UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA BIOMÉDICA
TRABAJO FIN DE GRADO

APPLICATION OF NATURAL LANGUAGE PROCESSING TECHNIQUES FOR BIOMEDICAL DOCUMENT CLASSIFICATION

Antonio Ochotorena Laynez

2020

TRABAJO FIN DE GRADO

Título:	Aplicación de técnicas de Procesamiento de Lenguage Natural para la clasificación de documentos biomédicos
Título (inglés):	Application of Natural Language Processing techniques for biomedical document classification
Autor:	Antonio Ochotorena Laynez
Tutor:	Álvaro Carrera Barroso
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN
Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

APPLICATION OF NATURAL LANGUAGE PROCESSING TECHNIQUES FOR BIOMEDICAL DOCUMENT CLASSIFICATION

Antonio Ochotorena Laynez

Mayo de 2020

Agradecimientos

En primer lugar, me gustaría darle las gracias a Álvaro por su paciencia y gran ayuda.

Gracias a mi familia, por permitirme estudiar este grado y apoyarme en todo momento. Gracias a mis amigos, por animarme cuando nadie más puede.

Además, me gustaría darle las gracias a todos los profesores que me han enseñado a lo largo de todos estos años. Finalmente, gracias a Helena y a Nuria por ayudarme con el inglés.

Acknowledgement

I would like to offer my special thanks to Álvaro for his patience guidance and his valuable advice during the whole project.

Thanks to my family, for giving me the chance to study this degree and supporting me in every way possible and to my friends, for pushing me forward every day. Besides, thanks to all the professors that educated me throughout the years.

Ultimately, thanks to Helena and Nuria for helping with written English.

Resumen

En este proyecto, estudiamos la capacidad de los ordenadores para clasificar documentos llenos de conceptos médicos. Encontrar formas de organizar automáticamente los datos en un esquema óptimo puede mejorar el trabajo del médico y, como consecuencia final, la posibilidad de atender a más pacientes. Incluso para un experto, estos conceptos pueden ser lo suficientemente complejos como para requerir de una investigación previa antes de dar un diagnóstico adecuado. Con el objetivo de limitar la complejidad del problema en cuestión, decidimos centrarnos en un tema en particular: las enfermedades.

El conjunto de datos, cuya estructura era idónea para la realización de un intento de clasificación biomédica, fue la colección Ohsumed. Gracias a técnicas como el Procesamiento del Lenguage Natural (PNL) y el Aprendizaje Automático (AA) comenzamos a construir los diferentes modelos. En primer lugar, todos los documentos son ordenados y tokenizados para cada clase. En segundo lugar, para transformar los 'tokens' en vectores, se utilizaron tres *embedding models*: Tf-idf, Word2vec y Simon. Finalmente, los modelos resultantes, analizados por distintos clasificadores, determinaron la calidad de las predicciones. En promedio, los resultados en todos los ensayos parecían sugerir que algunos modelos no asimilaban las relaciones médicas de manera efectiva. Otro inconveniente encontrado fue que los documentos tenían múltiples etiquetas, lo que significa que, en promedio, un texto dado podría pertenecer a tres o más clases distintas. La solución requirió transformar dichos identificadores y realizar una clasificación multi-etiqueta, en la cual se obtuvo una puntuación del 68% que corresponde con la más alta hasta el momento (utilizando un clasificador SVC lineal).

Este resultado muestra que la complejidad del campo biomédico requiere la adopción de técnicas sofisticadas para mapear de manera efectiva las relaciones entre conceptos. Sin embargo, dados otros artículos, nuestros resultados parecen ser acertados y ligeramente mejores si consideramos que estamos usando todo el conjunto de datos en lugar de una fracción de los mismos.

Palabras clave: Ohsumed, PNL, AA, clasificación, multi-etiqueta.

Abstract

In this project, we study the computers' capacity for the classification of documents crowded with medical jargon as if they were experts on such topics. The technology era has brought several challenges to many different areas; one of the most affected ones is the biomedical field. Finding ways to organise the data in an optimal scheme, can enhance the physician's work and as a final consequence, the possibility to attend more patients. Even for a doctor, medical concepts can be complex enough to require some previous research before giving a proper diagnosis. To limit this complexity, we decided to focus on one particular topic: diseases.

The dataset, whose structure was ideal for the realisation of a biomedical classification attempt, was the Ohsumed collection. Thanks to techniques such as Natural Language Processing (NLP) and Machine Learning (ML), we started building the different models. Firstly, we had to create a dataframe where all the documents were sorted by their classes and successfully tokenised. Secondly, to transform tokens into word vectors, three different word embedding techniques were used: Tf-idf, Word2vec and Simon. Finally, the resulting embedding models, learned by a classifier, determined the quality of their predictions.

On average, the results all over the classifiers seemed to suggest that some of the models were not assimilating medical relations effectively. Another drawback found was that the documents were multi-labelled, meaning that on average a given text could belong to three or even more different tags. Solving this issue was achievable by transforming the dataset's identifiers into a 23 length binary array. Then, multi-labelled classifiers specially designed for these cases performed the estimation. After this trial, we computed the highest score so far attained in this project of a 68% F-score using a linear C-support vector as the classifier.

This result shows that the complexity of the biomedical field requires the adoption of sophisticated feature extractors to map relations between concepts effectively. However, given other articles, our results seem to be on point and slightly better if we consider that we are using the whole dataset instead of a fraction of it.

Keywords: Ohsumed dataset, NLP, ML, classification, multilabel

Contents

List of Figures

List of Tables

Glossary

AI	Artificial Intelligence
AUC	Area Under the Curve
CAGR	Compound Annual Growth Rate
CBOW	Continous Bag of Words
CV	Cross Validation
DM	Data Mining
EHR	Electronic Health Record
EU	European Union
GDPR	General Data Protection Regulation
GNB	Gaussian Naive Bayes
IoT	Internet of Things
JSON	JavaScript Object Notation
LOO	Leave One Out
MESH	Medical Subject Headings
ML	Machine Learning
MNB	Multinomial Naive Bayes
MRI	Magnetic Resonance Imaging
NER	Named Entity Recognition
NLTK	Natural Language Toolkit
NLP	Natural Language Processing
POS	Part of Speech
RE	Regular Expressions
ROC	Receiver Operating Characteristics
SGD	Stochastic Gradient Descend
SGNS	Skip-Gram Negative Sampling
SVC	C-Support Vector
SVM	Support Vector Machine
OvR	One vs Rest
ROC	Receiver Operating Characteristics
SDS	Support Decision System

CHAPTER 1

Introduction

In Section 1.1, we present the struggles of the medical field in data management as a first approach to our problem. Then in Section 1.3, we explain the objectives of the project. Finally, we describe the structure of the document in Section 1.4.

1.1 Context

Nowadays, healthcare is one of the most developed sectors in the world, experiencing substantial growths each year. To show this increase, the International Data Corporation has given some insight into the current state of the sector. Their researchers estimate the data generated to reach a Compound Annual Growth Rate (CAGR) of 36% in 2025, which is 10% higher than some prominent sectors like media. However, data quantity does not necessarily correspond with quality, meaning that even though the industry is generating data on a large scale, the technologies fall below average in some data competencies. Being more precise, the company Seagate alongside the International Data Corporation positions the healthcare sector with a DATCON (Data Readiness Condition Index) score of 2.4.[2]

These lead us to the next question: How can we extract value from data in the biomedical field? Well, currently, the two main approaches that have more engagement are two. On the one hand, the implementation of human interactive systems as tools to create an information interchange from a person to a computer and vice versa. On the other hand, data extraction tasks, commonly based on machine learning, aims to convert all this information into a structured format so that it can be valuable and easily accessed. [1]

The first one covers disciplines such as the implementation of support decision systems that can help reduce costs and improve quality, the creation of chatbots to minimize the use of workforce in trivial tasks, the design of robots that can reproduce complex procedures without failure and the development of intelligent surgical rooms that can adapt to different case scenarios.

The second field relates to the use of machine learning methodologies. With them, computers can develop the ability to learn without being explicitly programmed (Arthur Samuel, 1959). Human experts are good at recognizing patterns but limited to the dimensionality of the problems (≤ 3). The data quantity and complexity of the medical field generates tasks with sometimes more than a thousand dimensions. Sophisticated algorithms and decent computing capacity are needed to overcome these issues.[3]

In medicine and many other fields, the data is not homogeneous. Healthcare data is one of the most varied and complex of them all, due to the different methodologies used in the sector.

Firstly, there are imaging techniques such as magnetic resonance imaging, computed tomography and ultrasounds. The size of the data can change a lot from one format to another; therefore, to give a concrete example MRI is used. In these procedures, the images' resolution varies from a wide range of values that go from 64x64 up to 1024x1024 pixels. Thus, what could be a small study of breast cancer, it can generate 1500 images which correspond to 300 MB.

Secondly, there is the genomics field, where advances in high-throughput biotechnologies have produced a drastic change in the data size. Four DNA base pairs can fit on a single byte, which means that if there are 6E9 base pairs on a diploid genome, each sample supposes 1.5 GB of storage capacity. This amount of information presents several challenges such as multiple comparisons issues, high dimensional data, computational limitations, and noise. [4]

Lastly, there are the unstructured medical data texts, which can appear in the form of medical informs, abstracts and clinical notes. In this situation, the task is to give a standard format to the data for its posterior use in different information models.

1.2 Motivation

NLP techniques are the ones used to solve these problems.

"Natural Language processing is a sub-field of linguistics, computer science, information engineering and artificial intelligence concerned with the interactions between computers and human natural languages, in particular how to program computers to process and analyse large amounts of natural language data" [5]

It covers a varied range of tasks that analyze syntax, semantics, discourse, and speech. When talking about medical corpora, a series of considerations need to be made to progress successfully analyzing these documents. The medical jargon can be quite complex and varied, which is the reason why there is extensive research done using different approaches. Some of the most known ones are the use of meta-maps and ontologies combined with statistical methods that can organize medical concepts and apply them to specific machine learning techniques, for instance, classification, clustering and regression tasks.

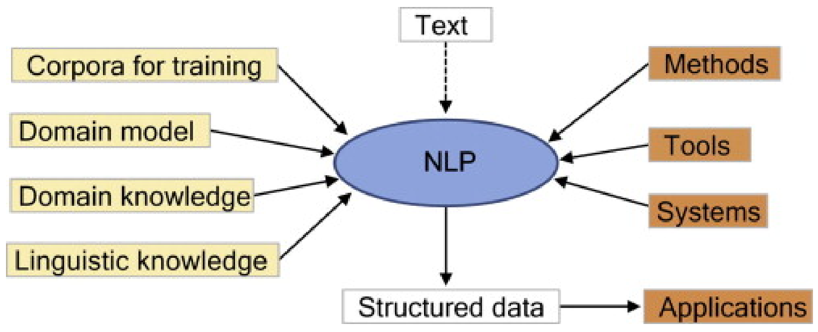

Figure 1.1: NLP basic scheme [1]

These studies have a robust statistical facet that requires large scale datasets to be well-founded. Currently, with the adoption of the Electronic Medical Record as a tool for collecting data, the task of extracting these texts has been worsened. Before, notes where abundant in the narrative description of the clinical encounter, which is critical for the development of linguistic models. Right now, inserting data on a discrete format can result in a drastic reduction of useful corpora for the NLP analysis. [1]

In this work, we are going to use the most common methods applied for unstructured text analysis to measure the effectiveness and accuracy of classifying medical corpora.

1.3 Goals of the Project

A series of goals are defined to ensure a successful evaluation of the models:

1. Look for a large dataset with unstructured medical data that fits the requirements to perform a classification task. The subject matter of the text can cover any medical category from treatments, diagnosis and diseases to organs, anatomy and physiology.
2. Regarding the different datasets from the anterior task, select a corpus that fits best the requirements; giving preference to labelled datasets in order to lessen the preprocessing work.
3. Prepare the data using NLP techniques, allowing the ML algorithms to compute the classification tasks by training and testing the documents, already preprocessed, in a variety of pipelines.
4. Obtain statistical metrics from the different pipelines to test their robustness and efficiency. Once those results are explored, we will conclude showing all the different findings and interesting achievements obtained throughout the project.

1.4 Structure of the document

The rest of the document is structured as follows. In Chapter 1, the current state of the medical sector is exposed alongside the different project goals. Then, in Chapter 2, the tools and programs required for the development of this work are described. Next, Chapter 3 presents and explains the dataset selected for the project along with its advantages and disadvantages. Later, in Chapter 4 we detail the implementation of different pipelines that propose a varied range of approaches to perform the classification. Afterwards, in Chapter 5 the results of the models are shown in the form of different scoring metrics. Finally, in Chapter 6, some conclusions are presented and some possible paths for future work are proposed.

CHAPTER 2

Enabling Technologies

In this chapter, section 2.1 introduces the fundamentals of language features.Then, section 2.2 describes the techniques used to exploit them. Finally, section 2.3 defines the machine learning technology designed for unstructured data.

2.1 Language features

Written text is full of meaning and presented to the reader in many different ways. Our brain has developed through learning how to extract the intention from such writings. However, implementing this sort of intelligence on a computer is a task far from being achieved. This section describes the methods that allow a machine to extract meaning from various pieces of literature. The underlying structure of written texts has a varied set of characteristics [6] that serve as features for a computer to analyse:

- **Genre:** defines a document with a characteristic form. There are many different types, for instance, brochures, biographies, memoirs, forms and e-mails.
- **Text structure:** describes the organisation of writings; for example, factual texts have a categorical structure with sections and headings. Expressing content with structural patterns determines how difficult it is to understand readings; the com-

bination of descriptions, chronological sequences, comparisons and cause/effect are some of them. It is the writer's task to make an effective combination so that it is understandable for the target audience.

- **Content:** relies on understanding the subject matter of the text. On factual extracts, it refers to the topic. The level of understanding of an excerpt depends on the level of the reader in that particular topic. For example, a reader's comprehension level does not matter at all if he/she is reading a clinical note without being familiar with medical jargon.
- **Themes and ideas:** themes relate to the ideas that a writer wants to transmit to the audience. Those can be concrete and easy to get, or abstract and hard to extract from the reading.
- **Language and literary features:** there is an extensive set of literary elements subject to the writing form. Factual writers use description and technical communication.
- **Sentence complexity:** represents how to map meaning in an essay. The simpler the sentences, the easier it is for a reader to understand. If phrases are embedded together with conjoined clauses, comprehension will worsen.
- **Vocabulary:** gathers the particular lexicon used to give meaning and express content and ideas in a sentence. Contains a list of concepts which represent the most basic structure in literature and speech, words.
- **Others:** apart from raw text, on a book, there are also tables of contents, indexes, notes and illustrations that give extra meaning to the main corpus and may contribute remarkably to the understanding of an extract.

All these language features play an essential role in the implementation of natural language processing systems. Depending on the purpose of the study, some of them will be more relevant than others. It is a developer's task to choose which features will help to extract value from the text.

2.2 Natural language processing

A book called 'Speech and Language Processing' [7] served as a guideline for the development of this section. Therefore, if you are interested in a complete explanation of a specific topic, we recommend reading through it.

2.2.1 Analysing words

The description of the methodologies used in NLP starts now, beginning with the analysis of the most basic structure of language (words). Detecting a particular token is a task usually managed by the use of the so-called regular expressions.

2.2.1.1 Regular expressions

It is the language used in computer science to search for a specific text string. A particular algebraic notation with many variants is the one used to perform this analysis. Computer languages, word processors and text processing tools use this notation to operate.

The nature of the target can be varied; therefore, to be precise while writing the commands, we have to consider the use of several tokens that will help expressing it clearly. The first thing to point out is that the term between which will be written the target word is the double slash (/.../). Inside those bars, the next thing to have in mind is that the Regular Expressions (RE) are case sensitive. To solve this ambiguity, the squared brackets (/[]/) represent disjunctions, allowing us to assimilate together upper and lower case tokens (/[Aa]/). The next term that saves time in the developing of the RE is the range expression (/-/), which wraps all the words within two points. Other characters can have several uses depending on the place they are situated, for example, the caret (ˆ) has three: it can mark the beginning of a sentence, the negation of a term and the real text token ˆ.

