Master in High Performance Computing

Multi-label Classification of Computed Tomography Scan Reports

Supervisor(s):
Prof. Luca Heltai,
Prof. Luca Bortolussi

Candidate:
Matteo Zampieri

5th edition
2018–2019

Abstract

The digitalization of clinical reports and the ever-growing usage of electronic health records make possible the collection of huge amounts of data. This data can be used to explore strategies to come in aid of both the patients and the clinical personnel, in terms of inference tools that could hint diagnostic decisions in a relevant manner, or as a general research pool. This project specifically makes use of reports of Computed Tomography Scans of patients with metastatic breast cancer. The aim of the thesis is to explore methods for multi-label text classification. The reports of interest are classified with a varying number of tags, depending on the location of the metastasis inferred from the report, that comes in the form of a free text description. To address this problem, I used a set of algorithms, namely logistic regression (multinomial and one-vs-rest), k-Nearest-Neighbors (with 'uniform' and 'distance' weight), Multi-k-Nearest-Neighbors, and Support Vector Classifier; these algorithms were fed with different types of word embeddings (TF-IDF and doc2vec). Moreover, the fastText library was explored in its integrated word embedding and text classification capabilities. At last, I used Fast-Bert, an open-source extension of Google's BERT to specifically perform text classification. The results were not satisfying, due to the small size and the high class imbalance of the dataset. However, the investigation of different techniques has shed light to the promising possibilities of some of the strategies used.

Contents

Chapter 1

Introduction

1.1 Thesis Motivations

Notwithstanding significant systemic adjuvant therapies and improved early detection strategies, breast cancer is still the major cause of cancer-related death in women worldwide. Around 5-10% are metastatic at diagnosis, while 20-30% of early-stage patients develop Metastatic Breast Cancer (MBC) during follow-up [1]. Despite breakthroughs in cancer treatments, the main goal of MBC management is aimed at burdening the course of the disease, with constant monitoring to avoid ineffective anti-cancer therapies or unnecessary side-effects. Progression is usually tracked at intervals of 10 to 12 weeks, whereas clinical trials monitor treatment development ranging from 6 to 9 weeks, until disease progression or end-of-study. Radiology reports are used in the daily practice to manage MBC's treatment, but the manual extraction of the relevant information is quite time consuming. The present work is inserted in the context of the progressive digitalization of medical reports and the expanding use of electronic health records (EHR), that have led to the availability of huge amounts of clinical data. Indeed, this type of data offers great possibilities to investigate new opportunities aimed at improving clinical workflows, diagnostic decisions [2], or, more broadly speaking, new lines of research. Nonetheless, a comfortable access to such contents is possible only through the extraction of the meaningful components from the original source, which, in most cases, comes in the form of clinical narratives. Implementing systems for the automatic extraction of interesting information could help build a comprehensive database that would at last provide benefits to research lines focused on domain-specific variables, on horizontal studies, follow-ups, clinical decision making, resource management, and logistics.

This thesis deals with radiology reports written in Italian of Computed

Tomography scans of patients with Metastatic Breast Cancer; more specifically, it aims at classifying the documents based on locations of the metastasis with a multi-label approach, through the investigation of different Natural Language Processing (NLP) techniques. This work wants to measure the efficacy of novel approaches with the final goal of providing help in the process of clinical reports storing. If such a task is possible, the pipeline could be directly integrated in the software the expert use to draft their reports; the online classification could then be immediately validated or corrected by the radiologist, thus providing another feedback to the classifier. In time, this process would reduce the manual effort of the classification for the storing of a comprehensive document.

1.2 Short Introduction to Natural Language Processing

Natural Language Processing (NLP) comprises a wide set of techniques and tools to perform language-based tasks, such as language translation, speech recognition, spell-checking, named entity recognition, sentiment analysis, documents classification, question answering, relationship extraction et cetera. Even though NLP research exists since the 1950s as a set of rule-based methodologies (like predefined grammar or stemming heuristics), significant inflation in its possibilities can be attested at first in the 80s, after the advent of statistical inference techniques and an ever-increase in computational power, and then in the 2010s with the important prevalence of deep learning unsupervised and semi-supervised methods (for overviews, [3], [4]). Machine learning algorithms allows the inference of linguistic rules thanks to the creation of models that are fed with large *corpora* (sets of documents) based on real-life examples. Some of the advantages over rule-based systems are:

— learning procedures that automatically focus on the most common cases

— robustness to unfamiliar or erroneous input (whereas hard-coded rules are time consuming and hardly comprehensive)

— to increase accuracy, many times is sufficient to input a larger amount of data (whereas hard-coded rules require more hard-wired complexity with the risk of unmanageability)

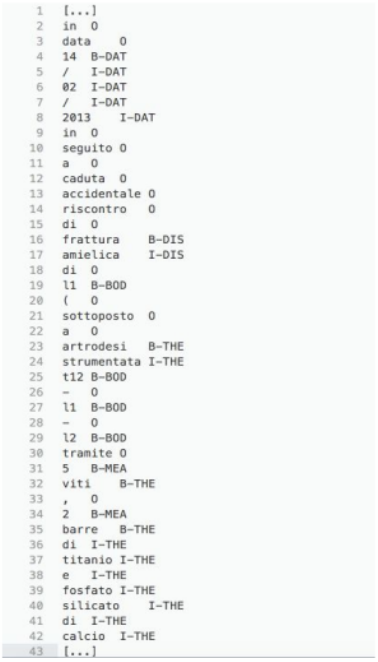
As I will later explain, in this thesis I will make use of both statistical inference and deep-learning methods to tackle a problem of multi-label text classification.

1.3 Background and Related Works

In this context, a computed tomography scan is a radiodiagnostic technique that provides an anatomical 3-D stratified image. Associated to the image, the radiologist also writes a comprehensive report of what he considers of relevance. Even though there are guidelines on how to draft a clinical radiology report (e.g. the description starts from the head and gradually moves towards the feet), it usually comes in the form of free narrative text. This means that, apart from technique-related ICD codes (an ICD code - International Classification of Diseases - is a numerical encoding of all the diseases, pathologies, and also exams), the summary of the diagnostic event is up to the rhetorical abilities of the writer. On top of this, there are usually acronyms and abbreviations that can be local to the specific hospital or clinical facility. In addition, misspelling or ungrammatical language due to human or software (e.g. voice dictating tools) are not rare to be encountered.

A number of strategies to tackle the analysis of free text are based on Natural Language Processing techniques. For the purposes of this work, the task is looking towards the extraction of relevant information regarding the location of the metastasis in a narrated clinical report; such an endeavor is feasible not only if it's possible to point at spatial coordinates from direct written references to anatomical locations, but also if it is attainable to infer such references from more indirect complex periphrases. There are some major issues when working with natural language in free texts. One is the high variability between documents written by different people. In absence of a standardized filling format, it's dubious that two subjects could write almost perfectly overlapping descriptions of the same object. To this we should add the very domain-specific technical language that this work deals with. Clinical and biomedical texts very often seem more convoluted than normally spoken and written language. The most common available tools for NLP are trained on and for regular, everyday language schemes. Even though there have been huge improvements on providing a most comprehensive vocabulary and set of semantic relations for many common English Natural Language Understanding tools, errors are not uncommon (e.g. Google Translate does a wonderful job at translating even biomedical descriptions, but we shouldn't be surprised of seeing repetitions or non translated unrecognized tokens).

On top these issues, even though it is in continuous growth, there is not a large body of work done with the Italian language compared to the English one, even less when dealing with the biomedical domain. Nonetheless, significant research has been conducted on important topics like Named Entity Recognition (NER), meaning the annotation of the single tokens in a docu-



```
[...]
in  O
data    O
14  B-DAT
/   I-DAT
02  I-DAT
/   I-DAT
2013    I-DAT
in  O
seguito O
a   O
caduta  O
accidentale O
riscontro   O
di  O
frattura    B-DIS
amielica    I-DIS
di  O
l1  B-BOD
(   O
sottoposto  O
a   O
artrodesi   B-THE
strumentata I-THE
t12 B-BOD
-   O
l1  B-BOD
-   O
l2  B-BOD
tramite O
5   B-MEA
viti    B-THE
,   O
2   B-MEA
barre   B-THE
di  I-THE
titanio I-THE
e   I-THE
fosfato I-THE
silicato    I-THE
di  I-THE
calcio  I-THE
[...]
```

(a)

Figure 1.1: Example of text annotation with Named Entity Recognition in biomedical domain. Figure taken from [8]

ment depending on semantic relations that are of relevance for given tasks and domains. Words can be tagged based on their belonging to categories like 'disease', 'anatomical part', 'organization' etc... Open source libraries (like `NLTK` [5], `spaCy` [6]) that automatically annotates texts already exists, but they are mainly limited to basic categories ('person', 'city', 'date', and such), and are trained on huge but general-domain data (or very diluted, like Wikipedia). Within these, a notable mention is due to `TextPro`, a toolsuite that deals with Italian, English, and also German on a variety of linguistic tasks, namely tokenization (the process of splitting sentences in their units), sentence splitting, morphological analysis, Part-of-Speech tagging, lemmatization, stemming, and even temporal decomposition and more [7]. Recent advancement regarding NER for the Italian biomedical domain has been achieved by [8].

The group built a multi-layer classifier, in which the first two layers were both Bidirectional Long-Short-Term Memory Recurrent Neural Networks (Bi-LSTM), the first taking as input both the training set and a custom-made domain-specific corpus and outputting Word Embeddings (WE), the second layer feeding contextualized WEs within the training sentences to the

final one; this last layer models the tagging decision by means of a Conditional Random Field algorithm, that minimizes the probability of incorrect NER sequences based on the training set. Notwithstanding the important idea of feeding a domain relevant *corpus* based on successful work on relation extraction by [9], the main take-away element of this research relies on the strategy with which they used the network, i.e. that of *active learning*. NER is a task whose outcome is of utmost importance for many modeling topics, but a great issue lies in the need of manually annotating the training text when needing to build domain specific models. Active learning allows to significantly cut human endeavor by initially feeding the model just a tiny fraction of manually tagged text. After an initial and probably mostly erroneous classification, the network is again fed with another fraction of training samples that have been manually evaluated and corrected after the first outcome. This procedure is repeated until the accuracy converges.

Named entity recognition can be useful not only in itself, but also as a starting point to produce more poignant entity extractions from free texts. By providing a General Recurring Unit (GRU) based classifier an annotated training set, and combining results from this with those of a dictionary lookup, i.e. a knowledge-based entity match, work by [10] was able to annotate and extract *clinical events* on cardiology reports, events that in this case are characterised also by temporal features, with a final precision of 88.6%, recall of 91.7%, and F-1 score of 90.1%.

A study that is more relevant to the topic of this thesis is [11]. This work specifically deals with the classification of radiological reports. The aim was that of extracting different kinds of information of the examination, namely:

— type, as in first or follow-up

— result (positive, negative, stable, relapse)

— nature of lesion (neoplastic, not-neoplastic, uncertain)

— site of lesion

— type of lesion (infectious, aspecific, uncertain, primary, metastasis)

After the manual annotation of a number of documents via a custom made interface for training purposes, several intersecting and cascade methods of classification based on sentence tagging were deployed. The final classification is based on radiologists-provided heuristic rules that for example give precedence to *relapse* classification over *stable* if there is at least one sentence classified as *relapse*; *metastasis* prevails on *primary* if also *neoplastic* is present in another level, and so on.

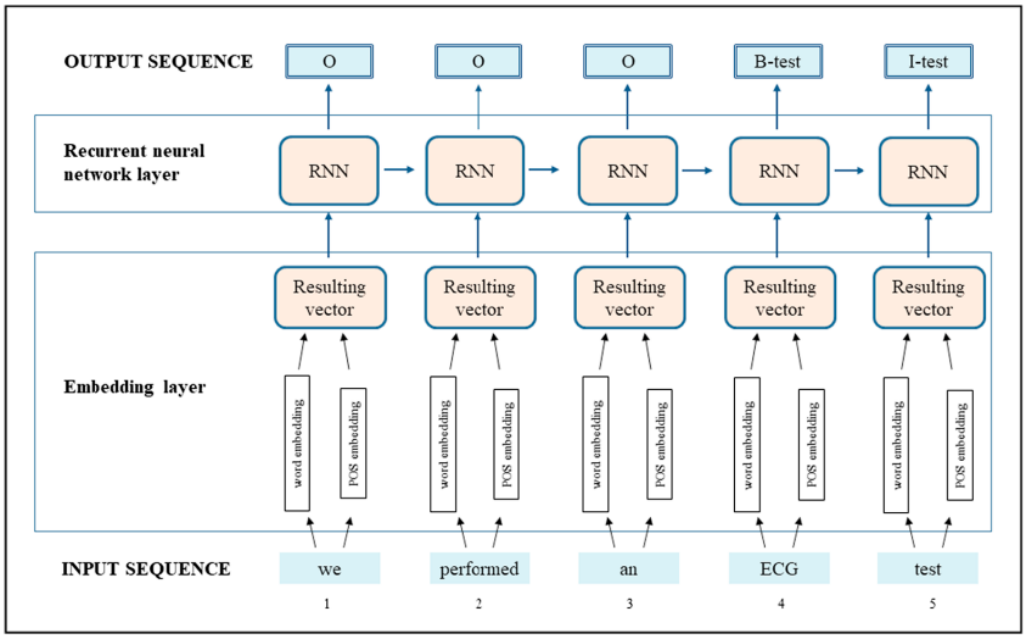


Figure 1.2: Model architecture used in [10]. The final notation is in B-I-O sequence notation (Begin, Inside, Outside).

1.4 Issues and Directions Regarding the Present Work

The few studies I just presented look very promising for the Italian biomedical NLP. Surely, they are more than starting points for future works, as they provide essential cues. However, their strength points pose some issues for the present thesis project, namely the size of the data and the level of annotation of the data itself.

The previous works reached quite successful results thanks to the use of machine learning and deep learning algorithms applied to a huge body of documents or a thoroughly annotated dataset, or both. For example, in [11] they had the availability of 10'000 unlabeled reports, of which 346 were meticulously tagged by several experts. In [8], for the active learning based NER, they utilized 1000 electronic health records, with manual annotations supervised by a pool of professional physicians. Even when the dataset is tiny, as in [10], the time spent by the clinical expertise to create a *golden standard* of tagged documents and later verify the annotation was quite important.

As I will present later in the section dedicated to the dataset, my body of work was relatively small, counting 331 non-annotated radiology reports.

