Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

---

Tesi di Laurea Magistrale in Ingegneria Informatica

Prediction of variables from cancer reports

Candidato
Federico Magnolfi

Relatori
Prof. Paolo Frasconi
Prof. Simone Marinai

Correlatori
Leonardo Ventura, ISPRO
Stefano Martina, PhD

Anno Accademico 2019/2020

Abstract

Doctors who follow oncological patients write a medical report in natural language after each visit.

There are institutions (such as ISPRO) that collects these reports to do statistical analyses. They have experts that manually analyze these reports to extract variables: a big problem in this process is that there are about 5 years of delay.

In this thesis, we investigate whether it's possible to algorithmically analyze these reports. This could help humans to keep up and analyze all data without so big delays. We focus on a dataset of breast cancer reports, collected in Tuscany.

After an in-depth analysis of the dataset, we propose different approaches for two distinct variables groups. For the first group we compare many Machine Learning and Deep Learning models, experimenting that XGBoost and Random Forests perform better than Transformer-based Neural Networks. For the second group of variables, we extract predictions from text with a regex-based algorithm.

We show that it is possible to achieve very good results for many of the variables of interest.

Contents

Chapter 1

Introduction

The idea of this thesis arose from the need to support an ISPRO project funded by the pharmaceutical company Roche.
ISPRO (Istituto per lo Studio, la Prevenzione e la Rete Oncologica; Institute for the Study, Prevention and Oncological Network) is the institution that collects oncological reports in Tuscany.
We investigate different algorithms to predict variables from Tuscany cancer reports.

1.1 Motivations

This thesis is a part of a project where the aim is to find the right tools to ensure that cancer patients (breast and colorectal tumors in particular) follow the diagnostic and therapeutic care pathways (PDTA) according to the guidelines issued by the Tuscany Region. We would like to know if, how, where and when these guidelines are respected in Tuscan hospitals.

ISPRO (Institute for the Study, Prevention and Oncology Network) is the institute that collects and analyzes the reports of cancer patients from

all over Tuscany.

These reports are transferred with the help of computer systems, and then manually analyzed by the experts to extract interesting information such as the location of the tumor, the stage, the size, etc.

From this extracted information, it's possible to create statistical indicators that allow us to understand if the trend of the therapeutic pathways is aligned with the guidelines of the region.

A major problem in this process is that cancer registries accumulate delays of many years in analyzing reports. In Tuscany there are about 4/5 years of delay, in the other Italian regions there are always at least a couple of years of delay.

These delays have the effect of postponing the study of any problem in the therapeutic pathways, preventing a timely resolution.

1.2 Previous work

In [1] the authors developed several classifiers to predict topography and morphology ICD-O codes [2] on cancer reports collected by ISPRO between 2004 and 2013. They observed that deep learning approaches did not significantly outperform classic machine learning methods. Furthermore, they noticed that hierarchical models do not work better than non-hierarchical ones, as opposed to what is observed in other languages [3] [4]. This can be explained by how are these reports written: there are mainly keywords, almost no verbs.

1.3 This thesis

The objective of this thesis is to verify whether it's possible to predict variables from reports.

This study is limited to breast cancer reports: in future works, obtained results must be extended to other types of cancer.

There are many variables that can be interesting to predict on these kind of reports: we will analyze the predictability of each of them, in the data provided to us by ISPRO.

We firstly explore the dataset to understand the data we have to use: we investigate how the reports are written, what are distributions of the labels to predict, cleaning the data whenever necessary.

We compare different types of algorithms, from simple and interpretable as decision trees to more advanced algorithms that are the state of the art in Natural Language Processing, such as Transformer Neural Networks.

1.3.1 Question

In this thesis, we want to answer the following question:

- Is it possible to predict variables from these reports?

The answer is not obvious, since there is no previous attempt in predicting these variables from these data. In the previous work [1], the authors predicted other variables.

1.4 Document structure

In chapter 2 we will describe how cancer registers are created, with a particular accent on how it was made in Tuscany.
In chapter 3 we will expose the theory behind different methods to handle and represent text data.
In chapter 4 we will expose the theory of different kind of learning algorithms that can be used on text data.
In chapter 5 we will analyze the dataset extracting many statistical distributions.
In chapter 6 we will explain how the work is organized.
In chapter 7 we will list all the experiments done, reporting the obtained results.
In chapter 8 we will recap all the work done and suggest possible future works.

Chapter 2

Cancer registers

A doctor who visits a patient has to interpret the exams made, perform diagnosis and choose which is the best way to continue the therapeutic path of the patient. After each visit, the doctor writes a report in natural language.

The doctor cannot afford to catalog the report for various reasons: precisely determine the category of the report would require a long time. Furthermore, a report should be cataloged according to more criteria. These criteria can also be different depending on the tumor under review, and sometimes it is necessary to look for the information by looking at the whole patient's clinical folder, and the last report alone would not be enough.

For these reasons, the work of cataloging reports is not made by doctors who make diagnoses, but it is done in special institutions such as ISPRO.

Usually, these institutions collect data from the whole region (or simply from many hospitals) and have experts able to analyze reports.

2.1 Statistics

In Italy, there are about 1000 diagnoses of new cases of cancer per day, a value almost unchanged for some years. In 2017 alone, there were nearly 13000 deaths from breast cancer, the most common cause in women. In the same year, the total number of deaths due to cancer was 80000 for women, 100000 for men. The total number of deaths was about 650000: cancer was the cause in almost 28% of the cases. [5]

2.2 Motivations

Each patient must receive appropriate care for his/her situation. If they do not receive enough care, therapy may not be sufficient, with serious consequences. On the other hand, if they receive superfluous care, you create unnecessary anxieties and concerns with no advantage in the final result, as well as avoidable time losses (for patients and doctors) and economic (for the healthcare system and for pharmaceutical companies). If the patients receive incorrect care, you have both the above disadvantages.

2.3 Tuscany cancer reports

ISPRO is collecting cancer reports in Tuscany since 1985. From 1985 to 2012, reports came from Florence's and Prato's provinces only, except for 2004, when they made an attempt to collect reports from all over the region. Since 2013, reports are regularly collected from all the provinces of Tuscany.

Chapter 3

Handling text data

Natural Language Processing (NLP) studies the interactions between computers and human language data. In this thesis we focus on Natural Language Understanding, the subfield of NLP that regards extract useful information from text.

Our data are in the form of ASCII text: text processing requires several steps and hides several pitfalls. We will talk about document and clinical record interchangeably, leaving aside the fact that a record is made by various reports.

In the next sections we will describe some NLP tasks, discuss about difficulties of text processing, and explain how we can represent text to be understandable by an algorithm.

3.1 Common NLP tasks

NLP techniques can be used in many applications to solve a vast range of problems. Of all these tasks, we present some that can be formulated in the context of this thesis:

- **document classification**: is the act of determining the category of the document, over a pre-defined set of categories;
- **document regression**: is similar to document classification, with the difference that the prediction is a numeric value;
- **information extraction**: extract relevant information from the text.

Example of record classification is determining the *grading* of the patient's cancer: there are four different categories that can identify the grading [2].

We evaluated whether to formulate the prediction of some variables as a record regression problem: variables that indicates the size of the tumor or the percentage of a particular marker, do not have pre-defined classes, but they can assume any value (in a certain range). Though, we do formulate all the predictions as classification or information extraction problems rather than regression ones, as explained in section 6.2.

In *information extraction* the aim is to identify where are the relevant information in the text, and extract them. The focus of the algorithm is on determining the characteristics of the text that are helpful to locate the information of interest, rather than trying to immediately give the prediction.

3.2 Difficulties

From a written text, our brain is capable of understanding its meaning, imagine real-life objects, remind other concepts. As we read we use our knowledge to give context to the words, and we are able to read even in the presence of syntactical errors, missing words, typos, abbreviations.

If we want to automatically process a text while achieving human-level performance on a certain task, we need to design algorithms that present the

properties just mentioned. The steps that we will describe shortly must be designed with this objective in mind.

3.3 Preprocessing

Text preprocessing is an optional but important step for NLP tasks. It transforms text into a more digestible form so that machine learning algorithms can perform better.
Examples of preprocessing operations:

- make all letters upper-case or lower-case;
- remove strange characters, such as those originated by a bad encoding;
- stemming: truncate the final part of the word;
- lemmatization: substitute words with their lemma;
- handle accents, by either removing them or the whole letter;
- convert different words into the same word, if we know they have the same meaning for the desired task.

We have to decide which of these operations are suitable for the desired target, also considering the texts we work on, their language, and the amount of data we have. With a small dataset like the one we work on (it has about 100k reports) the preprocessing is even more important: the fewer data there are, the more difficult it will be for the learning algorithm to extract meaning and similarities among words. Also, note that this is a very specialized domain and there are no similar datasets publicly available.

The preprocessing step it's not completely independent of the next one, the tokenization: a different preprocessing yields a different tokenization,

even with the same tokenizer.

