able to predict correctly even with some of the other trees dropped, improving generalization and avoiding overfitting.

Finally, as the tree selection method, we have used the exact method which greedily enumerates all split candidates and selects the best one.

3.4 Combining results of the Matcher and Ranker

Having trained both the Matcher and the Ranker, the issue of how to combine the results is left. We follow other machine learning models and implement two different ways, one that outputs probabilities of all labels and another that simply predicts the labels of the document.

First, we can get the probabilities of belonging to a cluster and label of unclassified documents. It is important to note that the output probabilities of the Ranker are not precisely the probabilities of the label because it was trained only with documents in the cluster. More rigorously, they can be defined as the conditional probabilities of the label given that it belongs to the cluster. So to get the probabilities of the label, we can use the Bayes Theorem, which states:

$$\begin{aligned} P(L|C) &= \frac{P(C|L)P(L)}{P(C)} \quad , \\ P(L) &= \frac{P(L|C)P(C)}{P(C|L)} \quad , \end{aligned} \tag{3.9}$$

where L is label, C is cluster, $P(L|C)$ is the output probabilities of the Ranker, and $P(C)$ is the output probabilities of the Matcher. It is direct to note that $P(C|L)$, the probability of belonging to a cluster taking a label, is one if the label belongs to the cluster and zero otherwise. So this equation can be redefined as:

$$P(L) = P_{\text{Ranker}}(L) * P_{\text{Matcher}}(C), \quad L \in C \quad . \tag{3.10}$$

Another option is to output just the predicted classes. To generate this, we take a conservative approach and do an inner join of the labels predicted and the clusters predicted, which means that only predicted labels in a predicted cluster will be a predicted a label,

$$L = \text{Pred}_L \cap \text{Pred}_{LC}, \quad LC = L \in Pred_C \quad . \tag{3.11}$$

One of the problems of this architecture is that it has an error propagation from the Matcher to the Ranker. More specifically, the documents that are wrongly predicted of one cluster in the Matcher pass through to the Ranker, which has not seen similar documents in that cluster.

To avoid this issue, we have implemented a correction when training with a validation set (this can not be done if a validation set is not provided). This correction passes documents whose cluster prediction is incorrect to the training of the binary classifiers of the wrongly predicted cluster labeled with a negative class.

3.5 Ensemble

Ensemble methods are widely used methods in machine-learning and data-mining communities. By definition, an ensemble is a group of learning models whose predictions are aggregated to give the final prediction. It is widely accepted that an ensemble is usually better than a single classifier given the same amount of training information [119].

An ensemble learning framework or architecture features multiple weak predictive results based on features extracted through a diversity of projections on data. Therefore, it can be considered a standalone model for some purposes, especially regarding computational costs [41].

A different approach is to ensemble strong learners, like deep neural networks or transformers, done in the previous state-of-the-art of Codiesp and Cantemist [69]. We consider this approach unfair because it only leverages different runs of the same computationally expensive training process and thus confounds the advances obtained by creating better architectures and simply applying more computational power.

However, for the purpose of competing in similar conditions, we have also created an ensemble strategy in which we ensemble different instances of the DaC model and obtain better results by brute force. The ensemble strategy is also done at two levels, ranking the labels and predicting them.

The ranking of all the labels is done by summing the probabilities of the ensembled models for each label. The prediction of the labels is a union of the predicted labels in all the ensembled models.

$$\begin{aligned} \text{Score}(l, E) &= \sum_{m \in E} \text{Prob}(l, m) \quad , \\ \text{Predicted}(l, E) &= \max_{m \in E}(\text{Predicted}(l, m)) \quad . \end{aligned} \tag{3.12}$$

3.6 Data Augmentation using Named Entities

Data Augmentation describes a set of algorithms that construct synthetic data from an available dataset. This synthetic data typically contains small changes in the data that the model's predictions should be invariant to [103].

Most publications often overlook the importance of the data augmentation techniques used, thus making their experiments' reproducibility more challenging and claiming scores higher than those other researchers may get when testing their models. Interestingly, 70% of scientists report failure to reproduce someone else's results, and more than half can not reproduce their own [34]. To be able to reproduce all our results, the data augmentation techniques we have used are explained below. Also, the code implementing these techniques is part of the library so every implementation detail is transparent.

Furthermore, most libraries and implementations of different architectures see data augmentation techniques as a preprocessing step and thus fail to include them in the library. We have incorporated the data augmentation into the library and structured it so every user can quickly implement and test the architecture on other corpora.

We have created four types of data augmentation techniques:

- **NE Mentions**: Each mention of an entity in the corpus documents is added as a new document with the label being the mentioned entity.
- **NE Sentences**: Every document is split, by points and new lines, into all its compounding sentences. Then we add every sentence as a new document with the entity mentions in the sentence as its labels.
- **NE Stripped**: Every document is stripped of all the words that are not entity mentions. This is then added as a new document.
- **Definition codes**: For each code of the ontology we add documents with the dictionary definitions that can be found in the ontology.

The first three techniques need to have a corpus in which the different labels are associated with a span of the document (Named Entity), which is a widespread thing to happen because most corpora are created to solve Named Entity Recognition tasks and Text Classification tasks. An important term related to data augmentation is label preservation, which describes transformations of training data that preserve class information [17]. We can observe that these three augmentation techniques take the same documents and faithfully preserves the correct labels in the creation of new shorter documents.

One common issue with XMC is that some labels are present in very few documents or are not even present at all. The fourth augmentation adds roughly the same number of documents per label, so it also has the benefit of evening out the number of documents per label.

Finally, not all these data augmentation techniques are used both by the Matcher and the Ranker. In fact, the Matcher uses a transformer architecture that is trained using sentences and needs semantic context, so for the Matcher, NE Stripped would only make the results worse and is not used.

3.7 Library

The DaC library is an open-source implementation of our architecture for extreme multi-label classification leveraging semantic relations between the labels. The library is currently hosted and published on Github and can be found at link[2]. It can be downloaded using the pip package manager. The source code was structured according to the design and code patterns used by the Flair framework for managing word embeddings [10]. All the library code is written in Python and requires Python ≥ 3.6.

This library was created to fulfill the following objectives:

- Encapsulate the DaC architecture as a standalone model for ease of use
- Incorporate and make easily extensible to other corpora the data processing steps to be able to train a DaC architecture
- Incorporate and easily extend to other corpora the data augmentation steps

It also provides:

- Easily reproduce results on the different corpora tested
- Easily save and load different model instances
- Integration with Google Cloud Storage to save and upload models to/from the cloud
- The support for training different instances of the architecture and evaluating the ensemble of these instances
- Automatic creation of corpora data exploration
- Easily evaluate models using F_1 score and MAP.

3.7.1 Library important classes

To be able to extend the DaC architecture to other corpora, one python class is paramount: the DACCorpus. If these abstract class methods are implemented on other corpora, the DaC architecture can be easily trained on other corpora. The methods to implement or override are shown in Listing 1.

Listing 1: DACCorpus abstract model and methods to implement to adapt to other corpora.

[2] `https://github.com/plncmm/dac-divide-and-conquer`

```python
class DACCorpus(ABC):
    def __init__(
        self,
        corpus: str,
        data_path: Path,
        augmentation_matcher: List[Augmentation] =
         ↪  [Augmentation.descriptions_labels, Augmentation.ner_sentence],
        augmentation_ranker: List[Augmentation] = [
            Augmentation.descriptions_labels,
            Augmentation.ner_sentence,
            Augmentation.ner_stripped,
            Augmentation.ner_mention,
        ],
        augmentation_corpus: List[Augmentation] = [
            Augmentation.descriptions_labels,
            Augmentation.ner_sentence,
            Augmentation.ner_stripped,
            Augmentation.ner_mention,
        ]
    )
    pass

    # Mandatory methods to implement

    @abstractmethod
    def process_corpus(self) -> pd.DataFrame:
        """
        :return: A pandas DataFrame with the following columns:
            sentence: String with the document text
            labels: List of strings with the document labels
            split_type: train, test or dev
            filename: Filename or id of the document
        """
        pass

    @abstractmethod
    def assign_clusters_to_label(self, label: str) -> List[str]:
        """
        :param: Label string
        :return: A list of the label clusters
        """
        pass

    # Optional methods to override

    def download_corpus(self):
        pass

    def process_augmentation_ne_mentions(self) -> pd.DataFrame:
        """
        :return: A pandas DataFrame with the following columns:
            filename: Filename or id of the document
            mention_text: Mention string of an entity
            label: Label of the mentioned entity
            off0: Start offset of the mention
            off1: End offset of the mention
        """
        pass

    def process_augmentation_descriptions(self) -> pd.DataFrame:
        """
        :return: A pandas DataFrame with the following columns:
            label: Label string
            description: Text description of label
        """
        pass
```

If all these methods are implemented, testing the architecture is as simple as following the code in Listing 2.

Listing 2: Testing example when all methods are implemented.

```
data_path = Path(".").resolve()
corpus = DACCorpusImplemented("corpus_name", data_path)
corpus.reproduce_mean_models() # To reproduce 5 training rounds
corpus.reproduce_ensemble_models() # To reproduce ensemble of 15 models
```

Of course, the data can be manually downloaded, and tweaking and testing of hyperparameters of the architecture may be needed. For more significant manipulation, the code in Listing 3 can be used:

Listing 3: Testing example when download method is not implemented.

```
data_path = Path(".").resolve()
corpus = DACCorpusImplemented("corpus_name", data_path)
corpus.create_corpuses() # Does all pre processing and data augmentation
 ↪  steps
model = DACModel(corpus.indexers_path, corpus.models_path, corpus.corpus)
ranker_train_args = {}
matcher_train_args = {}
```

```
model.train(matcher_train_args=matcher_train_args,
↪   ranker_train_args=ranker_train_args)
```

All the models implement the save, train, predict, eval, and upload_to_gcp methods. The training and initialization arguments that can be passed to the Matcher and Ranker are shown in Listings 4 and 5.