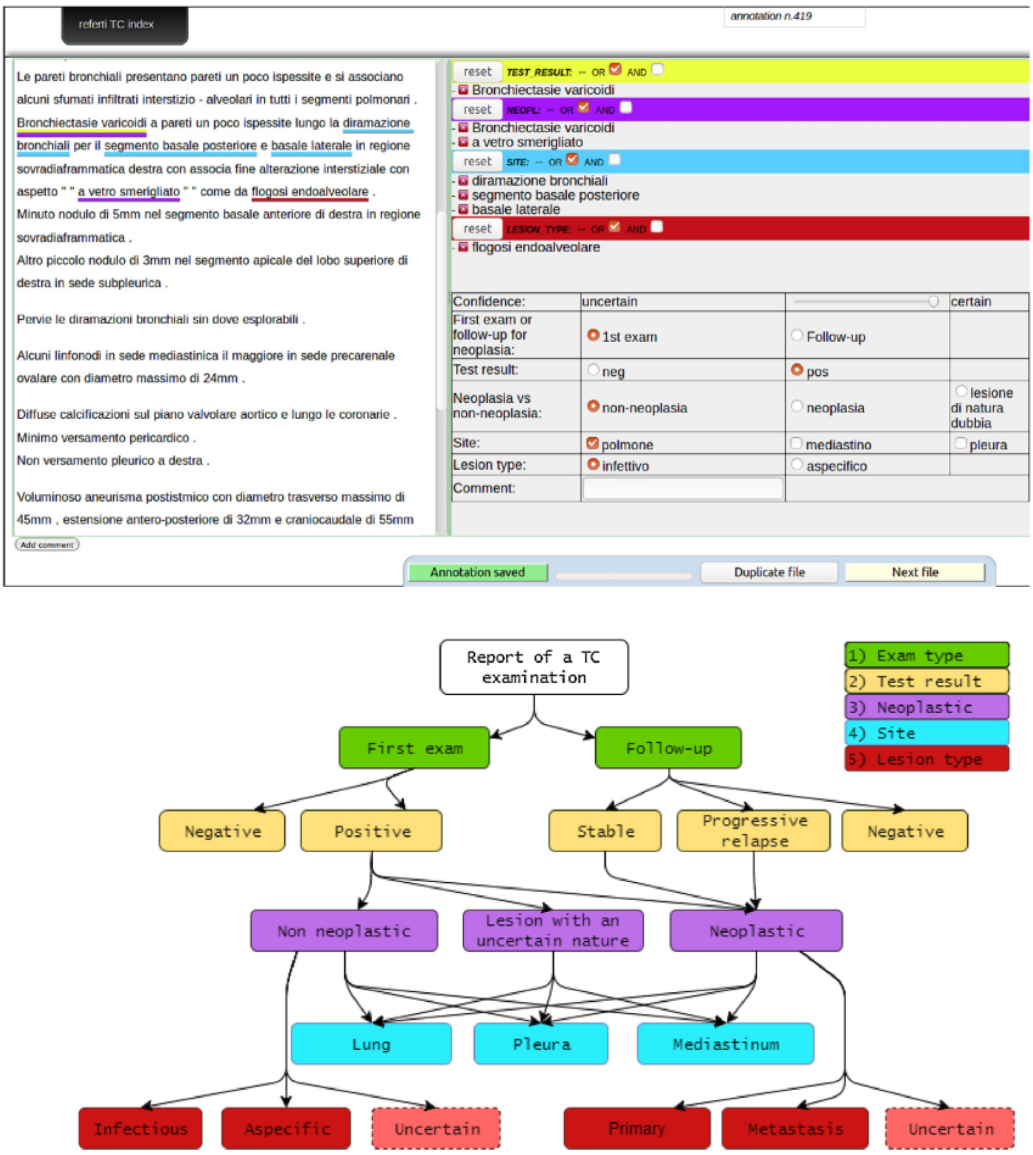


Figure 1.3: On the top, the interface used for annotating the text used in [11]. Below, a schematic view of the several layers of classifications used in the same work.

Another very important issue regards the final task itself, i.e. a multi-label classification: the issue stems not only by the high number of labels present in the dataset, but also from the important imbalance across the labels themselves. Again, I will later describe the dataset more thoroughly, but it is a key point to make in order to introduce the different classification strategies I

adopted with regards to the studies in the literature. As of now, the dataset counts 18 labels, several of which appear only once through the entire body of documents; others, on the contrary, appear in at least half of the cases. If we compare the multi-level classification scheme develop by [11] on radiology reports, when we focus on the *site* classification, we notice that they dealt with just 3 mutually exclusive classes.

Due to the lack of of a significant amount of balanced data, and due to time constraints, the present thesis tried to tackle the task of multi-label text classification by focusing on implementing existing supervised algorithms, and on testing existing state-of-the-art deep learning libraries that specifically deal with NLP problems.

Chapter 2

The Dataset

2.1 Radiology Reports

As stated in the previous chapter, the present thesis wants to investigate the possibilities of classifying radiology reports based on some criteria; more specifically, it deals with clinical records of Computed Tomography scans. A CT scan

> "[...] makes use of computer-processed combinations of many X-ray measurements taken from different angles to produce cross-sectional (tomographic) images (virtual "slices") of specific areas of a scanned object, allowing the user to see inside the object without cutting."[1]

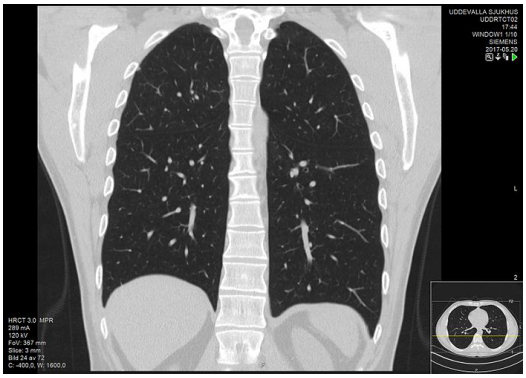
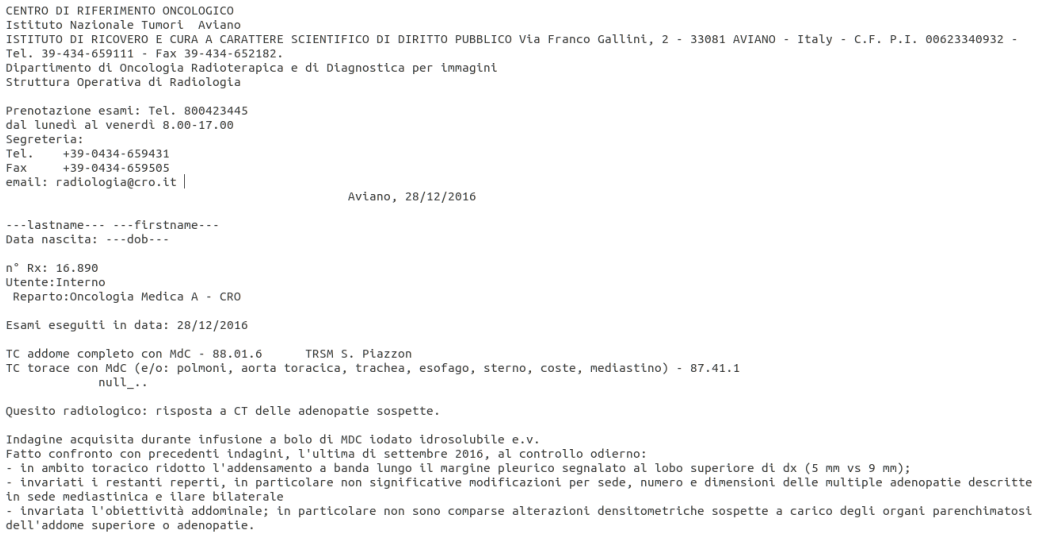


Figure 2.1: CT scan of a normal thorax taken from Wikimedia.

Along with the imaging usually comes a report in the form of a free text description, provided by the radiologist. There is no filling standard, but the

[1] https://en.wikipedia.org/wiki/CT_scan



```
CENTRO DI RIFERIMENTO ONCOLOGICO
Istituto Nazionale Tumori  Aviano
ISTITUTO DI RICOVERO E CURA A CARATTERE SCIENTIFICO DI DIRITTO PUBBLICO Via Franco Gallini, 2 - 33081 AVIANO - Italy - C.F. P.I. 00623340932 -
Tel. 39-434-659111 - Fax 39-434-652182.
Dipartimento di Oncologia Radioterapica e di Diagnostica per immagini
Struttura Operativa di Radiologia

Prenotazione esami: Tel. 800423445
dal lunedì al venerdì 8.00-17.00
Segreteria:
Tel.    +39-0434-659431
Fax     +39-0434-659505
email: radiologia@cro.it

                                                Aviano, 28/12/2016

---lastname--- ---firstname---
Data nascita: ---dob---

n° Rx: 16.890
Utente:Interno
 Reparto:Oncologia Medica A - CRO

Esami eseguiti in data: 28/12/2016

TC addome completo con MdC - 88.01.6      TRSM S. Piazzon
TC torace con MdC (e/o: polmoni, aorta toracica, trachea, esofago, sterno, coste, mediastino) - 87.41.1
            null_..

Quesito radiologico: risposta a CT delle adenopatie sospette.

Indagine acquisita durante infusione a bolo di MDC iodato idrosolubile e.v.
Fatto confronto con precedenti indagini, l'ultima di settembre 2016, al controllo odierno:
- in ambito toracico ridotto l'addensamento a banda lungo il margine pleurico segnalato al lobo superiore di dx (5 mm vs 9 mm);
- invariati i restanti reperti, in particolare non significative modificazioni per sede, numero e dimensioni delle multiple adenopatie descritte
in sede mediastinica e ilare bilaterale
- invariata l'obiettività addominale; in particolare non sono comparse alterazioni densitometriche sospette a carico degli organi parenchimatosi
dell'addome superiore o adenopatie.
```

Figure 2.2: A radiology report extracted from the dataset.

physicians usually commence the descriptive evaluation from the head and gradually move downwards.

The reports I worked on came without the respective X-ray image and were provided by the Centro di Riferimento Oncologico (CRO, Aviano (PN), Italy)[2] and were written accounts of radiology imaging of patients with metastatic breast cancer. These patients undergo periodic evaluations to keep track of the progression of the disease and, in case they adhered, of the progression of the clinical trial.

The first part of the report is composed of a set of headings that are followed by anonymized demographics, a unique ID for the exam, the date of the evaluation, a set of ICD codes corresponding to the types of test and means used. It's not uncommon the use of a *contrast medium* and a parallel evaluation by means of CT guided biopsy (the extraction of tissue with specific needles for futher exams). Below the list of procedures comes the description written by the radiologist; it's in the form of free text and may present the complete evaluation of the whole image or just a focus on the parts regarded of the utmost importance.

The reports I received came in two Excel spreadsheets (`.xlsx` format), that counted respectively 69 and 292 documents, for a total of **361**. More specifically, the first column collects the instances of the textual reports, while a second column lists the respective **labels** for each record.

It was noticed that several reports were not really informative of the

[2] http://www.cro.sanita.fvg.it/it/index.html

Report	Database metastatico::Sedi M iniziali
Aviano, 18/02/2013 \ ---lastname--- ---name--- \Data nascita: ---dob--- \n° Rx	polmone/linfonodi
Aviano, 01/09/2014 \ ---lastname--- ---name--- \Data nascita: ---dob--- \n° Rx	osso
Aviano, 13/03/2015 \ ---lastname--- ---name--- \Data nascita: ---dob--- \n° Rx	polmone
Aviano, 31/03/2015 \ ---lastname--- ---name--- \Data nascita: ---dob--- \n° Rx	linfonodi mediastino/osso
Aviano, 21/05/2015 \ ---lastname--- ---name--- \Data nascita: ---dob--- \n° Rx	osso
Aviano, 26/10/2015 \ ---lastname--- ---name--- \Data nascita: ---dob--- \n° Rx	linfonodi/osso/polmone

Figure 2.3: Excerpt from the Excel spreadsheet. The first column is contains the reports, the second contains the classification labels.

final classification, as they described very thoroughly the procedure of TC guided biopsy instead of the evaluation portrayed on a usual TC image. The radiologists from CRO indeed confirmed that these documents could not be used for the purposes of this thesis. I thus discarded all the reports with a reference to the ICD code for the TC guided biopsy (code: 50.19.1). Subsequent to drops, the final dataset consisted of **331** rows or reports and classificatory labels. The number slightly increases to 340 if we consider that some rows contain more than one report referring to the same patient, as a sequence of first evaluation and follow-up, but still drawing the same final labelling.

On average, considering just the interesting window of the radiology record, i.e. the description of the imaging, each report is approximately 200 words long without punctuation, for a total of 70'743 tokens in the entire corpus. Moreover, the number of labels that come with each report is, on average, almost 2.

Num. Report	Num. Tokens	Tokens/Report	Labels/Reports
340	70743	208.07	1.78

2.2 Labels

In this section, I briefly present an overview of the labels that appear in the classification column.

As I stated, the present dataset deals with TC scan reports of patients affected by metastatic breast cancer. The radiologist is thus investigating the image in search for signs that possibly indicate the presence of the spreading of the original mammal tumor to peripheral areas. The findings might be multiple and located in every part of the body. The labels of the dataset thus indicate the areas that were found to be malicious.

Originally, the classification column counted 22 tags; after consulting with the radiologists, we could group some of the labels within the same category and reduce the total number of classes down to 18.

Table 2.1: Original Categories

Labels
annessi
cervelletto
cervello
encefalo
fegato
linfonodi
linfonodi mediastino
mammella
mediastino
meningi
NED
osso
ovaio
peritoneo
pleura
polmone
regione parasternale
reni
SNC
surrene
tessuti molli
vulva

Table 2.2: Categories after grouping

Labels
annessi
fegato
linfonodi
linfonodi mediastino
mammella
mediastino
NED
osso
ovaio
peritoneo
pleura
polmone
regione parasternale
reni
SNC
surrene
tessuti molli
vulva

Table 2.3: Comparison between the original set of categories (left) and the set with some labels grouped (right). The bolded ones indicate which one were grouped.

Distribution of the labels. So far we saw that the total number of tags for the multi-label classification is 18 and that, on average, each report comes with 2 labels.

However, the labels are not evenly distributed across the documents. In fact, we are dealing with a highly imbalanced dataset, in that that some of the tags appear way more often that the others, and several appear only once throughout the 331 rows. A visual representation of the frequency of the occurrence of each label might help understanding the degree of imbalance.

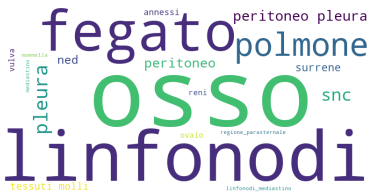


Figure 2.4: Word cloud based on labels' frequency

We immediately notice that the most frequent labels are *bone*, *lymph nodes*, *lung*, and *liver*, with bones being the most frequent site of metastasis' finding. On the other hand, sites like *breast*, *mediastinum*, and *mediastinal lymph nodes* appear only once. I am not sure whether other groupings were feasible without losing salient information, but without receiving any additional guideline on how to merge two or more tags other than those belonging to the Central Nervous System, I haven't proceeded in that direction.