Preprocessing at high level

`String rawText` → `String cleanedText`

3.4 Tokenization

Tokenization is a mandatory step for NLP tasks: it is a step required by both traditional machine learning algorithms and those based on deep learning. Tokenization consists of dividing the text into chunks called tokens.
There are different types of tokenization:

- character tokenization: each token is a character
- n-gram characters tokenization: each token is a sequence of n characters
- word tokenization: each token is a word
- n-gram words tokenization: each token is a sequence of n words

The set of all tokens obtained from tokenizing the whole available corpus is the dictionary. Each token in the dictionary must be associated with a unique index (an integer number) to be used by the learning algorithms.

Tokenization at high level

`String text` → `List<Integer> tokens`

3.5 Representing documents

If we are interested in classify documents, a possible strategy is to define a representation of a document, and then apply a classifier on this representation.

The simplest method is to represent a document with the set of tokens that are in the document, without caring about neither the tokens relative order, nor the number of occurrences of each token. This representation is called **bag-of-words**: the idea is that the presence or absence of certain tokens (words) is a good way to characterize the document. Mathematically, a bag-of-words of a document is a binary vector as long as the dictionary size, with value 1 at indexes of tokens that are present in the document, and 0 elsewhere.

One variant of the bag-of-words representation is the **TF-IDF**: the idea is that a token have to be more important if it appears many times in the document and if it is usually a rare token. The TF-IDF vector have floating point numbers, not boolean: the value to put in the vector is calculated using the Term Frequency (TF) and Inverse Document Frequency (IDF) of the token. Mathematically, the ith element of the jth document is:

$$v_{ij} = TF_{ij} * IDF_i$$

where:

$TF_{ij} = \#$ of occurrences of token i in document j

$IDF_i = -log(n_i/N)$

$n_i = \#$ of documents with token i

$N =$ number of documents

3.6 Representing words

Both bag-of-words and TF-IDF don't care about the relative order of tokens in text: this information is lost. A way to overcome this problem is to start from a representation for each token and aggregate them to construct the representation of the document, considering also the position of the tokens.

A possible representation for a token is a **word vector**. Firstly introduced in [6], the idea of word vectors is that each token (word) is a vector in the words space. In the original work, these vectors are randomly initialized and then learned by predicting contextual words (CBOW variant) or by predicting the word given his context (Skip-Gram variant). Word vectors have the ability embed multiple meanings of the word (such as gender, plurality...) [7].

Having a distributed representation in a continuous space allow models to achieve a high level of generalization, since similar words are likely to have similar vectors. That is not possible with classical n-gram language model, because each n-gram has no inherent relationship to one another.

A representation of the document is obtained by applying a function on them. This function can be pre-determined like the *max* function, or it can be learned, e.g. using Recurrent Neural Networks or Transformer-based Neural Networks.

Chapter 4

Learning

4.1 Supervised Learning

In supervised learning a dataset $\mathcal{D}$ is given, which is made by N pairs (x_i, y_i), where x_i is the ith sample and y_i is the corresponding label (ground truth). It is assumed that these pairs have been generated by an unknown distribution $\mathbb{P}(X, Y)$. We want to find a function (predictor) $f : X \mapsto Y$ that well-approximate the conditional distribution $\mathbb{P}(Y|X)$.

To evaluate f we can define a loss function $L(f(X), Y)$ that quantifies the dissimilarity between $f(X)$ and Y. We would like to minimize the expected value of the loss, i.e. we would like to find $f^\star$ s.t.:

$$f^\star = \arg\min_f \mathbb{E}[L(f(X), Y)]\,, \tag{4.1}$$

but, since the $\mathbb{P}(X, Y)$ distribution in unknown, we can't calculate the expected value of the loss. The best approximation we can do is to minimize the empirical loss, i.e. find $f^\star$ s.t.:

$$f^\star = \arg\min_f \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} L(f(x), y)\,. \tag{4.2}$$

4.1.1 Dataset split

In practice, we are not interested in perfectly minimize the *loss* function on all available data. The performance of a model on data that were used to create it is usually better than those that would have been on unseen data.

Therefore, the dataset is divided into three disjunct subsets: *training set*, *validation set* and *test set*. Training set is used by the learning algorithm to adapt the values of the model parameters. Sometimes, the model is also characterized by some hyperparameters, which are parameters whose value cannot be automatically learned during the training process, but they must be set before the training starts.

The validation set can give an impartial evaluation of a model, and so can be used for tuning the hyperparameters. It is said that the model generalizes well when the loss calculated on data not used for training is similar to the loss calculated on the training data set. Ideally, we would like to have high and similar classification capabilities on training and validation data.

The validation set gives you a good idea of the model's ability to generalize, but cannot provide a precise indication of its performance because it indirectly influences the learning process. For a truthful estimate of a model's qualities, a third set (the test set) should be used to evaluate the model, but one time only and after the final choice of the hyperparameters.

In case you can't reach a fairly low loss, we talk about *underfitting*, while when there is a lot of difference between the loss on the validation/test set and the loss on the training set, we talk about *overfitting*. A good model has *underfitting* and *overfitting* as little as possible.

4.1.2 Classification and regression

Classification and regression are two fundamental problems in machine learning.

Classification is the problem of identifying to which class (over a predefined set of classes) an observation belongs. If the prediction is wrong, the loss has the same value regardless of the true class.

Regression is the problem of predicting a numerical score. Often, the prediction is restricted to an interval. The difference from classification is that the loss is higher when the mistake is bigger.

4.2 Machine Learning Algorithms

4.2.1 Decision Trees

A decision tree is a simple and easy-interpretable machine-learning algorithm. Each example x_i is a vector, and each element of it is an attribute. The classification (or regression) of x_i can be done by following a path along the tree, starting from the root. Every internal node of the tree is a decision over an attribute: the next child on the path is chosen depending on the attribute value. At some point, the path comes to a leaf node: a prediction is associated with each leaf node.

The construction of the tree can be done using algorithms such as ID3 [8], C4.5 [9], C5.0, CART [10]. The common idea behind these algorithms is to greedily construct the tree by choosing at each node a split on an attribute that minimize some measure of impurities, such as Gini impurity or entropy.

4.2.2 Random Forests

A random forest is a strong learner made by an ensemble of decision trees (weak learners) [11]. Each tree is different from each other: the final prediction is obtained by aggregating (e.g. using mode or mean) the predictions of the individual trees.

To differentiate the trees, random forests use the bootstrapping technique (random sampling with replacement) both on examples and on features. So each tree is trained on a random subset of the dataset, and at each split it can choose the best split only among a random subset of the features.

Random forests correct the high variance of decision trees, often reducing the overfitting.

4.2.3 XGBoost

XGBoost is an optimized library that implements machine learning algorithms under the Gradient Boosting framework [12].

Gradient Boosting is an ensemble method that typically uses decision trees as base learners. Gradient Boosting trains many week learners in a gradual, additive and sequential manner. Trees are added one at a time, the existing ones are not changed. A gradient descent procedure is used to minimize the loss when adding trees. The final output is given by the sum of the predictions of the individual trees.

XGBoost has clever techniques to penalize and randomize the trees, with the aim of reducing the overfitting. XGBoost usually performs better than Random Forest.

4.2.4 Support Vector Machines

Support-vector machines (SVMs) [13] are supervised learning models usable for classification or regression. Given a dataset with examples belonging to two categories, SVM finds the hyperplane that separates the points of the two categories with the maximum margin, to be as robust as possible. The hyperplane is found by solving an optimization problem.

If the dataset is not linearly separable, the algorithm can converge if we accept to misclassify some training examples, while paying a penalty for each misclassification in the objective function. Another possibility is to project the dataset examples in higher-dimensional space using the kernel trick.

A SVM classifier can classify two classes. An extension to multi-class classification problems can be done using multiple binary classifiers that distinguish between one of the labels and the rest (one-versus-all) or between every pair of classes (one-versus-one).

4.3 Neural Networks

An artificial neural network is a model inspired by the functioning of the brain. An artificial neural network is made by a set of interconnected units: there are units of various kind. A simple example of a single unit is the mathematical model of a neuron (Figure 4.1).

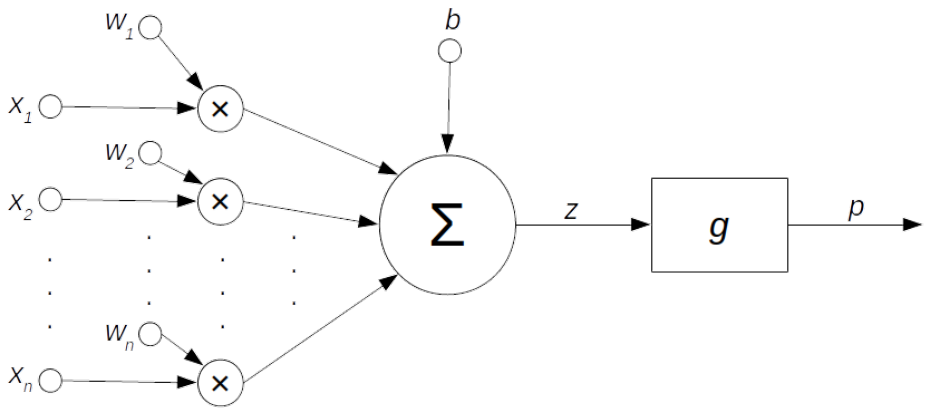

Figure 4.1: Scheme of the mathematical model of an artificial neuron, fundamental unit in neural networks.