In the typical dilemma of American/British English, words like color/colour need to be spotted. The token that works well with such cases is the question mark (/?/) which marks the optionality of the previous character. Lastly, the distinctive character period (/./) represents any possible term that appears in a position like a wildcard (except the carriage return character). In the table 2.3, we can see the application of some of the RE described above.

These examples represent isolated cases in language, useless at all in real case scenarios. The context of a word can induce mistakes in the detection of a particular term, to show a practical and straightforward example we will explain section 2.1.3 from [7] where the task was to detect the word 'the'. The procedure is the following:

1. /the/ appears to be the simplest and straight solution. However, given this query, we will be missing the upper cases 'The' at the beginning of sentences, which is a widespread case. Solution: /[Tt]he/
2. Once this issue is solved, another one appears. Words that contain on them the

Regular Expression	**Text example (with multiple matches)**
/[0-9]+/	My phone number is **646525434**
/[Bb]eg.n/	**Begin**,**began**, **begun** are the possible tenses
/colou?r/	What word comes from British **color** or **colour?**
/[A-Z]\s/	The package was delivered by **SERANCO** **SA**

Table 2.1: Explanation of some RE special characters

term 'the' represent a false positive scenario, for example, the names ´theorem' or ´theology'. Solution: /\b[Tt]he\b/

3. Now we want to detect the word even if numbers follow it 'the24'. To do so, we have to replace the previous token blank space (\b) for a RE that means not a character. Solution: /[ˆa-zA-Z][Tt]he[ˆa-zA-Z]/
4. The last problem is that this query cannot detect words either at the beginning or end of a sentence. As a solution placing an (ˆ) token covering both possible scenarios solves this task. Final solution:/(ˆ|[ˆa-zA-Z])[Tt]he([ˆa-zA-Z] |$)/

2.2.1.2 Words

The next step before processing a corpus will be to determine what counts as a word and what does not. The first logical approach is to consider one as a string surrounded by white spaces. However, several exceptions make this task more complicated. What shall we do if the sentence we are looking for is at the beginning/end of a sentence? Also, what about several languages like Japanese that do not use spaces in their writings? Taking into account all these factors is critical for tokenising successfully.

Punctuation is essential for capturing meaning and locating sentences; hence, the first process of tokenisation considers punctuation as 'words'. Another factor to study is capitalisation, are 'They' and 'they' the same words? It will vary depending on the methodology used; part of speech and named entity tagging try to maintain them while speech recognition ignores those disparities.

What about these 'trees' and 'tree'? In this case, these words have the same lemma but different word-forms. A lemma is a set of words that possess the same stem, word sense and PoS. A word-form is the way a word can exist in the context of a language to fit both with their form (grammatically) and their meaning (semantically).

Therefore, defining a set of words requires the definition of two concepts which are 'types' and 'tokens'. Types represent the number of unrepeated words on a corpus while tokens represent the total amount of running words. The more terms or vocabulary size you have, the more types there will be. The Heaps' Law defines this relation as 2.1

$$|V| = kN^{\beta} \tag{2.1}$$

where $|V| \equiv$ the number of types, $N \equiv$ the number of tokens, k and $\beta \equiv$ are positive constants where $0 < \beta < 1$

With all these terms defined, we can begin explaining the NLP methods that allowed the completion of this project.

2.2.2 Lexical processing

The first operation is called text wrangling. These procedures aim to clean what is considered noisy data in a systematic way. Thanks to python libraries such as Natural Language Toolkit (NLTK), Gensim and TextBlob is possible to perform these tasks smoothly.

1) Cleansing
Sometimes data is stored on a web page and has markup code on it. Therefore, to obtain the raw text, the unneeded lines without value have to be deleted.

2) Tokenisation
This process relies on the extraction of tokens as sentences or as isolated words. As it was explained in the previous section, depending on the purpose, we will apply one method or the other.

3) Stemming and Lemmatisation
Both processes compute words and reduce them to their stem or lemma. Technically, the two represent the same concept, which is a word reduction to its root. In practice, the difference is that lemmatisers use as an extra tool, the Part of Speech on each word. These last ones provide better results in the process but are slower computationally talking.

4) Stop word removal
In a corpus, the so-called stop words are the ones that appear with the highest frequencies. Before, it was reasonable to believe that the high appearance rate of a concept gave useful information to the processing. To prove this wrong, checking on

a large dataset the terms that appear most are words like ‘and’ and ‘the’. Those are familiar words, used to connect sentences with no meaning at all, and therefore, detachable.

5) Punctuation removal
Now, when punctuation marks are no longer needed, it is time for removing them. At this point in the preprocessing, every punctuation token is detected and isolated from the rest of the words. Thus, removing them by checking a stop word list represents an effortless task.

6) Rare words
Typos are a common thing in written texts that can add noisy data to the sample. Removing them is a simple task that involves counting words. The frequency of misspellings is usually low (1-2 incidences in total). Therefore, printing a frequency list, removing the unique ones from the dataset will be enough to reduce this noise.

2.2.3 Syntactic processing

The syntax is the arrangement of words and phrases to create well-formed sentences in a language. The NLP methods in charge of analysing these features are Part of Speech (POS) tagging, Named Entity Recognition (NER) and parsing.

2.2.3.1 Part of speech

POS tagging is the method in charge of assigning a grammatical category to a word. The most common approach is to use an annotated corpus such as the Penn Tree Bank. Each corpus has its notation; therefore, knowing its structure is vital to understand the results of this process.

In some occasions, words can be ambiguous. That is the central problem POS tagging has to face. On the English language, 85% of words are unambiguous, which leads to thinking that this incertitude is not a common problem in writings. However, the specific terms that are ambiguous are the ones most frequently used to express ourselves, leading to the presence of such terms up to 55-67%.

There are three main approaches to overcome this challenge, two of them based on sequence modelling and the third one relies on a neural network approach. Those are the following:

1. Hidden Markov Model is a generative model that uses for decoding the Viterbi algo-

rithm or a beam search variant.

2. Maximum Entropy Markov Model is a discriminative model that uses logistic regression as a tagging tool and the Viterbi algorithm for decoding.
3. The neural approach is a discriminative model that uses word embeddings as inputs, while the outputs are tag probabilities obtained through the available tagset.

2.2.3.2 Information extraction

One of the most useful features that NLP has is the possibility of information extraction. Nowadays, with such an amount of data circulating through the internet confirms how crucial is to gather all this information logically.

The process has several steps that aim to tag the data and extract the valuable one. Firstly, NER is in charge of labelling and classifying what we call entities (persons, locations and organisations). Secondly, obtaining a relation between them is crucial to give meaning to the information. Thirdly, these so-called entities perform actions in time; and therefore, the moment of its execution represents a valuable feature extracting. Finally, what is left to do is a template filled with all the relevant content retrieved.

2.2.3.3 Parsing

It is useful for the extraction of semantic structures on a given sentence and therefore applied in tasks such as grammar checking, question answering and information extraction. In this last case, partial parsing is enough for obtaining NER and uses machine learning techniques for its functioning.

2.2.4 Word embeddings

2.2.4.1 N-grams

One of the most basic ideas of NLP is predicting words in a sentence. This principle of ‘which word will go next’ is crucial in disciplines such as speech recognition, grammar correction and language translation. Its basis relies on the probability of appearance of a term given a previous series of words that will be called grams. Depending on the number of grams, there are bi-grams (2), tri-grams(3), and so on.

Identifying a word and its full meaning in a context can be a challenging task. There are two principles present in a lexicon, the conventionality principle and the principle of contrast. The first one states that each word has one or more ‘conventional’ meanings; while the second, on the contrary, tells that each term differs from every other in some sense. A word by itself is unable to transmit information; is the combination of words what gives coherence to a text.

A language model has to be able to represent relations between words for solving meaning related tasks. Hence, every model must be able to retrieve features such as similarity, relatedness, roles and connotations.

Similarity measures the closeness between two words; it is not the same as synonyms, but in some cases can occur. Relatedness detects how two different terms (with no similarity) can match, for example brain and skull. Roles is the task of identifying relations between actions. Lastly, the meaning of connotations varies depending on the task at hand; in sentiment analysis, it detects emotions and classifies them by three numbers called valence, arousal and dominance. This notation introduces the most common standard used NLP nowadays, which is the vectorisation of words, also called word embeddings.

There are loads of methods to build word embeddings, but all of them have a thing in common; they rely on the analysis of the context to represent words. The simplest ones are the term-document matrices that function by counting the frequency of apparition of a given name in each corpus.

Word	**Freq doc 1**	**Freq doc 2**	**Freq doc 3**
Bone	5	0	5
Muscle	1	20	4
Heart	0	15	3
Femur	13	1	2

Table 2.2: Intuitive example of the Term Document Matrix

From this matrix we can extract the following word vectors:

$$bone = [5, 0, 5] \quad muscle = [1, 20, 4] \quad heart = [0, 15, 3] \quad femur = [13, 1, 2] \tag{2.2}$$

On the upper table 2.2, we saw a case of four words, however, given a big dataset the number of columns for those matrices will be equivalent to the number of documents; which is usually bigger than 10K. The number of rows/words will be equal to the size of the vocabulary $|V|$(also a large number). The supposition done in this case is that two documents will tend to be similar if they use the same words and therefore classifiable. The current techniques used are a more complicated than this simplistic approach; those are:

2.2.4.2 Cosine similarity

This metric measures the similarity of two words thanks to the properties of the dot product operator. Two orthogonal vectors have a result of 0 while the ones in the same plane are equal to 1. There is also a problem given the word frequency that alters this similarity measure and gives higher scores to repeated terms. The solution to this problem is normalising this dot product that turns out to be the cosine angle equation.

$$cosine(v,w) = \frac{vw}{|v||w|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{2.3}$$

Where v and w are the word vectors selected for the similarity measure

2.2.4.3 Tf-idf

It is the most common word embedding used as a baseline in NLP. It works based on two different equations multiplied between each other at the end. The first one, called term-frequency (tf), works by counting the apparition rate of a word into a document as a term-document matrix. Another factor to consider is that the frequency of a term is not directly proportional to its relevancy, and therefore, the equation adds the log function to correct this tendency.

$$tf = \log_{10}(count(t,d)) + 1 \tag{2.4}$$

The second term is the inverse document frequency (idf) that gives higher weights to the words that appear less in the whole collection. A name that is present in every single document does not provide relevant information about the characteristics of the corpora. The equation of this term is also a logarithmic function given the high number of documents that can appear in a collection. The fact that the matrix terms depend on the particular context of a target word results in sparse matrices with lots of zero components on it.

$$idf = \log_{10}(\frac{N}{df_t}) \tag{2.5}$$

The final weighting co-occurrence matrix defines the whole collection, and it is useful for tasks such as information extraction. Another method, and the one that is relevant for this project, is the computation of document similarity. In this case the procedure is to take each document and given their co-occurrence matrices calculate its centroid; these centroids represent vectors that we can classify by cosine similarity measurements.

2.2.4.4 Word2vec

Unlike tf-idf, word2vec produces word embeddings in the form of dense matrices. This different approach has proven that machine learning algorithms work worse on sparse matrices. The dimensionality reduction of the embeddings reduces computing times and is capable of capturing the similarity between words.[8]

There are two main algorithms used in word2vec; the first one Continous Bag of Words(CBOW) predicts a word given a context. The second one, Skip-Gram Negative Sampling (SGNS), proceeds the other way round; it predicts the context given a specific word.

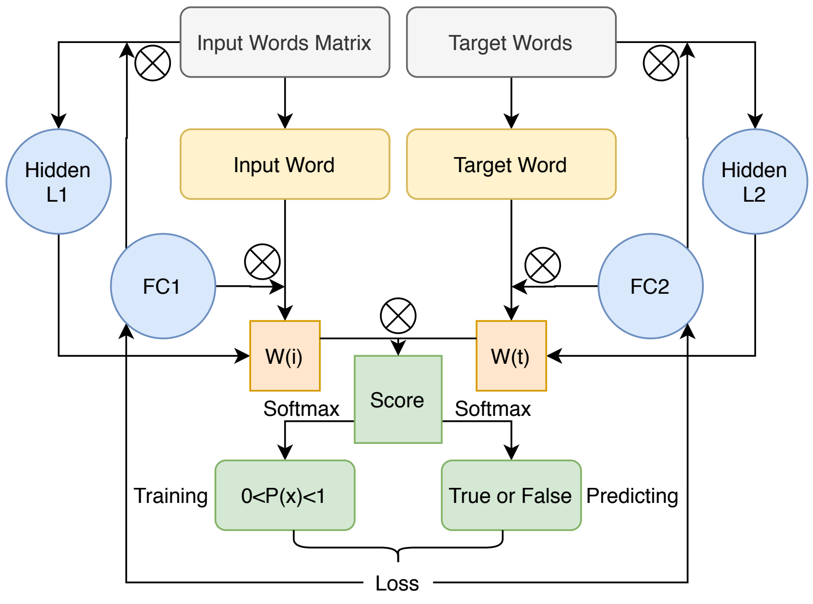

Figure 2.1: Word2vec SGNS scheme

This last algorithm, developed at Google by Tomas Mikolov and his team in 2014, bases its working in a shallow neural network with two layers. A fully connected one is in charge of transforming input words to the embedding vectors, and a second one produces weights that turn tagged words to context arrays. These two results combined give a score that applied to a sigmoid activation function makes the model learn by comparing the output to the target terms. Backpropagation updates the weights on each iteration until it finishes training. The input is selected based on a window size that compares those words with other randomly chosen from the corpus.

The resulting embedding can predict target words in the following queries; however, the real objective of word2vec was to extract the weighted embedding and use it as a dense matrix for the subsequent NLP tasks.

2.3 Machine learning for Natural Language Processing (NLP)

The objective of this section will be to explain the available tools that make possible the accomplishment of classification tasks. This technique is useful for many different labours such as sentiment analysis, spam detection, authorship and library category assignment.

In machine learning, a classifier works by assigning an output value to a given series of inputs; supervised techniques are the ones that support these procedures. The definition of those output classes can be a feature already implemented in the dataset or a manual task to do for the developer. The corpora defines a set of tuples $((d_1, c_1), ..., (d_n, c_n))$ where each document d is related to a particular and unique class. The ultimate purpose of the classifier is to take a new document $(d_{n+1}, ?)$, and with the previous knowledge from $((d_1, c_1), ..., (d_n, c_n))$ predict its output class ?.

Learning all our dataset seems to be the better way to optimize the results of the classification to its maximum. However, overfitting is a common drawback from limited sized datasets. The solution to avoid this problem is to divide the data into training and test sets with a method called cross-validation.

The idea is to divide the available data into a series of train and test splits. Each group of (train, test) is what we call a fold. When performing cross-validation, the standard procedure is to select ten folds, which entails the execution of ten classification processes. The developer gets ten different metrics from each different fold and will have to combine them to get an averaged score. Using this tool increases the reproducibility of the results, which is vital for model implementation.

The next step is selecting which metrics will be useful for the evaluation of a NLP model. The construction of a confusion matrix is the starting point towards evaluating a classifier. The rows represent the predictions made by the computer and the columns the correct labels of those documents. Therefore, the standard classification matrix of 2 classes will

	Actually Positive (1)	**Actually Negative (0)**
Predicted Positive (1)	True Positive	False Positive
Predicted Negative (0)	False Negative	True Negative

Table 2.3: Confusion matrix scheme

have a size of 2 × 2, where each position will represent true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) respectively.

The standardized measure extracted from this matrices is the accuracy, which represents the number of correct predictions over the entire documents. Its equation is the following:

$$Accuracy(ACC) = \frac{\sum TP + \sum TN}{\sum TP + \sum FP + \sum TN + \sum FN} = \frac{\sum TP + \sum TN}{\sum documents} \tag{2.6}$$

However, this metric is not enough for defining a good classifier. The first thing is noticing that they are two different types of error which are false positives or false negatives. This distinction, ignored by the accuracy expression, can lead to confusion. Imagine predicting a virus infection, here; the expectation is to detect all the positive cases. In other words, our classifier needs to have 0 false negatives, and a high accuracy does not necessarily mean this. As a solution, two metrics can fulfill these requirements. Those are:

$$Precision(Predicted\,Positive\,Value) = \frac{\sum TP}{\sum TP + \sum FP} \tag{2.7}$$

$$Recall(True\,Positive\,Rate) = \frac{\sum TP}{\sum TP + \sum FN} \tag{2.8}$$

Depending on the task at hand, the objective will be having either a good recall or precision. It is the task of the developer to determine the classifier requirements. Some projects do not necessarily need one of these measures because sometimes having a good recall can provoke a low precision and vice versa. A balanced metric that represents the harmonic mean of these two is the F-score, which has this expression:

$$F\ score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.9}$$

The calculations are straight when dealing with two classes. However, what about having a classification problem with a higher number? In this case, there are two ways of obtaining the desired metrics; both of them slightly different from each other; those are macro and micro averaging.