It might be interesting to the reader to visualize the degree of co-

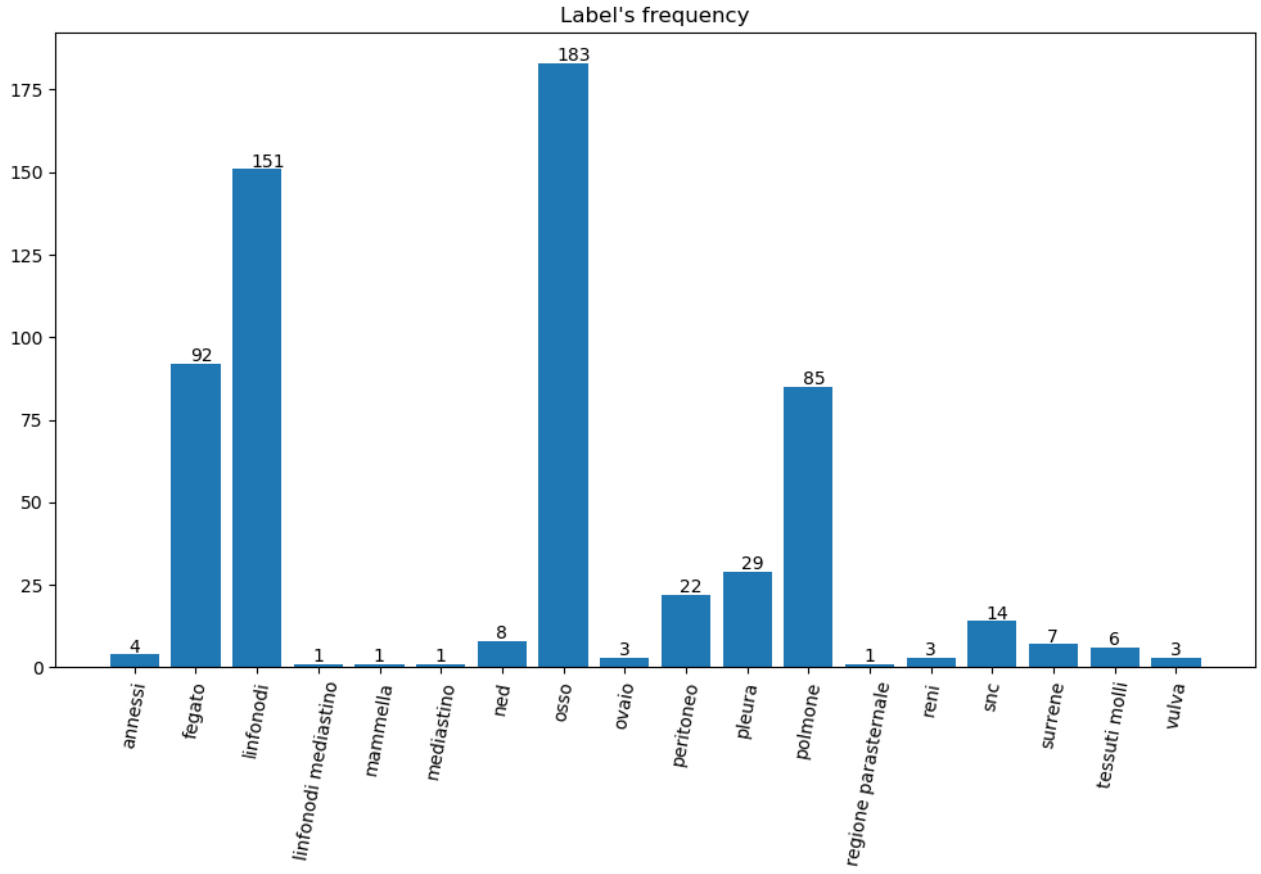


Figure 2.5: Frequency of occurrence of the 18 tags

occurrence of the tags, which might shed a light on the probability of co-manifestation of the metastasis in two sites. The graph in Figure 2.6 shows the connectivity of the labels and represents categories that occur more often together as more proximal to each other. The network is built around the tag *bone*, as it is the most frequent site. Specifically, *bone* is present 82 times together with *lymph nodes*, 48 times with *lung*, it appears alongside *liver* in 40 documents, and 17 with *peritoneum*.

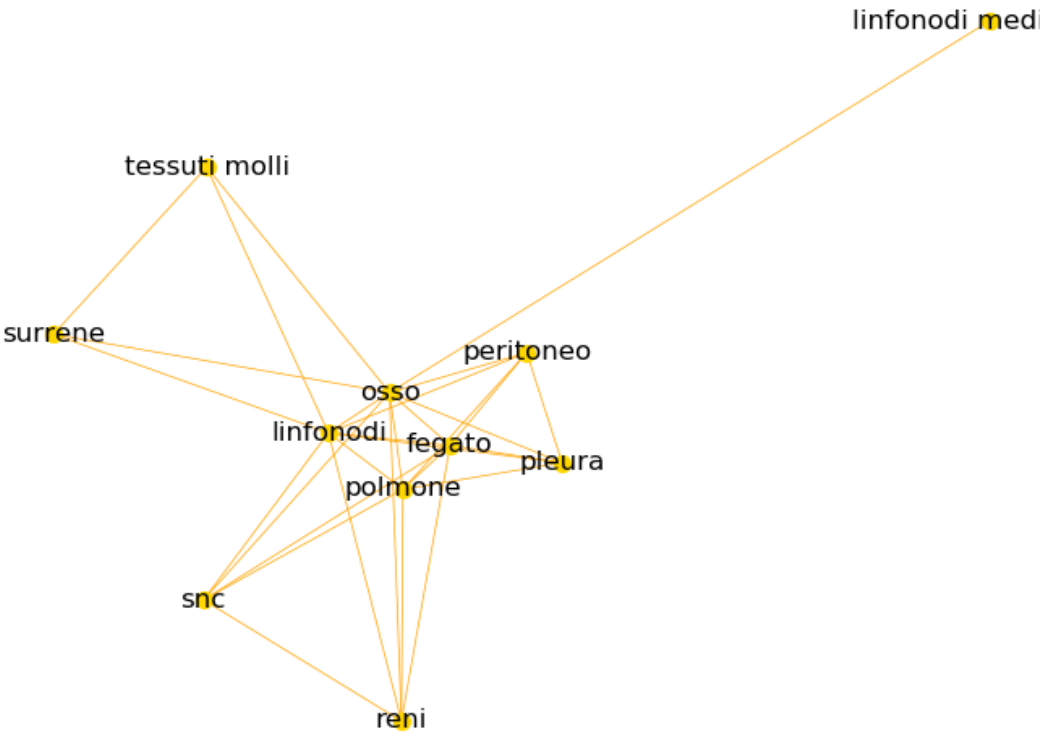


Figure 2.6: Graph drawing the degree of co-occurrence of the labels with repsect to *bone*

Chapter 3

Multi-Label Text Classification

3.1 Introduction to the Problem of Multi-Label Classification

Multi-Label Classification (MLB) is a machine learning problem that deals with assigning one or more label to each input distance. MLB generalizes a similar problem, i.e. Multi-Class Classification, which instead tries to assign one and only one class to each instance. To sum up, in the first case (MLB) there is no constraint on the number of classes an instance can belong to[1].

A simple example can be that of posts on Stack Overflow[2]. Each post can have multiple tags, as in the following:

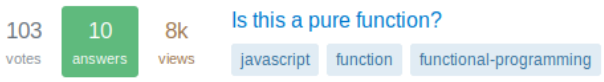


Figure 3.1: Example of a Stack Overflow post with relative tags

We notice that the post has multiple tags, namely `javascript`, `function`, and `functional-programming`. The same tag can be applied to many posts, and different tags can be applied to the same post. There is no theoretical exclusivity on the compresence of labels.

On the other hand, if we are dealing with e.g. flowers and wanted to build a recognition algorithm that provided what kind of flower we are inputting to the machine, we know that we must coerce the decision to one and only one kind of flower (as far as I know, a sunflower cannot simultaneously be a

[1] https://en.wikipedia.org/wiki/Multi-label_classification#cite_note-ml_knn-10

[2] https://stackoverflow.com

rose, unless some mad scientist decided to CRISPR its way to a hybrid - but nonetheless we could argue that it would be a new class altogether).

There is not a clear cut best method to deal with multi-label classifications, even less when dealing with textual data (there is no decisive evidence that more complex, stratified techniques produce better results [12]). There are a number of strategies that allow to break down the problem of multi-labeling:

— **transforming the problem into a binary classification problem**: this method, also known as *binary relevance*, basically states the independent training of one binary classifier for each label. On an input sample, the model predicts all labels for which the respective classifiers predict a positive result;

— **transforming the problem into a multi-class classification problem**: every combination of labels in the training set is accepted as a binarized class instance in itself. For example, a training sample brings a set of label like [A, B], another has [B, C], and a third one [B], the resulting set of classes will be [1, 1, 0], [0, 1, 1], and [0, 1, 0];

— **transforming the problem into a One-vs-Rest multi-class problem**: the One-vs-Rest (OvR, or One-vs-All, OvA) strategy states that each class (each label in our case) is combined with a single classifier, which is trained considering the sample of that class as positives, and all the other samples as negative. The final classification then corresponds to the label whose classifier reports the highest confidence score:

$$\hat{y} = \underset{k \in \{1 \dots K\}}{\operatorname{argmax}} f_k(x) \tag{3.1}$$

Where K is the total number of classes.

The second strategy, i.e. that of creating a power set of all labels' combinations, could be useful in the case the training dateset contained almost (or better) all possible combinations. However, in this case all the combinations with 18 labels would be a number in the order of the $262k$. Having only 330 reports, it is not a feasible strategy to use. More promising outcomes would come by adopting the other listed strategies, namely binary relevance and OvA. Indeed, I made use prevalently of the One-vs-All classification way, as I wanted to test already proven implementations on my dataset. Later, I will briefly explain the algorithms used and some existing deep learning libraries that exploit those algorithms in their classification layers.

3.2 Text Representation

Before delving into the algorithms utilized for the classification of the documents, i want to briefly explain the most common methodologies that are used to represent textual content in computational problems.

Us humans are quite comfortable in dealing with words and recognizing letters, written numbers, known foreign language terms and so on. We also have quite innate statistical inference tools [13] in understanding when a word exists and it conveys a meaning, or if it makes sense that it can exist, as it seems the composition of two or more know words; or if simply it is not a possible term. But we are not machines and whenever we need to study relations that can arise from different documents, infer the mood of a thousand twitter comments, analyse the possibility of being in front of a network of spread fake news - all of these tasks are accomplished thanks to the power of computers, algorithms for the recognition of patterns and such; in few words, we need to feed these tools with textual data. However, pure raw strings are hardly comprehensible to algorithms that were developed to crunch numbers. This is the reason why textual data is usually converted into numeric formats that can be comfortably supplied to computational models.

The mapping of the words into vectors of real numbers is known as **word embedding**. What's the idea behind mapping words in a vector space? To put it simply, words that occur in similar contexts should *occupy* close spatial positions, whereas words that hardly share context should be more distant to each other. The measure of similarity that is most often use is *cosine similarity*, defined as:

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|a\|\|b\|} \tag{3.2}$$

Where vector a and vector b are two word vectors (or document vectors).

Word embedding is a term the comprises different possibilities to compute the conversion. Following, I will briefly list the most used.

3.2.1 Word Embeddings

There are a number of ways documents are mapped into vectors, the most common ones being:

— **bag-of-words**

— **tf-idf**

— **distributional neural embedding**

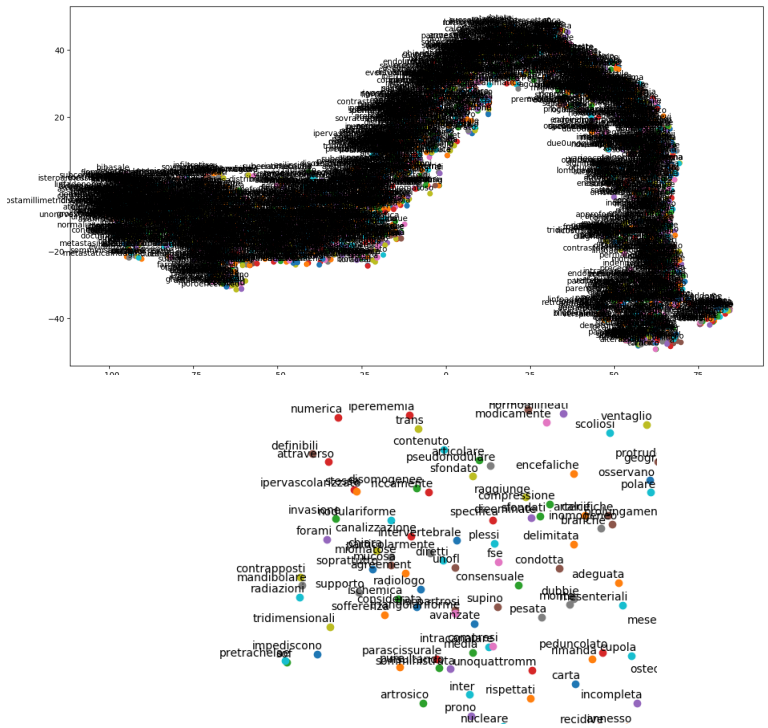


Figure 3.2: 2-D projections of word vectors. On top, the entire distribution of words from the reports; below, a zoom-in inspection.

Bag-of-words. This is the most traditional model of word embedding. It works by first building a dictionary of all the words present across the documents of the corpus of interest; then, for each document it generates a vector of integers that respectively represent the frequency of the word indexed in that position for that document. If we take as example the following sentences:

"The cat ate the mouse hidden in the empty can."

"Can you lend me the mouse of your computer?"

Each and every word present in the two sentences are used to build a dictionary vector, as in:

[an, ate, can, cat, computer, hidden, in, lend, me, mouse, of, the, you]

For the first sentence, based on frequency enconding, the word vector is:

$$[1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 3, 0]$$

Such a strategy can be useful for small datasets, where the problem can be emphasized by term frequencies. However, we can notice how each word token is completely isolated from its original context, as word order is not relevant, and there is the risk of giving too much importance to meaningless tokens that appear more often.

TF-IDF. Term Frequency Inverse Document Frequency (or TF-IDF) is similar in principle to the bag-of-words model, but other than counting just the occurrences of the single tokens within their documents, it also gives relevance to the occurrence of the tokens across the whole corpus. Basically, it provides a weighting to the words within the context of interest. The formula that conveys the value of TF-IDF for a single word is:

$$W_{ij} = tf_{ij} \times \log\left(\frac{N}{df_i}\right) \tag{3.3}$$

where tf is the term frequency of ith word in the jth document; df is the number of documents containing i, and N is the total number of documents.

In other words, TF-IDF states that the more a term is present across all the documents, the less relevant it probably is. On the contrary, the less frequent a term is, the more valuable it probably is.

One might argue that *una tantum* words has the chance of having the highest TF-IDF value, as well as more frequent terms can count the highest number of occurrences in the bag-of-words model. Usually, to contrast these circumstances, some *pre-processing* steps are applied to the text, like removal of *stop-words*, i.e. words that commonly are the most frequent (articles, determinants, prepositions and such); also, terms that appear less than a desired threshold can be discarded. These appear to be useful techniques to refine the aforementioned models; however, we can notice how the deletion of some tokens could completely disrupt the ultimate context of the whole sentence. For example, if we decided to remove articles and common verbal conjugations that we know are very frequent in documents, we would delete the token 'can' in the sentences provided above. While in the second phrasing we wouldn't lose much meaning, the first series of tokens would be deprived of an important element. Where was the mouse hiding? If someone terrorized told me "Snake ate mouse hidden empty..." I would probably take him for a joker, and then move to my room where I keep my collection of empty cans.

Distributional Neural Embedding. Although the mapping of words into vector spaces by means of feed-forward neural networks is not a relatively recent 'discorvery' [14], it was not until 2013 that word vectors as we

usually refer nowadays came to play (with `word2vec` and `GloVe` [15]). The main feature of this modelling is the willingness to capture encoding contextual and semantic relationships amongst tokens. Moreover, differently from the first experiments with neural embeddings, these new models are tuned to be usable in many tasks, contrary to the task-specificity of the former.