The output p of the neuron is given by:

$$p = f(x) = g(w^T x + b)\,, \tag{4.3}$$

where:

- x: input vector;
- w: weights vector;
- b: bias;
- g: activation function, it makes the computation non-linear.

The unit calculates a different function depending on g, w and b. The activation function g is chosen at the beginning, there are many possible choices. The weights vector w and the bias b are parameters whose value is determined during the learning process. Note that b could be integrated in w by simply associate to b a dummy input always equal to 1.

The first algorithm based on the model of the neuron was *perceptron*, proposed by *Frank Rosenblatt* in 1958 [14]. As activation function it usually has the Heaviside step function: $g(z)$ is 1 when $z \geq 0$, otherwise 0. Thus, depending on x, w and b, the perceptron can be either switched on or off. In the learning process w and b are determined in a way that the perceptron switches on or off to distinguish the class for x. The parameters are randomly initialized at the beginnng and updated with the purpose of reducing the prediction errors.

By combining more units, we can obtain more complex neural networks (Figure 4.2). Units are usually organized into layers: the units of a layer receive as input the output of the previous layer. A layer whose units are connected to all the units of the previous layer is called *dense*, or *fully connected*. The first layer of the network is the *input layer*, the last one is the *output layer*, the intermediate ones are the *hidden layers*.

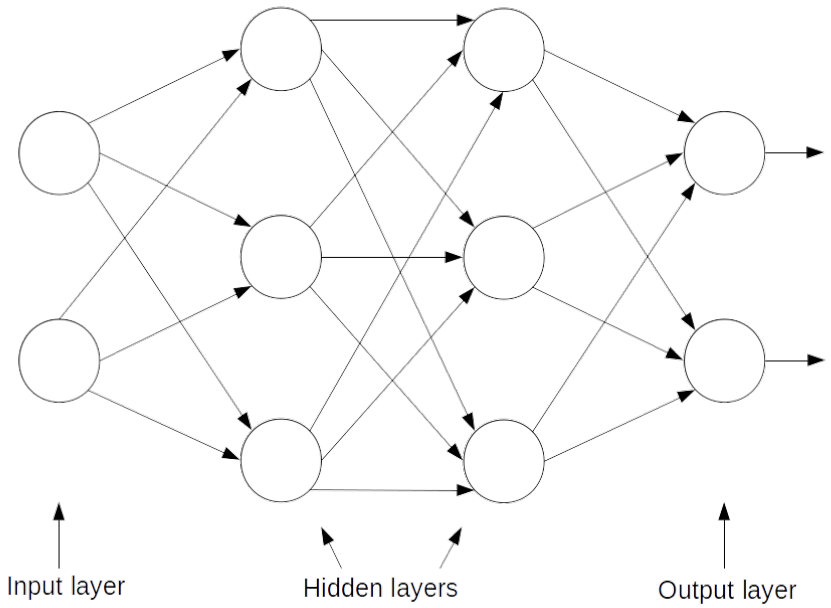

Figure 4.2: Example of neural network with 2 input, 2 hidden layers, and 2 output.

4.3.1 Training of a Neural Network

A neural network is parameterized by a vector W which contains all the weights and biases. The network takes a vector x as input, computes a function f and outputs a vector p: $p = f(x, W)$. In classification problems, p are the probabilities that the input belongs to the different classes[1].
Depending on the problem, you can choose different loss functions. For classifications, it's common to use the *Cross-Entropy Loss* (Eq. 4.4).

$$L(f(x, W), y) = L(p, y) = -\sum_i y_i \log p_i \,, \tag{4.4}$$

[1]If the classes are mutually exclusive, the activation function of the output layer is a sigmoid (for binary classification) or a softmax (for multi-class classifications).

where the label y, linked to example x, is one-hot encoded: it is a vector with value 1 in correspondence of the correct class, and 0 elsewhere.
Note that the codomain of this loss is $[0, +\infty)$: the value is 0 when the classifier gives 1 as probability of the correct class, i.e. $y = p$.

Models tend to become complex, with weights of high absolute values, causing sensibility to noise in the data that leads to overfitting. To overcome this problem, we add an explicit regularization function to the empiric loss in the second member of equation 4.2, multiplied by a hyperparameter λ.
Adding the regularization function to 4.2, the objective is to find $W^\star$ s.t.:

$$W^\star = \arg\min_W \frac{1}{N} \sum_{(x,y)\in D} L(f(x, W), y) + \lambda R(x, W)\,. \tag{4.5}$$

It's not necessary that all the elements of W have the same penalty λ: they can have different λ_1, λ_2, etc.
The parameters W are randomly initialized at the beginning, there are many techniques [15]. Then, W is iteratively updated by leveraging the gradient $\frac{\partial L}{\partial W}$ of the loss w.r.t. W, which is calculated through the *backpropagation* [16]. The algorithm that uses the gradient to update the weights is the *optimizer*. There are many optimizers based on *gradient descent* [17], which uses the following update rule:

$$W \leftarrow W - l_r \frac{\partial L}{\partial W}\,, \tag{4.6}$$

where l_r is the *learning rate* hyperparameter: if l_r is too low, the learning is very slow and can stuck in a local minimum; if l_r is too high, it's impossible to minimize the loss.

In principle, we can use any function as a computing unit in neural networks, as long as it provides a gradient to be used during the backpropagation.

4.3.2 Transformers

In [18] the authors proposed a network, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.

Attention mechanism in Transformers

Given a query and a set of key-value pairs (as vectors), an attention function outputs a weighted sum of the values: the weights for the values are computed by a compatibility function of the query with the corresponding key.

In other words: the compatibility of the vector q with the vector k_i of the ith key, gives the weight to use for the ith value v_i in the weighted sum.

For efficient computation, we can pack the keys into a matrix K, the values into a matrix V and the queries (if there are more than one) into a matrix Q, resulting in Equation 4.7:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V\,, \tag{4.7}$$

where d_k is the length of each key vector. The scaling factor $\sqrt{d_k}$ counteracts the effects of having an extremely small gradient in the softmax when the input is large.

It could be beneficial to concatenate the results of multiple attention heads (Figure 4.3).

This **attention mechanism** is used by the Transformer as a building block of the entire architecture (Figure 4.4, which is made by two different parts: the **encoder** and the **decoder**. The encoder transforms the input sequence into another sequence of the same length. The decoder can be used to generate an output sequence (for sequence-to-sequence tasks), but we do not use it in this thesis.

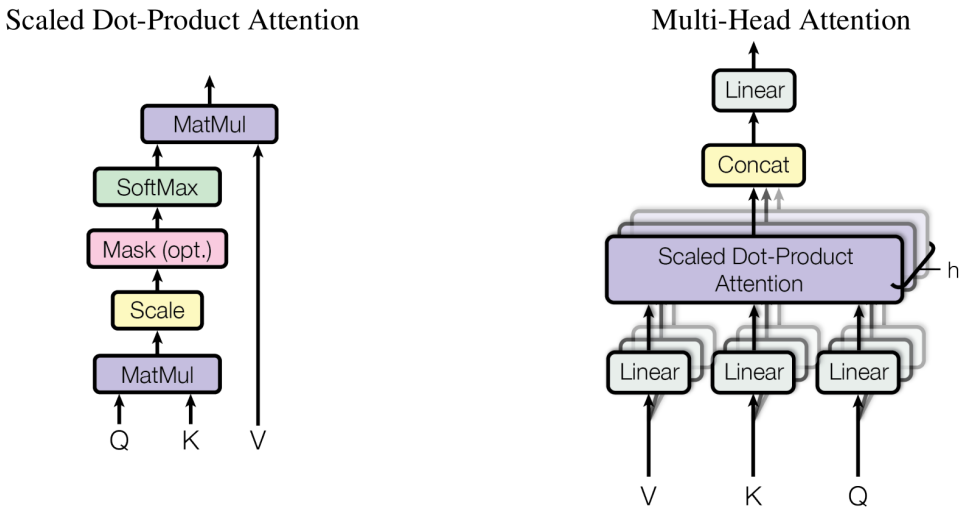

Figure 4.3: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. (Image from [18])

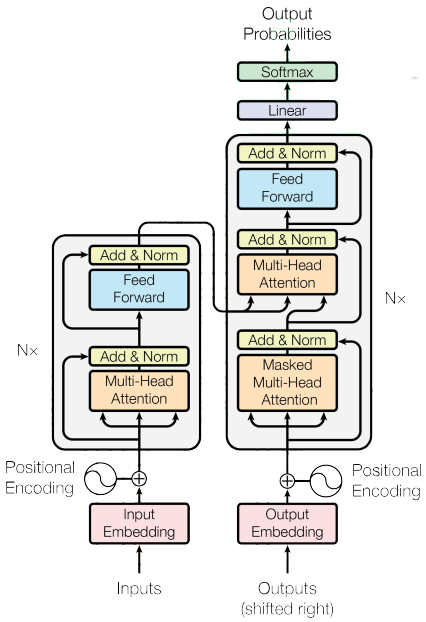

Figure 4.4: The Transformer model architecture. The left part is the encoder, the right part is the decoder. (Image from [18])

An important mention to the **positional encoding** layer: it adds information about the position of tokens, since the the attention itself do not make use of the order of the sequence.