Listing 4: Initialization and training arguments of Matcher model.

```
class Matcher:
    def __init__(
        self,
        indexers_path: Path,
        models_path: Path,
        indexer: str,
        transformer: str =
        ↪  "PlanTL-GOB-ES/roberta-base-biomedical-clinical-es",
        seed: int = 0,
    ):
        """
        :param indexers_path: path of preprocessing files created by
↪   DACCorpus
        :param models_path: path of where to save the models
        :param indexer: corpus name given to DACCorpus
        :param transformer: transformer name to use of HuggingFace
        :param seed: Seed to set for Flair
        """
        ...

    def train(
        self,
        upload_to_gcp: bool = False,
        augmentation: List[Augmentation] = [
            Augmentation.ner_sentence,
            Augmentation.descriptions_labels,
        ],
        max_epochs: int = 15,
        mini_batch_size: int = 10,
        remove_after_running: bool = False,
        downsample: int = 0.0,
        train_with_dev: bool = True,
        layers="-1,-2,-3,-4",
        num_workers=2,
        save_model_each_k_epochs: int = 0,
        optimizer: torch.optim = torch.optim.AdamW,
        scheduler=LinearSchedulerWithWarmup,
        **kwargs,
    ):
        """
        :param upload_to_gcp: if True uploads the model to Google Cloud
↪   Storage after training
        :param augmentation: list of Augmentations to use
        :param max_epochs: Max epochs to use
        :param remove_after_running: Removes saved model after running,
↪   may be useful to save disk space if it was already uploaded
        :param downsample: downsample of data
        :param train_with_dev: Whether to use the validation set in
↪   training
        :param layers: Layers to be finetuned
        :param num_workers: to use loading corpus
        :param save_model_each_epochs: Epochs to pass between model saves
        :param optimizer: Optimizer to use for training
        :param scheduler: Scheduler to use for training
        :param **kwargs: Other arguments that can be passed to flair
↪   trainer
        """
        ...
```

Listing 5: Initialization and training arguments of Ranker model.

```
class Ranker:
    def __init__(
        self,
        indexers_path: Path,
        models_path: Path,
        indexer: str,
        seed: int = 0,
    ):
        """
        :param indexers_path: path of preprocessing files created by
↪   DACCorpus
        :param models_path: path of where to save the models
        :param indexer: corpus name given to DACCorpus
        :param seed: Seed to set for XGBoost
        """
        ...

    def train(
        self,
```

```python
        upload_to_gcp: bool = False,
        split_types_train: List[str] = ["train", "dev"],
        augmentation: List[Augmentation] = [
            Augmentation.ner_mention,
            Augmentation.ner_sentence,
            Augmentation.ner_stripped,
            Augmentation.descriptions_labels,
        ],
        use_incorrect_matcher_predictions: bool = False,
        subset: int = 0,
        transformer_for_embedding: str = None,
        log_statistics_while_train: bool = True,
        train_starting_from_cluster: int = 0,
        train_until_cluster: int = None,
        n_jobs_ova: Union[float, int] = 1,
        n_jobs_xgb: Union[float, int] = -1,
        scale_pos_weight: str = "max",
        tree_method: str = "auto",
        booster: str = "dart",  # gbtree gblinear
        subsample: float = 0.6,
        colsample_bytree: float = 0.6,
    ):
        """
        :param upload_to_gcp: Wether to upload model to Google Cloud Storage after training
        :param split_types_train: Which splits to use for training
        :param augmentation: Which Augmentation techniques to use for training
        :param use_incorrect_matcher_predictions: Use incorrect matcher predictions for training
        :param subset: Subset of data to train
        :param transformer_for_embedding: Add transformer contextual embedding to tf-idf embedding, should provide transformer name to use for embedding
        :param log_statistics_while_train: Output stats of test evaluation during training
        :param train_starting_from_cluster: From which cluster index to start. Useful for pausing and resuming training.
        :param train_until_cluster: With which cluster index to end. Useful for pausing and resuming training.
        :param n_jobs_ova: How many hard forks to create for one vs rest. Usefull for paralellizing the training.
        :param n_jobs_xgb: How many threads to use in XGBoost train. Default all.
        :param scale_pos_weight: Wether to use the max or the mean of the cluster as scale_pos_weight
        :param tree_method: Tree method for XGBoost
        :param booster: Booster for XGBoost
        :param sumsample: Row subsample for XGBoost
        :param colsample_bytre: Column subsample for XGBoost
        """
        ...
```

3.7.2 Library files

We proceed to do a small summary of what every library module does; the library structure can be seen in Figure 3.6. First, we have the model folder where the Matcher, Ranker, and DACModel are implemented. Next, we have the dataset folder, which has a base file that implements all data processing and augmentation in a single abstract class, DACCorpus. Codiesp, Cantemist, and Falp files are the implementation of the DACCorpus abstract class for each one of those corpora.
Finally, we have different files in the root directory that contain various utils:

- **corpora_downloader.py:** Creates simple functions to download files.
- **custom_io.py:** Implements various methods to do input-output operations.
- **evaluation.py:** Implements the evaluation of ensembles and of different training rounds.
- **flair_utils.py:** Common integration methods with Flair framework.
- **gcp.py:** Integrates with Google Cloud Storage.
- **metrics.py:** Calculate metrics for the predicted sentences.
- **utils.py:** Miscellaneous utils.

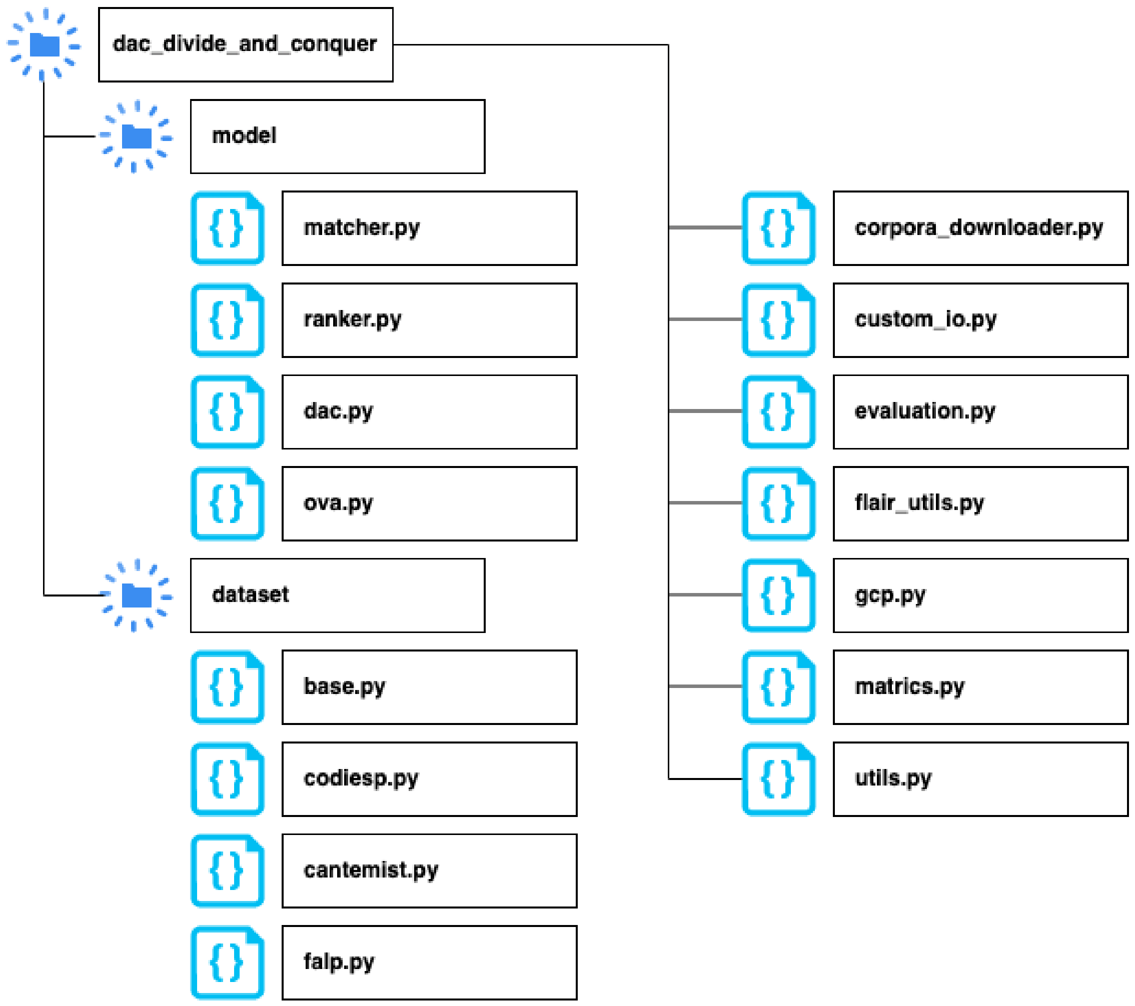

Figure 3.6: Overview of the DaC Library structure.

Chapter 4

DaC on multiple medical corpora

Having already explained the architecture, its modules, and parameters in the previous chapter, we proceed to evaluate our architecture on four different corpora that exist for clinical coding in Spanish; CodiEsp Diagnostics, CodiEsp Procedures, Cantemist and FALP. We also do a brief analysis of the different modules of the architecture and provide general conclusions about when the architecture thrives.

4.1 Corpora

The corpora were chosen to try to stay loyal to the scope of this work, which is to tag medical documents with codes from different medical ontologies. It is also relevant to notice that we have decided only to approach this issue in Spanish written documents since it is a highly used language but with less available data than English. Finally, one crucial feature we looked at in the candidate corpora was the straightforward evaluation and comparison of our architecture. This means that every corpus chosen has previous work to compare with our model, and even three of the four corpora used were created for a shared task, thus, having various models to compare. The corpora characterization can be seen in Table 4.1.