In particular, the first neural model consisted in a one-hidden-layer feed-forward network that predicted the next word in a sequence via a *softmax* function.

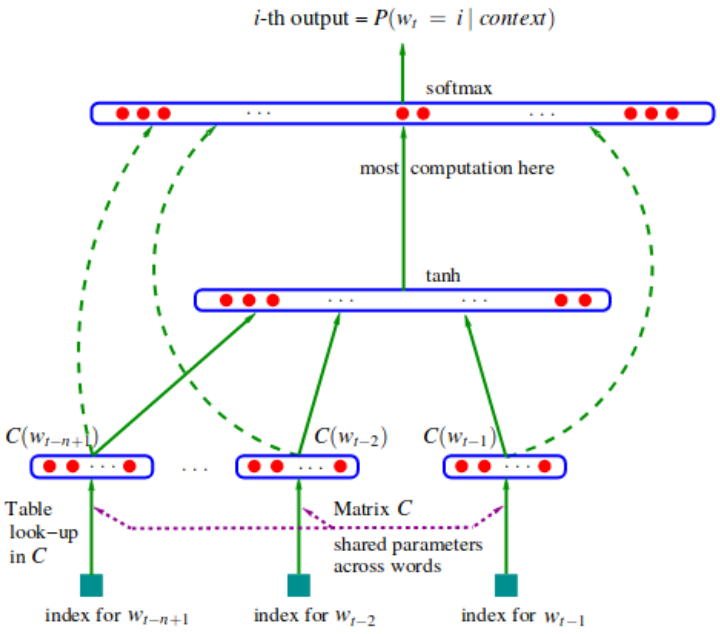


Figure 3.3: Neural architecture in Bengio et al. [14]

The cost of this model lies in the hidden layer that provide a non-linearity to the concatenation of word embeddings of previous words. Nowadays, state-of-the-art word embeddings make use of LSTM neural network [16]. Another issue that still bothers Natural Language Processing tools is the use of the softmax function, that returns a probability distribution over all the words, thus having a cost proportional to the number of all the words present in the vocabulary.

The `word2vec` model, one of the most popular word embedding models used today, discards the intermediate non-linear layers and allow more contextual content to be surrounded. This is made possible by following two different strategies:

— **continuous-bag-of-words (CBOW)**: instead of using the previous

n words to predict the following item, this model uses a *window* of surrounding n words around the target token at each time step.

— **skip-gram**: flipping the CBOW model, the Skip-gram model is aimed at predicting the *surrounding* tokens at each token time step.

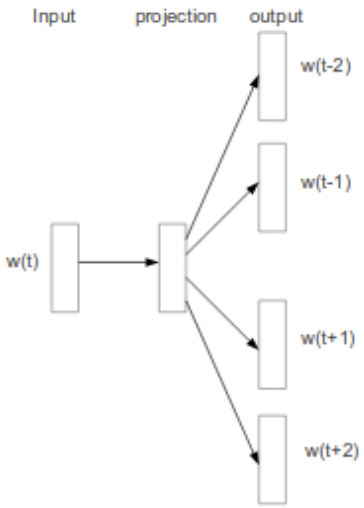


Figure 3.4: Ski-gram model, where at each token w(t), the prediction is towards the surrounding window.

A model by Facebook's `fastText` incorporated the Skip-gram strategy and slightly modified it by adding **sub-words information** [17]. Their work stems from the fact that embedding models map words that can have different forms in different vectors; 'dog', 'dogs', 'house', 'farm', 'farmhouse', are all usually encoded in *unique* vectorial spaces. What they did was extending, in the context of the skip-gram model, the concept of *n-grams* as a window of n tokens surrounding the item of interest to the concept of *n-grams* as windows of characters that compose the single tokens. So, for example, b choosing a window of 3, we can split the word 'where' as:

'whe', 'her', ere'

By doing this, even though the final vector is still mapped to the whole word, the dictionary of the corpus is comprised by the complete set of the sub-words of all the words. This allows to theoretically include the territories of morphology, by accounting, in tasks that might thrive from it, for the encounter of apparently unknown or complex compositions of words.

Quite recently, Google implemented its own model, that gives state-of-the-art results on many NLP tasks. The model is called `BERT` (Bidirectional Encoder Representations from Transformers [18]) and exploits its own sub-word representation by means of word-piece tokenization [19]. The difference with *fastText* rests upon the averaging of the sub-word vectors to form whole-word vectors. Moreover, due to the intrinsic deep bi-directionality of the learning architecture, the word representations are *contextualized* by the tokens that both precede and follow. Capturing context can be a key feature when dealing with syntactic information.

These methodologies to produce mappings of words into vectors are all useful for a variety of downstream NLP tasks, depending on whether the emphasis should be place on semantics, on context, on structural information inherent to the grammar and so on.

For the scope of this thesis, and given that there is no definitive solution for the task at hands, I make use of almost the whole set of embedding models, to investigate which one can provide better results for the purpose of text classification with multiple labels.

3.3 Models and Algorithms for Classification

In this section I will briefly introduce common algorithms that are used for classification tasks and that I implemented in my work. Some inherently support multi-class or multi-label classification, while others work when the problem is transformed in a multi-class one, e.g. via a one-vs-all strategy.

3.3.1 Logistic Regression

Also known as **logit** model, it's a statistical tool that models the probabilities of belonging to one of two classes.The logistic regression exploits the sigmoid **logistic function** to model a binary dependent variable with two possible values. The function is the following:

$$f(x) = \frac{L}{1 + e^{-(x - x_0)}} \tag{3.4}$$

The logistic regression estimates the parameters of a logistic model to return a value for the target variable that lies between "0" and "1" (absolute certainty). The logarithm of the odds of for the value named "1" is a linear combination of one or more predictors that can either be in a binary form themselves or in a continuous form.

This model can be exploited in a one-vs-all strategy, by applying the logistic classifier to each label.

3.3.2 Multinomial Logistic Regression

It is also called **softmax regression** and it is an extension to the aforementioned logistic regression, in that it generalizes the previous binary model to a multi-class one, meaning that it's used to predict the probabilities of the possible outcome of a number of categorical dependent variables from a set of predictors.

The multinomial logistic regression uses a *linear predictor function* in the form of *f(k,i)* to predict the probability that observation i belongs to category k. Without diving into mathematical formulations or proofs, that are beyond the scope of this thesis, it is worth noting that this model is implemented in `fastText` to compute the outcome of a multi-class problem or, following a one-vs-rest application, to predict a multi-label classification. In its 'simplicity', the model produces very good results [12], provided that some assumptions are met, e.g. a minimum number of cases per category in the dataset (at least 10).

3.3.3 Support Vector Classifier

Support Vector Machines (SVM) are a kind of classifiers that compute classifications by constructing hyper-planes in a multidimensional space that separates samples of different classes. The model supports both continuous and categorical variables, but in the latter case it creates dummy variables with a binary value. An iterative strategy is then adopted to optimized an error function. The implementation of different error functions defines several types of SVMs for both regression and classification; for the classification purpose, the choice was towards the **C-SVM**, where C is a regularization parameter, which is inversely proportional to the strength of the regularization. With a SVM, you usually wish to set a margin as wide as possible to include right classifications, while minimizing the misclassifications. The C parameter tells the model how much you want to to avoid misclassifying samples. Small values of C will cause the optimizer to look for large margins in the hyper-plane, thus augmenting the chances of misclassification.

The C-SVM can be used in a one-vs-rest strategy too.

[3] http://www.statsoft.com/textbook/support-vector-machines

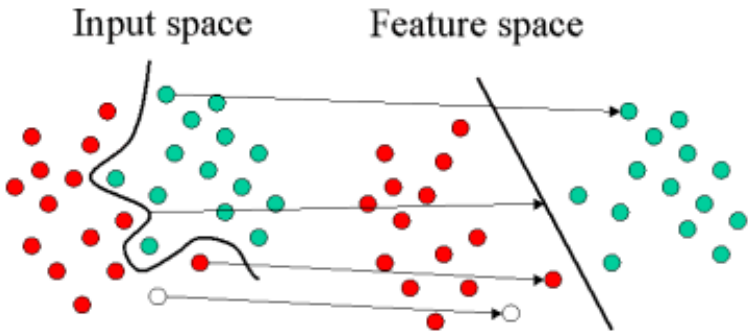


Figure 3.5: Ideal final linear separation with a Support Vector Machine.[3]

3.3.4 k-Nearest Neighbors and Multi-kNN

With an extreme simplification, the algorithm **k-NN** provides a classification scheme based on the *similarity* of the features of the input with those of the training samples. It's usually based on the *euclidean distance*[4] and it works on an arbitrary provisioned k number of neighboring data points, meaning that the class that obtains the highest number of k minimal distances will be chosen as prediction.

The k-NN algorithm doesn't need any assumption on the distribution of the data, thus it represents a good choice for an initial classification adventure. However, the arbitrariness of the choice of k neighbors poses a problem similar to that of the C-SVM, in that a low k determines a narrower search surrounding the input data point, thus resulting more 'blind' to the general distribution of the data points. On the other hand, a higher k risks to ignore details of the data that might be of importance.

Multi-kNN has its roots on the k-NN algorithm and extends it to a multi-label classification problem [20]. First, it finds the input's k nearest neighbors based on the training set; then, based on the number of neighboring points belonging to each possible tag, the algorithm performs a *Maximum A Posteriori* method to retrieve the label set.

I used these models and algorithms in my classification task, by transforming the problem in a one-vs-all one when due, namely with the logistic regression and the support vector classifier. Some procedures implemented the k-NN and the multi-kNN algorithms, whereas other made use of specific libraries' own implementations, that mostly used the one-vs-rest multinomial logistic regression.

In the next chapter, I will present the libraries I used to investigate the

[4] https://en.wikipedia.org/wiki/Euclidean_distance

main problem, along with the results I obtained.

Chapter 4

Implementation

In this chapter I will explain how I handled the data for the pre-processing and the packages that were used. I will then introduce the implementation of the algorithms introduces in the previous chapter, along with the description of the respective libraries and the results.

General Environment For the purposes of this thesis, I used `Python 3.7` in a virtual environment created with `conda 4.7.12`.[1] All the packages and libraries that were needed for the different steps of the work were thus installed in that environment via `conda` or `pip`. Development-wise, I tried to keep the repository as tidy as possible. I wanted to keep similar operations confined in specific modules; for example the pre-processing functions are placed a separate module (as in `reports_parser.py`). Moreover, I assume that the operations are self-contained in dedicated functions. Overall, I tried to keep the scope of each function focused on clear and separate objectives.

4.1 Data Processing

As I introduced in Chapter 2, the dataset consists of 361 radiology reports with the respective labels. The documents came in two Excel spreadsheets, containing one column for the reports and one for the classification.

Given the small size of the dataset, I decided to manage it using `Pandas` (0.24.2), by first loading the two spreadsheets in two `DataFrame` and then merging them into one. `Pandas` comes in handy when dealing with contained dataset sizes, as it allows to run methods on the entire column by exploiting `Numpy`'s vectorization. Other than this, I preferred to keep a tidy dataframe

[1] https://docs.conda.io/en/latest/

with separate columns containing the output of different pre-processing procedures, as they were requested for running different algorithms later on and were useful for experimentation. On top of this, it was possible to save and then load single columns, depending on the necessities.

As mentioned, several reports were not really informative of the respective classification tags, as they dealt with TC guided biopsies; specifically, the textual description provided an account of the procedure of extraction of the tissue of the site of interest, without necessarily focusing on all the areas that might have been set as a final classificatory label. For this reason, the first step to was to drop all these documents. I used a operation to the entire column, a method that is directly implemented in `Pandas`' `DataFrame`'s objects; the target of the `regex` was a specific ICD code that indicated the TC guided biopsy, i.e. `"50.19.1"`. After the drop, the dataset consisted of **331** records.

Another crucial procedure was that of isolating the labels that came divided by a `"/"`, but were compacted as a unique `string`. Following is the format with which they came inside their column:

osso/polmone/fegato

I used the method `column.str.split()` with some additional `regex` patterns to clean the rows from whitespaces. In the end, each row contained a list of labels:

'osso', 'polmone', 'fegato'

Additionally, just before the splitting, I replaced the strings 'cervello', 'cervelletto', 'encefalo', 'meningi' with 'SNC', according to the grouping introduced in the second chapter. Other than this, I corrected some spelling mistakes. I decided to place the newly-formatted labels in a separate column, that I called `split_labels`.

The next important step to take was to extract the meaningful portion of the report containing the description of the imaging session. Before the extraction, a crucial operation when dealing with textual data is to remove any consecutive whitespaces, tabs, newline characters, excessive backslashes, and make sure that spaces around punctuaction is respected. After this, I was able close the window enough to focus on the important part without adding too much noise. The operation wasn't an easy task with `regex` patterns, as the two datasets than compose the set of all the documents came encoded in different formats and didn't show any clear regularity to be used as marking point for a precise cut. The extracted reports were placed in a dedicated column. Following is a sample:

: 28/12/2016 TC capo con MdC (e/o encefalo, cranio, sella turcica, orbite) - 87.03.1 TC addome completo con MdC - 88.01.6 . Quesito radiologico: rivalutazione dopo 6 cicli di terapia in ca mammella HER2 positiva con mts epatiche. Indagine acquisita durante infusione a bolo di MDC iodato idrosolubile e. v. Fatto confronto con precedente ultima indagine dell'8 novembre 2016, al controllo odierno non significative modificazioni a carico delle multiple lesioni ipodense di natura sostitutiva note, disseminate ad entrambi i lobi epatici, che permangono millimetriche e ai limiti della visibilità radiologica. Invariata anche l'area ipodensa di circa 25 mm di diametro assiale massimo in sede di ilo epatico. Vie biliari non dilatate. Non sono comparse alterazioni densitometriche sospette a carico di milza, pancreas, surreni e reni. Non adenopatie in sede lombo-aortica. In scavo pelvico non masse o adenopatie in sede iliaco-otturatoria. Invariata la falda fluida nel Douglas. A livello encefalico non aree di patologico iperaccumulo di mdc in sede intra- od extra-assiale. Nei limiti di norma gli spazi liquorali della base, della volta e le cavità ventricolari. Strutture mediane in asse.

Depending on the library used to build the embedding vectors, which can be context-independent or context-dependent, it might be useful to have different degrees of text cleaning. While a simple pre-processing like the one above, where excessive whitespaces were removed along with a punctuation check, can be used in a context-based embedding like that of `BERT`, a deeper cleaning is necessary for embedding models like TF-IDF, `word2vec`, and `fastText`. Thus, I created another column where the extracted reports were further cleaned, by means of removing all the punctuation, case lowering, encoding of accents into unaccented forms, converting the digits into words.

I also created functions the deal with the extraction of the dates, both in numerical and literal form, and of the ICD codes. These could be useful for quick lookups.

These are preliminary and standard manipulations on texts. Further processing and formatting of the documents is needed to work with the specific libraries I will use, but I will explain them where due.

4.2 Key-Search Preliminary Analysis

Before jumping into more sophisticated solutions, it might interesting to perform a very basic and apparently dumb analysis to check the presence of

the labels inside the free text data, i.e. a *key-search*.