In practice, matrices Q, K and V are obtained by multiplying the word embeddings with different weight matrices for each head W^Q, W^K and W^V, respectively.

When we talk about *self-attention*, we intend that the query sequence is the same as the key-value sequence. In this thesis we use self-attention only.

4.3.3 Multi-task learning

Multi-task learning is a machine learning setting in which multiple tasks are learned at the same time. The underlying intuition is that if tasks are similar, it's possible to extract a better representation of each example of the dataset. This can result in improved prediction accuracy when compared to training the models separately.

4.3.4 Multi-instance learning

In the usual supervised learning setting, each instance is individually labeled. In multiple-instance learning instead, there is a set of labeled bags, each containing many instances.

Chapter 5

Breast cancers dataset

This dataset is a csv file of more than 120000 rows concerning breast cancer reports, collected between 1990 and 2019, and labeled between 2003 and 2015.

Unfortunately, in the ISPRO database, labels was not stored beside the report, or beside a well-defined group of reports. The reports are linked to the patient, and so are the labels, but there is no direct link between reports and labels. Therefore, it is not always straightforward to determine which report is linked to which label. The dataset csv file is just a dump of the joined tables of this part of the database. This dump includes only patient who had at least one breast cancer are selected.

Note that the links between reports and labels do exist at some point, but they were not explicitly persisted in the database.

How does this problem reflect in the data we have available? Many times, the reports appear in the row beside the correct label, but sometimes it can happen that:

- the report was written after the labeling of the patient;

- the report was available during the labeling of the patient, but it was ignored by the labelers because it was about another type of cancer;
- the report was available during the labeling of the patient, but it was ignored by the labelers because, even if it was about the cancer of interest, it wasn't relevant in the choice of the value of the label.

Since we can't know when one of the previous cases happens, these mismatches remains in the data as a kind of noise. It is important to note that this type of noise, if it exists, will be present both in the data used to create the prediction models, and in the data used to evaluate their performance. We tried our best to improve the data quality by removing some rows, as explained in detail in section 5.3.

5.1 Dataset fields

In the csv file, there are three different kind of fields: input data, desired output variables, and metadata.

5.1.1 Metadata

Beside to each report, in addition to the text and to the variables of interest, there are other fields:

- `anno_diagnosi`: year in which the report was analyzed and labeled
- `anno_referto`: year in which the report was written
- `id_paz`: identifier of the patient
- `sede_icdo3`: ICD-O3 topography code
- `morfologia_icdo3`: ICD-O3 morphology code

5.1.2 Input data (text of reports)

The columns that contains the ASCII text of the report are `diagnosi` (diagnosis), `macroscopia` (macroscopy) and `notizie` (news). All three are strings. Their semantic is not always respected, but is clear that the field `notizie` is the less important one: in fact, it is not used even by the labelers.

5.1.3 Variables to predict

The objective of the predictors is to extract these variables from the text. Many data are missing: not always a patient is associated with all the values of the variables to predict. Each variable has a different percentage of missing values (Figure 5.1). These variables are:

- `cerb`: proteins involved in proliferation and cell growth (percentage)
- `dimensioni` (size): diameter of the carcinoma, in millimeters. In the reports, it is sometime expressed in millimeters, other times in centimeters;
- `grading`: grade or differentiation of malignant neoplasms;
- `ki67`: marker of tumor cells proliferation speed (percentage);
- `metastasi` (metastasis): indicates whether the tumor spread from its home to other organs of the individual;
- `mib1`: marker of tumor cells proliferation speed (percentage);
- `numero_sentinella_asportati` (removed_sentinels_number): number of removed sentinel lymph nodes;

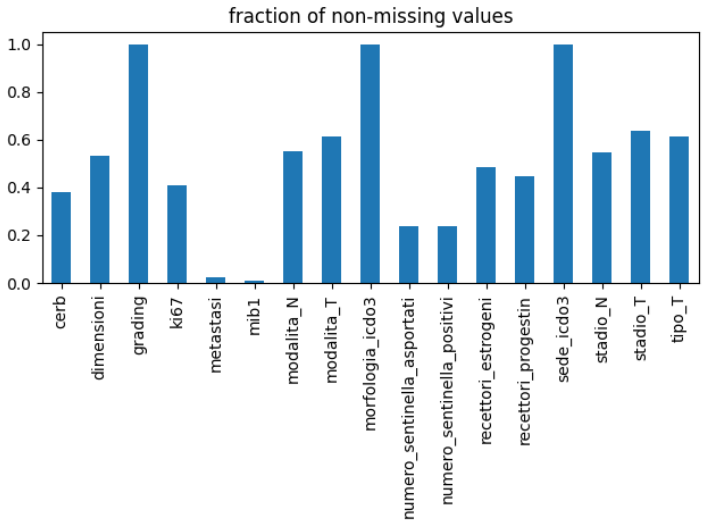

Figure 5.1: Fraction of non-missing values per variable. Unfortunately, there are a lot of missing values.

- `numero_sentinella_positivi` (positive_sentinels_number): number of lymph nodes (among those removed) in which tumor cells were detected;
- `recettori_estrogeni` (estrogen_receptors): percentage;
- `recettori_progestin` (progesterone_receptors): percentage;
- `stadio_N`: quantification of the involvement of lymph nodes near the tumor;
- `stadio_T`: quantification of the extension of the primary tumor;
- `tipo_T`: binary variable, can be P or PY.

Distributions of the variables

From figure 5.2 to 5.14 we report the distributions of the variables. These distributions are calculated after the cleaning steps described in section 5.3.

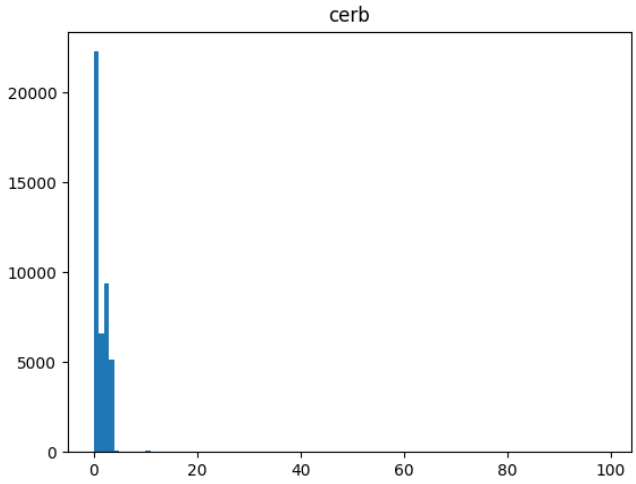

Figure 5.2: Distribution of the *cerb* variable: values are very low, and the standard deviation is very small

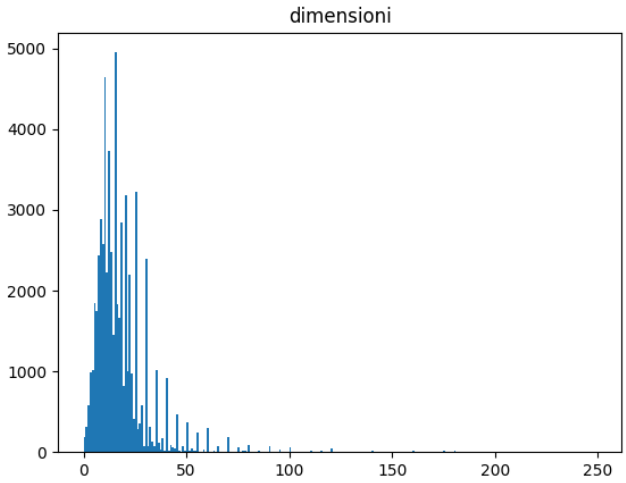

Figure 5.3: Distribution of the *dimensioni* variable: the majority of values are between 0 and 50, with spikes in correspondence of values that ends with the 0 or 5 digit.