Macro averaging extracts the scores for each class and averages the results at the end, micro averaging, takes all the predictions from all categories and compute them together. The different ways of proceeding are the reasons why micro averaging results tend towards the most frequent class in the dataset and macro averaging ideal when all the labels are equally important in the classification.[9]

There are several types of classifiers designed for many different purposes. Nevertheless, to perform NLP techniques, some standard algorithms are the ones that present better results because they were developed explicitly to such tasks.

2.3.1 Multinomial naive bayes

The purpose of this classifier was to use it in text categorization tasks such as spam detection and sentiment analysis; which is the reason why there is abundant research done on those matters.

This classifier works by making two assumptions. Firstly, the word position does not count, and the only feature that matters is the frequency of a term (Bag of Words). Secondly, there is the 'naive' assumption where each word conditioned probabilities to a class are independent of each other.

Naive Bayes uses probabilities to make predictions which is the reason why a specific document d can be from class c when $P(c|d)$ is high. This classifier develops this expression using the assumptions seen above to simplify the algorithm.

$$\hat{c} = argmax\ P(c|d) = argmax\ P(d|c)P(c) = argmax\ P(f_1, f_2, ..., f_n|c)P(c) \tag{2.10}$$

Where $P(d|c)$ is the likelihood and $P(c)$ the prior. Applying the Naive Bayes assumption:

$$\hat{c} = argmax\ P(f_1, f_2, ..., f_n|c)P(c) = argmax\ P(f_1|c)P(f_2|c)...P(f_n)P(c) \tag{2.11}$$

To avoid overflow and to increase the speed, the expression can be transformed into the log space.

$$\hat{c} = argmax\ logP(c) \sum P(f|c) \tag{2.12}$$

Apart from knowing how to predict based on probabilities, the algorithm needs training. The Naive Bayes learning process has two parts; the first one has to do with computing the probability of each class as:

$$P(c) = \frac{N_c}{N_{doc}} \tag{2.13}$$

Where N_{doc} represents the total number of documents and N_c is the number of documents that represent a particular class.

The second one, joins all documents from a given class and count their word frequency, then, compare it against the whole Bag of Words:

$$P(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)} \tag{2.14}$$

There are also some particular scenarios to take into account related to the probability 0 case which happens, for example, when we want to classify a word like femur into a class bone, with no femur words in all documents from that class. A valid solution is to use the add one Laplace smoothing by adding one both in the denominator and numerator of the expression when training.

$$P(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} count(w, c) + 1} = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|} \tag{2.15}$$

2.3.2 Logistic regression

The difference between Naive Bayes and Logistic Regression (LR) is that the first one has a generative approach where it computes probabilities based on the likelihood term $P(d|c)$ given in equation 2.10. In contrast, the LR classifier has a discriminative approach that inspects this expression $P(c|d)$ directly. However, they also have things in common, as they are both linear classifiers.

The model works by assigning weights given a series of inputs in a training phase along with a bias term. In this process, a classification function called cross-entropy loss determines the likelihood between a prediction and the ground truth term. For optimizing this function and converge fast to its global minima, stochastic gradient descent iterative method is the one used in these procedures.

In LR each weight, w_i is a real number associated with one input feature x_i, where a positive number represents a high correlation with the class and a negative one discrepancy. The expression results as follows:

$$\hat{y} = \sum w_i x_i + b \tag{2.16}$$

However, this prediction $\hat{y}$ is not a real probability because $\hat{y} \in \mathbb{R}$ and therefore, its value is not between 0 and 1. LR converts the term through a sigmoid (binary classification tasks) or a softmax function (multinomial) to solve this problem.

2.3.3 Support vector machines

This machine learning technique is useful in regression and classification tasks. In NLP it is commonly used in POS tagging and information extraction processes.

The algorithm looks towards separating the different features by spotting a hyperplane with a large margin between classes. The length of the margin, which corresponds to the minimum distance between the training samples and the hyperplane, allows new features to fit the same labels even if they are different from the trained input.

There is a varied set of Support Vector Machine (SVM) algorithms that differ in how the hyperplane is defined. Hard SVM, tries to find a perfect separation between features; soft SVM is less strict with the boundaries and does not assume a severe separability between samples. Lastly, kernel SVM is the one with more repercussion because their functions are capable of working with high dimensional data which was a critical problem that SVM algorithms had to face.

2.3.4 Ensembles

Finally, the ensemble models are in charge of combining the different outputs from the varied set of classifiers so that the overall performance could be improved. There are two main categories of ensembling methods, averaging and boosting. The first one averages the predictions of a set of classifiers separately and combine the results in a parallel way; the second builds a sequence of classifiers that learn by correcting the bias from the prior in an iterative process.

2.3.4.1 Averaging ensembles

The most renowned ensembles of this section are the following ones:

Bagging

Uses a set of N classifiers that train by extracting subsamples from the training set with replacement, doing so, ensures diversity and robustness in the decisions. The typical algorithms used in bagging are linear, for instance, SVM and logistic regression. Depending on the sampling, there are four possible variations: the pasting, the random subspaces and the random patches. In the case of large datasets, bagging is not the best because it is only optimal for small datasets, and as an alternative, pasting works better. In the end, a majority voting decides the best prediction from all the trained classifiers.

Stacking

This renowned ensemble technique improves single classifiers and combines them by reducing bias and over-fitting. These first classifiers, also called layer 1, learn by using a part of the training set and the rest for predicting. Then, it combines these resulting predictors via a meta-classifier that uses the results as new inputs for a new learning process.

2.3.4.2 Sequential ensembles

Boosting

Iteratively trains a set of 'N' weak classifiers, lowering the training error on each step. In each cycle, the mistaken samples from the previous phases move on to the next classifier to learn better from the most conflictive features. Adaboost is one of the most used algorithms; its working is similar to the original boosting method but differs from it in the use of weights for training and decision-making. On each training phase, the correct classified samples do not represent a big problem to the model, as a consequence, the algorithm focuses on learning from the mistaken inputs resulting in a reduction on the training error (similar to the loss function seen in Section 2.3.2). It also repeats an iteration until such error rate gets lower than a 50% threshold. Finally, to make predictions, boosting methods perform a weighted average of the N estimates.

Ensembling is the last resort in machine learning that comes up when there is nothing left to try. All the previous steps involved, such as cleaning the data, feature extraction and multiple classifiers trials, are vital to ensure a successful ensemble. The predicting ability of a model is entirely dependant of the features; therefore, having extracted a varied set of them is fundamental to exploit the data successfully.

CHAPTER 3

Ohsumed dataset

The internet is a powerful tool for extracting data in many different fields. The problem is that, usually, this data needs a large number of working hours in order to be useful for machine learning. Finding a dataset that fits the requirements of a plan is a difficult task due to the heterogeneity of data. Usually, a final dataset for a specific scheme results from merging more than two completely different sets. Hopefully, the project task at hand was broad enough to find suitable corpora for its fulfilment. In this chapter, section 3.1 describes the corpora selected for this project, the Ohsumed dataset. Its selection was carried out by looking for essential features that allow the implementation of classification tasks. Additionally in section 3.2, taking into account that the final objective is to extract medical meanings, the documents we seek should be abundant in well represented biomedical concepts.

3.1 Dataset overview

The construction of this dataset finished in 1991 where titles and abstracts from 270 medical journals of the Medline database were gathered together (to download the dataset go to [10]). The texts describe a set of 23 disease categories extracted from the Medical Subject

Top level categories	**Label**	**Top level categories**	**Label**	**Top level categories**	**Label**
Anatomy	A	*Biological Sciences*	G	*Persons*	M
Organisms	B	*Physical Sciences*	H	*Health Care*	N
Diseases	**C**	*Social Phenomena*	I	*Publication Characteristics*	V
Chemicals and drugs	D	*Technology and Food*	J	*Geographic Locations*	Z
Techniques and Equipment	E	*Humanities*	K		
Psychiatry and Psychology	F	*Information Science*	L		

Table 3.1: Top level MeSH categories

Headings (MeSH) thesaurus. This hierarchic vocabulary aims to enhance the search and acquisition of biomedical information [11]. The dataset consists of 56984 medical abstracts with a smaller version pruned to classify cardiovascular diseases.(see table 3.1)

The size of the documents varies between a range that goes from 1KB to 4KB storage; adding all the documents together results in a total volume of 66.3 MB. This volume is directly proportional to the complexity of the text; usually, in a low size corpus (1 KB) the structure consists of a single paragraph, whereas in documents from 3-4KB it is commonly noticeable a subdivision of the information based on headings represented by capital letters. Taking a random document of 4KB from the dataset, resulted in an abstract divided into eight different sections named the following way: *objectives*, *design*, *setting*, *patients*, *interventions*, *main outcome measures*, *results* and *conclusions*. These final two are habitually the most relevant headings when talking about the whole dataset because the main features of the labelled diseases regularly belong to these drafts.

To conclude, if we look at figure 2.1, we can see that the distribution between classes is highly biased with tags that have more than 2K documents (10) and others that do not reach the 1K mark (5). The eight labels that are not mentioned stay around 1.3K on average. Therefore, in the next chapters, while we are developing the models and extracting results, we will have to be careful with this variance and analyse until which point this is affecting the classifiers. We will also use tools to minimise the impact of this distribution like the Stratified K-fold method to prepare the models, and the macro and micro averaged metrics to check for irregularities in the estimators' results.

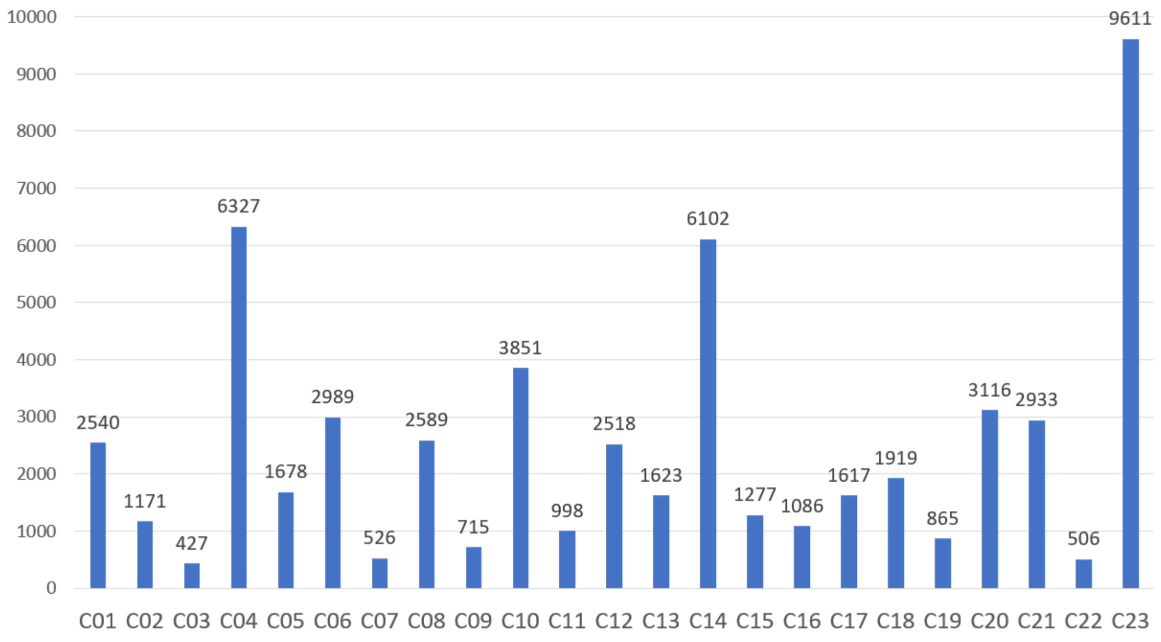

Figure 3.1: Ohsumed distribution of documents

3.2 Multi-labelled data

Another remarkable characteristic of a dataset is the presence of multi-labelled documents. When we are talking about real world-data, uniqueness is not a standard feature [12]. For example, a document in sentiment analysis can express positive and negative emotions without a clear distinctive of which one is dominant. However, due to the reduced dimensionality of the task at hand (positive, neutral or negative), classifiers can manage to sort these problems.

If the dimension of the labels increases, the appearance of more complex relationships between classes becomes a reality. In our project, with 23 different labels, several documents are placed in more than one class. In medicine, one disease can affect physiologically and anatomically several areas of the body, which is the reason why the developers assign multiple tags to the documents of the dataset.

The next step will be computing the number of repeated documents, which results in 16087. This number means that the amount of unique texts in the dataset only corresponds to 32.1%. The next job is to detect the labels from which these repetitions are coming. After this analysis, we check if the multi-tagging makes sense according to the MeSH categories. The process goes as follows:

1. Firstly, we check each class independently to make sure there are no repeated docu-

Labels	**1**	**4**	**6**	**10**	**12**	**14**	**17**	**18**	**20**	**21**
1	**2540**	150	308	160	156	165	89	44	281	155
4	150	**6327**	415	361	547	208	192	128	362	120
6	308	415	**2989**	123	124	141	75	127	160	135
10	160	361	123	**3851**	81	401	79	107	175	302
12	156	547	124	81	**2518**	234	44	145	89	72
14	165	208	141	401	234	**6102**	115	285	106	185
17	89	192	75	79	44	115	**1617**	36	156	62
18	44	128	127	107	145	285	36	**1919**	55	48
20	281	362	160	175	89	106	156	55	**3116**	151
21	155	120	135	302	72	185	62	48	151	**2933**

Table 3.2: Prunned confusion matrix of the top 10 label's relationships

ments.

2. Secondly, we count the number of repeated texts in the whole corpora (already shown in the previous paragraph).
3. Finally, we studied how frequent those repeated texts are distributed between classes.

From this process, we conclude with a comparative table 3.2 that shows the incredibly high relation between the different diseases and their associated classes (to see the whole matrix go to section C.4).

With medical knowledge, we can see how two different tags from the dataset can be related. Selecting some of the most noticeable groups of labels present in the corpora was useful to check the integrity of the classified documents. Some of the relations seen were:

- Bacterial Infections and Mycoses (C01) with Digestive System Diseases (C06)
- Virus Diseases (C02) with Immunologic Diseases (C20)
- Respiratory Tract Diseases (C08) with Neoplasms (C04)
- Musculoskeletal Diseases (C05) with Skin and Connective Tissue Diseases (C17)
- Otorhinolaryngologic Diseases (C09) with Respiratory Tract Diseases (C08)

There are plenty of these relations in the dataset that goes up to a 67.9%. This number shows the complexity and cohesiveness of medical concepts. In this last table 3.2, we confirmed that the relations were consistent with the data. Finally, to end this chapter, we will briefly develop the medical categories to explain some of the strongest relations.

3.3 Exploring dataset's diseases

Bacterial Infections and Mycoses (C01): Bacteria are prokaryotic microorganisms with a varied set of forms (spheres, rods and spirals). The most distinguished bacterias are the ones present in our digestive system (Escherichia Coli, Helicobacter Pylori), enabling certain functions. However, if their number exceeds a specific threshold, they become harmful to our organism producing an infection. On the other hand, mycoses represent fungal infections propitiated by physiological and environmental changes.

Virus Diseases (C02): A virus is an infectious agent that depends on other organisms to survive. The Human Immunodeficiency Virus (HIV) is one of the most famous ones; it affects the immunologic system and is very abundant in our dataset, which explains the strong relation between C02 and C20 (Immunologic diseases).

Parasitic Diseases (C03): A parasite is an organism that invades a living host taking nutrients from them. Three main classes can cause human diseases, those are: protozoa, ectoparasites and helminths. The main difference from a virus is that parasites can leave outside of the hosts for a significant amount of time.