The strategy involves looking for the presence or the absence of the label's string within the document. It might be that the tag is openly pointed at and repeated. Surely, one should immediately grasp the risk of matching not only the labels of that form the final classification of the respective report, but also those that do not belong there. The record whose tags are 'liver' and 'mediastinum' might include a thorough description of the areas of the Central Nervous System, of the bones, of the lymph nodes, but without addressing those sites as malicious. Thus, when looking for each unique label within the document string, we must differentiate between the found occurrences of **True Positives** and those that instead are **False Positives**; the latter, to repeat, indeed belong to the set of all the 18 unique labels, but are not part of the classificatory list of the document of interest.

Figure 4.1 shows the frequency of occurrence of each label as a False Positive and as a True Positive, compared to the total.

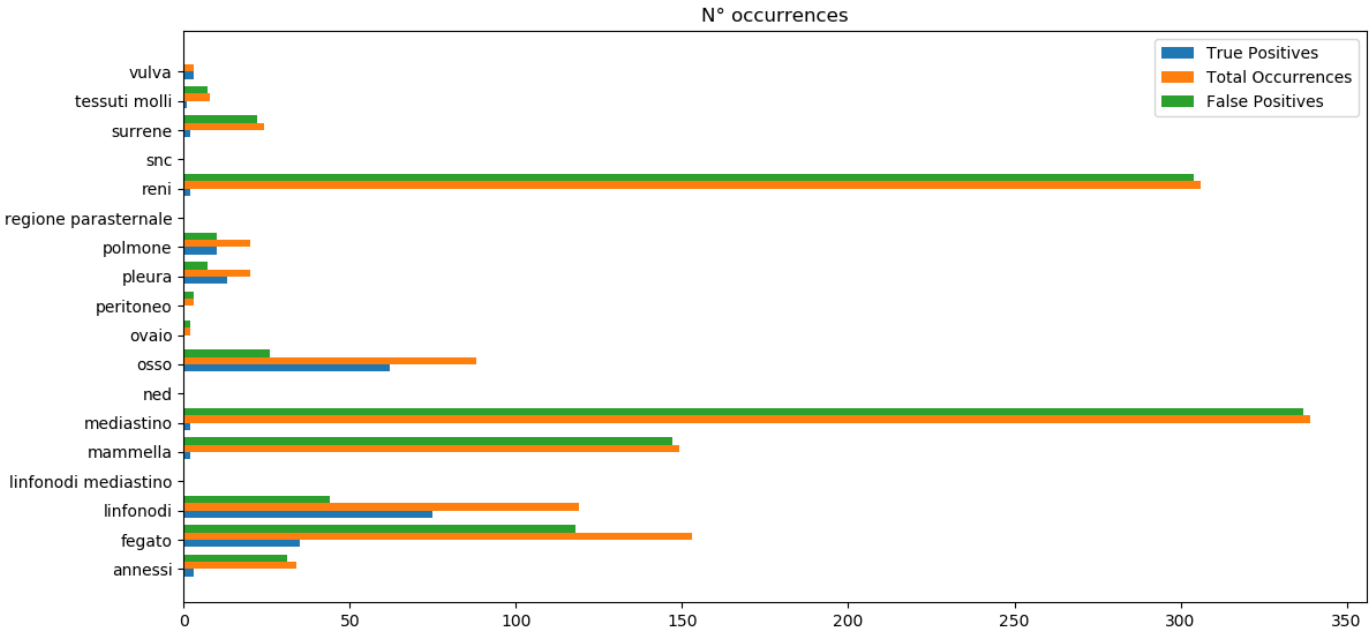


Figure 4.1: Key-Search on the extracted reports. The x-axis represent the number of occurrences, while the y-axis represents the 18 labels.

One should immediately notice the extreme prevalence of false positives over the true positives, especially for the cases of labels that are not so

frequent across the dataset.

The key-search strategy could be expanded to include not only the literal label, but also its plural and composite forms. This is possible with a simple tweak of the tag's string called **stemming**, which involves truncating the word in its last character. By doing this, we allow the inclusion of other possibilities when searching for matching of the string in the text. For example, the word 'polmone' (lung) would become 'polmon', thus with the potential of matching 'polmon*i*' (lungs) and 'polmon*are*' (of/relate-to the lung). We could also include synonyms and link them to the respective label, e.g. 'hepatic' could be linked to 'liver'. Consequently, we could apply the same stemming procedure to the synonyms.

I have run these variants of the strategy, but the results were even worse, so I will not show the plot.

4.3 Logistic Regression, Support Vector Classifier, k-NN, Multi-kNN

In this section I will describe the implementation of the algorithms introduced in Chapter 3. **REFERENCES**

Without reinventing the wheel, I decided to take advantage of a well-estabilished machine learning library, `scikit-learn` (or "sklearn", version 0.21.3).[2] This free package is written in Python and makes extensive use of high-performance linear algebra and array operations, by means of building its core on `Cython`, `NumPy`, and `SciPy`. Moreover, the logistic regression and the support vector machine are built with a Cython wrapper around `LIBLINEAR` and `LIBSVM`, two machine learning libraries written in `C++`. Additionally, I made use of the library `scikit-multilearn` (or "skmultilearn") [3], which itself is built on top of `sklearn` and extends its functionalities to comprehend a gamma of algorithms to deal with multi-label classification.[4]

As I previously mentioned, raw text data is not suitable to be handled by algorithms designed to process numeric data. Thus the need to transform words into numerical vector through an embedding model. For this set of algorithm implemented in both `sklearn` and `skmultilearn`, namely *logistic regression*, *support vector machine*, *k-nearest neighbors*, and *multi-kNN* I decided to test two different models of embedding, namely **TF-IDF** and

[2] https://scikit-learn.org/

[3] http://scikit.ml/index.html

[4] I found the existence of this library very late in the process, so unfortunately I haven't had the time to go deep into it apart from the k-NN comparative algorithm.

doc2vec. The former is implemented directly into sklearn, whereas the second exploits `word2vec` and generalizes it to entire documents[21].

For both models, I used the thoroughly cleaned reports, as I wanted less noise as possible. The reports were previously saved in a text file, one report per line, in order to be accessible to the library dealing with doc2vec.

Furthermore, I talked about exploiting the multi-label problem by transforming it into a multi-class problem by dealing with class by class, each taken as a binary variable. Indeed, the *logistc regression* and the *support vector classifier* are dealt with with a **one-vs-rest** strategy. Because of this, I found myself in need of transforming the way the labels are presented to such algorithms. How? By *binarizing* them in a **one-hot encoding** fashion. This consists in creating, for each document, a corresponding array whose length is the total number of tags, and whose elements are either 0 or 1 depending on the presence or absence of the index-encoded label for that particular report. The way this could be achieved is through `sklearn`'s `MultiLabelBinarizer()`, an encoder that must be instantiated and then fitted with the labels through the `fit_transform()` method. This procedure returns a multi-dimensional array with the converted tags. To access the list of the original labels, one just needs to run `<encoder>.classes_`. In my case, if I take one document's labels as an example, I get the following sequence of original labels, encoded labels, and list of ordered labels by the binarizer:

['polmone', 'linfonodi', 'fegato']

[0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]

['annessi' 'fegato' 'linfonodi' 'linfonodi mediastino' 'mammella'
'mediastino' 'ned' 'osso' 'ovaio' 'peritoneo' 'pleura' 'polmone'
'regione parasternale' 'reni' 'snc' 'surrene' 'tessuti molli' 'vulva']a

We can notice that the binarizer collects the labels in alphabetical order and that the second element of the binarized array corresponds to 'liver' (so it is a 1), the third with 'lymph nodes', and the twelth with 'lung'.

The binarized labels are used also for the k-NN and the multi-kNN algorithms.

4.3.1 TF-IDF with Sklearn

TF-IDF, or Term Frequency Inverse Document Frequency as a refresher, computes the frequency of words inversionally proportional to their occurrence across the entire corpus of documents of interest. Thus, it basically give less importance to tokens that appear the most, whereas highlighting

low-frequency words that might be of higher importance for relevant information. `sklearn` implements its own TF-IDF vectorizer, in the form of the `TfidfVectorizer()` object, to which we need to apply the `fit_transform()` method for every report in the corpus. This method first builds a vocabulary from the corpus and then returns a matrix of tuples containing the document ID and the token ID, followed by the TF-IDF score of the given token in the given document; tuples that are not present in the document have a score equal to 0.

Training and testing sets were generated with sklearn's `train_test_spit()` function, by feeding both the TD-IDF vectors and the binarized labels, and by providing a `test_size` value of 0.2.

4.3.2 doc2vec with Gensim

Gensim is an open-source library, implemented in Python and Cython, that deals with a series of NLP models and tasks. It implements the introduced `word2vec` embedding model, that I remind consists of two variants, namely *cbow* and the *skip-gram*. The former computes token predictions based on the preceding sequence of tokens, whereas the latter build predictions of a window of words around the token itself. On top of the *cbow* embedding, the same author of `word2vec` have build a new model, by simply adding another document-unique feature vector for the prediction of the tokens. When the words vector are trained, the document vector trains as well.

The corpus, one document per line as stated above, is passed through Gensim's `TaggedDocument()` parser, that tokenizes the reports and attaches an ID tag to each of them. A `Doc2Vec()` object is then instantiated, from which the method `build_vocab()` is called by feeding the loaded corpus read from the document tagger as argument. Then the model is trained for 50 *epochs* by stating a *vector size* of 300. The model is then saved for future retrieval.

The list of document vectors can be retrieved by iterating through `model.docvecs.vectors_docs`.

Training and testing sets were generated with sklearn's `train_test_spit()` function, by feeding both the document vectors and the binarized labels, and by providing a `test_size` value of 0.2.

4.3.3 Metrics

One more thing to consider, before finally moving to the results, are the metrics used to assess the outcomes.

In few words, classification tasks usually evaluate the results in terms of **precision**, **recall**, and **f1-score**. In order:

— **precision** is also known as *positive predictive value* and represents the fraction of retrieved instances that are relevant to the task. In other words, it's the number of *true positives* over the sum of *true positives* and *false positives*

$$Precision = \frac{tp}{tp + fp} \tag{4.1}$$

— **recall** represents the fraction of instances that are correctly retrieved, or the number of *true positives* over the sum of *true positives* and *false negatives*

$$Recall = \frac{tp}{tp + fn} \tag{4.2}$$

— **f1-score** or *f-measure* is the harmonic mean between precision and recall

$$F_1 = \left(\frac{2}{recall^{-1} + precision^{-1}} \right) = 2 \times \frac{precision \times recall}{precision + recall} \tag{4.3}$$

These measures can be computed by the services of `sklearn`'s `metrics` module, specifically by the method `classification_report()`, that takes as argument the labels used as *test set* and the *predicted* tags. Moreover, it computes each metric for every label, where assumptions of size are met (otherwise, wherever the samples are not enough, it throws a warning and leaves the scores at 0). On top of these, it also displays `macro_avg` (weighted mean per label), `micro_avg` (mean of the total true positives, false negatives and false positives), and `sample_avg`.

4.3.4 Logistic Regression

As I explained in Chapter 3, it is possible to apply the logistic regression to a multi-label problem that is transformed to multi-class one, by means of the strategy called *one-vs-all* (or one-vs-rest). `sklearn` allows to implement both variants with some simple tweaks. First of all, we need to initialize the object `OneVsRestClassifier()`, that takes as argument an estimator object (in this case the `LogisticRegression()` model), which is used by the wrapping class to fit and predict multiple labels per instance. Regarding the `LogisticRegression()` classifier provided by the library, it was run in two variants based on the feature provided to the argument `multiclass`: in one variant the feature assigned was *"ovr"*, meaning that a binary problem is fit

for each label; in the other, the argument took *"multinomial"*, which is self-explanatory based on the description of the multinomial logistic regression provided in Chapter 3. For both, the chosen `solver` was `saga` (advocated as the best solver for multinomial problems; other solvers are useful for larger datasets, which is not the case for this specific work) and the maximum number of iteration was set to 4000, as a lower value would cause the fitting to throw warnings on the possibilities of running bad trainings due to the sample size.

4.3.5 Support Vector Classifier

For this model, the same strategy *one-vs-rest* was applied via the object `OneVsRestClassifier()` provided by sklearn. The `SVC()` classifier was assigned the feature `linear` for the argument `kernel` based on the literature for the task at hand. The other parameters, like the regularization parameter `C` (=1.0) were kept at default value, given that after some experimentation there was no significant change.

4.3.6 k-Nearest Neighbors and Multi-kNN

The algorithm k-NN provided by sklearn inherently supports a multi-label problem, thus it didn't need the initialization through the OneVsRestClassifier(). The `k` number was set at 2, while the weights chosen were both *uniform* and *distance*: the former means that each point in the neighborhood weigh the same, whereas the latter calls for nearer query points to have a stronger influence. This means that the algorithm was run twice, once per type of weight.

The `Multi-kNN` algorithm is instead provided by the `scikit-multilearn` library, and is supposed to be provide an improved implementation built on top of `sklearn`'s k-Nearest Neighbors classifier. Even though a grid search returned the k number to be best set at 1, I decided to set it at 2 to compare it to the sister's implementation.

4.3.7 Results

I will make use of graphical support to report the results.

For each embedding type, namely sklearn's **TF-IDF** and Gensim's **doc2vec**, the aforementioned list of algorithms was applied, for a grand total of 12 sets of results. The complete list, as a reminder, is composed of:

— **logistic regression (multinomial**

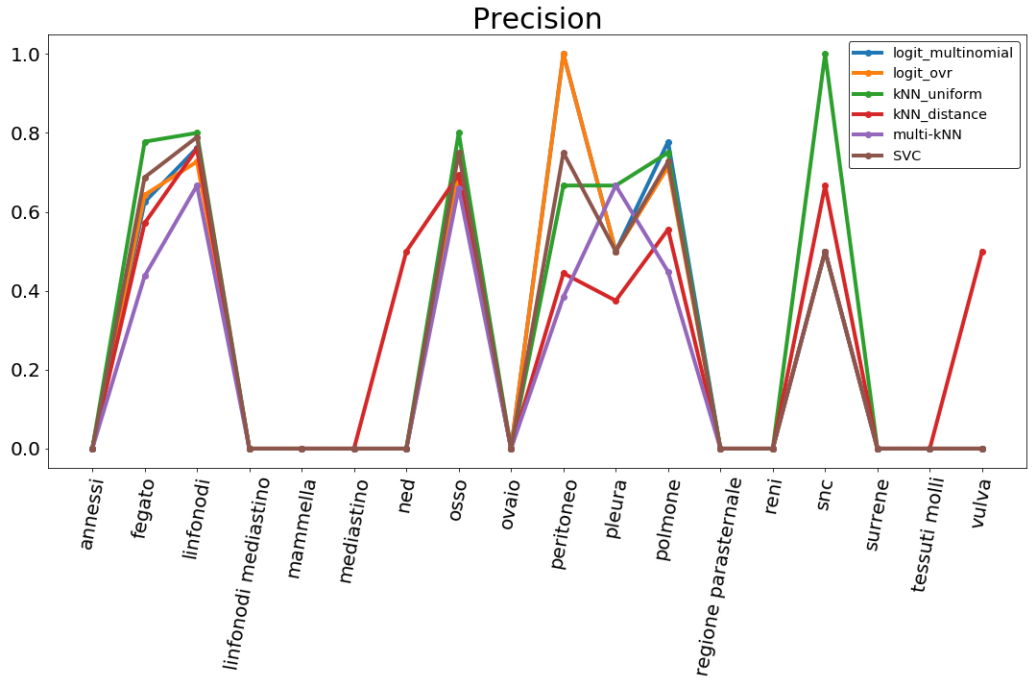


Figure 4.2: Visual representation of precision for documents embedded with doc2vec. On the x-axis, the set of labels, whereas the y-axis represents the ratio. Each line belongs to a different classifier, as shown in the legend.