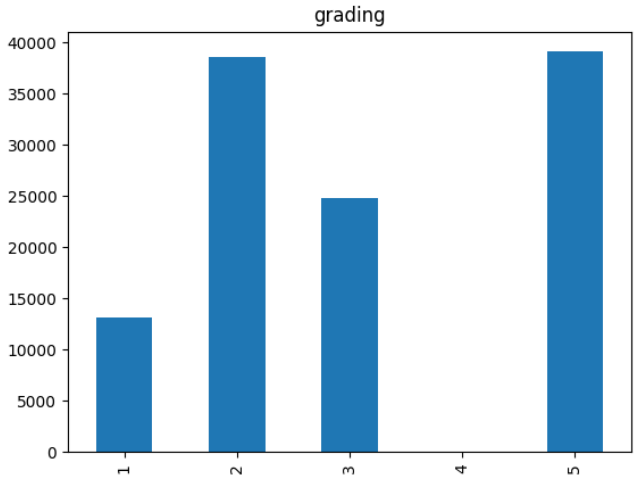

Figure 5.4: Distribution of the *grading* variable: the value 4 is present but in too few cases to be modeled by a predictor.

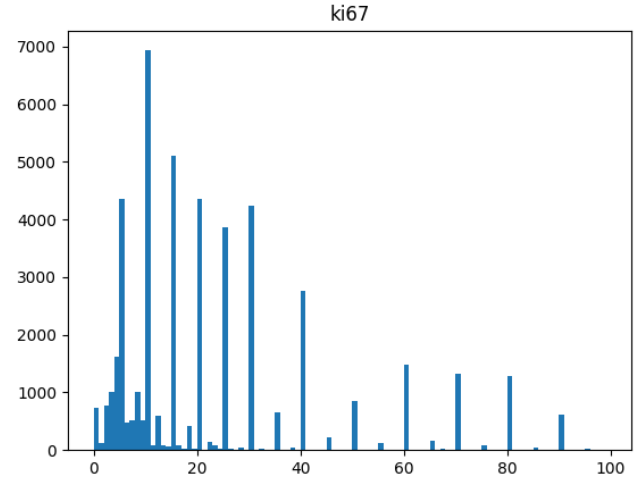

Figure 5.5: Distribution of the *ki67* variable: there are more values between 0 and 50 than between 50 and 100. There are spikes in correspondence of values that ends with the 0 or 5 digit.

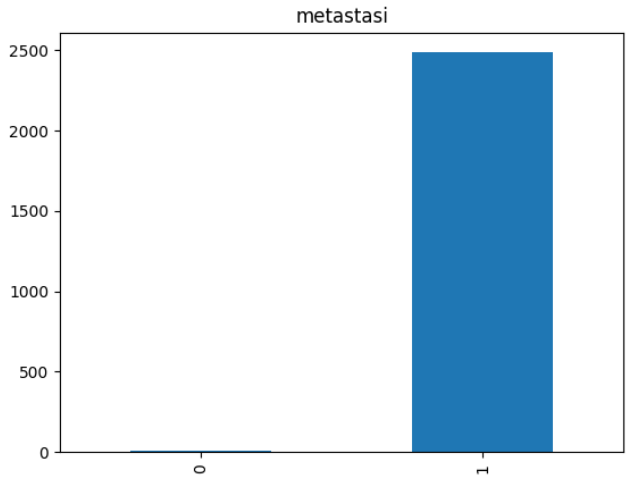

Figure 5.6: Distribution of the *metastasi* variable: non-metastatic reports are probably too few to be real. Probably something in the labeling process went wrong. In any cases, there are too few labeled reports to create a reliable predictor.

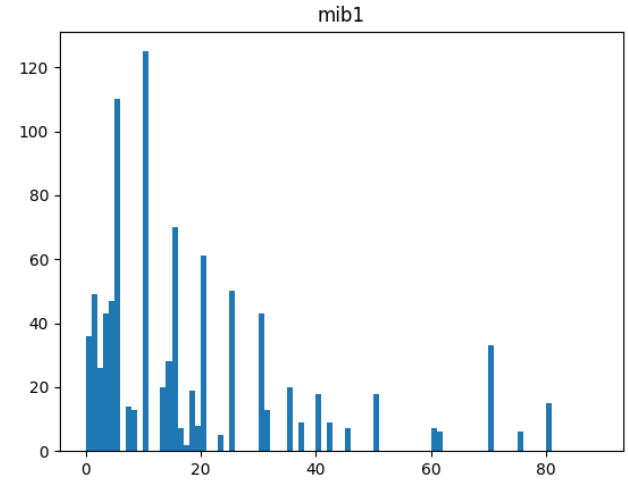

Figure 5.7: Distribution of the *mib1* variable: the shape of the distribution look similar to the one of *ki67* variable (Figure 5.5). Though, in comparison, there are fewer labels.

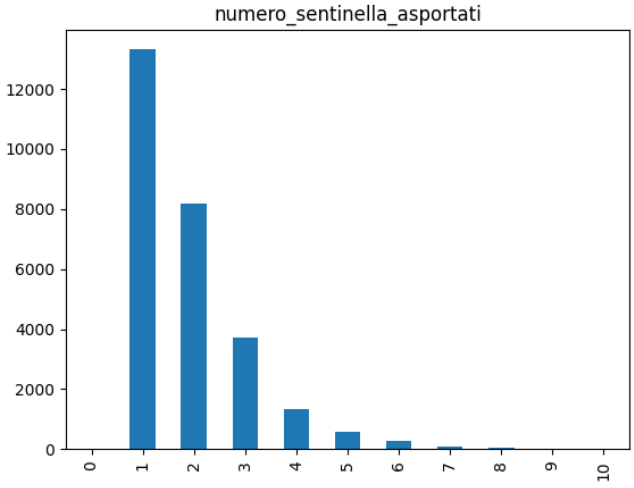

Figure 5.8: Distribution of the *numero_ sentinella_ asportati* variable: usually they are no more than 4 or 5. Only values ≤ 10 are shown. Notice the strangely low occurrences of value 0.

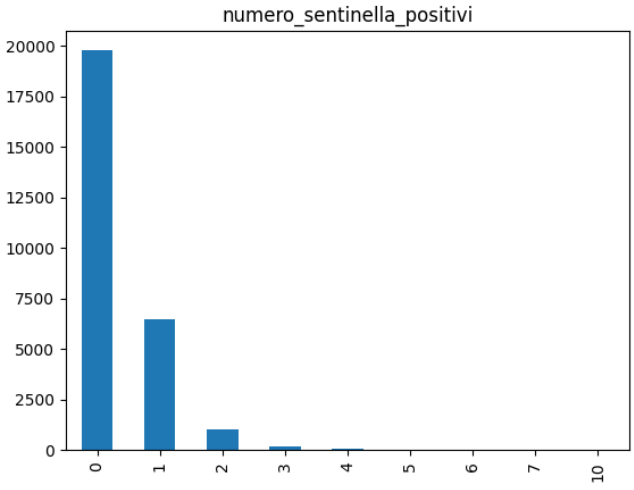

Figure 5.9: Distribution of the *numero_ sentinella_ positivi* variable: usually they are no more than 2 or 3. Only values ≤ 10 are shown. Notice that some values (like 8 and 9) are not present in the data.

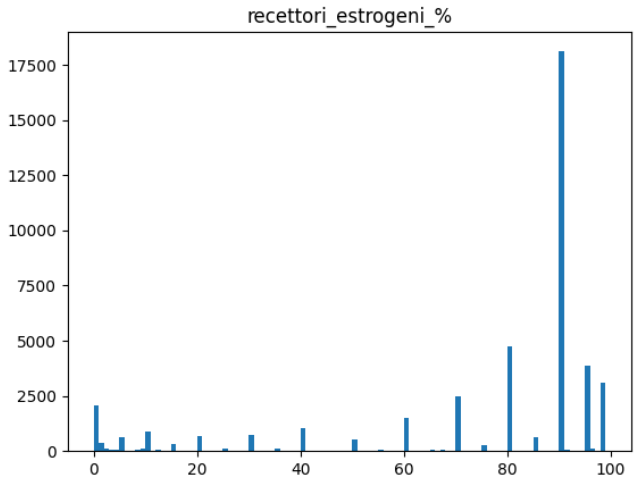

Figure 5.10: Distribution of the *recettori_estrogeni* variable: the value 90 is the more frequent by a large margin. The local peaks are in correspondence of values that ends with the 0 or 5 digit.

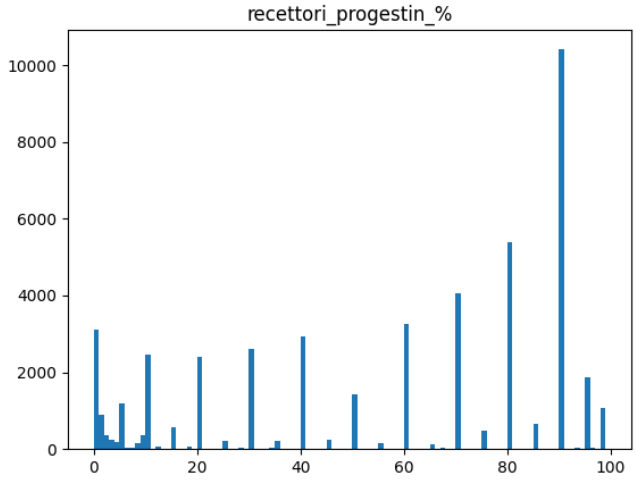

Figure 5.11: Distribution of the *recettori_progestin* variable: the value 90 is the more frequent. The local peaks are in correspondence of values that ends with the 0 or 5 digit.