Neoplasms (C04): Represents all the different abnormal and uncoordinated growths that can happen in the organisms. When the mass is uncontrolled and reaches a considerable size, it is called a tumour. The table suggests that the most common neoplasms from this dataset are pulmonary, digestive and prostate ones.

Musculoskeletal Diseases (C05): Encompasses all the pathologies that affect the locomotor system, conformed by bones, muscles, tendons, cartilage, ligaments, connective tissue and joints. Motion is controlled by the Somatic Nervous System (SNS), which is the reason why a consistent part of the diseases are related to class 10.

Digestive System Diseases (C06): The system that allows us to extract nutrients from the food by decomposing them to their basis molecules. It is formed by the gastrointestinal tract, and organs like the pancreas, the liver and the gallbladder. The bacteria of the digestive tract represent the primary source of diseases of this label in the corpora.

Stomatognathic Diseases (C07): The term stomatognathic refers to the anatomic system formed by the jaw, the teeth and associated tissues. Neoplasms seem to be the ones with the strongest relation to this topic.

Respiratory Tract Diseases (C08): Represents two connected anatomical areas form the respiratory tract; the upper one (nasal cavity, pharynx and larynx) and the lower part (trachea, primary bronchi and lungs), this system allows us to extract the oxygen from the atmosphere and transport it to our cells. Some of the most related diseases come from neoplasms and the cardiovascular system.

Otorhinolaryngologic Diseases (C09): Englobes all the diseases that affect the ears and equilibrium, paranasal sinuses and some disorders of the upper and lower respiratory tract, which is the reason why 8 and 9 correlate in the dataset.

Nervous System Diseases (C10): The nervous system represents the complex network of neurons in charge of transmitting signals between the different parts of the body. They have two main parts: the central and the peripheral nervous system. It is very related to cardiovascular diseases because the brain is the organ that requires more oxygen and nutrients (intensely irrigated).

Eye Diseases (C11): The eye is the body organ that enables our sight sense. It is composed of pupil, cornea, iris, optic nerve, vitreous, sclera, fovea and the ciliary muscle. The most related diseases in the corpora seem to be caused by bacteria and neoplasms. The eye microbiome represents the collection of bacterial cells that resides in the eye and maintain its functions. An unbalanced number of them causes several eye diseases.

Urologic and Male Genital Diseases (C12): Urology is the field that treats all the events related to the urinary system and male/female genital problems. The strong relation with neoplasms suggests that the documents treat one of the most extended tumours in men: prostate cancer.

Female Genital Diseases and Pregnancy Complications (C13): Similar to the male scenario and adds to it pregnancy complications. It englobes one of the most known and studied neoplasms in women, breast cancer.

Cardiovascular Diseases (C14): The cardiovascular system is composed of heart and blood vessels which are in charge of transporting nutrients to all the cells in our body, ensuring their survival. The most influenced organs are the lungs (C08) and the brain (C10).

Hemic and Lymphatic Diseases (C15): The lymphatic system is in charge of cleaning our body from waste, toxins and other unwanted materials like viruses and bacteria. The term hemic refers to the aggregates formed in this interaction with unwanted molecules, for example, blood proteins disorders and erythrocyte aggregation. One of the most common diseases is the Hodgkin lymphoma, and that is why it is related to the label number (C04).

Neonatal Diseases and Abnormalities (C16): The term neonatal refers to the first four weeks of a baby being out of the woman's womb. These abnormalities can be either for a genetic cause or a pregnancy complication, which is the reason why it is related to label 13.

Skin and Connective Tissue Diseases (C17): The skin constitutes the first layer of protection from environmental pathogens. It has three layers: the epidermis, the dermis (with connective tissue, sweat/hair glands) and the hypodermis. The most common related topic, neoplasms, might be because skin cancer is one of the most prevalent diseases in the whole world, and therefore, intensely studied.

Nutritional and Metabolic Diseases (C18): Nutrition is the science that studies how food influences our body (growth, reproduction, health and diseases). Metabolism englobes all the chemical processes that occur in a living being to maintain their vital functions. The function of the cardiovascular system is to distribute the nutrients and enable metabolic processes which explains its relation to label 10.

Endocrine Diseases (C19): The endocrine system is made up of a collection of glands that enables metabolic processes like growth, sexual function, sleep mood, among others. This also explains the strong relation to the 18th label.

Immunologic Diseases (C20): The immune system represents all the different defences that the body has against infectious agents. It englobes many of the labels seen before (skin, lymphatic system, bacteria). However, the most shared articles come from viruses and neoplasms, which suggests that a lot of the different texts are treating the HIV disease topic.

Disorders of Environmental Origin (C21): The environment represents the surrounding situations that a living being will always have to endure. It can be the cause of several diseases propitiated by mechanical damage or pathological agents like viruses, bacteria, pollution and pollen. It is strongly related to label 5 because mechanical damage affects directly to the musculoskeletal system.

Animal Diseases (C22): Rats are the most frequent animal in this section, mainly because the central part of research experiments use them as a baseline before moving on to clinical trials. As the number of documents related to animals is low, no definite conclusions could be made. However, rats and pigs are the main subjects for the study of cardiovascular diseases, which is where the most significant number of documents comes from (91 texts).

Pathological Conditions, Signs and Symptoms (C23): This last label is so broad that many of the documents of the dataset could be labelled here. Therefore we could not extract any clear relation between any of the tags seen in the corpora.

As the number of documents per label is unbalanced, some relations between tags that are very strong in the medical scenario could no be represented.

CHAPTER 4

Development of classification models

In this chapter section 4.1 describes all the preprocessing steps required to clean the text and prepare it for the ML procedures. Then section 4.2, explains the different feature extractors. Once the word embeddings are ready, section 4.3 defines the chosen estimators for the classification task. Additionally, section 4.4 introduces a new approach to deal with multi-labelled data. Finally, section 4.5 sums up all the different techniques presenting the latter approach, the ensembles.

The implementation of the models is the core of this project, which means that the results and conclusions entirely rely on how effective and precise these methods are. For the first step, we focus on preprocessing the corpora.

4.1 Preprocessing phase

This section aimed to extract the data and store it in a computable setup. The Ohsumed dataset saves its documents into a .txt format which is one of the most common ways to store information alongside .csv files. For the extraction, libraries such as NLTK, GSITK and Pandas supported the process (for more detailed information go to the appendix C.1).

The first step was to upload the data to jupyter's framework; to lessen the time spend submitting the extensive corpora, the solution was to compress the folders into a zip format so that it could be easily extracted once it is uploaded. The labels of the data worked as folders inside of which all the documents were carefully arranged.

Then, the objective was to transcribe all the data into a pandas dataframe for its management using python code. It is important to note that the name of the labels changed from a string format 'C01' into integer '1' because classifiers in sklearn compute this tasks based on integers tags. While doing this insertion of documents into the dataframe, each text excerpt was tokenised following these steps explained in section 2.2.2:

1. Tokenisation: Using NLTK to split the given text into individual tokens.
2. Stop words removal: Removing all the words contained on a stopword list.
3. Punctuation removal: Catches all the punctuation marks and delete them from the tokens' list.
4. Noise removal: there are always remaining terms that add noise to the dataframe and require its dismissal.

Testing the effectiveness of these two tokenisers was done by plotting a frequency list of the extracted terms and analysing which one showed better outcomes. As a result of this procedure, we discovered that a noisy term (–) appeared in a noticeable frequency. This noise happens because of the structure of the documents described in chapter 3, documents of 3-4KB from Ohsumed's dataset contained internal subdivisions given by titles. These titles, remarked by capital letters, were all followed by these noisy separators (–). Submitting an additional list of undesired terms was enough for the successful removal.

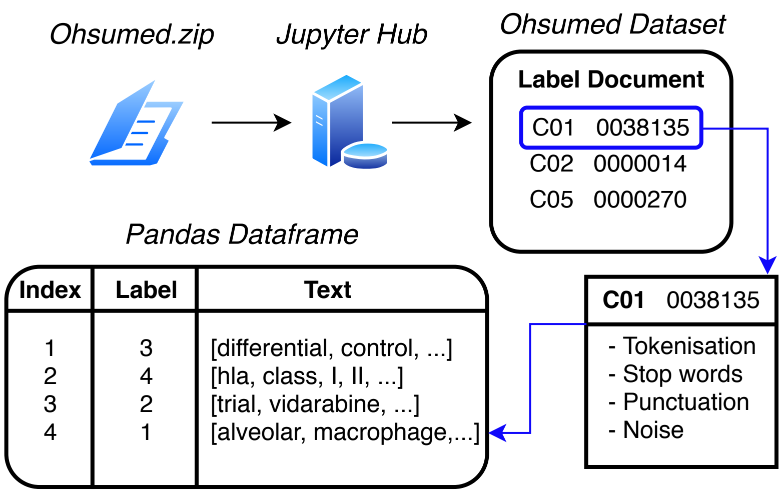

Figure 4.1: Preprocessing the Ohsumed's dataset

4.2 Feature extraction phase

Once the documents are ready, it is time for the application of ML techniques. The main difference between each of the pipelines is the feature extracting method; starting as a baseline with the tf-idf word embedding.

4.2.1 Tf-idf

At this point in the processing, we have the text documents tokenised and labelled with its correspondent class. These raw tokens are unprocessable by a computer, to solve this issue, tf-idf transforms them into vectors also called word embeddings. The fundamentals behind this feature extractor are already explained in section 2.2.4.3.

The methods provided by scikit learn [13] that allow the vectorisation of tokens are $fit()$ and $transform()$. The $fit()$ method generates learning parameters for the training data. Then $transform()$ converts the data based on the parameters previously defined by $fit()$. When talking about the generation of word embeddings, a combination of both of them called $fit-transform()$ is the fastest option. If we talk about classifiers, the name of these methods appears to be the same. However, in feature extraction, the whole vocabulary is fitted and then transformed. On the contrary, classifiers only use $fit()$ on the training data and $transform()$ for the test set.

The vectorisation of the documents using tf-idf has a varied set of parameters that condition the resulting embeddings. The most relevant ones tested in this project are:

1. $Ngram\_range$ represents the lower and upper boundary for the extraction of n-grams sets. By doing this, we can modify the method to extract unigrams (1,1), bigrams (2,2) or even a mixture of them (1,3).
2. $Max\_features$ selects the number of elements that the method should extract and stops when it reaches that limit.
3. Max_df sets the upper boundary on the term frequency, which means that this parameter is in charge of thresholding the most common words of the corpora.
4. Min_df sets the lower boundary on the term frequency, which means that this parameter is in charge of thresholding the least common tokens.

A sparse matrix represents the resulting features; which correspond to a low proportion of non-zero components. At this point, and with the features extracted, starts the process of splitting the data into training and test sets.

For this task, scikit-learn has already several tools that perform this division. In this project, the cross-validation method employed (C.2) to split the documents was the Stratified K-fold algorithm. This method divides the data in a stratified way, which means that the proportion of train/tests splits remains constant for every class in the dataset, avoiding imbalances and improving the robustness of the results.

Once divided, the next step is to prepare the classifiers for the learning process of the training set. Firstly, we initialise them to define their parameters, which modifies the way the algorithm learns the training features. In machine learning, to optimise time and supervise which variables are the ones that show better results on the dataset, a technique called hyperparameter tunning is commonly used.

In this project, the method selected for this task is the GridSearchCV() from scikit learn. With it, we can modify the classifier's parameters as well as the ones defined in the feature selection (ngram_range, max_frequency, max_df and min_df). Defining a pipeline is the way to concatenate these two processes into a bigger one enhancing results and saving time. The typical way of using this algorithm is to select another split of the data, called dev set, to tune the parameters correctly (no longer used thanks to CV techniques). This method also has a CV parameter that gives consistency to the values obtained. Finally, to check the results, we can look at the score (best_score_), the parameters (best_params_), and if we are using a varied range of classifiers, the estimator (best_estimator_).

4.2.2 Word2vec

Following the scheme of the tf-idf model, the first step is to describe the feature extractor used in word2vec. In this first case, the embedding model used to initialise the extractor was the Google news corpus, generated by a collection of 3 million words that resulted in a 300 dimension embedding of English word vectors. GSITK implements the gensim library and a tutorial that allows the initialisation of the embeddings without further complications. Once extracted, the only thing left to do is calling the fit_transform method to convert the documents into the final word embedding as an input for the corresponding estimators. The fundamentals of Word2Vec are extensively described in section 2.2.4.4.

4.2.3 Simon

The features extracted from this model add a new step to the one previously seen, which is the use of a lexicon. The method to generate it works by tokenising all the documents together and creating a frequency list like the Bag of Words algorithm.

Then, with the novel lexicon extracted from the dataset, it combines it with the embedding of the previous section 4.2.2 to build what we call the Simon model. Calling fit_transform after the initialisation of the extractor transforms the documents into its correspondent Simon features which have a dense representation due to its origins in word2vec.

4.3 Classifiers generation phase

The last part of a ML process is the use of classifiers to tune the word vectors into predictions. The quality of these predictions is what differentiates one classifier from another. However, in this section, we will only check the parameters used to initialise them, while the classifier's results will be discussed in the next chapter 5.

4.3.1 Tf-idf classifiers

The estimators selected to deal with the tf-idf features are Stochastic Gradient Descend (SGD), Multinomial Naive Bayes (MNB), C-Support Vector (SVC) and One vs Rest (OvR); all provided by the scikit learn package. At this point, we begin the description of the classifiers along with the relevant parameters used for each particular case:

4.3.1.1 Stochastic Gradient Descend (SGD)

Implements a bunch of linear classifiers with a stochastic gradient descend learning process. The algorithm can work with either dense or sparse features of floating-point values.

Parameters

1. *Loss* defines the loss function to use. For classification tasks, there are five possible options which are 'hinge', 'log', 'modified_huber', 'squared_hinge' and perceptron. Choosing hinge implements the loss function of a linear SVM; log incorporates the logistic regression one.
2. *Penalty* refers to the regularisation term used. There are three possible options 'l1', 'l2' and 'elasticnet'. 'L2' for example uses the regulariser for SVM models.
3. *Alpha* constant value that modifies the penalty and also, if selected, the computation of the learning rate.

4.3.1.2 Multinomial Naive Bayes (MNB)

Suitable for classification with discrete features extensively explained in section 2.3.1. In practice, also works with tf-idf fractional counts.

Parameter

- *Alpha* additive Laplace smoothing parameter. If set to zero, the classifier does not apply any smoothing.

4.3.1.3 C-Support Vector (SVC)

Consist of a classifier based on libsvm (C.1) that uses kernel functions for its deployment. Fitting beyond 10K samples may result in unpractical times and therefore, in those cases, is recommended changing to linear SVC.

Parameters

1. *C* is a regularisation parameter inversely proportional to the strength of the penalty.
2. *Kernel* select the type of kernel to use in the classifier. There are four main options which are: ‘rbf’, ‘linear’, ‘poly’ and ‘sigmoid’.
3. *Gamma* is the coefficient for the selected kernel function.

4.3.1.4 One vs Rest (OvR)

This particular classifier bases its mechanisms in an iterative process where it trains each class as a binary classification problem where the negative cases are the left out labels. It performs this procedure for the ‘X’ classes of the dataset and extracts unique features for each class.

Parameter

- *Estimator* selects the classifier to perform the binary classification task. The computing time of this algorithm is usually low because of the binary nature of the classifier. However, it is imperative to mention that the selected estimator and the number of labels are some variables that can also increase the computing time considerably.