— **logistic regression (ova)**

— **kNN (uniform weights)**

— **kNN (distance weights)**

— **multi-kNN**

— **SVC**

For each run, *precision*, *recall*, and *f1-score* was computed for each label, together with a comprehensive *micro-average* for each metric.

First, I will show precision, recall, and f1-scores regarding the documents embedded with doc2vec.

By inspecting figures 4.2, 4.3, and 4.4, one can immediately notice how the metrics precipitate to zero for some labels. This is probably due to the lack of enough samples, that does not allow the classifier to compute an estimate. Indeed, sklearn throws warnings in conjunction to the uncomputable labels. Apart from this, I notice that the more frequent labels (like bone, liver, lymph

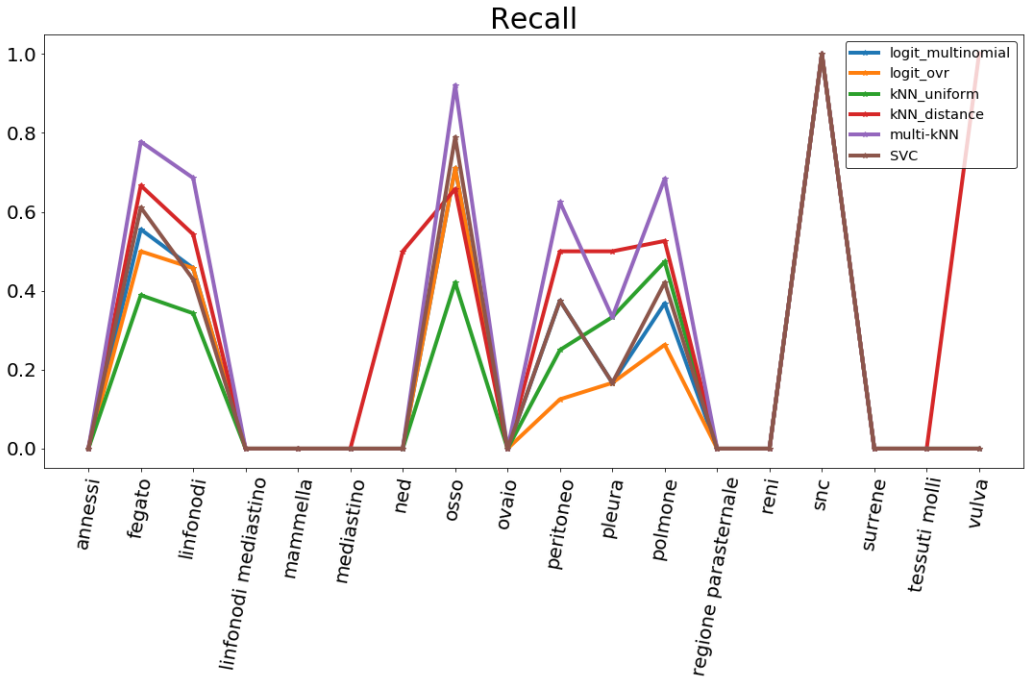


Figure 4.3: Recall for documents embedded with doc2vec.

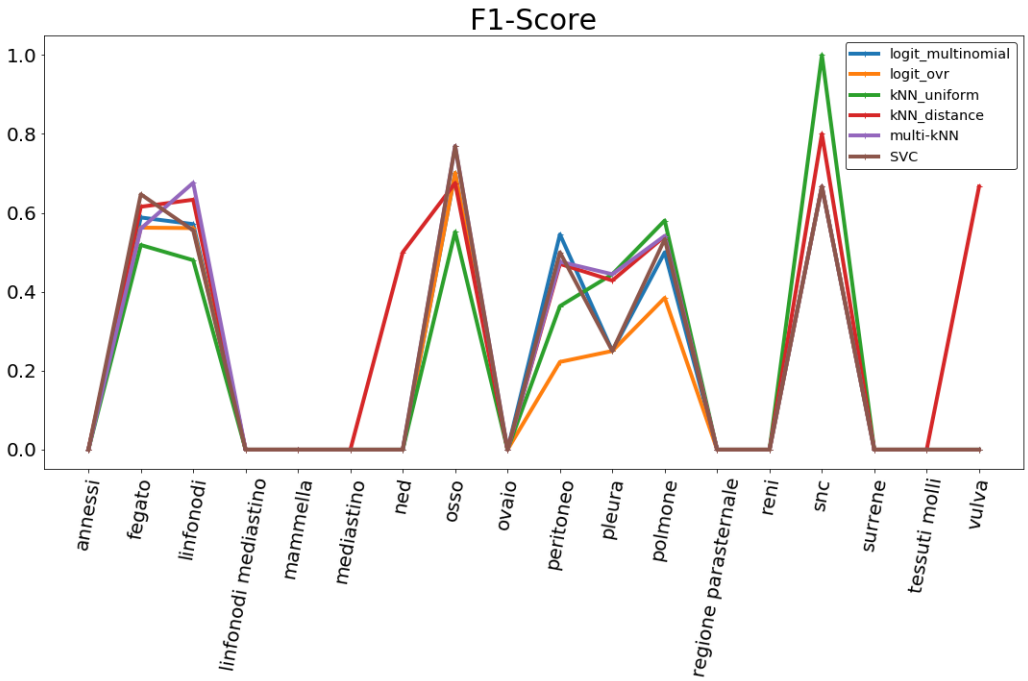


Figure 4.4: F1-score for documents embedded with doc2vec.

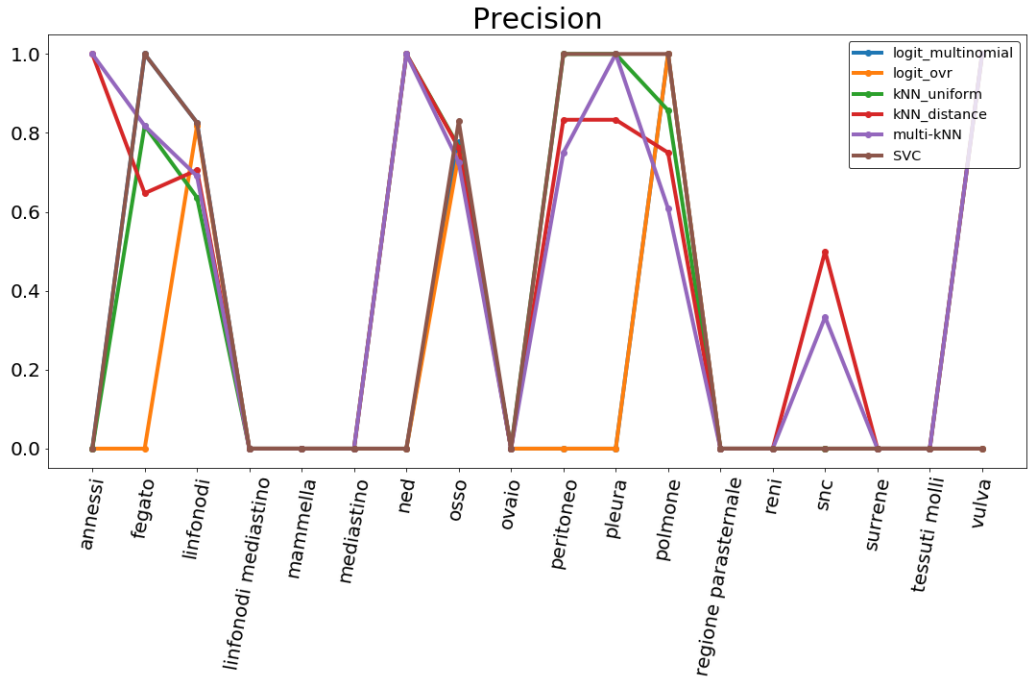


Figure 4.5: Visual representation of precision for documents embedded with TF-IDF. On the x-axis, the set of labels, whereas the y-axis represents the ratio. Each line belongs to a different classifier, as shown in the legend.

nodes) reach a decent recall and a slightly above chance-level F1-score. Not to be pessimistic, but such 'high' scores are probably due to very small sample size, as the majority of the documents appear with those labels. It would be quite like blindingly choosing those labels each and every time, knowing that most of the tosses are probably hits. On the other hand, I can't quite explain why a label like 'vulva', that appears in the dataset just 3 times, has such high metrics. I guess it depends on 'lucky' hits. This is the case also for other instances, as I deeply believe the size of the dataset and the imbalance of the labels are causing many issues to the classifiers. Regarding the differences between the classifiers, I would say that, even though I wouldn't base a qualitative evaluation with these results, the ones performing better seem to be the multi-kNN, the k-NN with weights 'distance', and the support vector classifier.

By inspecting figures 4.5, 4.6, and 4.7, I see that the results do not change significantly from those regarding doc2vec. Also here, many labels embrace zero values, while others present perfect 1s. The multi-kNN and the k-NN with weights of the 'distance' type still perform slightly better that the other classifiers. I'm not comfortable yet in assessing the goodness of the high f-1

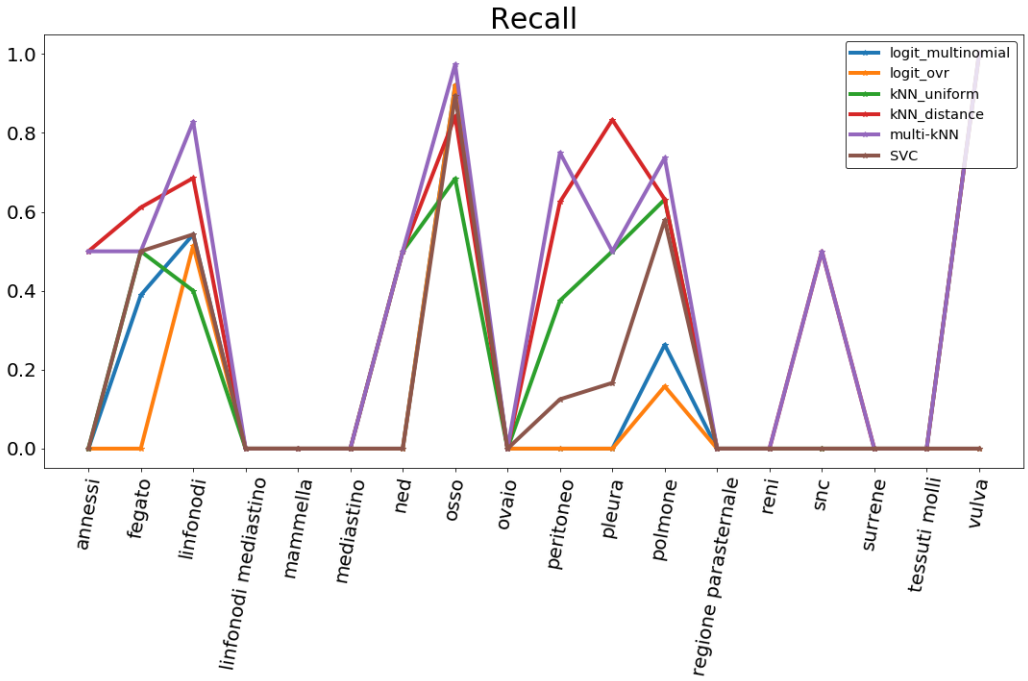


Figure 4.6: Recall for documents embedded with TF-IDF.

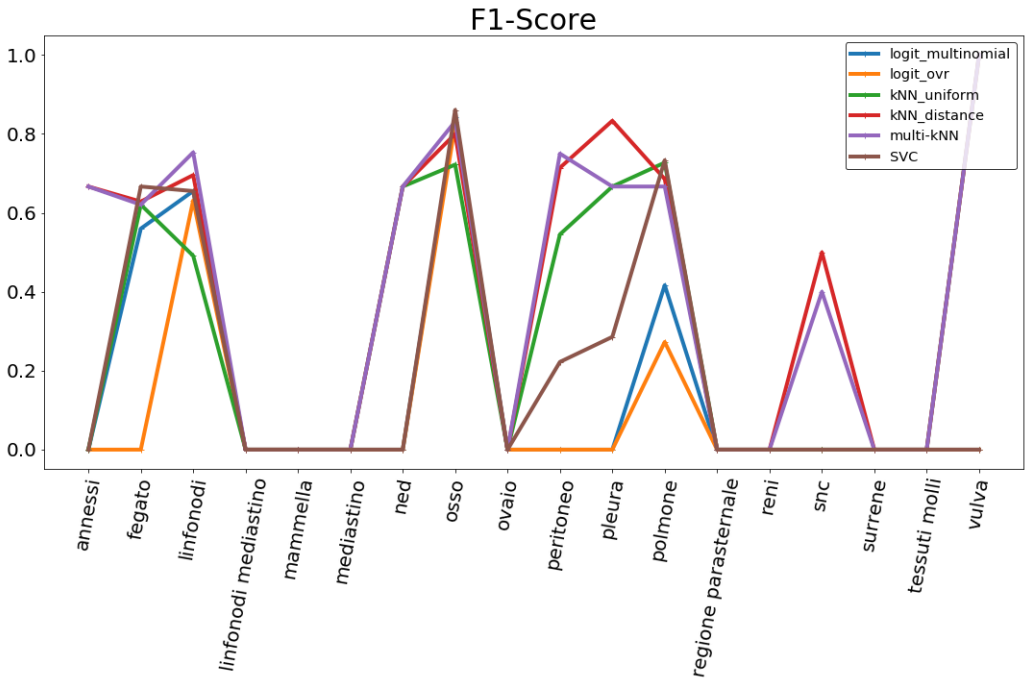


Figure 4.7: F1-score for documents embedded with TF-IDF.

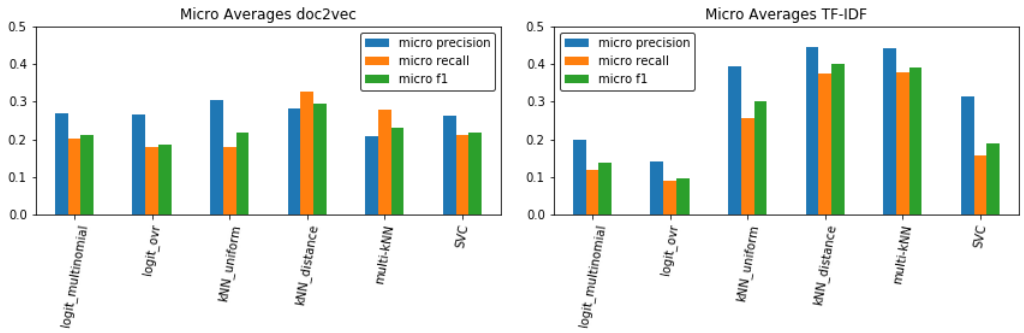


Figure 4.8: Comparison between the micro averages of precision, recall, and f1-score. On the left, the scores obtained with the doc2vec corpus; on the right, the ones with TF-IDF. On the x-axis, the algorithms used; on the y-axis, the score.

scores, as the problem of the small sample size, in addition with that of the imbalance, persist.