	CodiEsp-D			CodiEsp-P			Cantemist			FALP		
	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test
Documents	500	250	250	500	250	250	501	500	300	869	124	108
Avg document length	410	411	414	410	411	414	894	804	812	859	915	834
Avg codes per document	14.4	13.7	14.7	3.9	4.2	4.4	12.8	12	12.1	8.6	9.5	8.9
Avg clusters per document	4.9	4.9	4.8	1.9	2.0	2.0	2.8	2.8	2.8	2.8	2.8	2.9
Number of different codes (N_c)	2557			870			850			225		
Number of clusters	21			17			51			51		
Avg codes per document (Cardinality)	14.3			4.1			12.3			8.7		
Density = Cardinality / N_c	0.006			0.005			0.014			0.39		

Table 4.1: Statistics of the corpora.

It is relevant to note that the corpus that most suits the extreme multi-label classification task is Codiesp-D. It has three times more different codes than Codiesp-P and Cantemist. For this reason, it is also the most difficult of the tasks above, with more codes and fewer documents to train. All the corpora have a similar amount of documents to train the classifiers and evaluate them. However, the length of the cancer diagnoses in Cantemist and FALP is almost double that of Codiesp EHRs. This poses some challenges for transformers which have a limit of 512 tokens. However, the sentence separation data augmentation addresses this challenge, so it does not affect the overall performance of the architecture. Finally, the most simple task is the FALP task, having only 225 different codes and relatively the same amount of documents.

4.1.1 CodiEsp

The CodiEsp corpus [1] is a collection of 1,000 clinical case reports written in Spanish that cover a diversity of medical specialties. The training subset consisted of 500 documents, while the development and test set consisted of 250 documents each. Professional clinical coders exhaustively and manually annotated all documents with codes from the Spanish version of ICD-10 (procedure and diagnostic) [77].

The manual annotation process followed official clinical coding guidelines published for Spain. CodiEsp documents were coded with the 2018 version of CIE-10 (the official Spanish version of ICD-10-Clinical Modification and ICD-10-Procedures) and inspired by the "Manual de Codificación CIE-10-ES Diagnósticos 2018" [2] and the "Manual de Codificación CIE-10-ES Procedimientos 2018" [3] provided by the Spanish Ministry of Health. A document describing the annotation guidelines [4] was published to cover aspects and particularities relevant to the sub-tasks and documents used for CodiEsp [77].

Clinical codes (diagnostic and procedure) were linked to textual evidence fragments that support their assignment. This textual evidence was used as named entities for our architecture purposes, defining the textual evidence as the mentioned label.

While CodiEsp comprises three different subtasks, Diagnostics, Procedures, and Explainability, we have tested our architecture on two subtasks that use this corpus: CodiEsp Diagnostics and CodiEsp Procedures. The Explainability subtask consisted of giving the supporting spans of text for each code predicted, and our model does not have a module to obtain the highest weighted words of each code predicted. It is relevant to note that this could be obtained using each label XGBoost model's most important parameters, as it is an ensemble of trees, but this is subject to future work.

4.1.2 Cantemist

The Cantemist corpus is a collection of 1301 oncological clinical case reports written in Spanish. The training subset contains 501 documents, the development subsets 500, and

[1] `https://doi.org/10.5281/zenodo.3625746`

[2] `https://www.sanidad.gob.es/eu/estadEstudios/estadisticas/normalizacion/CIE10/CIE10ES_2018_norm_MANUAL_CODIF_DIAG_.pdf`

[3] `https://www.sanidad.gob.es/estadEstudios/estadisticas/normalizacion/CIE10/CIE10ES_2018_norm_MANUAL_CODIFICACION_PROCEDIMIENTOS_EDICION_2018.pdf`

[4] `https://zenodo.org/record/3730567`

the test subset 300. All documents of the corpus have been manually annotated by clinical experts with mentions of tumor morphology. Every tumor morphology mention is linked to an eCIE-O code (the Spanish equivalent of ICD-O) [76].

To increase the usefulness and practical relevance of the Cantemist corpus, the creators of the corpus selected clinical cases affecting all genders that comprised most ages (from children to the elderly) and of various complexity levels (solid tumors, hemato-oncological malignancies, neuroendocrine cancer, etc.). The Cantemist cases include clinical signs and symptoms, personal and family history, current illness, physical examination, complementary tests (blood tests, imaging, pathology), diagnosis, treatment (including adverse effects of chemotherapy), evolution, and outcome [76].

While Cantemist offered two different subtasks: NER and coding, we only used our model for the coding subtask. Following the same intuition as in CodiEsp we could use the explainability features of an ensemble of trees to provide NER mentions. Still, it would be far-fetched and probably not the best approach to this issue. The named entities from the NER subtask were used for data augmentation, as it is explained in 3.6.

4.1.3 FALP

The FALP corpus was created by collecting 1,101 anonymized pathology reports from the Oncology Institute Fundación Arturo López Pérez (FALP) cancer registry written in Spanish by pathologists. The reports already came with ICD-O morphologic and topographic codes assigned at the document-level by FALP expert coders. Subsequently, all tumor morphology and topography mentions inside the documents were manually annotated by clinical experts [110].

Two FALP clinical experts with extensive experience in ICD-O coding performed manual annotation of the FALP corpus using INCEpTION. This platform offers an intuitive user interface, flexible configuration of the annotation scheme, and workflow support with annotation and adjudication stages. It also offers an assistance system through recommenders and active learning, which improves the efficiency of the manual annotation process [110].

Morphology annotations were performed following the Cantemist guidelines, while topography annotation guidelines were developed from scratch following the design of the Cantemist project [110].

The DaC architecture was tested only on the morphology mentions and could be extended to the topology mentions. As the FALP corpus was primarily created for NER we defined the codes in a document as all the codes of the named entities of the document and used the named entities as a data augmentation technique.

It is relevant to note that the FALP corpus is private and is intended to be used for a shared task, so the data is not yet available for reproducibility but will be in the forthcoming future.

4.2 Ontologies and Cluster Choice

Every corpus previously mentioned has documents tagged with medical codes in some ontology. The ongoing rapid growth of the amount of available data and its use in a wide range of domains to solve different complex tasks requires sophisticated techniques of intelligent information and knowledge management. These efforts have resulted in a significant number of data integration and management systems and tools enriched with vocabularies, thesauri, terminologies, and ontologies [52].

There are no universally accepted definitions for ontology, but in computer science, it is usually defined as an explicit, formal specification of a shared conceptualization [48].

Associating diseases and medical entities to unique codes started in medicine around 1880, and for an extended period, the ICD was the only medical terminology resource. The newest edition (ICD-10) is translated into 42 languages and maintained by the World Health Organization (WHO) [86]. All the corpora we have analyzed are coded in different variants of the ICD ontology; disease codes (ICD-10-CM), procedure codes (ICD-10-PCS), and cancer morphology codes (ICD-O-3). Since every ontology is different, every ontology has its own clustering method that is based on the categories of the ontology.

4.2.1 Ontology - ICD-10-CM

The ICD-10-CM ontology comprises around 68,000 codes that relate to the different diagnoses in the clinical domain. Work on ICD-10-CM began in 1983, became endorsed by the Forty-third World Health Assembly in 1990, and was first used by member states in 1994 [8].

The ICD-10-CM coding system allows users of clinical healthcare data to get more specific than the previous ICD-9 ontology. The features included in the ICD-10-CM coding system allow for expansion and fine-grained differentiation between diagnoses [28].

ICD-10-CM starts with alpha (uses all letters except "U"); 2nd character is always numeric; 3rd–7th characters can be alpha or numeric; decimal is always after 1st three digits [28].

We have decided to use the first three digits prior to the decimal as the relevant part of the code for clustering. This was agreed upon following the chapters of the ICD-10 coding guide [86]. As it is shown in Table 4.2 the clusterization is done by assigning codes to different ranges of the first three digits, which share a semantic relation.

Table 4.2: Clusters defined for ICD-10-CM Ontology.

Label partitioning	**Assigned Cluster**
From a00 to b99	Certain infectious and parasitic diseases
From c00 to d49	Tumors [neoplasms]
From d50 to d89	Diseases of the blood and hematopoietic organs, and certain disorders that affect the mechanism of immunity

Continued on next page

Table 4.2 – continued from previous page

Label partitioning	Assigned Cluster
From e00 to e89	Endocrine, nutritional and metabolic diseases
From f00 to f99	Mental and behavioral disorders
From g00 to g99	nervous system diseases
From h00 to h59	Diseases of the eye and its adnexa
From h60 to h95	Diseases of the ear and mastoid process
From i00 to i99	Circulatory system diseases
From j00 to j99	Diseases of the respiratory system
From k00 to k95	Digestive system diseases
From l00 to l99	Diseases of the skin and subcutaneous tissue
From m00 to m99	Diseases of the musculoskeletal system and connective tissue
From n00 to n99	Diseases of the genitourinary system
From o00 to o9a	Pregnancy, childbirth and puerperium
From p00 to p96	Certain conditions originating in the perinatal period
From q00 to q99	Congenital malformations, deformities and chromosomal abnormalities
From r00 to r99	Abnormal clinical and laboratory symptoms, signs, and findings, not elsewhere classified
From s00 to t88	Injuries, poisoning and some other consequences of external causes
From v00 to y99	External causes of morbidity and mortality
From z00 to z99	Factors influencing health status and contact with health services

4.2.2 Ontology - ICD-10-PCS

ICD-10-PCS is a coding system designed to accommodate the rapidly changing world of procedures better. The code system was developed in the 1990s, but the use of the continually updated codes started almost 20 years later [105]. This coding system was developed to differentiate diagnostics from procedures in the new ICD-10 ontology. While the ICD-9 had procedure codes, ICD-10-CM did not, so the need to create ICD-10-PCS was noticeable.