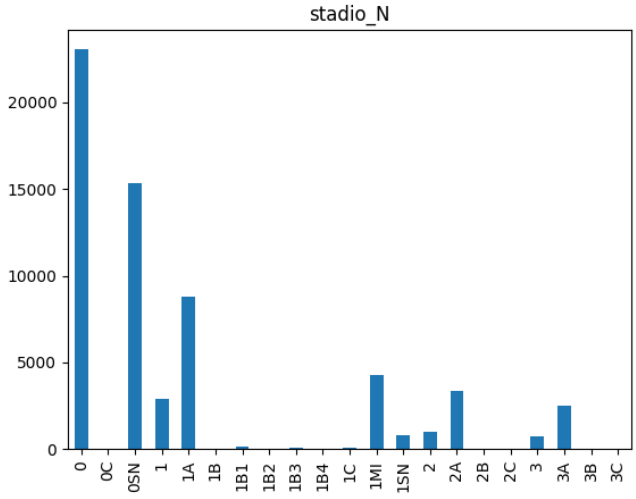

Figure 5.12: Distribution of the *stadio_ N* variable: many of the subclasses (letters after the first number) have very few labels. A prediction of the first number only seems more doable.

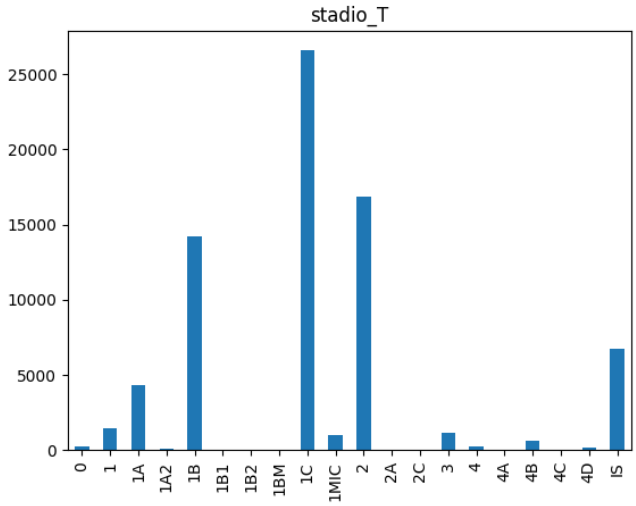

Figure 5.13: Distribution of the *stadio_ T* variable: many of the subclasses (letters after the first number) have very few labels. A prediction of the first number only seems more doable.

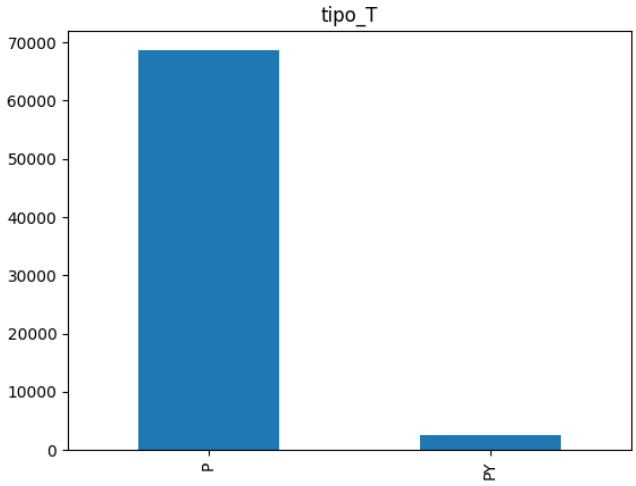

Figure 5.14: Distribution of the *tipo_ T* variable: this is an unbalanced binary variable.

5.2 Example of data

In table 5.1 we report an example of a report of a patient. In this example, we can easily spot where do some labels come from. The entire clinical record of the patient is made by many reports like this one, but of course, not all of them contain the necessary information to extract the variables.

5.3 Dataset analysys and cleaning

In this section we analyze the data in the dataset, describing and motivating the cleaning steps done. Each row of the csv file contains a report and the corresponding labels, the patient is specified by a special field `id_paz` which is unlinkable to the true identity.

There are initially 123709 reports, belonging to 25612 patients. Due to a problem in the data, a patient as a report with an incorrect number of columns: we remove this patient. Then we check the number of missing

field	value
id_paz	5******
anno_diagnosi	2012
sede_icdo3	C509
morfologia_icdo3	85003
dimensioni	23
tipo_T	P
metastasi	
modalita_T	E
modalita_N	E
stadio_T	2
stadio_N	0SN
recettori_estrogeni	90
recettori_progestin	60
numero_sentinella_asportati	1
numero_sentinella_positivi	0
mib1	
cerb	0
ki67	10
grading	2
anno_referto	2012
id_isto	5******
notizie	
macroscopia	Q.I.C. mammella sn (cm 12x8x5):\nT1-4) neoplasia (mm 23), distanza dai margini >mm 10; MS) margine superiore; MI) margine inferiore; MM) margine mediale; ML) margine laterale; MP) margine profondo; CU) margine cutaneo.\n(eseguita colorazione ematossilina-eosina e valutazione parametri biologici con controllo di qualitï¿½).
diagnosi	CARCINOMA DUTTALE INFILTRANTE (N.O.S.) (G2) DELLA MAMMELLA CON ASSOCIATE ESPRESSIONI INTRADUTTALI DI BASSO GRADO (T1-4)\nNON EVIDENTE PERMEAZIONE NEOPLASTICA VASCOLARE \nNESSUNA PROLIFERAZIONE CANCERIGNA NEI MARGINI DI SEZIONE CHIRURGICA (MS, MI, ML, MM, MP), NELLLA CUTE (CU), NEL LINFONODO SENTINELLA (vedi es.B 10508/12).\n(pT2 N0(OSNA -))* \n*(TNM, VIIï¿½ ed., 2009)\nParametri biologici: ER: + 90% ; PGR: + 60% ; ki67: + 10% ; Her 2: - .

Table 5.1: Example: report of a patient, with all fields. The colored underlining highlights some of the variables, and from where they can be extracted from the text. Possibly sensitive information has been obscured.

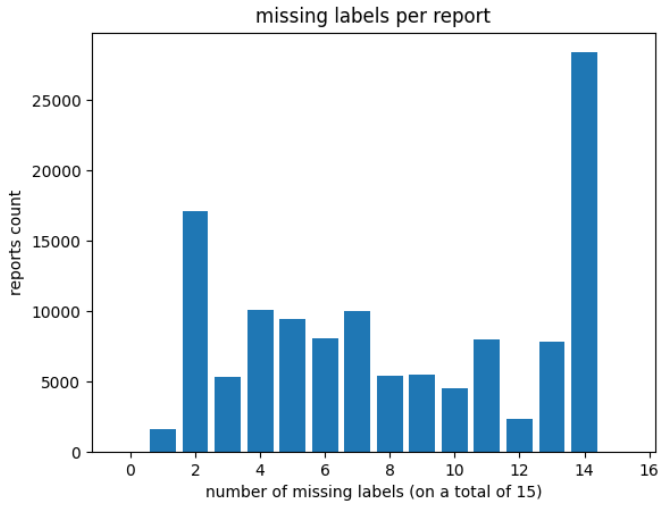

Figure 5.15: Number of missing labels per report

labels for each report (Figure 5.15): remind that the number of labels is 15.

We drop all duplicated reports, which were originated by patients who had at least two breast cancers.

Then we proceed the cleaning by removing invalid values either by replacing them with *NaN* or by dropping the entire report. A value is considered invalid if it is not allowed for that variable (e.g. a letter instead of a number or vice-versa, or a percentage above 100).

The variables "numero_sentinella_asportati" and "numero_sentinella_positivi" had in reports values too high to be real (e.g. more than 80), or values too little frequent to be be used in a classification (e.g. only one patient with 12 removed lymph nodes, and zero patient with 11 removed lymph nodes): we replace all these values with *NaN*, while keeping more than 99% of the labels.

We also drop all reports that have empty `diagnosi` and `macroscopia` fields, regardless of the content of the `notizie` field. In the end there are 25153 patients, 115655 reports.

5.4 Dataset split

Now we look at the distribution of reports per year, using the `anno_diagnosi` field (Figure 5.16). We split the dataset into *training*, *validation* and *test* in the following way:

- all patients that have at least a report with `anno_diagnosi = 2015`, go into the *test* set (3828 patients, 18961 reports);
- all patients that have at least a report with `anno_diagnosi = 2014` and are not in the test set, go into the *validation* set (3777 patients, 18975 reports);
- all remaining patients (all reports on 2013 or before) go into the *training* set (17548 patients, 77719 reports).