I also show a comparison between the different averaged metrics over all the labels. Of course, the scores are very low. In any case, it appears that overall ulti-kNN and k-NN 'distance' run on the TF-IDF vectors perform better that the others. However, on average they still are below chance. I haven't inspected slices of the dataset involving only the most frequent labels though; it's a work that can be done in the future.

4.4 Embedding and Classification with fastText

`fastText` is a library developed by Facebook AI that focuses on word embedding and text classification, written in `C++`, now supporting a Python API, and bounded to just CPU usage. Thus it is possible to train the models and run the classification both via the `Command Line Interface` (CLI) and through a Python script.

The library allows the training of cbow and skip-gram embedding, and can perform text classification for either a multi-class or a multi-label problem; in the former case the default loss function is a *multinomial logistic regression* (or softmax), whereas in the latter it employs a *one-vs-all* strategy. In the multi-class type of problem, if the number of classes is high, it also allows

[4] https://ai.facebook.com/

the use of a *hierarchical softmax* to speed-up the training without losing too much precision.

Another important feature that `fastText` exploits is that of supporting the enriching of the word vectors begin trained on the dataset with additional **pre-trained vectors**. For testing purposes, I decided to build my own custom pre-trained vectors, build from a corpus assembled with domain-specific biomedical material.

4.4.1 Domain-Specific Corpus and Vectors

Building a custom, domain-relevant corpus might benefit the training of word vectors for a series of tasks, from named entity recognition to question answering or text classification. Due to the absence of an open-source, biomedical oriented corpus, I followed some tested guidelines from [8] and [9] to build my own.

First of all, I executed a selective dump of approximately $60k$ Wikipedia articles, which was done by using Wikipedia PetScan to create a boundary around the preferred categories, and then by extracting the articles with Wikipedia's Export tool. Subsequently, I used a `perl` script, that was came with `fastText`'s repository, to preprocess the dumped articles by removing the XML tags and operating some standard text cleaning. To these, I added the results of a series of scraped medical online dictionaries. The scraping was done by means of custom Python scripts, that exploited the library `BeautifulSoup`. Each dictionary required ad-hoc HTML processing to extract the definitions. Finally, I added a series of texts and documents on a variety of medical and biomedical topics (e.g. on biotechnologies, on imaging techniques in oncology et cetera). The documents and the definitions were cleaned with removal of punctuation, conversion of accents, numbers, removal of whitespaces, tabs, newlines, and other standard preprocessing steps.

The homemade corpus now counts more than $33M$ tokens, with $443k$ being unique.

By no means this is a big corpus compared to the ones built by other laboratories and research groups, or compared to the general-purpose word vectors trained on Wikipedia by `fastText` itself; on the contrary, it is probably still quite small, but it has the potential to grow and serve better for its purposes. Nonetheless, notwithstanding some noise that hopelessly enters with e.g. the dumping of the articles, this is a vocabulary that is well oriented to the biomedical domain, with particular attention to oncology and

[4] https://petscan.wmflabs.org/
[4] https://it.wikipedia.org/wiki/Speciale:Esporta

medical imaging.

I trained the word vectors for both the continuous-bag-of-words and the skip-gram models; I run `fastText`'s unsupervised training with `C++` in the command line, by providing the following parameters:

```
./fasttext skipgram \
    -input /path/to/corpus \
    -output /desired/output/name \
    -lr 0.3 \
    -dim 300 \
    -epoch 50
```

To train the cbow, just replace "skipgram" with "cbow". The learning is set to 0.3, higher rates would produce `NaN` that crashed the training. The output are two files, a `.bin` file that contains the whole model, while a `.vec` is a text file and contains just the words vectors.

4.4.2 Data Format

To perform classification tasks, `fastText` requires the data to be formatted in a specific fashion. Both the validation and the training data must be provided in the following manner:

__label__<label_tag> <text>

One document per line. If a document comes with more than one label, the `__label__<label_tag>` for each present label simply are separated by a whitespace.

Training and testing sets were generated by first shuffling the whole dataset in the format just mentioned (with `shuff bash` utility), and then by splitting the first 80% of the documents to training set and the bottom 20% to the validation set (I used `head -l num_lines` and `tail -l num_lines`).

4.4.3 Training and Testing Procedures

For multi-label text classification, I trained the supervised model on the training reports using a window of `wordNgrams` of 3 (in the literature, a window of 2 or 3 is suggested), a learning rate of `1e-6`, a vector dimension of 300, and for a different number of epochs, namely 25, 50, 100, 250, 500, 1000. The learning rate was chosen after testing with higher values, but resulting in the killing of the job due to the generation of `NaN` values. The choice of the epochs instead was due to the fact that for a small of a dataset as the one of the present work, the result might be broken with a low number

of epochs (e.g. when testing around 5-25 epochs, the probabilities returned for the predicted labels were exactly the same for each test sample). Thus, the decision of testing different configurations. Furthermore, the chosen loss function was `one-vs-all`, that applies a multinomial logistic regression in a one-vs-all strategy.

Finally, for purpose of testings, I trained the model both with `pre_trainedVectors` computed with the custom corpus and without.[5]

```
for epc in 25 50 100 250 500 1000;
do
    ./fasttext supervised \
        -input path/to/input/training \
        -output path/to/output/name_${epc} \
        -wordNgrams 3 \
        -lr 1e-6 \
        -epoch ${epc} \
        -bucket 200000 \
        -dim 300 \
        -loss one-vs-all \
        -pretrainedVectors path/to/vectors \
        -seed 42
done
```

The output is a `.bin` file containing the model, that can be used for testing the classifier on the validation set by providing the number of labels the algorithm should predict at max (a value of -1 means that it should predict as many labels as possible) and a probability threshold under which the labels are not considered not predicted (in this case 0.5):

```
./fasttext test model.bin -1 0.5
```

The testing returns two metrics, namely **precision@1** and **recall@1**, which simply are the precision and the recall introduced in the previous section about the `sklearn`'s metrics.

We can also test single documents and be provided with the predicted tags:

```
./fasttext predict-prob - -1 0.5
```

The single dash will let you input a string as `stdin`, after which one can get the predicted labels with the respective probability.

[5] An interesting comparison would with the model trained on the Italian Wikipedia's vectors, but at the moment of writing I am having some issues with crashes, due to the size of vectors (more than 4GB)

4.4.4 Results

I need to warn the reader about this section, as the results are not at all satisfying, and probably not yet properly understood. My guess is that the poor size of the dataset, in conjunction with a high imbalance in the labels and the lack of minimum sample sizes to meet the assumption of the softmax, have produced very poor and strange results. In figure 4.9, one can see respectively how, based on the vector training with the custom built corpus or without any pretrained vectors, precision and recall vary when the number of labels to predict augments (the final number is 17, as one label was eaten by another: 'mediastinum lymphnodes' into 'lymphnodes'). Overall, precision is very poor and recall is inversionally proportionally high, which to me is quite strange. One reason for this could be that `fastText` is simply outputting all the `k` possible labels for `k` in range 1-18, without having a stopping decision on whether not giving a label if the probabilities are too low. By doing this, it basically providing the whole set of tags, which indeed represent meaningful hits when computing the recall. One possible stop criterion could be inputting the previously mentioned *probability threshold*, but in this case only the label '*bone*' would survive the 0.5 threshold, and by a very tiny margin, at which point it would be a coin toss. However, it is nice to notice the marginal but hopeful effect of pretrained vectors.

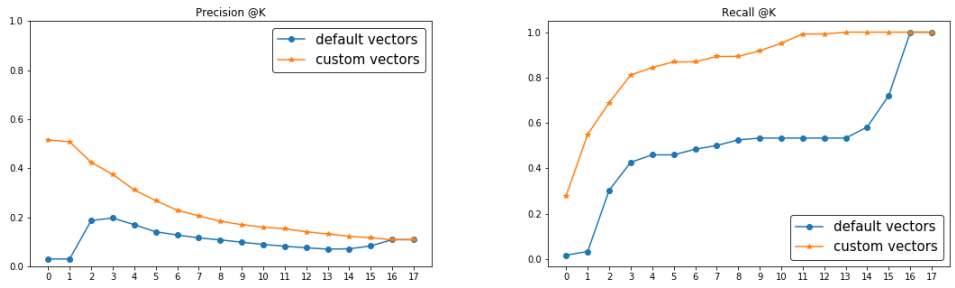


Figure 4.9: On the left, the behaviour of precision by varying the number of labels to predict. On the right, the recall. Both plots show the result based on both type of vectors' training, namely with the pretrained custom corpus and without.

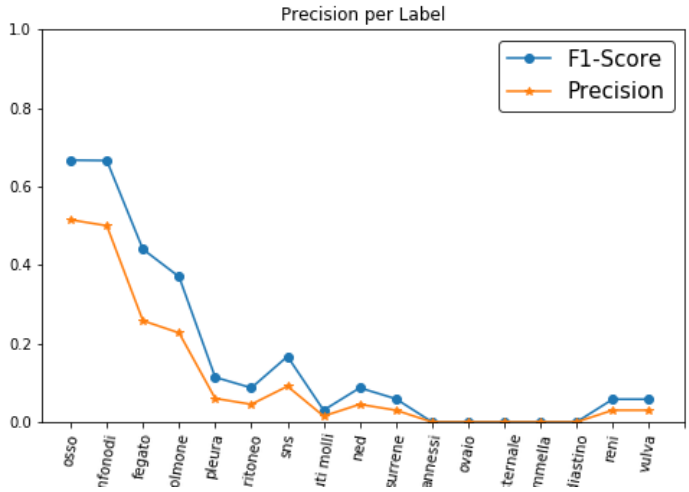


Figure 4.10: F1-Score and Precision for each label on predictions based on the dataset trained with the pretrained vectors. Again, some labels are not at all considered, probably due to lack of samples.

4.5 Multi-Label Classification with Fast-Bert

In this section I introduce `fast-bert`[6], a library based on Google's `BERT` (Bidirectional Encoder Representation from Transformers)[7] that was created to deal with multi-class and multi-label text classification.

4.5.1 The Attention Mechanism

In few words, in tasks that need to produce a sequence output form a sequence input, for example in language tasks, we want the model to 'remember' certain states, in order to compute the training in light of those states. Recurrent Neural Networks (RNNs) try to remember those states by passing the information to 'copies' of the same network (by looping) in different time steps. One the main culprits arises whenever the network is asked to generate a step of the sequence contextual to some steps that appeared with an important time gap. RNN simply are not comfortable with this bigger gap, due to the possible loss of the information during the chaining process. Furthermore, the network modifies existing information every time new information is added, leaving no room for giving importance to specific information. For this reason, a special type or RNN was proposed to remedy to these issues,

[6] https://github.com/kaushaltrivedi/fast-bert
[7] https://github.com/google-research/bert

called Long Short Term Memory (LSTM). The main advantage it has is that of 'remembering' certain things considered important thanks to *cell states*, the mechanism through which the information flow. Each cell state takes as inputs the previous cell state and its output, and generates a new cell state along with an output. However, LSTM network seems to suffer of the same limitations of RNN, i.e. the probability of forgetting important context when the sequences get too long.

A solution might come by building the network with an *Attention* technique, i.e. with a focus on parts of a subset of the information provided. For example, RNN, instead of encoding whole sentences in a hidden state, can encode each word in separate hidden states, which are then all passed through to decoder stage. However, RNNs cannot parallelize the sequences processing. This can instead be done by Convolutional Neural Networks (CNNs), given that each word input can be processed at the same time, not depending on previous words to be operated on. However, even CNNs lack a solution on the problem of short and long term dependencies.

4.5.2 BERT and Transformer

`BERT` (Bidirectional Encoder Representation from Transformers) is a recent NLP model that has reached outstanding results in many tasks [18]. The peculiarity of the model is being based on the bidirectional training of the Transformer architecture. In their original paper, the encoder stage is composed by a stack of six enconders, with the same structure holding true for the decoding stage. Each encoder is built with a *self-attention* layer that looks at the other elements of the sequence while holding on to the current word; and with a feed-forward network layer, that receives the output of the previous layer. The decoders are similar, but integrates an intermediate attention layer that focuses on meaningful parts of the input sequence.

Without delving too much into the intricacies of the whole architecture[8], I just want to add that the bottom-most encoder in the stack takes as input vectors of sequences (of max length 512), which represents the embedding layer. On the other hand, the up-most decoder outputs a vector of floats that is fed to a *linear* layer that that projects the vector into a larger space that in return is fed to the final *softmax* layer that returns the predicted token based on the log probabilities computed.

The strength of `BERT` lies in the *pre-trained* models that come with it. For example, one can make use of the `bert-large-cased` model, that was trained

[8]This blog post has a marvelous description of the architecture and provides a nice visual inspection http://jalammar.github.io/illustrated-transformer/

with an architecture of 24-layers, 1024-hidden, 16-heads, 340M parameters on a huge corpus with punctuation, accents, cased letters and such.

`BERT` comes with a series of pipelines for several tasks, such as NER, question answering, text generation, and sequence classification. This last procedure is important for the scope of this work, but the model itself doesn't fully support whole documents classification. Fortunately, there are good souls in the world that have extended the sequence classification functionality to address whole-text labeling.

4.5.3 Fast-BERT

`fast-bert` is a library built on `BERT`, that exploits the HuggingFace's PyTorch API [22].[9]. Specifically, it extends the sequence classification pipeline (based on the class `BertForSequenceClassification()`) to include whole documents classification. The main change is the use of **Binary Cross-Entropy with Logits** as a loss function instead of the original *Cross-Entropy* loss. The new loss function produces independent probabilities for each label.

The architecture is made of 1 embedding layer, based on Bert's tokenizer; 12 BERT's attention layers, and a classifier layer.

The classification pipeline runs through a `run_classification.py` script, that can be tuned with a set of hyperparameters, such learning rate, maximum sequence length (max is 512), number of training epochs, use of GPU, warmup method et cetera. An important argument to provide is the pre-trained model to use. For a task involving general-purpose language, the use of `bert-large-cased` could be desired. However, when dealing with a specialized topic such as radiology reports, a more domain relevant vocabulary could be used. Fortunately, different research groups have trained BERT architectures on large corpora of scientific documents, creating a set of specialistic BERT-based models. Two of them are **SciBERT**[10] and **BioBERT**[11], which reached state-of-the-art evaluations on a set of tasks, including NER and sequence classification. Both models are open-source with the respective weights, vocabulary, and configuration file.

For this work, I made use of `BioBERT`'s **biobert_v1.1_pubmed**[12] model, after having it converted for PyTorch from the original TensorFlow

[9] https://github.com/huggingface/transformers/tree/master Originally, Google open-sourced the TensorFlow implementation, that was used by HuggingFace developers to port it in PyTorch.

[10] https://github.com/allenai/scibert

[11] https://github.com/dmis-lab/biobert

[12] https://github.com/naver/biobert-pretrained

implementation.