ICD-10-PCS provides a multi-axial design to the codes, with each code consisting of seven alphanumeric characters. The first character is the section of the code. The second through seventh characters means different things in each section. The ten digits 0-9 and the 24 letters A-H, J-N, and P-Z may be used in each character. The letters O and I are excluded to avoid confusion with the numbers 0, and 1 [9].

To assign clusters, we have decided to use the first character of the code, the section. Although the multi-axial design meant we could have chosen any of the seven characters to clusterize, we decided to use the first one because the section is at the top of the hierarchy.

Table 4.3: Clusters defined for ICD-10-PCS Ontology.

Label partitioning	**Assigned Cluster**
Starting with 0	Medical and Surgical
Starting with 1	Obstetrics
Starting with 2	Placement
Starting with 3	Administration
Starting with 4	Measurement and Monitoring
Starting with 5	Extracorporeal or Systemic Assistance and Performance
Starting with 6	Extracorporeal or Systemic Therapies
Starting with 7	Osteopathic
Starting with 8	Other Procedures
Starting with 9	Chiropractic
Starting with B	Imaging
Starting with C	Nuclear Medicine
Starting with D	Radiation Oncology
Starting with F	Physical Rehabilitation and Diagnostic Audiology
Starting with G	Mental Health
Starting with H	Substance Abuse Treatment
Starting with X	New Technology

4.2.3 Ontology - ICD-O-3

The International Classification of Diseases for Oncology (ICD-O-3) is the third revision of a domain-specific ICD ontology for tumor diseases (ICD-O). This classification is widely used by cancer registries and has two axes: topography which refers to cancer location, and morphology which refers to the cancer histology and behavior. To the best of our knowledge, identifying morphology and topography has not been solved together in Spanish. In fact, in Cantemist the topography codes are not even annotated.

The clusterization for the morphology axis has been defined using a range of the first three digits of the codes, and it is shown in Table 4.4. This clusterization was chosen following the categories provided in [87].

Table 4.4: Clusters defined for ICD-O-3 Ontology.

Label partitioning	**Assigned Cluster**
Label partitioning	Assigned Cluster
From 800 to 800	Neoplasms, Sai
From 801 to 804	Epithelial Neoplasms, Sai
From 805 to 808	Epidermoid Neoplasms
From 809 to 811	Basal Cell Neoplasms

Continued on next page

Table 4.4 – continued from previous page

Label partitioning	Assigned Cluster
From 812 to 813	Papillomas and Transitional Cell Carcinomas
From 814 to 838	Adenomas and Adenocarcinomas
From 839 to 842	Skin Adnexa Neoplasms
From 843 to 843	Mucoepidermoid Neoplasms
From 844 to 849	Cystic, Mucinous and Serous Neoplasms
From 850 to 854	Ductal, Lobular and Medullary Neoplasms
From 855 to 855	Acinar Cell Neoplasms
From 856 to 857	Complex Epithelial Neoplasms
From 858 to 858	Epithelial Neoplasms Thymomas
From 859 to 867	Specialized Gonadal Stromal Neoplasms
From 868 to 871	Paragangliomas and Glomus Tumors
From 872 to 879	Nevi and Melanomas
From 880 to 880	Soft Tissue Tumors and Sarcomas, Sai
From 881 to 883	fibromatous neoplasms
From 884 to 884	Myxomatous Neoplasms
From 885 to 888	Lipomatous Neoplasms
From 889 to 892	Myomatous Neoplasms
From 893 to 899	Complex Mixed and Stromal Neoplasms
From 900 to 903	Fibroepithelial Neoplasms
From 904 to 904	Synovial Neoplasms
From 905 to 905	Mesothelial Neoplasms
From 906 to 909	Germ Cell Neoplasms
From 910 to 910	Trophoblastic neoplasms
From 911 to 911	mesonephromas
From 953 to 953	meningiomas
From 912 to 916	Blood Vessel Tumors
From 917 to 917	Lymphatic Vessel Tumors
From 918 to 924	Bone and Chondromatous Neoplasms
From 925 to 925	Giant Cell Tumors
From 926 to 926	Other Bone Tumors
From 927 to 934	Odontogenic Tumors
From 935 to 937	Other Tumors
From 938 to 948	gliomas
From 949 to 952	Neuroepitheliomatous Neoplasms
From 954 to 957	Nerve Sheath Tumors
From 958 to 958	Granular Cell Tumors and Alveolar Soft Tissue Sarcoma
From 959 to 972	Hodgkin and Non-Hodgkin lymphomas
From 973 to 973	Plasma Cell Tumors
From 974 to 974	Mast Cell Tumors
From 975 to 975	Histiocyte and Accessory Lymphoid Cell Neoplasms
From 976 to 976	Immunoproliferative diseases
From 980 to 994	leukemias
From 995 to 996	Other Myeloproliferative Syndromes
From 997 to 997	Other Hematologic Disorders

Continued on next page

Table 4.4 – continued from previous page

Label partitioning	Assigned Cluster
From 998 to 999	Myelodysplastic syndromes

4.3 Metrics

One way of generalizing effectiveness metrics to the multi-label scenario consists in modeling the problem as a ranking task, i.e., the system returns an ordered label list for each item according to its suitability. Reducing the classification to a ranking problem is especially appropriate in extreme classification scenarios and simplifies the definition of metrics. [12]

Ranking problems are more commonly associated with Information Retrieval tasks, where the output is the relevant documents based on a query but can be extended to multi-label using the following definition. An XMC problem is learning a score function $f : X \times Y \to R$, that maps a (document, label) pair (x, y) to a score $f(x, y)$. The function f should be optimized such that highly relevant (x, y) pairs have high scores, whereas the irrelevant pairs have low scores.

Mean Average Precision is often used as an indicator for evaluating the ranked output of documents in standard retrieval experiments [59] and has shown good discrimination and stability [102]. It has also been used as the primary metric to rank participants in both Cantemist [76] and CodiEsp [77].

The Average Precision (AP) of a document is defined as,

$$\begin{aligned} \text{AP} &= \frac{1}{R}\sum_{i=1}^{R} x_i * P@i \quad , \\ P@n &= \frac{1}{n}\sum_{k=1}^{n} x_k \qquad\quad , \end{aligned} \tag{4.1}$$

where x_j is a variable that for the label ranked at position j is one if the label is in the gold set and zero otherwise, $P@n$ is precision at n labels and R is the total amount of labels. The Mean Average Precision (MAP) is the mean accross all documents,

$$\text{MAP} = \frac{1}{D}\sum_{d=0}^{D} AP_d, \tag{4.2}$$

where D is the total amount of documents. An essential feature of AP compared to $P@n$, as shown in 4.1, is that the gold labels do not have to be a ranked list. It can be a label set because it only weights the relevant labels.

Another way of measuring performance in multi-label classification is using the typical micro-F_1 score. Although it is not specially fitted for these tasks, it also conveys critical information to compare models. We use this as a secondary metric following the example of CodiEsp and Cantemist.

Precision is the first component of the F_1 score; it can be interpreted as counting the correct percentage from everything that has been predicted as positive. A precise model may not find all the positives, but the ones that the model classifies as positive are very likely to be correct. An unprecise model may find most of the positives, but its selection method is noisy, detecting many positives that are actually negatives. It is defined as,

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{TP} = \text{True Positives}, \quad \text{FP} = \text{False Positives} \quad . \tag{4.3}$$

Recall is the second component of the F_1 Score; it can be interpreted as how many predicted positive examples from everything that is actually positive the model managed to obtain. A model with high recall succeeds well in finding all the positive cases in the data, even though they may also wrongly identify some negative cases as positive cases. A model with a low recall cannot find most of the positive cases in the data. It is defined as,

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{TP} = \text{True Positives}, \quad \text{FN} = \text{False Negatives} \quad . \tag{4.4}$$

F_1 score can be defined as a harmonic mean of precision and recall. A model will obtain a high F_1 score if both precision and recall are high, a low F_1 score if both precision and recall are low, and a medium F_1 score if one of precision and recall is low and the other is high. It is defined as,

$$F_1 = 2 * \frac{P * R}{P + R} \quad . \tag{4.5}$$

Another widely used metric in multi-label scenarios is the Hamming Score, which can be related to accuracy taking into account partially correct predictions. The Hamming Score is defined as one minus the Hamming Loss. The Hamming Loss can be calculated by summing the number of labels incorrectly predicted (false negatives and false positives) for each document and then dividing it by the sum of the number of gold labels in each document.

Although this metric has high interpretability (being that it can be interpreted as the accuracy in a multi-label scenario), it was not calculated. This decision was taken because the competitions and the prior relevant work did not use the metric, so we had no comparison with the other models.

4.4 Evaluation

To compare the predictions against the actual data, we used the metrics described in Section 4.3. These metrics were evaluated on the test set provided by the shared tasks, so comparability to other models is assured. To avoid data leakage, the test set was completely separated from the training process; it is not used to train the models, define the encoding of the documents, or even in data augmentation techniques such as the description of the labels.

However, the process of training a transformer and boosting trees is not deterministic, so reporting a performance score of one training round is an unfair comparison with the other participants. They had to predict one single run but were blind to the test set, so they could not choose the runs where they had better outcomes.

To correctly determine whether the differences between the performance of our model and the other models are reliable or just due to statistical chance, we have done five different training rounds, each with a different seed, ensuring different results. The results reported are the mean of these five training rounds, and the standard deviation is also reported for completeness.

Regarding the performance of the ensemble models, the report of different training rounds is unfeasible due to the fact that we would have to train an immense amount of models. However, the statistical chance is interiorized in the ensemble because it uses 15 different instances of the architecture.