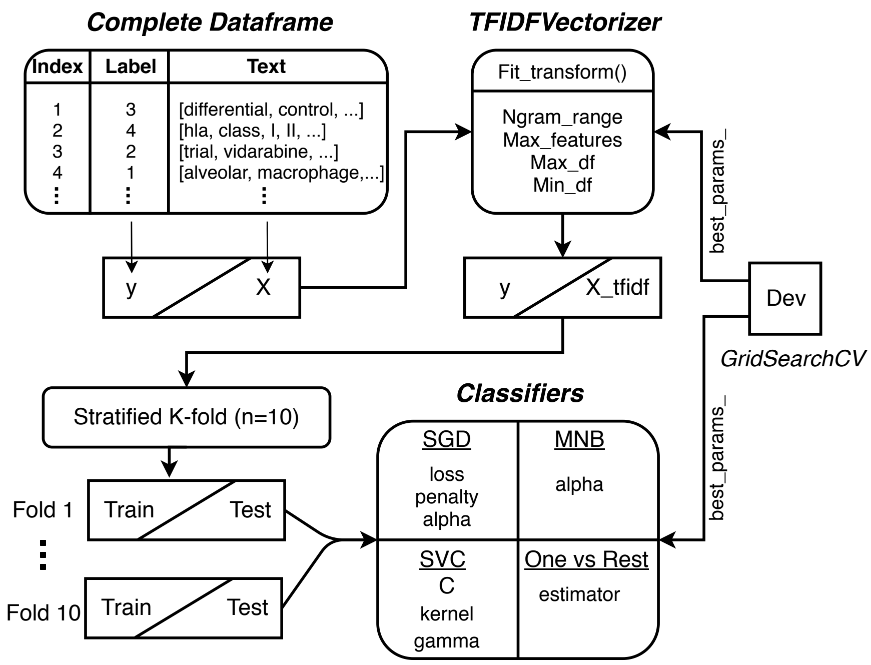

Figure 4.2: Tf-idf developed pipeline

4.3.2 Word2vec classifiers

The classifiers used in this project are practically the same for all cases, which is the reason why we mention them briefly in this section. The ones used in this case are Gaussian Naive Bayes (GNB), SGD, SVC and OvR.

The main difference from the previous model is that the features, represented by dense embeddings, have a lower dimensionality and values that are no longer strictly positive. MNB does not support negative values as input features; thus, GNB was used as an alternative.

4.3.3 Simon classifiers

With all the features ready for the classification, first, we split the data into test and training sets using the Stratified Kfold method. Afterwards, prepare all the target classifiers for the parameter tunning to finally asses the classification task by fitting on the training data and predicting from the test ones. The classifiers used in the Simon model are the GNB, SGD, SVC and OvR.

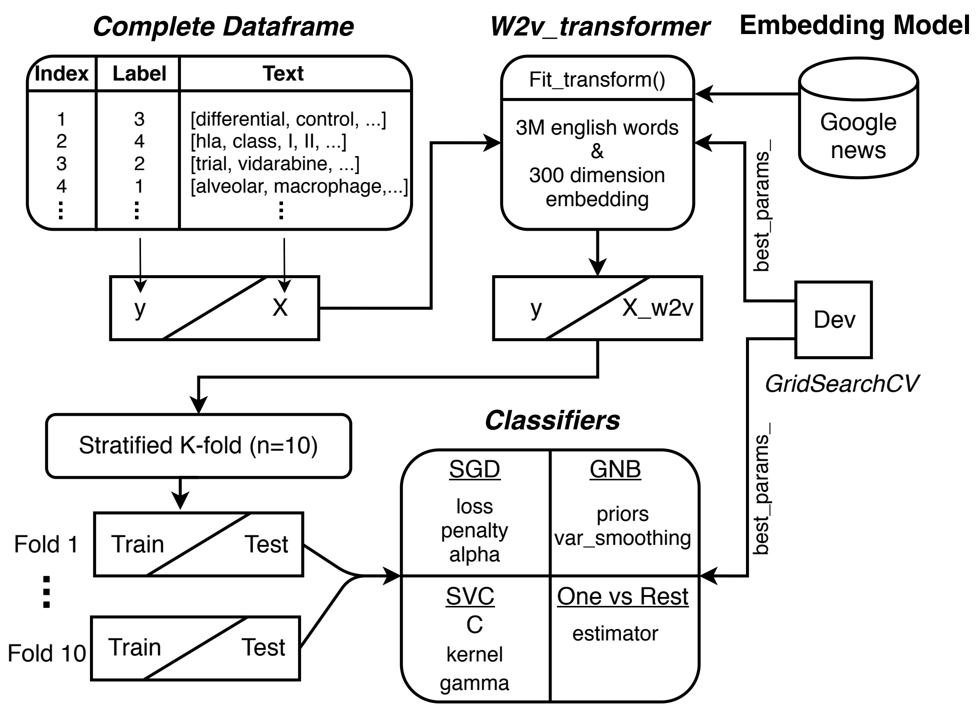

Figure 4.3: Word2Vec developed pipeline

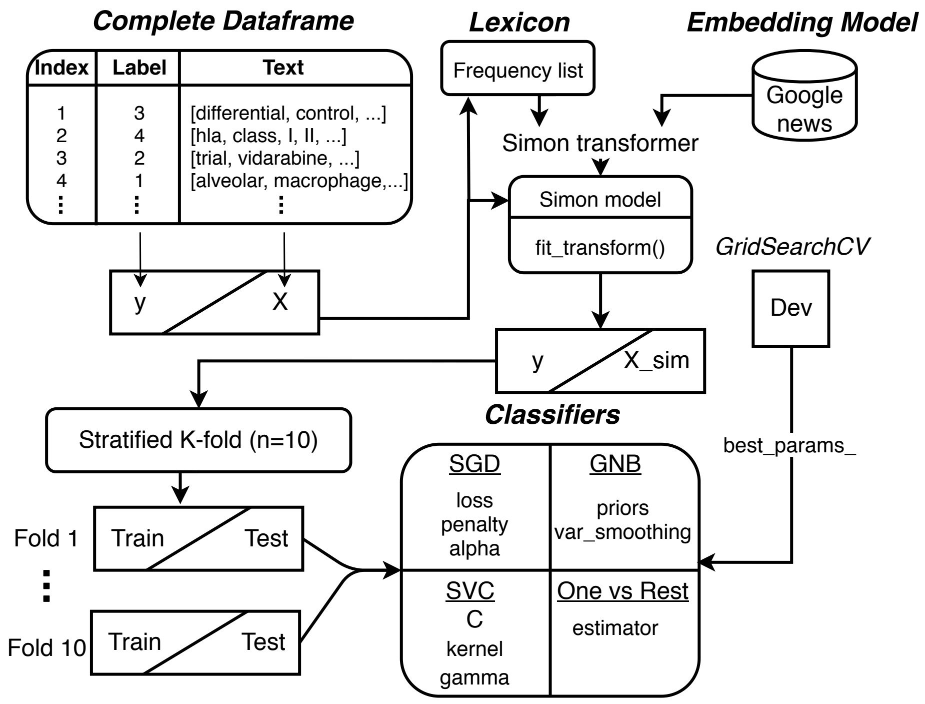

Figure 4.4: Simon developed pipeline

4.4 Multi-labelled model

Initially, documents in the Ohsumed dataset are single labelled (4.1). Multi-labelled classifiers from sklearn require transforming these classes into binary inputs. The process to enable the realisation of this model goes as follows:

1. Firstly, we initialise the label array, which must be of the same length as the number of classes (23).
2. Secondly, we transformed the single labelled documents of the dataset (with a total number of 18302) into their correspondent binary array.
3. Thirdly, we gathered all the repeated documents (with a total number of 16087) in the dataset and assigned them their multiple labels; the same way as we did in step 2.
4. To conclude, the two resulting datasets, one made up from the unique documents and the other with the multi labelled ones, are joined together into a final pandas dataframe of 34389 different texts.

Afterwards, in the process of transforming the features, as the text is not modified, it remains identical to the other models. Features extractors like tf-idf and word2vec work utterly transforming the data. However, the Simon feature extractor requires also the document labels to perform the fit_transform method, and since they are multi-labelled, the feature extractor does not support this conversion.

	Label before	**Label after**
Base array	[0]	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Unique documents	[5]	[0,0,0,0,**1**,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Multi-labelled documents	[3], [8], [16], [22]	[0,0,**1**,0,0,0,0,**1**,0,0,0,0,0,0,0,**1**,0,0,0,0,0,**1**,0]

Table 4.1: Multi-labelled transformation example

Finally, we split the data using the KFold CV method to train and predict from the classifiers (Stratified-Kfold does not support multilabel tags). The ones used in this model are the OvR (already explained in the previous models) and the MultiOutputClassifier.

4.4.1 Multioutput Classifier

This classifier achieves multi-labelled classifications by fitting one estimator per class. Its working is similar to the one expected from OnevsRest, extending the multi-labelled property to estimators that natively do not support it.

Parameter

- *Estimator* defines the classifier over which fitting and predicting probabilities.

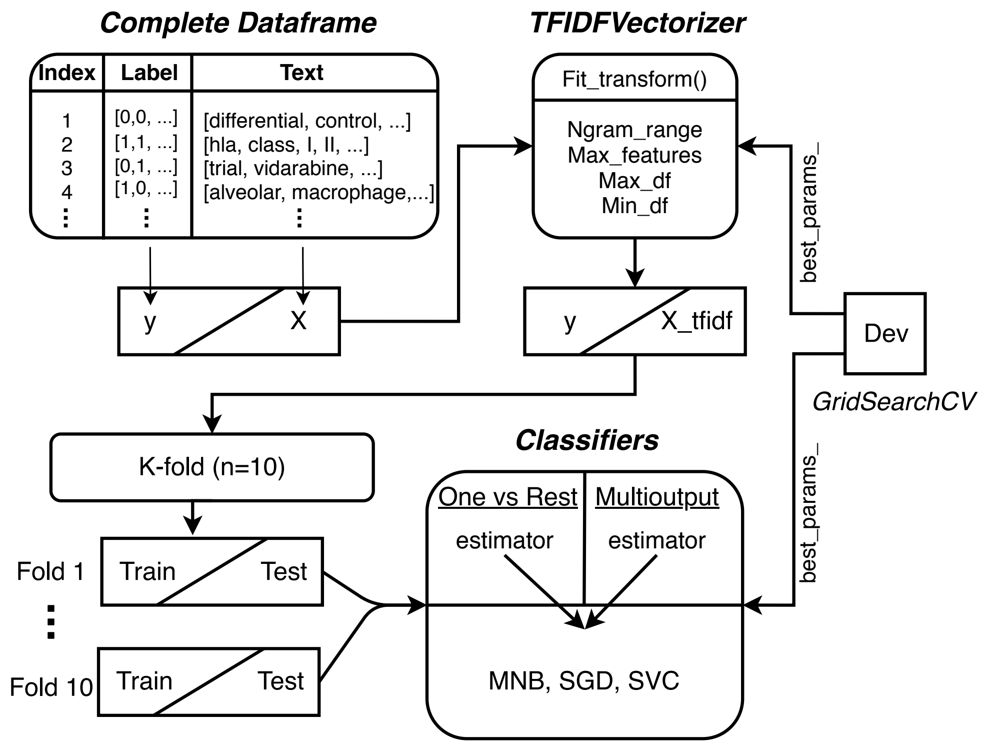

Figure 4.5: Multi-labelled developed pipeline with tf-idf example

4.5 The ensembles

The last effort to improve the results in this project relies on the use of ensemble techniques. As explained in section 2.3.4, ensembles are the last chance to improve the results in a machine learning project. However, they do not always necessarily achieve this. The basis of an ensemble is to use classifiers already defined and combine their strengths to cover their weaknesses. It is also essential to mention again that all the processes developed before the ensembles should be optimised in the best way possible (preprocessing, features and first layer classifiers).

The relevance of each process is also crucial to understand how to improve an ensemble. As we could learn from this project, using a single set of features with multiple classifiers and combining them at the end is not the best way to make improvements. The core of machine learning belongs to the features and how well they represent the data; classifiers are only a tool that helps us extract the meaning from them. Therefore, a strong ensemble is built up from a varied set of features, extracted by different classifiers and joined together into a final estimator.

Scikit learn has a broad range of ensembles for machine learning tasks. Usually, the ensemble that works well in classifications is the staking algorithm; nevertheless, in this section, we developed a model for each of the most renowned ensembles to see if they can improve the best score so far achieved.

4.5.1 Voting Classifier

If we compare this ensemble with stacking, we can see their strong similarities, since the differences between them only rely on how the final predictions are made. The use of unfitted estimators represent the input of this ensemble, and the final estimation is done by what we call voting.

Parameters

1. *Voting* decides the predictions based on a voting rule. If it is set to 'hard', the voting acts by assigning a majority to a prediction. Therefore, in the case of three classifiers, if two of them have the same output means that the prediction of the third classifier is discarded. In case of a tie (possible if using a pair number of estimators), the final prediction is decided on ascending sort order. On the contrary, using 'soft' voting, returns the final prediction taking the argmax of the sum of the predicted probabilities of each classifier.
2. *Weights* is the parameter that can modify the predicted value of the estimators by assigning them a weight (int or float). It can be used either in hard voting to weight occurrences of predictions or soft voting to tune probabilities before averaging.

4.5.2 Bagging

This ensemble takes subsets of the dataset for training X estimators. Section 2.3.4.1 explain all the different ways of implementing it.

Parameters

1. *Base_estimator* selects the classifier to use in the ensemble.
2. *N_estimators* creates X 'base_estimator' instances.
3. *Max_samples* defines the upper threshold number of samples that each estimator draws in its iteration.
4. *Max_features* defines the upper threshold number of features that each estimator draws.

4.5.3 Adaboost

It is one of the most common boosting methods in the literature explained in section 2.3.4.2. In a few words, Adaboost sequentially uses a set of X estimators to train the dataset.

Parameters

1. *Base_estimator* selects the classifier to iterate through the ensemble.
2. *N_estimators* creates X 'base_estimator' instances.
3. *Learning_rate* regularises the contribution of each classifier. There is a balance between the 'n_estimators' and the 'learning_rate'.

4.5.4 Stacking

This ensemble selects first layer classifiers (like in voting) and then combine their predictions into a final meta-classifier that makes another prediction over the dataset.

Parameters

1. *Estimator* is defined by a list of the estimators that we aim to use in the ensemble.
2. *Final_estimator* is the final classifier that performs the final estimation based on the ensemble from the 'estimator' list.

CHAPTER 5

Results

In this chapter, we are going to show the different outcomes obtained through the proposed classification pipelines. Section 5.1 analyses the structure of the word embeddings and focuses on recovering all the different metrics. Finally, section 5.2 sums up the best results so far achieved.

5.1 Analysing results

5.1.1 Resulting word embeddings

Extracting the text features was the first task of the process. In this project, three different feature extractors (tf-idf, word2vec and Simon) are the ones extensively studied. The following table 5.1 shows the specifications of each embedding.

We can see that according to theory, the generated tf-idf matrix has a sparse nature and, on the contrary, word2vec and Simon are represented by dense embeddings. The difference between the three models directly determines the performance of classifiers. Another feature not covered in the table is that the tf-idf values are strictly positive (as its basis relies on counting words). At the same time, the other two representations are characterised for

	Dimension	**Non-zero components**	**%**	**Matrix type**
Tf-idf	56984 x 7000	68 non zero over 7000 dimensions	1	Sparse
Word2vec	56984 x 300	299'99 non zero over 300 dimensions	99.99	Dense
Simon	56984 x 1116	1116 non zero over 1116 dimensions	100	Dense

Table 5.1: Characteristics of the multiclass embeddings

	Dimension	**Non-zero components**	**%**	**Matrix type**
Tf-idf	34389 x 7000	14 non zero over 7000 dimensions	0.2	Sparse
Word2vec	34389 x 300	299'99 non zero over 300 dimensions	99.99	Dense

Table 5.2: Characteristics of the multilabel embeddings

having either positive or negative values (dense matrix).

It is also important to note that features from the multiclass classification pipeline differ a little bit from the multi-labelled ones. These differences are because we have manually reduced the dimensionality of the problem task at hand and therefore, modified their features (table 5.2). In the tables is not appreciated but if we count the number of zeros, it results that the second word2vec embedding is denser than the first one that has 42 zeros over 2 from the second. The Simon embedding is not represented in the latter case since it does not support a multi-labelled input.

5.1.2 Multiclass results

Once we have our features saved and ready, we start the exploration of the different classifiers following chapter's 4 structure. Thereby, the first results come from the tf-idf embedding model, which uses the multiclass approach. The estimators used in this model are SGD, SVC, MNB and OvR. The metrics considered for the following sections are the recall, the precision and the F-score. Besides, the AUC and the ROC plot are also a great tool to compare classifiers. However, due to the dataset's characteristics and the extension of those results we had to move them to the appendix C.5

Before computing the results, we created a GridSearchCV instance to check which were the best parameters for each classification. The algorithm worked based on the F-score,

	Micro average			Macro average		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
MNB	0.40	0.40	0.38	0.42	0.28	0.29
SGD	0.36	0.40	0.37	0.33	0.34	0.33
SVC	0.31	0.31	0.31	0.29	0.26	0.27

Table 5.3: Comparing macro vs micro metrics

and the resulting outcome was:

- Feature extractor: max_df (0.7) , max_features (7000), ngram_range (1, 3).
- SGD: loss (log).
- MNB: alpha (0.6).
- SVC: kernel (linear), C (10).
- OvR: estimator(SVC(kernel (linear))).