We have chosen a temporal division to better quantify prediction errors. We can't assume that the dataset samples are *independent and identically distributed (i.i.d.)* because the data archiving process has changed through the years. Notice the peak at year `2004` in Figure 5.16: in that year there was an effort to collect reports from the whole region. Furthermore, we can easily see the rising number of reports in recent years: in fact, from 2013 the reports are collected from all over the *Tuscany*, whereas until 2012 data was coming from the provinces of *Florence* and *Prato* only. The number of collected reports it's not the only thing that changes over the years: analyzing the average number of missing labels per patient (Figure 5.17), we see that there is something different in the data.

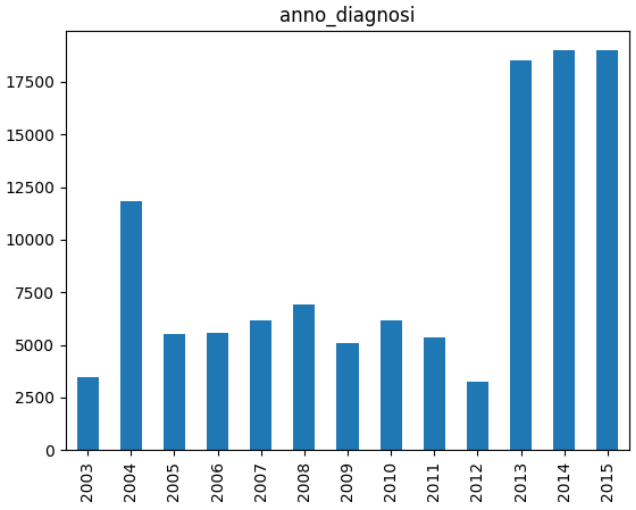

Figure 5.16: Number of labeled reports per year.

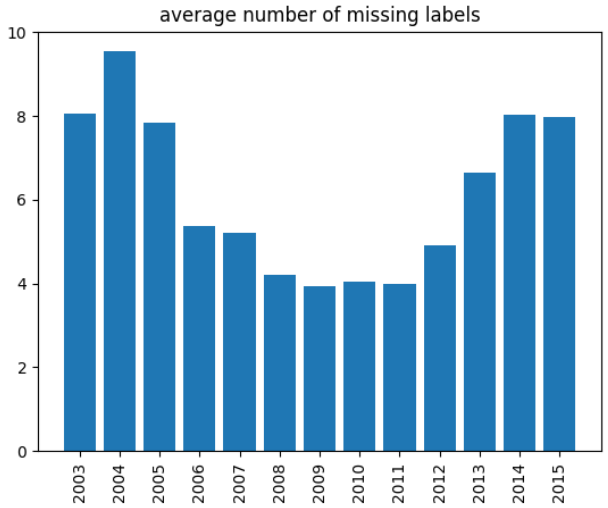

Figure 5.17: Average number of missing labels per patient in different years.

Chapter 6

Methods

Source code: `https://github.com/fedem96/predict-cancer-variables`.

6.1 Pre-trained models on other datasets

Unsupervised pre-training usually adds robustness to a deep neural network, helping in finding better local minima, with lower generalization error. [19] Recent works that obtain very good results in many Natural Language Processing tasks [20, 21] have pre-training as a crucial step.

However, we decide to not take any pre-trained model on other datasets, and to not pre-train our models with unsupervised tasks on other datasets. We believe that the eventual benefits would be lower than the amount of required effort, due to the data we work on:

- in the reports, there are many domain-specific keywords that are certainly relevant for the prediction of variables, but they are not present in the dictionary of the pre-trained models we had found;
- the syntax of the phrases in these reports is completely different from

the syntax we can find in scientific articles, Wikipedia articles and other common kinds of texts: it is some sort of list of keywords and numbers;

- the most similar datasets would be the cancer registers of the other Italian regions: though, these data are not publicly available, for privacy reasons.

6.2 Variables

In section 5.1.3 we described the meaning of the variables to predict. In this section we explain the strategies to predict them.

Each variable appear in the text in a different way, we can identify 4 different groups:

1. **keyword-identified** (grading, stadio_N, stadio_T, tipo_T): few keywords are relevant for classification;

2. **text understanding** (numero_sentinella_asportati, numero_sentinella_positivi): these variables are more difficult to predict, as there are not precise keywords;

3. **find-number** (recettori_estrogeni, recettori_progestin, mib1, cerb, ki67, dimensioni): the desired value is written as a percentage after and not very distant from words that define a variable. The *dimensioni* variable is a little different: it is not a percentage, it can have decimal digits, and can appear in two different units (millimeters and centimeters), the ground truth is in millimeters. The *cerb* variable can appear as a percentage and as an integer score, the ground truth is the score;

4. **unpredictable** (metastasi): not enough labeled data.

6.2.1 Keyword-identified variables

For these variables, usually few tokens are enough to make a good prediction. We set up the predictions of these variables as a classification problem.

Grading

Doctors usually use standard terms to specify the class:

1. "g1", "grado 1", "grado i" or "ben differenziato";
2. "g2", "grado 2", "grado ii" or "moderatamente differenziato";
3. "g3", "grado 3", "grado iii" or "scarsamente differenziato";
4. "g4", "grado 4", "grado iv" or "non differenziato".

However, we exclude class 4 from classification because it's too underrepresented. Some patients are labeled with value 5, which is an unusual grade [2]: by manually exploring the dataset, we notice that many reports with grading 5 have words in the text that suggested another grade. Thus we decide to exclude these patients from the classification.

Stadio

The main class of both stadio_N and stadio_T are usually identified by few characters: n0, n1mi, n1, n2, n3, tis, t0, t1, t2, t3, t4.

We do not investigate sub-classes (e.g.: t1a, t1b, t1c) because many of them do not have enough labeled data. For the same reason, we exclude the t0 class from stadio_T classification.

Tipo

The tipo_T variable has two possible values: P and PY. It seems that when the value is PY, there is a token "yp" in at least one of the reports.

6.2.2 Text understanding variables

For these variables the wording is less standard than it is for the "keyword-identified" variables: this probably requires a better understanding of the text to predict the correct value. From the example in table 5.1:
"Nessuna proliferazione cancerigna nei margini di sezione chirurgica (MS, MI, ML, MM, MP), nella cute (CU), nel linfonodo sentinella",
which translates in:
"No cancerous proliferation in surgical section margins (MS, MI, ML, MM, MP), in the skin (CU), in the sentinel lymph node",
we have to understand that only one lymph node was removed and it wasn't cancerous (the relevant terms are the underlined ones).

We set up the predictions of these variables as a classification problem, ignoring rare labels. For the variable "numero_sentinella_asportati" we select the values 1, 2, 3, 4, whereas for "numero_sentinella_positivi" we select the values 0, 1, 2. In both cases, we keep about 99% of the original labels.

6.2.3 Find-number variables

The value of some variables can be found in the text near tokens that identify the presence of that variable. We'll call these tokens *markers*.
From the example in Table 5.1:
"...Parametri biologici: ER: + 90% ; PGR: + 60% ; ki67: + 10% ; Her 2: - ."

This example is very representative of what we kind expect in the records,

field	markers
dimensioni	“neoplasia (mm ...)”, “diametro carcinoma ... centimetri”
recettori_estrogeni	“ER”, “estrogeni”, “recettori estrogeni”, “recettore estrogeno”
recettori_progestin	“PG” “PGR”, “progesterone”, “recettori progesterone”
mib1	“mib1” “mib-1”, “mib - 1”
cerb	“cerb”, “c-erb”, “c-erb”, “c-erb-b2”, “c-erbb-2”, “cerbb”, “her”, “her 2”, “her-2”
ki67	“ki67”, “ki-67”, “ki - 67”

Table 6.1: Common markers for each “find-number” variable. In the text, the correct value of the variable is usually right after to or near to one of these markers.

even if sometimes it’s more difficult: other words or abbreviations can be used to identify the variable, there may be annotations in parenthesis between the variable name and the number, there may be no semicolon to separate the variables. In Table 7.6 we list some common markers for each variable.

This indicates that a raw classification or a raw regression does not have much sense here. The problem would be better formulated as a localization/segmentation one, but in our dataset there is no info on where the correct prediction is in the text.

We do not set up the predictions of these variable as a Machine Learning problem: we use a regex-based algorithm to extract the correct numbers from the text, as explained in section 6.4.3.

Remark that the *dimensioni* variable is a little more complicated than the others in this group. The two main troubles are:

1. the unit of measure can be millimeters or centimeters: for each number in the text, we have to find the right token that identifies the unit. The unit is not always in the same position relative to the number;

2. the location of the number of interest can be both on the right that on the left of the marker, in contrast to what happens to the other variables, that are always on the right side.