Notably, the FALP corpus has not been divided by the creators yet, so the training, testing and validation subsets were randomly defined. We used 80% of the documents for training, 10% for validation, and 10% for testing.

4.5 Results

	CodiEsp-D		CodiEsp-P		Cantemist		Falp	
Model	MAP	F_1	MAP	F_1	MAP	F_1	MAP	F_1
IXA-AAA [23]	0.593	0.009	0.425	0.008	-	-	-	-
IAM [36]	0.521	0.687	0.493	0.522	-	-	-	-
FLE [46]	0.519	0.679	0.443	0.514	-	-	-	-
The Mental Strokers [37]	0.517	0.591	0.445	0.488	-	-	-	-
Vicomtech [45]	-	-	-	-	0.847	**0.855**	-	-
ICB-UMA [70]	0.482	0.009	-	-	0.847	0.013	-	-
FlatNER + Search Engine [110]	-	-	-	-	-	-	0.416	-
Clinical Coding Transformers - Best [69]	0.616	-	0.514	-	0.862	-	-	-
Clinical Coding Transformers - Ensemble [69]	0.662	-	0.544	-	**0.884**	-	-	-
Divide and Conquer (DaC)	**0.665**	**0.746**	**0.545**	**0.553**	0.788	0.712	0.957	0.919
Divide and Conquer - Ensemble (DaC-E)	**0.682**	**0.744**	**0.562**	**0.560**	0.804	0.695	**0.964**	**0.907**

Table 4.5: Overall results on three clinical coding datasets. Results of Clinical Transformers are taken from the author's paper. FlatNER + Search engine results were calculated by us using predictions provided to us by the authors. All the other results are from the competitions overview. Some results are missing because those approaches were not implemented for the corresponding tasks.

Table 4.5 shows the overall results of our model. We reported two different results for the DAC architecture: the average of five distinct training rounds using the original approach and another result from a version that ensembled 15 different model instances. We did not do a mean of five executions for the latter because that would involve training 75 instances, and the variation is interiorized in the 15 different instances.

Interestingly, our base model achieves state-of-the-art results in both CodiEsp tasks, surpassing the best base model (Clinical Coding Transformers - Best [69]) by 8% in CodiEsp diagnostics and by 6% in CodiEsp Procedures.

Even in comparison with an ensemble of strong learners, which obtains a similar performance (Clinical Coding Transformers - Ensemble [69]), our base model surpasses their results by a small margin of 0.5% in CodiEsp Diagnostics and 0.2% in CodiEsp Procedures. Their results correspond to 15 distinct runs of 3 different strong learners, where each language model was trained with a private oncology corpus. Unlike the mentioned work, we used only publicly available resources and a simpler architecture in terms of computational cost.

Most notably, our ensemble-based version of 15 different instances of our model outperformed previous results in the CodiEsp tasks by a wide margin, outperforming state-of-the-art methods, including ensembles of strong learners, on CodiEsp-D and CodiEsp-P by 3.0% and 3.3%, respectively.

In the FALP corpus, our base model has surpassed the previous method by an astonishing 131% according to the MAP metric. However, this corpus has not been tested using multiple different approaches, more models will be tested in the FALP documents in the forthcoming future, and this work can be considered a benchmark for those.

Although we did not obtain state-of-the-art in CANTEMIST, our models still achieve competitive performance. In fact, in the original shared task [76], we would have finished in third place, only surpassed by ICB-UMA [70] and Vimcotech [45], the two tied winners of the competition.

Regarding the comparison between our base model and the ensemble, we can observe in Table 4.5 that the ensemble-based version is better than the base version according to the MAP metric but not necessarily for the F_1 metric. This finding opens space for a better choice of voting mechanism for what classes to predict when using an ensemble of models.

We hypothesize that the high performance of our model is because the original text classification task is reduced to two subtasks, where the number of possible labels is smaller. First, the Matcher module performs a text classification in which the number of labels is equal to the number of clusters. Secondly, the Ranker is trained only with documents belonging to a cluster, which allows for a fine-grained differentiation between similar codes.

The differences between the datasets' results, not only in our model but in all the others as well, can be explained by taking into account two factors: the number of different codes (Cardinality) and the number of different codes in the dataset (N_c). These 2 factors can be comprised into the label density of the corpora, which greatly explains the overall results that every model can have in each dataset.

As can be seen in Table 4.1, when comparing the results between Codiesp-D, FALP, and Cantemist, the fewer codes there are in the entire dataset, the easier it is to do the classification. This conclusion is straightforward, taking into account that fewer codes make the task less extreme and thus easier to resolve.

The other factor that can explain the differences between results (the cardinality) becomes apparent by comparing the results obtained in Cantemist and Codiesp-P. While both have a similar number of different codes, Codiesp-P has fewer examples to learn from because it has roughly one-third as many codes per document as Cantemist.

These two metrics can be combined into a single metric: label density. It can be noted by looking at the label densities of the different corpora in Table 4.1 and the results in Table 4.5 that label densities explain the degree of difficulty associated with the task at hand. This conclusion is shared for the clinical coding task in [22].

4.6 Module Analysis

To provide a more comprehensive analysis of the architecture, we have computed metrics for each one of the modules. These metrics help us gain insights into which part of the architecture levels are acceptable and allow us to know when high scores for the architecture as a whole can be expected.

Regarding the Matcher module, we report the MAP and the F_1 score when the gold labels are the clusters. In the case of the Ranker module, we had to approach the issue of creating metrics that could evaluate its performance independently from the Matcher step, which is not straightforward. To overcome this issue, we have defined a weighted metric in which, for each cluster, we calculate the metric for that cluster's sentences. Then the clusters metrics are aggregated and weighted by the number of sentences in each cluster. This metric indicates how well the Ranker is labeling the documents. However, by itself, it does not give information that can relate to a practical task because it assumes that the previous step is correctly solved. It can be interpreted as what the metric would be if the Matcher had a perfect performance and thus acts as a ceiling for the DaC model's final performance.

In Tables 4.6, 4.7, 4.8, and 4.9 we report all the experiments made with the architecture for each corpus. We have performed experiments for 15 instances of the architecture with different seeds, five using BioClinical RoBERTa, five using BioMedical RoBERTa, and five using BETO. We also report the mean of the executions for each type of transformer and also the ensemble results of the architecture.

We can see in Tables 4.6, 4.7, 4.8, and 4.9 that the MAP and F_1 scores for the Matcher are high in all the experiments. This is required for the architecture to be competitive; otherwise, the error propagation leads to a low-quality final model. Most notably, in the FALP corpus, the scores are almost perfect, meaning that the chosen clusterization was correct and created clusters whose identification is relatively simple.

It is trivial that the Ranker metrics are not correlated to the transformer used because the Ranker does not use a transformers model. Though what is interesting is that the Ranker scores are much more consistent than the Matchers. This can be noted even within the same transformers models like in Table 4.6 where for BioClinical RoBERTa, we have the best and worst Matcher, and a 5-point gap separates them. This is probably due to the fact that the initial weights given by the seed may interfere with the correct learning of the transformer, leading to some instances that fail to achieve competitive performances. Even

	Matcher		Ranker		DaC	
Model	MAP	F_1	MAP	F_1	MAP	F_1
BioClinical RoBERTa - 1	0.943	**0.880**	0.727	0.726	**0.670**	0.737
BioClinical RoBERTa - 2	0.889	0.783	0.729	0.725	0.654	0.697
BioClinical RoBERTa - 3	0.926	0.842	0.730	0.725	0.666	0.725
BioClinical RoBERTa - 4	0.945	0.878	0.731	0.729	0.668	**0.739**
BioClinical RoBERTa - 5	**0.948**	0.876	0.729	0.726	0.667	0.736
BioMedical RoBERTa - 1	0.936	0.866	0.729	0.724	0.663	0.730
BioMedical RoBERTa - 2	0.943	0.877	0.726	0.725	0.663	0.736
BioMedical RoBERTa - 3	0.944	0.871	0.731	0.726	0.667	0.734
BioMedical RoBERTa - 4	0.942	0.868	0.731	0.726	0.665	0.731
BioMedical RoBERTa - 5	0.926	0.842	**0.732**	0.727	0.665	0.713
BETO - 1	0.915	0.819	0.729	0.731	0.657	0.712
BETO - 2	0.924	0.840	0.727	**0.733**	0.650	0.724
BETO - 3	0.920	0.828	0.729	0.723	0.654	0.706
BETO - 4	0.921	0.837	0.727	0.731	0.652	0.719
BETO - 5	0.900	0.798	0.729	0.722	0.653	0.702
BioClinical RoBERTa - Mean	0.930	0.852	0.729	**0.726**	**0.665**	0.727
BioMedical RoBERTa - Mean	**0.938**	**0.865**	**0.730**	**0.726**	**0.665**	**0.729**
BETO - Mean	0.916	0.824	0.728	0.728	0.653	0.713
Ensemble	-	-	-	-	**0.682**	**0.744**

Table 4.6: Report of metrics for each module and model trained in CodiEsp Diagnostics. The F_1 scores of both the DaC model and the Ranker use only the first three characters of the code as the label, following the procedures of how to evaluate the models created by the competition. The bolded results indicate the best metric score for each module and the underline mark the worst performance.

with this "failed" Matcher, the mean of BioClinical RoBERTa managed to obtain state-of-the-art results for the CodiEsp Diagnostics task.

Another interesting finding is that, we can see no significant difference between the domain-specific language models (RoBERTa BioMedical and BioClinical) in the tasks we performed experiments on. However, the general-domain language model we have tested (BETO) has significantly lower performance on all the tasks except in the FALP corpus. This can be explained because the FALP task was more straightforward according to our model results.