The method, executed on each model, showed three different ways of initialising the features depending on the classifier. However, as the variations were minimal, we could generalise the extraction to reduce time and have a standard embedding model. The parameter *max_features* is the one that sets the dimensionality of the tf-idf matrix seen in table 5.1.

After computing the predictions for each classifier, we plotted a table to analyse the different metrics to decide which one will be the standard for the rest of the project. The first decision made was that our classification problem does not require a distinction between false positives and false negatives (both represent an equally lousy result). Therefore, instead of showing precision and recall, we use the F-score as the regular metric. The next point is to distinguish between macro and micro parameters. These two ways of calculating a metric are described in section 2.3. In the results, we can appreciate that micro average scores are always higher than the macro ones, leading to the fact that the classifiers are estimating the smaller labels of the dataset unsuccessfully (the Stratified K-fold algorith reduces these disparities).

In other words, in the next results, the metrics are going to use the F-score as a reference, with three different representations of it (accuracy, micro average and macro average). The last two ones will serve as a guide to check how well represented are all the labels on each classifier. Because of the document's extension we had to move the resulting tables to the appendix C.6. Therefore, before continuing reading the rest of the section we highly recommend going through those tables first (C.3).

As we can observe in table C.3, the best estimator is the OvR using the SVC as an underlying classifier. This score shows a noticeable improvement in the results (+18%) thanks to its algorithm, which treats the multiclass classifications labels as an independent binary classification problem. With this transformation, SVC can linearly separate the different tags and manages to rise the results up to **58%**. (notice that the standalone LinearSVC classifier only achieves a score of 31%).

The model that follows is the word2vec one. In this case, the classifiers used are SGD, GNB, SVC and OvR. The GridSearchCV is a time-consuming algorithm, and since the results seemed to remain on the same values, we decided not to implement it on every estimator. The parameters used to obtain those results in this case were:

- GNB: default.
- SGD: loss (modified_huber).
- SVC: kernel (linear), C (default).
- One vs Rest: estimator(SVC(kernel (linear))).

Table C.4 shows that the best results achieved using the Word2Vec method come from the LinearSVC classifier (**42%**). The similarity between this algorithm and the OvR one, (apart from the fact that they use almost the same estimator) might be because the Google Word2Vec embedding model performs poorly in the definition of medical concepts. In the micro and macro scores, we can still see the fact that the tags imbalances are affecting the classification.

The Simon model is the remaining pipeline that closes this first part of the results. The classifiers used are identical to the ones seen in word2vec (SGD, GNB, SVC and OvR). At first, the results of the classification with Simon did not reach a 20% score. Once we introduced the lexicon described in 4.2.3, the metrics started raising by changing the number of vocabulary words of the embedding up to 5000. However, if we compare the word2vec model with Simon, we can see that the latter does not surpass the previous scores defined by word2vec (**41%**). The intuition suggests that by adding the lexicon to the same word embedding, which is the one used in word2vec, should improve the outcome. Still, since the embedding is not entirely representative of medical diseases, this expected upgrade was not achieved. To conclude, the classifier that works best is the same than the one from word2vec, which is the SVC estimator with a linear kernel.

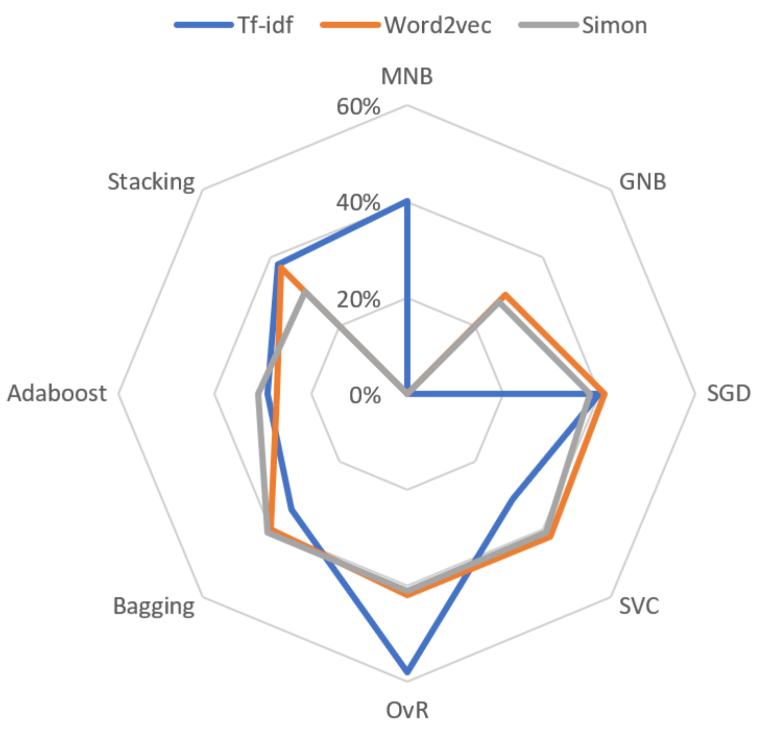

Figure 5.1: Evaluating the F-score on multiclass classifiers with a radial graph

This final figure of the multiclass models sums up perfectly all the results. With it, we can easily compare each of the embeddings. Firstly, we can notice that Word2vec and Simon are identical (since they were created from the same embedding model). Unfortunately, the new lexicon of Simon does not improve its results, as we would have expected in the first place. Secondly, we can appreciate that MNB classifier outperforms its homologue GNB in the dense representation by a 10% score. Finally, the decisive score of the tf-idf word vectors of 58% suggests that these features are better categorising diseases than the other two models, mostly because the embedding from Google was not the ideal source for the biomedical document classification.

5.1.3 Multi-labelled results

The first advantage of classifying with this modified dataframe is that the dimensionality of the embedding is reduced considerably. Another noticeable one is that thanks to this transformation, the documents' relations with the labels are more accurate. As described before, the embedding models used in this case are going to be tf-idf and word2vec. After their generation, One vs Rest and the multioutput classifier will be the ones predicting from the test set.

	Micro average			**Macro average**		
	Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
MNB	0.69	0.53	0.60	0.67	0.46	0.53
SGD	0.82	0.57	0.67	0.82	0.49	0.6
SVC	0.79	0.60	**0.68**	0.79	0.54	**0.63**

Table 5.4: OvR multi-labelled results

In this case, each classifier (MNB, SGD and SVC) appears in the estimator variable of the One vs Rest and Multi-output algorithm. After hyper tuning those parameters, we conclude that the best set up for each of them was:

- MNB: alpha (0.1).
- GNB: loss (modified_huber).
- SVC: kernel (linear), C (default).

Once the predictions finished, we can see an overall improvement in the scores confirming that the transformation of the data was a success. The multioutput classifier, described in section 4.4, appears to have the same algorithm than OvR and therefore, their results are not shown. In table 5.4, we can see that the tendency in the scores remains the same, meaning that the Linear SVC classifier is the best performer used so far. Another point to notice is the reduction of the differences between macro and micro scores which means that the estimators are classifying the classes with a better balance. Finally, we plot the standard table 5.5 used in this project where all the F-scores are gathered together. On it, we can observe that the SVC estimator has the best score so far, improving the one previously obtained with the tf-idf model (from 58% to a 68%).

	Macro F-score	**Micro F-score**	**Weighted F-score**
Multinomial Naive Bayes	0.53	0.60	0.59
Stochastic Gradient Descend	0.60	0.67	0.66
C-Support vector	0.63	0.68	**0.68**

Table 5.5: Multilabel OvR F-scores

The trials with the word2vec embedding model did not go as expected. In the multi-labelled task, the scores fell to 0 while 2 or 3 classes were correctly classified, showing strange overfitting. This atypical behaviour was present in almost every estimator parameter tested on the OvR. The GNB setting was able to attain an F-score of 0.26, which is a weak value if we compare it with the other models.

5.1.4 Ensembles results

The last attempt to improve the score in this project relies on the use of ensemble techniques. In the process of building those ensembles, two different approaches were planned. The first one tried to combine the classifiers of the different feature extractors into a stacking algorithm and also working with Adaboost and Bagging. The second method planned to combine these features into the union of the three different extractors (Tf-idf, Word2vec and Simon).

Starting with the first approach, the attempts to improve the results were unsuccessful. The three different sets of features performed equally or slightly worse than the standalone classifiers. The ensembles tested, in this case, were Bagging, Adaboost and Stacking. The computing time of Simon was considerably high if we compare it with the other embeddings, which might be because of the dimensionality of such extractor. The resulting table 5.5 shows the different results obtained.

	Tf-idf	**Word2vec**	**Simon**
Bagging	0.34	0.40	**0.41**
Adaboost	0.29	0.27	0.31
Stacking	0.38	0.37	0.30

Table 5.6: Ensembles F-scores

The second trial, whose objective was joining the features, was also unsuccessful. We think that the main reason for this is because of the different nature of the embeddings, one sparse and two dense ones. Converting them into a standard model (sparse or dense) was the only way to make a feature union and test the ensembles. Without that union, sklearn is not able to initialise the classifiers with the given input.

The most promising ensemble was the one built up from the results of the multilabel classifiers. One of the reasons was that some of them performed great in recall and other exceptionally well in precision. Therefore, we thought that combining them using a stacking classifier would improve the results. Unfortunately, the sklearn ensemble classifiers do not support a multilabel input so we could not test this model.

5.2 Summary

This section summarises the results obtained in the whole project so that it is more comfortable for the reader to compare the varied set of models that were developed. We will only show the F-score for readability purposes.

MULTICLASS	**Tf-idf**	**Word2vec**	**Simon**	**MULTI-LABELLED**	**Tf-idf**	**Word2vec**	**Simon**
MNB	0.40	-	-	**MNB**	0.59	-	-
GNB	-	0.29	0.27	**GNB**	-	**0.26**	-
SGD	0.40	0.41	0.38	**SGD**	0.66	0.01	-
SVC	0.31	0.42	0.41	**SVC**	**0.68**	0.01	-
OvR	**0.58**	**0.42**	**0.41**	**OvR***	-	-	-
Bagging	0.34	0.40	0.41	**Bagging**	-	-	-
Adaboost	0.29	0.27	0.31	**Adaboost**	-	-	-
Stacking	0.38	0.37	0.30	**Stacking**	-	-	-

Table 5.7: Results' summary

All the algorithms implemented in the multi-labelled half use OvR (*) as the base classifier. The estimator parameter is the variable that modified the scores, and therefore, is the one shown in the table. We can also see that GNB only works with dense features and MNB with sparse ones (only allows values strictly positive). To conclude, we can see that the ensembles did not support the transformation made in the multi-labelled dataframe as it was spotted in the previous section.

CHAPTER 6

Conclusions and future work

In this project, we have tested the computers' ability to classify biomedical documents. Thanks to ML techniques and decent computer capacity, we have been able to obtain satisfying results compared to the baseline [14]. In section 6.1, we will draw the different conclusions at which we have arrived. Finally, in section 6.2, we propose briefly possible future lines of development that can arise from this work.

6.1 Conclusions

The first thing noted when working with biomedical data are the complex relations between concepts that can be present in a single draft. In this particular project, we are dealing with diseases which have associated a wide distribution of treatments, vulnerable areas, side-effects, complications and so on. Therefore, solving this with a classification model that can depict all these relations can be tough work.

Another important thing learned from this practice is that before going straight to work with the corpora, we have to explore the dataset in depth, which means not only reading the description available in the site but also exploring the different classes making sure there are no mistakes on them. It is impossible to go through all of it one by one. However, the procedure taken in this project allowed us to confirm that the data was truthful and had no mistakes on it. The only strange thing we found was that there were three documents repeated with different ids which meant that our dataset had three random repeated texts on it.

This study also helped us realising that several documents on the Ohsumed dataset were multi-labelled, which made us divide our models into two different approaches a multiclass and a multi-labelled one. The intuition suggested that the second model would be better since it was reduced in dimensionality and the relations between labels were also implemented. After trying both models, we reassured this reasoning by getting a higher score in the multi-labelled pipeline along with a considerable reduction in computing time.

The final score achieved was 68% (F-score), using a multi-labelled dataset with a Linear SVC classifier. If we consider three, the average number of labels of the documents, the probability of a random classifier to get it right is about 13%, which means that our score outperforms this classifier by five times. Lastly, checking the score achieved in the baseline [14] document confirms that our procedure was correct (67%).

6.2 Future work

The prospect of the biomedical field is up-and-coming. New advances in medicine and technology suggest a stronger interaction between these two fields in the future. In particular, ML and NLP will play an essential role in the development of Artificial Intelligence (AI) software.

One example could be the development of bots that help during surgery by listening to the voice commands of the surgeon to store annotations, register material and make valuable comments. One day robots might be able to emulate doctors' movements while doing surgery all thanks to ML techniques, improving precision and efficiency. Another one could be a chatbot that assists a patient during their residence in the hospital answering doubts, solving problems and calling the physicians if required.

More focused on ML, thanks to new nano-implants and devices (Internet of Things (IoT), 5G, etc.) the development of Support Decision System (SDS) can help to monitor cases even outside of the hospital. For example, developing a portable Electroencephalography (EEG) device could help to diagnose the likelihood of an epileptic patient for having a seizure by training with the enormous amount of data that we have about this disease. The possibilities are countless; all we need is imagination and hard work to make them happen.

APPENDIX A

Impact of this Project

The different techniques applied in this project play an essential role in the construction of SDS. There are visible interests placed on the creation of algorithms that can classify data in a fast and organised way. Previously, workers had to sort all this data manually; with the development of these systems, the time spent on those matters was reduced, releasing the practitioners to spend their time doing other tasks. Distinctly, in the biomedical world, this data managing tools can help in many different fields, for instance, imaging, genetics, research and diagnosis. Robust decision systems can improve immensely the value provided to physicians and as final beneficiaries, to patients. However, errors in those applications could lead to misinformation and cause grave consequences. In this appendix, we study those different impacts in terms of society A.1, economy A.2 and environment A.3. Lastly, we discuss the different ethical implications in section A.4.

A.1 Social impact

The introduction of SDS into a company has severe implications concerning their workers. The first thing to note is that every change in the daily routine of a physician supposes an extra amount of hours that the worker will have to spend learning how to use the program. Not all the operators accept those changes positively because it will suppose a variation in their routine that has not been altered for years. Therefore, the implementation can be a slow process that produces extra stress in every practitioner but more intensely in those reluctant to such changes.

The inability to learn new technologies or the amount of time that requires apprehending them also plays an essential role in such matters. For example, in hospitals, the Electronic Health Record (EHR) which represented a drastic and extremely positive change in how hospitals worked, was first developed in 1965. Still, it was not until the 2000s when they started implementing them, and even nowadays, some doctors keep using paper as the tool for storing patient's information.

Data is an essential requisite in these projects, and taking into account that we are talking about medical information, the vast majority of it comes from patients. A physician needs to recover tons of personal information from subjects to help them overcome their diseases. The General Data Protection Regulation (GDPR) regulates the way this information is stored, helping to keep this knowledge unavailable to external sources. If we want to work with this information, it implies having it pseudo-anonymised so that no one knows from which person is coming. A leak on this data can have several implications to the patient affected and also several punishments to the institution that facilitate it.

A.2 Economic impact

Developing these systems can be an expensive task. Firstly, we have to make sure that the data quality and quantity are sufficient for the duty, which may require hiring an expert in healthcare data. Secondly, we need software engineers and data scientists to develop the operating program. Thirdly, we expect someone to teach professionals about how to use our system, to reduce costs, it could be a developers' extra task. Finally, those practitioners need time to learn the new software, taking out hours of their daily schedule could lead to a deterioration of their previous assignments. A plausible solution to this drawback could be augmenting their salary.