To use the model, it's necessary to first fine-tune it on the dataset of interest, by having it training for a number of epochs. After the fine-tuning, the classification can be run. To fine-tune the model, one just needs to provide the `run_classification.py` with the arguments `-do_train` and `-do_eval`. After this tuning, the argument that needs to be passed is `-do_predict`.

4.5.4 Reports' Translation

To exploit the goodness of this model with pre-trained weights specific to the biomedical domain, we need to make use of a vocabulary built on the English language. This poses a problem, as the reports at hand are written in Italian.

Due to time constraints and the limited custom corpus I built, which would be quite wasted for a model like this, I decided to translate my dataset into English. To achieve this goal, I used custom Python script that make use of Google Translate's API.

The result is quite outstanding, as the specificity of the biomedical language seems to be well maintained. Surely, some noise is added due to repetitive terms or unrecognized words kept in Italian; however, a visual inspection noticed very few occurrences.

4.5.5 Results

`fast-bert` was run in Google Colab on GPU. The dataset was splitted in training and testing set by providing two `.csv` files with a number of columns equal to `N_labels` + 1. This is because the first column contains the reports, and the rest contain the binarized labels, one per column, as specifically requested by the implementation. Contrary to the needs of previously presented embeddings, the radiology reports fed to the BERT's tokenizer were just the extracted portions, without further cleaning from punctuation or accents.

The model was fine-tuned on `BioBERT`'s weights on the dataset at hand, and then run on a set of epochs, namely 10, 20, 30, 40, 60, 80, and 100. The `max_sequence_length` parameters, i.e. the one that sets the limits to the size of the chunks of tokens processed at each step, was tested with the values of 256 and 512. The values being powers of 2 (up to 512), ideally the chunk size should be able to include whole sentences; if not, the sentence is truncated, with the risk of losing information. The *batch* size was set at 8, as a higher number would risk OOM (Out Of Memory) runtime errors on the

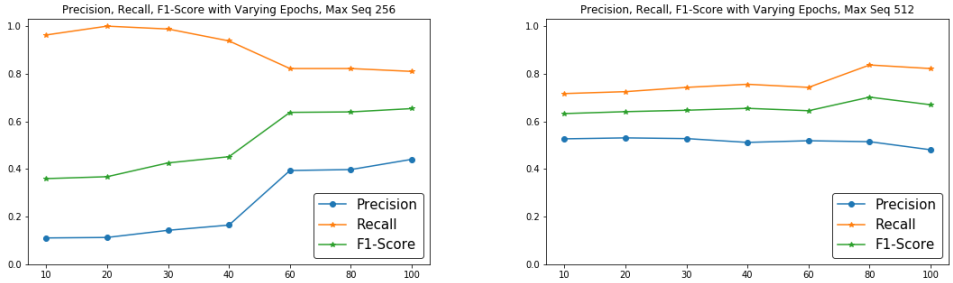


Figure 4.11: Progression of Precision, Recall, and F1-Score with varying number of epochs (10, 20, 30, 40, 60, 80, 100) and for different chunk sizes (256, 512).

GPU at use. The chosen optimizer was `LAMB` (to speed up the computation, but `Adam` is also supported); learning rate was set at `1e-5`

Overall, results are still very poor, with precision being at best almost at chance level. I notice that the F1-Score is ameliorating with higher numbers of epochs for both chunk sizes, with the best result obtained with chunk size 512 and 100 epochs (F1=0.702). I believe that these results show hope in the use of these models, but it would be better to evaluate them with in light of a bigger and well-balanced dataset.

Figure 4.13 shows a comparison of the averaged metrics across all the implementations, with the exception of fastText, for its results were the more obscure to interpret. One can clearly see how Fast-Bert peeks above the others. I am not sure this is due to poignant goodness of the model even in the circumstance of a dataset like the present; it could be that the high scores depend on 'lucky' decisions based on the prevalence of a limited number of labels, and on the contemporary absence of most of the others. Nonetheless, the results from this model are without a doubt promising.

[12]It would be interesting to investigate each metric for each single label, but at the time of writing this implementation is not supported, so a custom one is being studied.

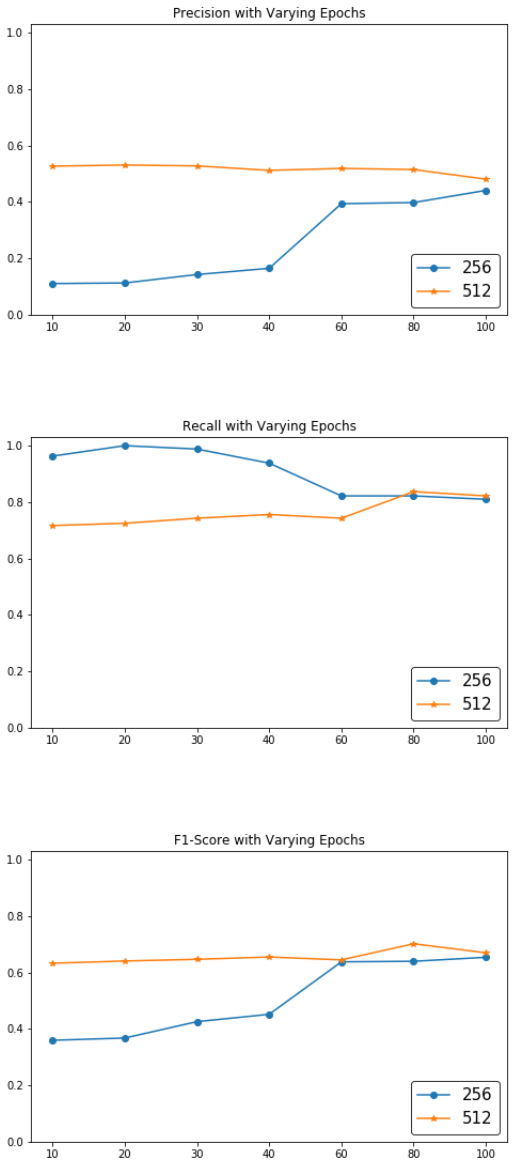


Figure 4.12: Precision, Recall, and F1-Score compared with different chunk size (256, 512).

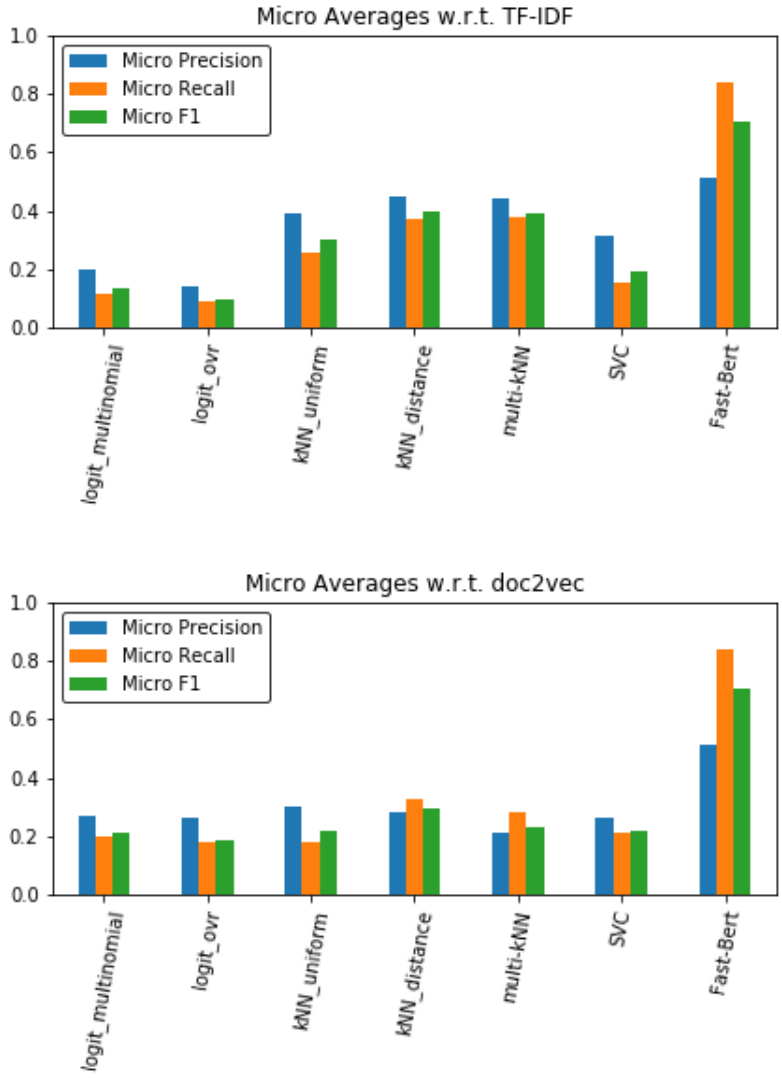


Figure 4.13: Comparison of the micro averages for Precision, Recall, and F1-Score across all the implementations (exception made for fastText). On the left, the non-BERT algorithms trained on TF-IDF vectors, whereas on the right trained with doc2vec.

Chapter 5

Conclusions and Future Directions

The present thesis project wanted to investigate the possibilities of performing a multi-label classification of radiology reports, classification based on the locations of the metastasis indicated within the free text descriptions contained in the records. Humans usually have no difficulties in retrieving information from natural language, being it spoken or written. Moreover, they overall have very good abstraction skills for interpreting complex linguistic structures that convey some specific information. For example, a series of apparently nested subordinated sentences might circumstantiate a concept without directly reference. However, the automation of such tasks of information retrieval (and classification) require the exploit of computational platforms. These complex linguistic, semantic, conceptual operations that are dealt with in natural language are not ordinarily well suited for algorithms that usually crunch numeric data.

In order to fully make use of the computational prowess of pattern recognition's techniques, we first need to transform the problem: we need to map words, sentences, and even documents in numeric vectors; this operation is called "embedding". To achieve this, different methodologies were created. In this work, I explored a set of them, namely TF-IDF (Term Frequency-Inverse Document Frequency), doc2vec, fastText's embedding, and BERT's. On top of these different manners for computing numerical vectors from natural language, I tested different algorithms for multi-label classification.

The topic of multi-label classification, and especially that of text multi-label classification, is yet not fully investigated. Only in recent years, with the advent of the aforementioned embedding techniques, loads of researches are being developed to exploit automation of language tasks. However, some methods for classifying samples with a series of tags already exist, and they

make use of some tweaks to transform the problem in either a binary or a multi-class problem. Specifically, I used multinomial logisitc regression, one-vs-all logistic regression, support vector classifier, k-Nearest-Neighbors, and multi-k-Nearest-Neighbors.

Overall, the results were not satisfying at all, most probably due to the extremely small dataset and the high categories' imbalance. By exploring the literature, these same techniques provide very good results whenever applied to decent-sized datasets (in the order of the tens of thousands of samples at least). Notwithstanding these limitations, I could notice that some methodologies cast hopes on the future of the topic. Especially, BERT based models, i.e. neural network architectures built on top of stacks of attention layers, have provided some good praises even within these rough circumstances.

Surely, with more time at hand, the models could have been explored more thoroughly, and integrative solutions could have been provided. One strategy that might be explored is that of tagging the text with ad-hoc notations (a task called *Named Entity Recognition*) to help the classifier layers of a network to focus on more relevant information [10][11]. This strategy is indeed being explored at the time of writing, though unfortunately results are still not educated enough to be presented. More specifically, I am making use of the previously mentioned HuggingFace's library to perform a named entity recognition task on my dataset, by tagging each token with respect to a specific notation, i.e. "B-I-O": "B" stands for "Begin", "I" for "Inside", and "O" for "Outside". Simply put, single tokens or sequences of tokens might delimit a particular instance, a concept, an important part of the sentence. For example, we might want to highlight, in the sequence "malignant masses with respect to the iliac structures", "malignant masses" and "iliac structures" as relevant to the oncological landscape (like the one of the present work). Thus, we could tag the series "malignant masses" as "B-tumor I-tumor", "iliac structures" as "B-anatomical_site" I-anatomical_site", and the rest of the tokens in the sequence as "O" (outside of the areas of interest).

Several libraries, especially BERT based ones, have released their own biomedical domain-specific weights to train models on a series of natural language tasks[1], one of these being named entity recognition. Different datasets are used for the training of this particular task, but I could find one specific to the oncological domain only very recently (the AnatEM dataset[23]). One very important information to provide is that these models are English language oriented, thus I am exploring such possibilities only with translated reports. In an ideal situation, the dataset would be manually annotated by experts, that would also supervise the correctness of the automatic labeling[8].

For sure, the most important element to be able to explore possibilities regarding multi-label text classification is the size of the dataset. Moreover, a uniformly distributed set of labels would be appreciated by the mentioned methodologies. The larger the data pool, the more exploitable are deep learning libraries that can work without too much manual work of refinement on the dataset itself; vice versa, a small sample size would most definitely need many relevant markers to be directed to the attention of the algorithms.

[1] At the present time, the writer has just uncovered more clinical domain-relevant models, like ClinicalBERT (https://github.com/kexinhuang12345/clinicalBERT)[24][25]

Bibliography

[1] Olive Peart. Metastatic breast cancer. *Radiologic technology*, 88(5): 519M–539M, 2017.

[2] JR Ball and E Balogh. Improving diagnosis in health care: Highlights of a report from the national academies of sciences, engineering, and medicine. *Annals of internal medicine*, 164(1):59–61, 2016.

[3] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[4] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning in natural language processing. *arXiv preprint arXiv:1807.10854*, 2018.

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

[6] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[7] Emanuele Pianta, Christian Girardi, and Roberto Zanoli. The textpro tool suite. In *LREC*. Citeseer, 2008.

[8] Francesco Gargiulo, Stefano Silvestri, and Mario Ciampi. Tecnica per l'annotazione di un corpus per l'addestramento di un sistema deep learning per biomedical named entity recognition.

[9] Anita Alicante, Anna Corazza, Francesco Isgrò, and Stefano Silvestri. Semantic cluster labeling for medical relations. In *International Conference on Innovation in Medicine and Healthcare*, pages 183–193. Springer, 2016.

[10] Natalia Viani, Timothy A Miller, Carlo Napolitano, Silvia G Priori, Guergana K Savova, Riccardo Bellazzi, and Lucia Sacchi. Supervised methods to extract clinical events from cardiology reports in italian. *Journal of biomedical informatics*, page 103219, 2019.

[11] Alfonso Emilio Gerevini, Alberto Lavelli, Alessandro Maffi, Roberto Maroldi, Anne-Lyse Minard, Ivan Serina, and Guido Squassina. Automatic classification of radiological reports for clinical care. *Artificial intelligence in medicine*, 91:72–81, 2018.

[12] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[13] Ram Frost, Blair C Armstrong, and Morten H Christiansen. Statistical learning research: A critical review and possible new directions. *Psychological Bulletin*, 145(12):1128, 2019.

[14] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[16] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[20] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[21] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[22] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[23] Sampo Pyysalo and Sophia Ananiadou. Anatomical entity mention recognition at literature scale. *Bioinformatics*, 30(6):868–875, 2013.

[24] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019.

[25] Emily Alsentzer, John R Murphy, Willie Boag, Wei-Hung Weng, Di Jin, Tristan Naumann, and Matthew McDermott. Publicly available clinical bert embeddings. *arXiv preprint arXiv:1904.03323*, 2019.