Finally, it is important to note that the ensemble-based architecture significantly outperforms all the base models at hand, at least in the MAP metric. In the F_1 metric, it surpasses the models in the CodiEsp tasks and fails in the Cantemist and FALP corpora. This adds room for improvement in how the class prediction is combined to calculate the F_1 metric.

	Matcher		Ranker		DaC	
Model	MAP	F_1	MAP	F_1	MAP	F_1
BioClinical RoBERTa - 0	0.939	**0.883**	0.614	0.588	0.543	0.537
BioClinical RoBERTa - 1	0.944	0.881	0.610	0.577	0.549	0.528
BioClinical RoBERTa - 2	0.941	0.873	0.612	0.582	0.542	0.533
BioClinical RoBERTa - 3	0.940	0.881	0.614	0.580	0.544	0.533
BioClinical RoBERTa - 4	0.941	0.877	**0.621**	0.592	0.545	**0.547**
BioMedical RoBERTa - 5	0.948	0.857	0.612	0.586	**0.549**	0.534
BioMedical RoBERTa - 6	**0.956**	0.869	0.620	**0.603**	0.546	0.549
BioMedical RoBERTa - 7	0.938	0.867	0.617	0.581	0.542	0.512
BioMedical RoBERTa - 8	0.940	0.868	0.615	0.587	0.546	0.531
BioMedical RoBERTa - 9	0.955	0.873	0.620	0.578	0.549	0.529
BETO - 10	0.944	0.860	0.616	0.582	0.534	0.517
BETO - 11	0.934	0.851	0.609	0.586	0.526	0.526
BETO - 12	0.933	0.853	0.612	0.588	0.538	0.534
BETO - 13	0.938	0.856	0.615	0.586	0.540	0.521
BETO - 14	0.931	0.846	0.611	0.592	0.528	0.527
BioClinical RoBERTa - Mean	0.941	**0.879**	0.614	0.584	0.545	**0.536**
BioMedical RoBERTa - Mean	**0.947**	0.867	**0.617**	**0.587**	**0.546**	0.531
BETO - Mean	0.936	0.853	0.612	**0.587**	0.533	0.525
Ensemble	-	-	-	-	**0.562**	**0.560**

Table 4.7: Report of metrics for every module and model trained in CodiEsp Procedures. The F_1 scores of both the DaC model and the Ranker use only the first four characters of the code as the label, following the procedures of how to evaluate the models created by the competition. The bolded results point to the best metric score for each module, and the underline marks the worst performance.

	Matcher		Ranker		DaC	
Model	MAP	F_1	MAP	F_1	MAP	F_1
BioClinical RoBERTa - 0	**0.956**	0.899	0.823	0.710	**0.791**	0.705
BioClinical RoBERTa - 1	0.955	0.903	0.817	0.714	0.786	**0.711**
BioClinical RoBERTa - 2	0.950	0.898	0.821	0.711	0.789	0.706
BioClinical RoBERTa - 3	0.951	0.899	0.823	0.711	0.787	0.703
BioClinical RoBERTa - 4	0.953	0.898	0.820	0.710	0.787	0.705
BioMedical RoBERTa - 5	0.943	0.900	0.827	0.715	0.784	0.710
BioMedical RoBERTa - 6	0.948	0.896	0.816	0.716	0.780	0.706
BioMedical RoBERTa - 7	0.950	0.895	0.817	0.712	0.785	0.708
BioMedical RoBERTa - 8	0.944	0.894	0.820	0.713	0.784	0.709
BioMedical RoBERTa - 9	**0.956**	**0.905**	0.817	0.709	0.788	0.708
BETO - 10	0.915	0.858	**0.830**	**0.716**	0.767	0.694
BETO - 11	0.922	0.863	<u>0.815</u>	<u>0.709</u>	0.763	0.696
BETO - 12	0.916	0.858	0.814	0.707	<u>0.761</u>	0.687
BETO - 13	0.914	0.856	0.824	0.711	0.765	<u>0.686</u>
BETO - 14	<u>0.909</u>	<u>0.851</u>	0.825	0.716	0.760	0.695
BioClinical RoBERTa - Mean	**0.953**	**0.900**	0.821	<u>0.711</u>	**0.788**	0.706
BioMedical RoBERTa - Mean	0.948	0.898	<u>0.819</u>	**0.713**	0.784	**0.708**
BETO - Mean	<u>0.915</u>	<u>0.857</u>	**0.822**	0.712	<u>0.763</u>	<u>0.692</u>
Ensemble	-	-	-	-	**0.804**	<u>0.695</u>

Table 4.8: Report of metrics for every module and model trained in Cantemist. The bolded results point to the best metric score for each module, and the underline marks the worst performance.

	Matcher		Ranker		DaC	
Model	MAP	F_1	MAP	F_1	MAP	F_1
BioClinical RoBERTa - 0	0.994	0.975	0.975	**0.940**	0.958	0.925
BioClinical RoBERTa - 1	0.991	0.979	0.976	0.935	0.957	0.921
BioClinical RoBERTa - 2	0.992	0.976	**0.978**	0.930	0.960	0.916
BioClinical RoBERTa - 3	0.990	0.973	0.975	0.930	0.956	0.915
BioClinical RoBERTa - 4	0.995	0.979	0.977	0.932	0.956	0.916
BioMedical RoBERTa - 5	0.992	**0.987**	0.970	0.937	0.955	**0.928**
BioMedical RoBERTa - 6	0.986	0.976	0.973	0.939	0.955	0.924
BioMedical RoBERTa - 7	0.988	0.981	0.974	0.935	0.956	0.923
BioMedical RoBERTa - 8	0.992	0.978	0.974	0.931	0.959	0.913
BioMedical RoBERTa - 9	0.990	0.983	0.973	0.934	0.953	0.923
BETO - 10	**0.996**	0.976	0.976	0.938	0.963	0.922
BETO - 11	0.990	0.982	0.969	0.933	0.950	0.922
BETO - 12	0.993	0.970	0.977	0.934	**0.962**	0.917
BETO - 13	0.994	0.974	0.972	0.934	0.957	0.916
BETO - 14	0.995	0.975	0.973	0.935	0.960	0.922
BioClinical RoBERTa - Mean	0.993	0.977	**0.976**	0.933	0.957	0.919
BioMedical RoBERTa - Mean	0.990	**0.981**	0.973	**0.935**	0.955	**0.922**
BETO - Mean	**0.994**	0.975	0.973	**0.935**	**0.958**	0.920
Ensemble	-	-	-	-	**0.964**	0.907

Table 4.9: Report of metrics for every module and model trained in FALP. The bolded results point to the best metric score for each module, and the underline marks the worst performance.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This work proposes a novel model for clinical coding in Spanish, outperforming previous results in three datasets; CodiEsp-D, CodiEsp-P, and FALP. Also, we obtained competitive results on the Cantemist task. Our method uses a Divide and Conquer approach that creates semantic groups of codes to build an architecture composed of two specialized modules: the Matcher and the Ranker.

The task of the clinical coding problem is separated into two simpler problems solved by each one of the modules. First, the Matcher predicts the clusters of each document, and then the Ranker predicts the codes of each document given a cluster. This division allows us to use state-of-the-art transformers to solve the easier task of cluster prediction and permits a fine-grained differentiation between similar codes in a cluster using XGBoost.

Interestingly, our base model obtains better results for these corpora than previous work, including ensembles of strong learners. However, for a fair comparison, we have included the results of an ensemble version of our model, demonstrating that this technique improves the performance of the models even more. Our model can be applied to any ontology having a hierarchy or partition of semantically related labels, as in the case of clinical ontologies.

Finally, the DaC architecture has been packaged into a library for ease of use, reproducibility, and to be able to extend the architecture for other extreme multi-label corpora in a straightforward manner.

5.2 Future Work

Future directions include implementing and testing the Divide and Conquer model on other multi-label text classification corpora. First, we expect to test the DaC architecture on clinical corpora in other languages, including languages with more resources like English. Second, we expect to test the architecture on other extreme multi-label classification corpora.

This poses a challenge since the number of labels we have processed thus far, though being very vast, falls into the category of small extreme multi-label classification datasets [20]. We expect to encounter issues with the training time required for processing other large corpora, forcing us to modify the library to optimize the speed.

In terms of improving the performance using this architecture, we identify opportunities to optimize the number of layers that we left fine-tuneable in the Matcher module given that we have seen research that shows that finetuning more layers provides better results [61]. Also, for the Ranker, we know that XGBoost can be trained with a ranking objective function, thus providing an alternative to the one-vs-rest approach. Implementing the Ranker using this approach would be faster to train (only one model per cluster would be trained) and may provide similar or better results.

Finally, the DaC architecture is a black box when defining which labels to assign for each document. Recently, explainability features of the different architectures are gaining more relevance. It is paramount that the model's predictions are understood to help the user make appropriate choices [42]. We expect to develop explainability to the labels predicted by providing textual queues of what features the model used to choose each label. The textual queues that support label assignment can be provided by the Ranker leveraging the explainability features of tree ensembles [90], and the textual queues that support the cluster choice can be obtained using the attention weights of the transformer model [67].

5.3 Contributions

This work has contributed by making a library that can be further extended and tested on other corpora. Also, our work is currently in the process of being published in the form of a short paper.

- Divide and Conquer: Extreme Multi-Label Classification for Clinical Coding in Spanish (In the process of being published).

Bibliography

[1] AdamW – PyTorch 1.11.0 documentation. `https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html`. [Online; accessed 2022-06-03].

[2] Artificial Neural Network. `https://samansiadati.blogspot.com/2019/03/artificial-neural-network.html`. [Online; accessed 2022-07-15].

[3] BCEWithLogitsLoss – PyTorch 1.11.0 documentation. `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html#torch.nn.BCEWithLogitsLoss`. [Online; accessed 2022-06-08].