If we are talking about the benefits, the first clear one is that an increase in the efficiency means a consequent improvement in the capacity of the company which can be reflected in attending more patients, developing clinical notes faster and facilitate work to researchers. Increasing productivity directly affects earnings. However, for a project to be viable, those earnings should be significantly higher than the costs described in the preceding paragraph.

A.3 Environmental impact

The first environmental consequence that our systems may produce is an increase in electricity consumption for having a computer or a server operative. This consumption induces an increase in the demand for energy sources, both non-renewable and renewable, affecting the environment negatively.

Either if we use computers or severs, those machines have built-in obsolescence which means that at the time they stop being useful they need to be dumped. Several components of PC's such as batteries, the power source, plastic covers, cables and internal circuits have contaminant products (selenium, cadmium, chrome, cobalt and mercury). Recycling them should be a widespread practice to reduce the impact that these devices may cause to nature.

Finally, having an excellent SDS, can reduce the number of tests that a patient has to take before getting a diagnosis which can be harmful to the patient (chemotherapy, X-rays, surgeries). Also, if we look at this problem through the hospital perspective, reducing the number of trials implies a reduction in the expenditure of sanitary materials, reducing costs and contamination.

A.4 Ethical impact

In the end, the objective of ML is outperforming workers thanks to computational skills. This first reason makes us think that one of the first ethical implications while developing machine learning systems is that we will be removing someone's job. However, SDS, as its name says, is designed to support the professionals as a helping tool, not as a replacement. Also, new technologies create new jobs giving workers the chance to adapt to changes by acquiring new skills.

Another ethical implication can be related to the medical GDPR. Collecting data from patients requires a sophisticated system that ensures that data is protected and encrypted so that no one else has access to it. Extracting this information for ML is a difficult task with many requirements. In hospitals, for example, data is stored in servers inside the building so that any outsiders have access to them. We have to be physically in the hospital to work with the information and also extract it anonymised in the best way possible.

Finally, as we are developing a classifier that may support decision systems in medicine, there are ethical implications related to the production of mistaken predictions. These errors can have enormous consequences on patients health; therefore, to reduce this impact, we need to make sure that those systems are reliable. However, it is essential to remember that those algorithms are prepared to help professionals as tools. Depending on the situation, it is always the final choice of the practitioner to follow or not those decisions.

APPENDIX B

Cost of the system

If we want to sell the software developed in this project, first we need to improve the results. Doing so could be achievable by using better feature extractors as well as a more significant amount of data. In this chapter, we will discuss the different costs associated with the development of a SDS software that will use the models designed in this work. Section B.1 talks about the price of the physical infrastructure necessary to compile our software. Then, section B.2 explains the different professionals that will be needed for the creation of the software. Finally, sections B.3 and B.4 talks about the licenses and taxes required for the commercialisation of our product.

B.1 Physical resources

The resources needed to ensure the correct functioning of the proposed software are strongly related to the amount of data used. Our dataset had a storage capacity of 66.3 MB. A computer or server with decent requirements would be enough for the fulfilment of this task. However, as we said before, if we want to commercialise a product of such kind we will need much more data to make a robust and truthful model. With the dataset used, the estimated requirements are:

- **Hard disk:** 500GB
- **RAM:** 16GB (2x8GB)
- **CPU:** Intel i5 9700K 3.7GHz

The cost of a machine with these characteristics could be around 1100€, considering the GPU and other components. These features are only symbolic since, as we have said before, if we want to commercialise it, we will need more data and larger computing capacity to develop such software.

B.2 Human resources

To develop an application of this kind, we estimate that hiring one software engineer and a data scientist specialised in healthcare would be enough for its completion.

We decided to divide the project into two parts within two years. In the first one, the data scientist will be in charge of creating a dataset with a consistent amount of information related to human diseases. Once it is ready (estimated eight months), he will create the machine learning models as we did in this project. Finally, in the year left, the software engineer will develop an application that uses the learned model and test it in real case scenarios. As a final result, we expect the commercialisation of the software to biomedical companies and hospitals. If we suppose a monthly salary of 1200€ to the data scientist and 1500€ to the software engineer (both in brute), the total cost of the project will be around 32000€.

B.3 Licenses

We are going to divide the licenses of the project into three sections:

1. The first one is related to the obtention of medical datasets. There are many different sources of free data like PubMed, data.gov, medicare, etc. However, many others require a subscription or a license to be accessible.
2. The second part, which is the one related to this project, does not require license since all the software used is of free access (Python and Jupyter Notebook). PyCharm can also be an interesting tool to work with Python code since it has an excellent debugger. The drawback is that the professional version that includes compatibility with Jupyter Notebook requires a license.
3. Finally, we did not include any particular program in the application's construction

since we assumed that it should be able to be fulfilled with a free access software.

B.4 Taxes

If we consider selling the product as downloadable software, we will have to apply 21% taxes supposing that our company is established in Spain. Additionally, the acquisition of an application represents a service which means that selling it to foreign countries is not considered exportation. Besides the seller, we have to consider the residence of the vendor, if it is from the European Union (EU) the tax remains the same. However, if it is from a foreign country selling this software will not have additional fees.

APPENDIX C

Libraries and others

This appendix covers some important concepts and results that were not explained in the main document. First, in section C.1, we talk about the different tools and libraries utilised. Then sections C.2, C.3, C.4, C.5 and C.6 are used to complete the content from the project's chapters.

C.1 Tools and libraries

This section explains the different software applications used in the development of this work. We start by describing the Jupyter's notebook framework.

Jupyter notebook Previously, scientific articles lacked reproducibility in terms of programming. Publications were full of prose, without specific tools to show the code that supported those conclusions. From this need raises the project Jupyter and with it the development of Jupyter notebooks. This tool covers both the prose and the code section, being able to be specific and facilitating reproducibility.

The code is organised in cells that can be individually modified and compiled. The result of each cell appears below as a part of the document, improving its readiness. Additionally, Jupyter notebook not only covers markdown code but also ways for representing mathematical equations and interactive graphs. Finally, markdown cells and the code ones can store comments which makes a notebook rich in specifications about either the code or the purpose of the writing.

These notebooks support multiple programming languages, which communicate using back-end programs that have a standard protocol. The first programming code used with this notebook was from a Python kernel. The access to the software can be done through a web browser; however, nowadays, some applications like Anaconda support it. The documents, stored with a JavaScript Object Notation (JSON) format and an extension 'ipynb' can be published via Github recovering the programming environment and facilitating the execution to other researchers.[15]

Natural Language Toolkit (NLTK) It is a source of open program modules developed to reduce the learning curve of NLP techniques [16]. Usually working with unstructured text requires a varied range of programs depending on the task at hand. The programming language that covered the requirements to support this library was Python. Its development aimed at:

- Ease of use: the learning is supposed to be centred in NLP itself and not in how to use the toolkit.
- Extensibility: the possibility of extending the toolkit to new tools.
- Documentation: due to the learning purpose, all the modules must be carefully documented.
- Modularity: another feature that helps to lessen the learning curve, allowing the students to know how to use the toolkit without further knowledge on how these modules interact.

With that said, the six main modules that support the toolkit are:

1. Parsing modules: produces trees that represent the structure of texts.
2. Tagging modules: Augments each token with supplementary information (PoS).
3. Finite State Automata: Interface for creating automata from RE.
4. Type checking: To facilitate programming (can be turned off)
5. Visualisation: graphic interfaces for the representation and manipulation of data structures
6. Text classification: is the module that allows the classification of texts into categories.

NumPy In Python, NumPy arrays are the way of representing numerical data. Using them facilitates the computation of extensive datasets by optimising the times of 'for' loops significantly. The three fundamental techniques that achieve this are the vectorising calculations, the avoidance of copying data in memory and the minimisation of operation counts. NumPy matrices also called ndarrays, uses the following attributes [17]:

- Data pointer: the memory address of the first byte in the array.
- Data type description: defines the nature of the element stored in the array.
- Shape: stores the dimension of the array in the form of coordinates.
- Stride: the number of bytes skipped to proceed to the next element.
- Flags: define if we are allowed to modify the array or not.

This efficiency, adopted by the Pandas library allows computing effortlessly operations in large datasets. Usually 'for' loops represent the bootle neck in the computation time, managing correctly with NumPy arrays avoids this problem.

Pandas It is a Python library used in many different fields, preferably on structured sets. It has all the means to perform an exhaustive analysis on data by performing manipulations and queries. It also has the tools to allocate the dataframes by converting .csv files into the notebook and vice-versa. To import the dataset, we have to create a Dataframe object on which we will initialise our required variables. Once it is uploaded, the data can be reshaped and transformed for the required task (ML in our case). [18]

In our particular situation, we transcribed our unstructured data into a dataframe by tokenising the text fragments on each document. In this case, the library was useful to visualise the data and spot some dataset's features that were not specified in the description like the multi-labelled origin of the data. Afterwards, thanks to ndarrays we were able to transform the dataset without complications allocating the new dataframe in a '.pkl' format.

The operations are not very efficient if we are using Dataframes or Series. Therefore, the best way to avoid this inconvenient is to work directly with the ndarrays by extracting the data with the attribute 'values'. Statistical functions like 'mean' and 'std' have been overridden to avoid missing values which is a common feature of many datasets.

Scikit-learn It is a Python module that implements a varied range of machine learning algorithms (supervised and unsupervised). It focuses on bringing these methods to a high-level language so that it is easily understandable by non-specialists; putting a strong emphasis on clear and complete documentation.

Its code provides a reliable implementation with a consistent set of parameters and variables which reduces the learning curve of the library. It also shows strict adherence to the Python coding guidelines and the NumPy style documentation. The main components of the library are estimators, transformers, the cross-validation iterator and a varied range of scoring algorithms. All of it made this an ideal tool not only for research but also as building blocks for a diverse range of approaches like medical imaging. [19]

Libsvm It is the library that holds SVM algorithms. As a supervised classifier, its typical use case relies on two different steps, training and testing the data. The package is structured as follows: [20]

- The main directory with all the testing and training algorithms.
- The tool subdirectory to check data format and select SVM parameters.
- Others contain prebuild binary files and interfaces for different languages.

It implements the C Support Vector classification as one of its algorithms for supervised learning. It works by varying the C parameter, which is called the regularisation term. This element defines the interval by which two models are divided. Ideally, we look for a large margin and a low misclassification rate. These two concepts are a little bit contradictory because augmenting the C parameter increases this error rate. However, that is not entirely true because misclassifications on the train set do not correlate to others in the test set. Therefore, to set the C parameter, the recommendation is to use the GridSearch to analyse the best resulting score directly. [21]

C.2 Cross Validation (CV)

The objective of the CV is to avoid predicting something that was previously learned by the computer. Training a model with the gross data is a common mistake on someone who starts studying ML. There are many different ways of overfitting, some of them, more noticeable than others. The one described above is the first typical example, computing the score on this overfitted model, will show an accuracy close to 100%.

The initial idea to solve this problem is to divide the data into what we call a test and a training split. Doing so allows the computer to prove its model with unseen data. Unfortunately, this method lacks generalisation performance because the only representative of the model is a specific split. CV allow us to generalise our models and make them more representative of our data.

Another overfitting problem occurs if we use GridSearch on the training data. This overfitting is harder to notice because it does not enhance the results drastically. Optimising the training split for a given test set will improve the results of the classifier, giving a false sensation of improvement. A solution to this kind of overfitting is to create a validation set to evaluate the GridSearch. However, this procedure implies an extra partitioning of the data which worsens the variety in the training and testing. Thanks to CV, this additional set is no longer needed.

There are many different types of CV iterators, some of the most common ones are:

- **CV for independent and ideantically distributed(i.i.d.) data**
  This approach assumes that all samples have been extracted from the same generative model. In real life, this approach is not very robust because data can be time dependant or modelled by a generative model with a grouped structure. Some of the algorithms used for this group are K-fold, Leave One Out (LOO) and Leave P Out (LPO).
- **CV with stratification**
  If the distribution of tags is unbalanced, this iterator creates each fold based on percentages assuring that the relative frequency between labels remains constant. Stratified K-fold is the most renowned and the one selected for this project due to the characteristics of the dataset.
- **CV for grouped data**
  If the distribution of tags is unbalanced, this iterator creates each fold based on percentages assuring that the relative frequency between labels remains constant. Stratified K-fold is the most renowned and the one selected for this project due to the characteristics of the dataset.
- **Time series split**
  This particular iterator can be used in a data model that has a constant flow of data. Therefore, what it does is putting the new inferences in the test set and trains the model with the previous samples.

C.3 Ohsumed dataset labels

The complete table with the labels, the disease category names and the number of documents is fully developed in this appendix, as the graphic shown in chapter 3 did not show the disease categories names associated with each class.

Label	**Disease description**	**N^o of documents**
C01	Bacterial Infections and Mycoses	2540
C02	Virus Diseases	1171
C03	Parasitic Diseases	427
C04	Neoplasms	6327
C05	Musculoskeletal Diseases	1678
C06	Digestive System Diseases	2989
C07	Stomatognathic Diseases	526
C08	Respiratory Tract Diseases	2589
C09	Otorhinolaryngologic Diseases	715
C10	Nervous System Diseases	3851
C11	Eye Diseases	998
C12	Urologic and Male Genital Diseases	2518
C13	Female Genital Diseases and Pregnancy	1623
C14	Cardiovascular Diseases	6102
C15	Hemic and Lymphatic Diseases	1277
C16	Neonatal Diseases and Abnormalities	1086
C17	Skin and Connective Tissue Diseases	1617
C18	Nutritional and Metabolic Diseases	1919
C19	Endocrine Diseases	865
C20	Immunologic Diseases	3116
C21	Disorders of Environmental Origin	2933
C22	Animal Diseases	506
C23	Pathological Conditions, Signs and Symptoms	9611

Table C.1: Ohsumed dataset documents per label

C.4 Complete confussion matrix

In this section, we can see the complete confusion matrix from table 3.2. As its 23x23 size was not pleasant to show in the main document, we decided to select the top 10 labels to enhance the matrix's display.

Labels	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
1	**2540**	188	30	150	129	308	59	304	60	160	85	156	131	165	123	51	89	44	31	281	155	73	662
2	188	**1171**	29	110	22	108	18	124	18	91	35	30	76	29	43	17	37	13	5	401	57	28	224
3	30	29	**427**	10	2	43	0	16	2	43	24	9	15	13	11	2	9	5	0	45	8	22	79
4	150	110	10	**6327**	158	415	139	457	191	361	106	547	296	208	208	98	192	128	194	362	120	32	1223
5	129	22	2	158	**1678**	51	43	43	22	211	25	47	16	83	74	33	170	69	41	79	212	23	445
6	308	108	43	415	51	**2989**	22	119	4	123	14	124	71	141	80	46	75	127	32	160	135	61	1186
7	59	18	0	139	43	22	**526**	26	14	32	5	5	5	15	16	16	47	18	4	36	23	3	144
8	304	124	16	457	43	119	26	**2589**	107	131	26	87	43	266	115	87	55	58	21	298	147	38	749
9	60	18	2	191	22	4	14	107	**715**	60	11	5	5	9	12	11	36	9	9	42	22	13	244
10	160	91	43	361	211	123	32	131	60	**3851**	115	81	85	401	102	87	79	107	70	175	302	35	1200
11	85	35	24	106	25	14	5	26	11	115	**998**	10	7	44	22	23	49	34	25	85	52	12	267
12	156	30	9	547	47	124	5	87	5	81	10	**2518**	130	234	94	36	44	145	99	89	72	29	537
13	131	76	15	296	16	71	5	43	5	85	7	130	**1623**	132	68	119	55	31	23	76	42	13	374
14	165	29	13	208	83	141	15	266	9	401	44	234	132	**6102**	132	132	115	285	132	106	185	91	1765
15	123	43	11	208	74	80	16	115	12	102	22	94	68	132	**1277**	33	83	25	9	152	29	10	401
16	51	17	2	98	33	46	16	87	11	87	23	36	119	132	33	**1086**	31	30	13	33	17	5	281
17	89	37	9	192	170	75	47	55	36	79	49	44	55	115	83	31	**1617**	36	28	156	62	13	358
18	44	13	5	128	69	127	18	58	9	107	34	145	31	285	25	30	36	**1919**	209	55	48	29	301
19	31	5	0	194	41	32	4	21	9	70	25	99	23	132	9	13	28	209	**865**	43	13	7	154
20	281	401	45	362	79	160	36	298	42	175	85	89	76	106	152	33	156	55	43	**3116**	151	34	480
21	155	57	8	120	212	135	23	147	22	302	52	72	42	185	29	17	62	48	13	151	**2933**	35	629
22	73	28	22	32	23	61	3	38	13	35	12	29	13	91	10	5	13	29	7	34	35	**506**	113
23	662	224	79	1223	445	1186	144	749	244	1200	267	537	374	1765	401	281	358	301	154	480	629	113	**9611**

Table C.2: Complete confusion matrix of the label's relationships

C.5 Computation of the Area Under the Curve (AUC) and the Receiver Operating Characteristics (ROC) curves

ROC curves are an excellent tool to represent the performance of classification models. They have a 2-D graphic representation where the X-axis corresponds to the False Positive Rate (FPR) and the Y-axis to the True Positive Rate (TPR).

$$Sensitivity = TPR = \frac{TP}{FP + FN} \tag{C.1}$$

$$1 - Specificity = FPR = \frac{FP}{FP + TN} \tag{C.2}$$

The plotted curves show the effect on varying a threshold that goes from 100% to 0% FPR. If the model is perfect, which means that it reaches an FPR of 0% with a TPR of a 100%, we will see a squared function that starts at the origin. However, if we cannot discern between the True Positives and False Positives at all, the plot will be a linear function of $y = x$.