[4] CART: Classification and Regression Trees for Clean but Powerful Models. https://towardsdatascience.com/cart-classification-and-regression-trees-for-clean-but-powerful-models-cc89e60b7a85. [Online; accessed 2022-07-07].

[5] Machine Learning Hierarchy. `https://gearsngenes.com/wp-content/uploads/2020/12/Machine-Learning-Hierarchy-2-768x768.png`. [Online; accessed 2022-07-15].

[6] ReduceLROnPlateau – PyTorch 1.11.0 documentation. `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`. [Online; accessed 2022-06-03].

[7] What is a Decision Tree - IBM. `https://www.ibm.com/topics/decision-trees`. [Online; accessed 2022-07-15].

[8] WHO International Classification of Diseases (ICD) Information Sheet. `https://web.archive.org/web/20121104064222/http://www.who.int/classifications/icd/factsheet/en/`. [Online; accessed 2022-06-29].

[9] Aalseth, P. T. *CodeBusters: A Quick Guide to Coding and Billing Compliance for Medical Practices.* Jones & Bartlett Learning, 1998.

[10] Akbik, A., Bergmann, T., Blythe, D., Rasul, and Vollgraf, R. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)* (2019), pp. 54–59.

[11] Alpaydin, E. *Introduction to machine learning.* MIT press, 2020.

[12] Amigó, E., and Delgado, A. Evaluating Extreme Hierarchical Multi-label Classification. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2022), pp. 5809–5819.

[13] Amin, S., Neumann, G., Dunfield, Chapman, K. A., and Wixted, M. K. MLT-DFKI at CLEF eHealth 2019: Multi-label Classification of ICD-10 Codes with BERT. In *CLEF (Working Notes)* (2019), pp. 1–15.

[14] Aronson, A. R., and Lang, F.-M. An overview of MetaMap: historical perspective and recent advances. *Journal of the American Medical Informatics Association 17*, 3 (2010), 229–236.

[15] Báez, P., Bravo-Marquez, F., Dunstan, J., Rojas, M., and Villena, F. Automatic Extraction of Nested Entities in Clinical Referrals in Spanish. *ACM Transactions on Computing for Healthcare (HEALTH) 3*, 3 (2022), 1–22.

[16] Balasubramanian, K., and Lebanon, G. The Landmark Selection Method for Multiple Output Prediction. In *ICML* (2012).

[17] Bayer, M., Kaufhold, M.-A., and Reuter, C. A Survey on Data Augmentation for Text Classification. *ACM Computing Surveys* (jun 2022).

[18] Beltagy. SciBERT: A Pretrained Language Model for Scientific Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (Hong Kong, China, nov 2019), Association for Computational Linguistics, pp. 3615–3620.

[19] Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The Long-Document Transformer. *ArXiv abs/2004.05150* (2020).

[20] Bhatia, K., Dahiya, K., Jain, H., Kar, P., Mittal, A., Prabhu, Y., and Varma, M. The extreme classification repository: Multi-label datasets and code, 2016.

[21] Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. Sparse local embeddings for extreme multi-label classification. *Advances in neural information processing systems 28* (2015).

[22] Blanco, A., Casillas, A., Pérez, A., and Diaz de Ilarraza, A. Multi-label clinical document classification: Impact of label-density. *Expert Systems with Applications 138* (2019), 112835.

[23] Blanco, A., Pérez, A., and Casillas, A. IXA-AAA at CLEF eHealth 2020 CodiEsp. Automatic Classification of Medical Records with Multi-label Classifiers and Similarity Match Coders. In *CLEF (Working Notes)* (2020).

[24] Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. Learning multi-label scene classification. *Pattern recognition 37*, 9 (2004), 1757–1771.

[25] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[26] Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., and Pérez, J. Spanish Pre-Trained BERT Model and Evaluation Data. In *PML4DC at ICLR 2020* (2020).

[27] Carrino, C. P., Armengol-Estapé, J., no, A. G.-F., Llop-Palao, J., Pàmies, M., Gonzalez-Agirre, A., and Villegas, M. Biomedical and Clinical Language Models for Spanish: On the Benefits of Domain-Specific Pretraining in a Mid-Resource Scenario. *ArXiv* (2021).

[28] Cartwright, D. J. ICD-9-CM to ICD-10-CM codes: what? why? how?, 2013.

[29] Chang, W.-C., Yu, H.-F., Zhong, K., Yang, Y., and Dhillon, I. S. Taming pretrained transformers for extreme multi-label text classification. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining* (2020), pp. 3163–3171.

[30] Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., et al. Xgboost: extreme gradient boosting. *R package version 0.4-2 1*, 4 (2015), 1–4.

[31] Cho. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Doha, Qatar, oct 2014), Association for Computational Linguistics, pp. 1724–1734.

[32] Choromanska, A. E., and Langford, J. Logarithmic time online multiclass prediction. *Advances in neural information processing systems 28* (2015).

[33] Chowdhary, K. Natural language processing. *Fundamentals of artificial intelligence* (2020), 603–649.

[34] Cohen, K. B., Xia, J., Zweigenbaum, P., Callahan, T. J., Hargraves, O., Goss, F., Ide, N., Névéol, A., Grouin, C., and Hunter, L. E. Three dimensions of reproducibility in natural language processing. In *LREC... International Conference on Language Resources & Evaluation:[proceedings]. International Conference on Language Resources and Evaluation* (2018), vol. 2018, NIH Public Access, p. 156.

[35] Collobert, R., Kavukcuoglu, K., and Farabet, C. Torch7: A matlab-like environment for machine learning.

[36] Cossin, S., and Jouhet, V. IAM at CLEF eHealth 2020: Concept Annotation in Spanish Electronic Health Records. In *CLEF (Working Notes)* (2020).

[37] Costa, J., Lopes, I., Carreiro, A. V., Ribeiro, D., and Soares, C. Fraunhofer AICOS at CLEF eHealth 2020 Task 1: Clinical Code Extraction From Textual Data Using Fine-Tuned BERT Models. In *CLEF (Working Notes)* (2020).

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

[39] Cunningham, H., Tablan, V., Roberts, A., and Bontcheva, K. Getting more out of biomedical documents with GATE's full lifecycle open source text analytics. *PLoS computational biology 9*, 2 (2013), e1002854.

[40] Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)* (2019), J. Burstein, C. Doran, and T. Solorio, Eds., Association for Computational Linguistics, pp. 4171–4186.

[41] Dong, X., Yu, Z., Cao, W., Shi, Y., and Ma, Q. A survey on ensemble learning. *Frontiers of Computer Science 14*, 2 (2020), 241–258.

[42] Duque, A., Fabregat, H., and Araujo. A keyphrase-based approach for interpretable ICD-10 code classification of spanish medical reports. *Artificial Intelligence in Medicine 121* (2021), 102177.

[43] Elisseeff, A., and Weston, J. A kernel method for multi-labelled classification. *Advances in neural information processing systems 14* (2001).

[44] Frigui, H., and Krishnapuram, R. A Robust Competitive Clustering Algorithm With Applications in Computer Vision. *IEEE transactions on pattern analysis and machine intelligence 21*, 5 (1999), 450–465.

[45] García-Pablos, A., Perez, N., and Cuadros, M. Vicomtech at Cantemist 2020. In *Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2020), CEUR Workshop Proceedings* (2020).

[46] García-Santa, N., Cetina, K., Cappellato, L., Eickhoff, C., Ferro, N., and Névéol, A. FLE at CLEF eHealth 2020: Text Mining and Semantic Knowledge for Automated Clinical Encoding. In *CLEF (Working Notes)* (2020).

[47] Goldstein, I., Arzumtsyan, A., and Uzuner, Ö. Three approaches to automatic assignment of ICD-9-CM codes to radiology reports. In *AMIA Annual Symposium Proceedings* (2007), vol. 2007, American Medical Informatics Association, p. 279.

[48] Gruber, T. R. A translation approach to portable ontology specifications. *Knowledge acquisition 5*, 2 (1993), 199–220.

[49] Gu, Y., Tinn, R., Cheng, H., Lucas, M., Usuyama, N., Liu, X., Naumann, T., Gao, J., and Poon, H. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH) 3*, 1 (2021), 1–23.

[50] Hirschberg, J., and Manning, C. D. Advances in natural language processing. *Science 349*, 6245 (2015), 261–266.

[51] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[52] Ivanović, M., and Budimac, Z. An overview of ontologies and data resources in medical domains. *Expert Systems with Applications 41*, 11 (2014), 5158–5166.

[53] Joachims, T. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning* (1998), Springer, pp. 137–142.

[54] Johnson, A. E., Pollard, T. J., Shen, L., Lehman, Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., and Mark, R. G. MIMIC-III, a freely accessible critical care database. *Scientific data 3*, 1 (2016), 1–9.

[55] Karimi. Automatic Diagnosis Coding of Radiology Reports: A Comparison of Deep Learning and Conventional Classification Methods. In *BioNLP 2017* (Vancouver, Canada,, aug 2017), Association for Computational Linguistics, pp. 328–332.

[56] Kaur, R., Ginige, J. A., and Obst, O. A Systematic Literature Review of Automated ICD Coding and Classification Systems using Discharge Summaries. *ArXiv abs/2107.10652* (2021).

[57] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems 30* (2017).

[58] Kingma, D. P., and Ba, J. Adam: A Method for Stochastic Optimization. *CoRR abs/1412.6980* (2015).

[59] Kishida, K. *Property of average precision and its generalization: An examination of evaluation indicator for information retrieval experiments.* National Institute of Informatics Tokyo, Japan, 2005.

[60] Larkey, L. S., and Croft, W. B. Automatic assignment of ICD-9 codes to discharge summaries. Tech. rep., Technical report, University of Massachusetts at Amherst, Amherst, MA, 1995.

[61] Lee, J., Tang, R., and Lin, J. J. What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning. *ArXiv abs/1911.03090* (2019).

[62] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., and Kang, J. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics 36*, 4 (2020), 1234–1240.