To compare different ROC curves, the standard thing to do is computing the AUC, which means calculating the area under the curve from 0 to 1. In the examples described above, we can see that the area of a square function will represent a perfect AUC with value one and in the worst-case scenario, the result will be the half of it (0.5). Therefore, calculating the AUC is interesting for choosing which categorisation method is better for a given task by measuring which one has a higher value. In a multilabel/multiclass classification, ROC curves are computed based on the number of labels being tested in the dataset, in our case 23.

Figures C.1 and C.2 demonstrates that the multilabel problem is the right choice as the AUC is substantially better on both SGD and SVC (MNB could not be tested because the classifier does not generate a decision function). However, due to the classes imbalances of the dataset, the AUC and the ROC curves are not the best methods to measure the effectiveness of a classifier. A better alternative is using precision and recall (or a combination of both: the F-score), which is the reason why we put these results on the appendix.

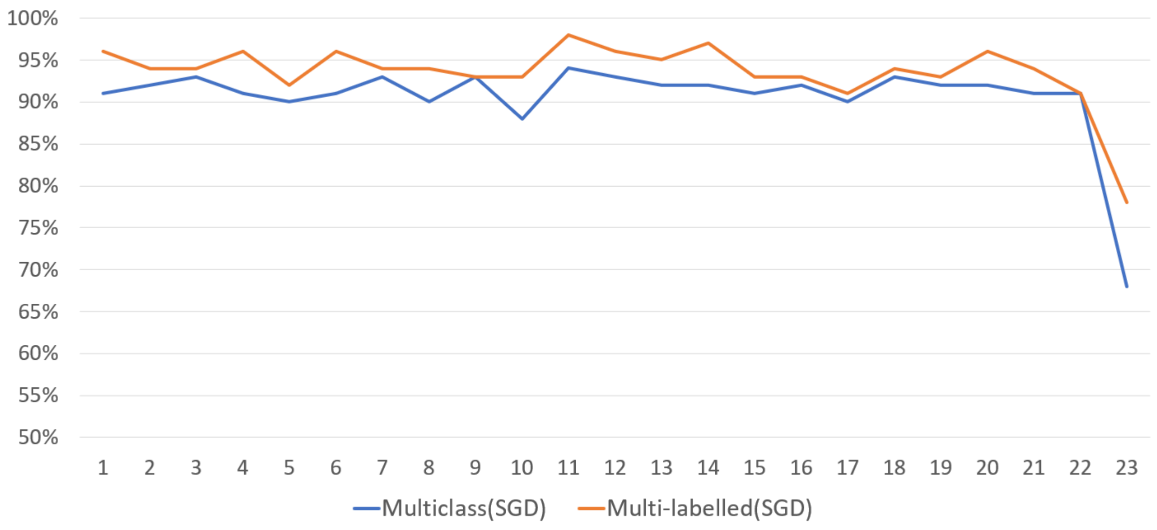

Figure C.1: AUC comparison of the SGD classifier

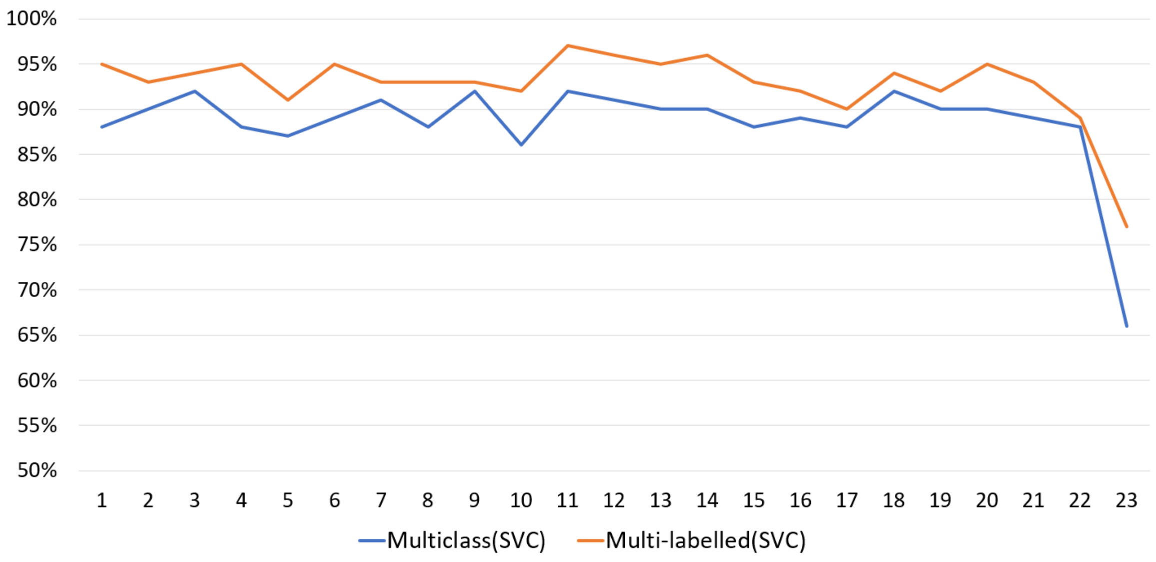

Figure C.2: AUC comparison of the SVC classifier

It is impossible to plot all the figures due to the high number of resulting graphs. Therefore, as many of the ROC curves are very similar we will try to show the ones more characteristic of our best classifier: Multi-labelled OvR LinearSVC().

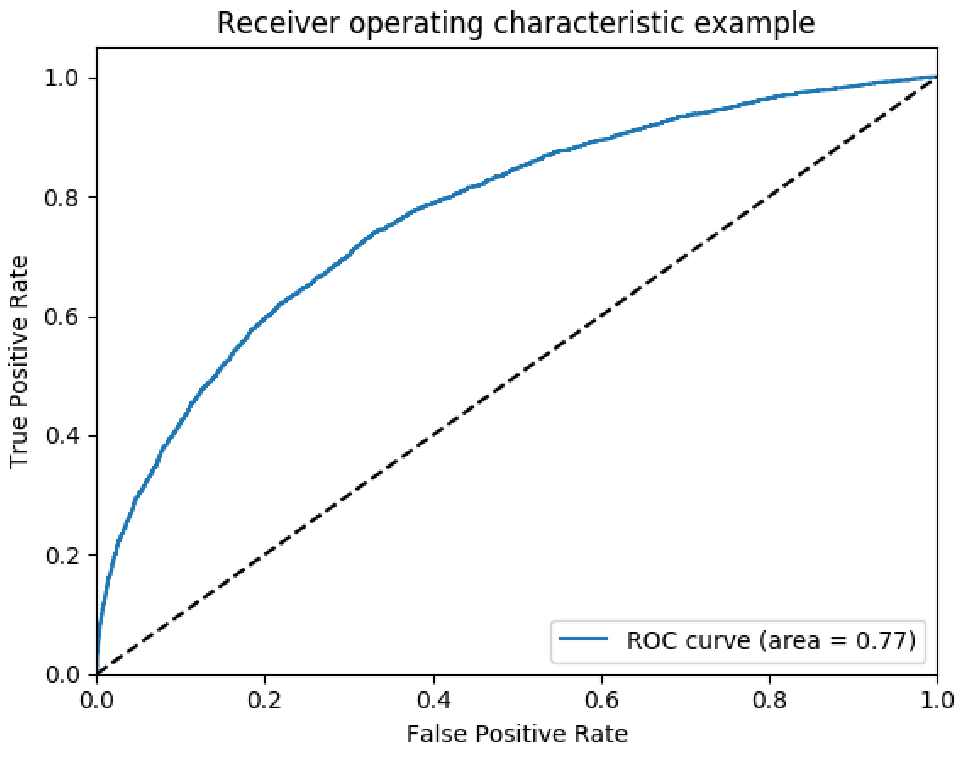

Figure C.3: ROC curve of Pathological Conditions, Signs and Symptoms (C23)

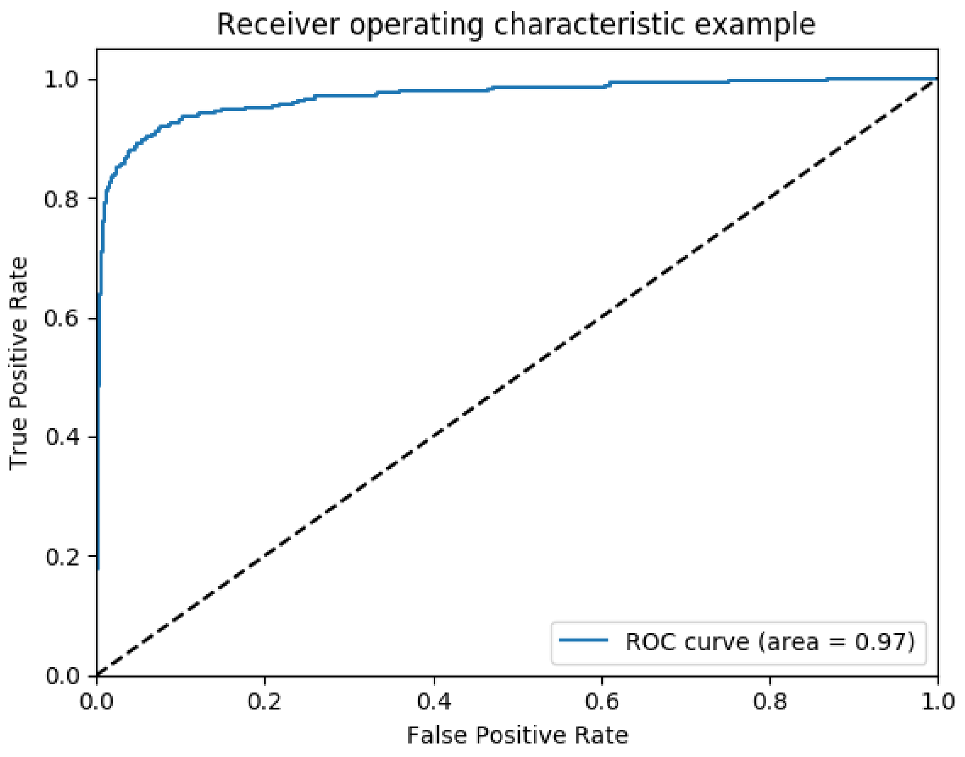

Figure C.4: ROC curve of Eye diseases (C11)

From these two figures, we can conclude that some classes have severe difficulties in the classification tasks. Label 23 outnumbers every other in document counts, which implies that their words probabilities on every other text are very high. Namely, classifiers are having troubles distinguishing from positives and negatives samples in label 23 because their ‘words’ can be correctly represented in every other category (assuming a OvR classification problem).

On the contrary, eye diseases, with a decent number of records, show that the classifiers can discern effectively between positives and negative samples; getting to the resolution that what determines the difficulty of estimating a document is the number of related labels in a class and not the number of texts that it has. The larger the relations between categories, the harder it will be to classify them.

C.6 Multiclass classification results extended

	Macro_F-score	**Micro_F-score**	**F-score**
Multinomial Naive Bayes	0.29	0.38	0.40
Stochastic Gradient Descend	0.33	0.37	0.40
C-Support Vector	0.27	0.31	0.31
One vs Rest	0.54	0.57	**0.58**

Table C.3: Tf-idf F-scores

	Macro_F-score	**Micro_F-score**	**F-score**
Gaussian Naive Bayes	0.25	0.30	0.29
Stochastic Gradient Descend	0.33	0.43	0.41
C-Support Vector	0.34	0.44	**0.42**
One vs Rest	0.35	0.44	**0.42**

Table C.4: Word2Vec F-scores

	Macro_F-score	**Micro_F-score**	**F-score**
Gaussian Naive Bayes	0.21	0.31	0.27
Stochastic Gradient Descend	0.33	0.39	0.38
C-Support Vector	0.35	0.42	**0.41**
One vs Rest	0.35	0.42	**0.41**

Table C.5: Simon F-scores

Bibliography

[1] Carol Friedman, Thomas C Rindflesch, and Milton Corn. Natural language processing: state of the art and prospects for significant progress, a workshop sponsored by the national library of medicine. *Journal of biomedical informatics*, 46(5):765–773, 2013.

[2] Idc_seagate_datcon_financial_services_v6. `https://www.seagate.com/www-content/our-story/trends/files/idc-seagate-datcon-financialservices.pdf`. (Accessed on 03/29/2020).

[3] Andreas Holzinger. Trends in interactive knowledge discovery for personalized medicine: cognitive science meets machine learning. *The IEEE intelligent informatics bulletin*, 15(1):6–14, 2014.

[4] Jae K Lee, Paul D Williams, and Sooyoung Cheon. Data mining in genomics. *Clinics in laboratory medicine*, 28(1):145–166, 2008.

[5] Natural language processing - wikipedia. `https://en.wikipedia.org/wiki/Natural_language_processing`. (Accessed on 03/29/2020).

[6] Gay Su Pinnell and Irene C Fountas. *The continuum of literacy learning, grades 3-8: A guide to teaching*. Heinemann, 2011.

[7] D. Jurafsky, J.H. Martin, P. Norvig, and S. Russell. *Speech and Language Processing*. Pearson Education, 2014.

[8] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[9] Amalia Luque, Alejandro Carrasco, Alejandro Martín, and Ana de las Heras. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition*, 91:216–231, 2019.

[10] Donwloadable corpora. `http://disi.unitn.it/moschitti/corpora.htm`. (Accessed on 05/12/2020).

[11] Carolyn E Lipscomb. Medical subject headings (mesh). *Bulletin of the Medical Library Association*, 88(3):265, 2000.

[12] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 254–269. Springer, 2009.

[13] scikit-learn: machine learning in python — scikit-learn 0.22.2 documentation. `https://scikit-learn.org/stable/`. (Accessed on 05/12/2020).

[14] Torsten Joachims. Categorization with support vector machines: Learning with many relevant features. In *machine learning: ECML-98 10 th European Conference on Machine Learning*, pages 137–142, 1997.

[15] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[16] Edward Loper and Steven Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[17] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[18] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14(9), 2011.

[19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[20] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[21] What is the significance of c value in support vector machine? `https://medium.com/@pushkarmandot/what-is-the-significance-of-c-value-in-support-vector-machine-28224e852c5a`. (Accessed on 05/08/2020).

[22] Carol Friedman, George Hripcsak, et al. Natural language processing and its future in medicine. *Acad Med*, 74(8):890–5, 1999.

[23] Curriculum design : Curriculum design philosophy. `https://www.kdd.org/curriculum/view/curriculum-design-philosophy`. (Accessed on 03/10/2020).

[24] Archiving, chapter 2: Medical image data characteristics - society for imaging informatics in medicine. `https://siim.org/page/archiving_chapter2`. (Accessed on 03/10/2020).

[25] How to use roc curves and precision-recall curves for classification in python. `https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/`. (Accessed on 05/10/2020).