[63] Leung, Y., Zhang, J.-S., and Xu, Z.-B. Clustering by scale-space filtering. *IEEE Transactions on pattern analysis and machine intelligence 22*, 12 (2000), 1396–1410.

[64] Li, Q., Peng, H., Li, J., Xia, C., Yang, Yu, P. S., and He, L. A Survey on Text Classification: From Traditional to Deep Learning. *ACM Transactions on Intelligent Systems and Technology (TIST) 13*, 2 (2022), 1–41.

[65] Liu, J., Chang, W.-C., Wu, Y., and Yang, Y. Deep Learning for Extreme Multi-Label Text Classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2017), SIGIR '17, Association for Computing Machinery, p. 115–124.

[66] Liu, P., Wang, X., Xiang, C., and Meng, W. A survey of text data augmentation. In *2020 International Conference on Computer Communication and Network Security (CCNS)* (2020), IEEE, pp. 191–195.

[67] Liu, S., Le, F., Chakraborty, S., and Abdelzaher, T. On Exploring Attention-based Explanation for Transformer Models in Text Classification. In *2021 IEEE International Conference on Big Data (Big Data)* (2021), IEEE, pp. 1193–1203.

[68] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv abs/1907.11692* (2019).

[69] López-García, G., Jerez, J. M., Ribelles, N., Alba, E., and Veredas, F. J. Transformers for clinical coding in Spanish. *IEEE Access 9* (2021), 72387–72397.

[70] López-García, G., Jerez, J. M., and Veredas, F. J. ICB-UMA at CANTEMIST 2020: Automatic ICD-O Coding in Spanish with BERT. In *IberLEF SEPLN* (2020), pp. 468–476.

[71] López-Garcia, G., Jerez, J. M., Veredas, F. J., Cappellato, L., Eickhoff, C., Ferro, N., and Névéol, A. ICB-UMA at CLEF e-health 2020 task 1: Automatic ICD-10 coding in Spanish with BERT. In *Proc. Work. Notes CLEF, Conf. Labs Eval. Forum, CEUR Workshop* (2020), pp. 1–15.

[72] Loshchilov, I., and Hutter, F. Decoupled Weight Decay Regularization. In *ICLR* (2019).

[73] Madhulatha, T. An Overview on Clustering Methods. *IOSR Journal of Engineering 2* (05 2012).

[74] Maron, M. E. Automatic indexing: an experimental inquiry. *Journal of the ACM (JACM) 8*, 3 (1961), 404–417.

[75] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. Efficient Estimation of Word Representations in Vector Space. In *ICLR* (2013).

[76] Miranda-Escalada, A., Farré, E., and Krallinger, M. Named entity recognition, concept normalization and clinical coding: Overview of the Cantemist track for cancer text mining in Spanish, corpus, guidelines, methods and results. In *Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2020), CEUR Workshop Proceedings* (2020).

[77] Miranda-Escalada, A., Gonzalez-Agirre, A., Armengol-Estapé, J., and Krallinger, M. Overview of Automatic Clinical Coding: Annotations, Guidelines, and Solutions for non-English Clinical Cases at CodiEsp Track of CLEF eHealth 2020. In *CLEF (Working Notes)* (2020).

[78] Moons, E., Khanna, A., Akkasi, A., and Moens, M.-F. A comparison of deep learning methods for ICD coding of clinical records. *Applied Sciences 10*, 15 (2020), 5262.

[79] Mullenbach. Explainable Prediction of Medical Codes from Clinical Text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (New Orleans, Louisiana, jun 2018), Association for Computational Linguistics, pp. 1101–1111.

[80] Nam, J., Kim, J., Mencí a, E. L., Gurevych, I., and Fürnkranz, J. Large-Scale Multi-label Text Classification — Revisiting Neural Networks. In *Machine Learning and Knowledge Discovery in Databases*. Springer Berlin Heidelberg, 2014, pp. 437–452.

[81] Naseem, U., Khushi, M., Reddy, V., Rajendran, S., Razzak, I., and Kim, J. Bioalbert: A simple and effective pre-trained language model for biomedical named entity recognition. In *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), IEEE, pp. 1–7.

[82] Naseem, U., Razzak, I., Khan, S. K., and Prasad, M. A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models. *Transactions on Asian and Low-Resource Language Information Processing 20*, 5 (2021), 1–35.

[83] Névéol, A., Dalianis, H., Velupillai, S., Savova, G., and Zweigenbaum, P. Clinical natural language processing in languages other than english: opportunities and challenges. *Journal of biomedical semantics 9*, 1 (2018), 1–13.

[84] Nilsson, N. J. Introduction to machine learning. An early draft of a proposed textbook, 1996.

[85] of Chile, A. U. Patagón Supercomputer, 2021.

[86] Organization, W. H., et al. *The ICD-10 classification of mental and behavioural disorders: clinical descriptions and diagnostic guidelines*. World Health Organization, 1992.

[87] Organization, W. H., et al. *International classification of diseases for oncology (ICD-O)–3rd edition, 1st revision*. World Health Organization, 2013.

[88] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research 12* (2011), 2825–2830.

[89] Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (2014), pp. 1532–1543.

[90] Petkovic, D., Altman, R., Wong, M., and Vigil, A. Improving the explainability of Random Forest classifier–user centered approach. In *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2018: Proceedings of the Pacific Symposium* (2018), World Scientific, pp. 204–215.

[91] Petrovskiy, M. Paired comparisons method for solving multi-label learning problem. In *2006 Sixth International Conference on Hybrid Intelligent Systems (HIS'06)* (2006), IEEE, pp. 42–42.

[92] Pollard, T. J., Johnson, A. E., Raffa, J. D., Celi, L. A., Mark, R. G., and Badawi, O. The eICU Collaborative Research Database, a freely available multi-center database for critical care research. *Scientific data 5*, 1 (2018), 1–13.

[93] Prabhu, Y., Kag, A., Harsola, S., Agrawal, R., and Varma, M. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference* (2018), pp. 993–1002.

[94] Prabhu, Y., and Varma, M. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), pp. 263–272.

[95] Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., and Huang, X. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences 63*, 10 (2020), 1872–1897.

[96] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training.

[97] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog 1*, 8 (2019), 9.

[98] Rokach, L., and Maimon, O. Clustering methods. In *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[99] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature 323*, 6088 (1986), 533–536.

[100] Russell, S. J. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[101] Salton, G., and Buckley, C. Term-weighting approaches in automatic text retrieval. *Information processing & management 24*, 5 (1988), 513–523.

[102] Schütze, H., Manning, C. D., and Raghavan, P. *Introduction to information retrieval*, vol. 39. Cambridge University Press Cambridge, 2008.

[103] Shorten, C., Khoshgoftaar, T. M., and Furht, B. Text data augmentation for deep learning. *Journal of big Data 8*, 1 (2021), 1–34.

[104] Smith, L. H., Rindflesch, T. C., and Wilbur, W. J. The importance of the lexicon in tagging biological text. *Natural language engineering 12*, 4 (2006), 335–351.

[105] Steindel, S. J. Learning and Using ICD-10-PCS. *Journal of AHIMA website* (2011).

[106] Sun, C., Qiu, X., Xu, Y., and Huang, X. How to fine-tune bert for text classification? In *China national conference on Chinese computational linguistics* (2019), Springer, pp. 194–206.

[107] Tagami, Y. Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (2017), pp. 455–464.

[108] Teng, F., Liu, Y., Li, T., Zhang, Y., Li, S., and Zhao, Y. A review on deep neural networks for ICD coding. *IEEE Transactions on Knowledge and Data Engineering* (2022), 1–1.

[109] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

[110] Villena, F., Báez, P., Peñafiel, S., Rojas, M., Paredes, I., and Dunstan, J. Automatic support system for tumor coding in pathology reports in Spanish.

[111] Vinayak, R. K., and Gilad-Bachrach, R. Dart: Dropouts meet multiple additive regression trees. In *Artificial Intelligence and Statistics* (2015), PMLR, pp. 489–497.

[112] Wiegreffe. Clinical Concept Extraction for Document-Level Coding. In *Proceedings of the 18th BioNLP Workshop and Shared Task* (Florence, Italy, aug 2019), Association for Computational Linguistics, pp. 261–272.

[113] Wolf, T., Debut, L., Sanh, V., Chaumond, Moi, A., Cistac, P., Rault, Funtowicz, M., Davison, J., Shleifer, Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T., Gugger, S., and Rush, A. Transformers: State-of-the-Art Natural Language Processing. In *EMNLP* (01 2020), pp. 38–45.

[114] Xie, P., and Xing, E. A neural architecture for automated ICD coding. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2018), Association for Computational Linguistics.

[115] Xu, J. Constructing a fast algorithm for multi-label classification with support vector data description. In *2010 IEEE International Conference on Granular Computing* (2010), IEEE, pp. 817–821.

[116] Xu, J. An extended one-versus-rest support vector machine for multi-label classification. *Neurocomputing 74*, 17 (2011), 3114–3124.

[117] Yogarajan, V., Montiel, J., Smith, T., and Pfahringer, B. Transformers for multi-label classification of medical text: an empirical comparison. In *International Conference on Artificial Intelligence in Medicine* (2021), Springer, pp. 114–123.

[118] Zhang, M.-L., and Zhou, Z.-H. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering 18*, 10 (2006), 1338–1351.

[119] Zhang, Y., Burer, S., Nick Street, W., and Bennett. Ensemble Pruning Via Semi-definite Programming. *Journal of machine learning research 7*, 7 (2006), 1315–1338.

[120] Zhang, Y., and Wallace, B. C. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. In *IJCNLP* (2017).

[121] Zinkevich, M., Weimer, M., Li, L., and Smola, A. Parallelized stochastic gradient descent. *Advances in neural information processing systems 23* (2010).