6.3 Data preparation

6.3.1 Preprocessing

Preprocessing is the same for all the algorithms:

1. all letters to lowercase;
2. remove RTF characters (they were in some reports);
3. keep only valid characters: numbers, non-accented letters, punctuation;
4. task-specific helps: convert similar sequences of characters into the same one;
5. put spaces around punctuation characters;
6. replace multiple spaces with a single one.

In the dataset, all accented letters and other characters were corrupted: at their place, there was the string "ï¿½". Instead of removing this specific string and other undesired characters, we keep only valid characters: in this way, we can process data with a different kind of corruption also.

Task-specific helps

grading: whenever occurs something like “grado 2”, “grado ii”, “grading 2”, “g 2”, we convert it into “g2” (the same for the other classes)
stadio_N: we put spaces around each “n0”, “n1”, “n2”...
stadio_T: we put spaces around each “t0”, “t1”, “t2”...
cerb: whenever occurs something like “c-erb”, “c-erb”, “c-erb-b2”, “c-erbb-2”, “cerbb”, “her”, “her-2”, we convert it into “cerb”
ki67: we remove dashes and spaces from “ki-67”, “ki - 67”

mib1: we remove dashes and spaces from "mib-1", "mib - 1"

find-number variables: we replace "-" with "0%" since we noticed that many records with "-" has the value 0 in the corresponding label.

6.3.2 Tokenization

Split sentence into tokens

Tokenization is done by wrapping the italian tokenizer of spaCy library [22].

After spaCy tokenization, tokens can be optionally grouped to return tokens n-grams: this is useful to add some context when using the bag-of-words or TF-IDF representation.

When we are using the n-grams of a report, we intend that we are using all k-gram up to order n (i.e. $1 \leq k \leq n$).

Tokens codec

The tokens encoding (string to index) and decoding (index to string) is created by:

1. preprocess all the reports in the training set
2. for each preprocessed report:
    2.1. tokenize the report
    2.2. for each token in the set of tokens
        2.2.1. if it is a new token, link it to the next available number
3. (optional) drop tokens that appear in too few documents
4. (optional) drop tokens that appear in too many documents

In all experiments we keep tokens present in at least 0.1% of the documents.

6.4 Models

We take different approaches to predict the different kinds of variables.

We **classify** the "keyword-identified" and "text understanding" variables using a Transformer-based model, with max pooling and classifier after the Transformer encoder. We compare the results with other simpler algorithms: decision trees, random forests, SVMs, MLP.

For "find-number" variables, we limit ourselves in trying a **regex-based algorithm**, which is not a Machine Learning model. The results obtained with this algorithm can be used in future works as baseline for Machine Learning models.

6.4.1 Deep learning models

Neural networks can handle both bag-of-words/TF-IDF and tokens indices data representation. In the former case, reports of the same patient are previously concatenated. In the latter case, an embedding layer is required as the first layer of the neural network, and it's possible to handle the multi-instance nature of the data by using arbitrary aggregation functions.
In our experiments, all neural networks have the same macro-structure:

1. tokens feature extractor
2. tokens aggregation function
3. reports transformation function
4. reports aggregation function
5. predictors

As feature extractor, we try a *Transformer* and a *Multi-Layer-Perceptron*.
As aggregation function we always use the $max()$ function.
As reports transformation function, we always use the identity function, since we didn't notice any improvement when using some learnable function.
All our neural networks are implemented in *PyTorch*. [23]

Transformer

With this model, we use the tokens indices representation of the data.

An embedding layer is applied on indices to obtain tokens vector. We add to this vector a positional encoding to give them the information on relative positions. Then, a certain number of Transformer encoder layers (with self-attention) are applied to obtain a deep representation of the tokens.

To find a good configuration we try different values for the main hyper-parameters that define the network architecture: embedding size, number of encoder layers, number of attention heads.

Multi-Layer-Perceptron

With this model, we use the bag-of-words representation of the data. TF-IDF representation does not significantly improve the performance. The multi-instance nature of the problem is eliminated by previously concatenate all the reports of the same patient: thus, the reports' aggregation function has no impact here.

Tokens vectors are obtained by applying a settable number of linear layers directly on the TF-IDF vectors. Each linear layer as a ReLU after it, to make a non-linear computation.

Loss and regularization

We use dropout [24] as implicit regularizer: apart from the dropout internal to the transformer encoder layer, we add another dropout just before the classifier. We use a l2 penalty as an explicit regularizer, slightly higher on the classifier than on the rest of the network. We also find small benefits by adding a l1 penalty on the activations of the layer that precedes the dropout of the classifier. The loss is a categorical cross-entropy with weights to compensate for class imbalance.

6.4.2 Machine Learning models

For Decision Trees, Random Forests, XGBoost and SVMs, data need to be in the bag-of-words or TF-IDF format.

The multi-instance nature of the problem is eliminated by previously concatenating all the reports of the same patient.

For these algorithms, we use the *scikit-learn* implementation. [25]
We noticed that sometimes it was helpful using weights to rebalance the classes, other times it wasn't helpful, even if the classes were unbalanced.

Decision Trees

We regularize the trees by limiting the maximum height of the tree and by requiring a minimum amount of examples to be in every leaf. The choice depends on the variable, but usually a height of no more than five was enough.

Random Forests

This algorithms works already very well with the default parameters of scikit-learn. We found helpful increasing the *max_features* parameter, which refers

to *features bootstrapping*, but the improvements have been marginal.

XGBoost

This algorithms works already very well with the default parameters of the library. We found helpful setting λ to 0 and increasing α: as for Random Forests, the improvements have been marginal.

Support Vector Machines

We use the linear version of SVMs with l1 penalty, since in the preliminary results we have noticed that the use of a kernel has not led to any improvement. We regularize the machine using the C parameter. Since SVM can't be multi-class out-of-the-box, in multi-class tasks is applied a one-vs-one strategy.

6.4.3 Regex-based algorithm

To predict variables of "find-number" group, we try an approach which is not a Machine Learning algorithm. In this approach we work directly on text, the reports are preprocessed but not tokenized.

The idea is to find with a regular expression the term or the terms (we call them markers) that identify the presence of one of the variables in the text. Of course, each variable has a different regular expression to identify its marker.

Then, we take a characters window that starts right after the marker, and whose size is a hyperparameter. With a second regular expression, we search for the first occurrence of a number in the window. There is a trade-off in the choice of the window size: if the size is too short, the majority of found numbers will be correct but many numbers will not be found; if the size is too

long, we will almost certainly find a number, but there is a higher probability to not take the right one.

If the marker matches more than one time, there will be multiple windows: in case of multiple matches, we take the bigger one.

To reduce the chances of making mistakes, we use a third regular expression that identifies markers of other variables in the window: we cut all the text right after the first foreign marker.

If no results are found, we proceed in the same manner with the next marker: in fact, each variable has multiple markers that can identify its presence. Note that analyze the results of a single marker at a time is different from having a unique regular expression that matches all possibilities: not all the markers of the same variable are good in the same way (e.g. if there is also a semicolon near the term, the probability of having the correct number next to him is higher).

It is possible that no correspondence has been found: in this case, there is no prediction. Of course, it would be trivial to return a default prediction for these cases.

6.5 Computational and memory optimizations

We pack all the dataset into a single tensor, to exploit the parallelism of the GPU when possible. To give the same dimension to all examples, we cut the long reports/records and pad with zeros the short ones.

6.5.1 Indices data

The vocabulary size goes from 1000 to 50000, depending on whether we are using n-grams and on how many too rare/common tokens we discard. In

every case, we are in the limits of 16-bit integers, which allow a maximum value of 65535. Therefore, we stored the dataset as a tensor of dtype unsigned int 16, containing all the encoded tokens [1].

6.5.2 Bag-of-words/TF-IDF data

Each report has very few tokens compared to the vocabulary size: hence, its vector is sparse. We stored the whole dataset as a sparse array, otherwise it would be difficult to even fit it in memory in some cases.

6.6 Hyperparameters choice

We didn't automatically optimize the hyperparameters. We manually tried different values for the hyperparameters and chose the ones that seem to fit better. However, we notice that in many cases there were many different valid hyperparameters that lead to similar results. It's obviously possible to find other hyperparameters that perform better, but we think that this choice is not crucial with these data, and it wouldn't change the outcome of the experiments.

6.7 Dataset split

As explained in section 5.4, we split the dataset into *training*, *validation* and *test* sets using a temporal criterion.

With every learning algorithm and in every experiment, we use the *training set* to learn the parameters of the model. We use the *validation set* to

[1] Unfortunately, *PyTorch* does not have the uint16 dtype: we store the dataset as int16, convert the batch tensor to long, and add 2^{15} to negative numbers

have an idea about the generalization capabilities of the models, and manually optimize the hyperparameters.

We haven't touched the *test set* in any way during the exploration of different techniques, neither when choosing the hyperparameters, since the beginning of the work. We left it aside on purpose, to use it only in the end, only after having already decided which models to use, relative hyperparameters and experiments to do: in this way, we can use it as a reliable set to give a good estimate of the errors that the predictors will do in practice.

However, we have to keep in mind that, since we expect the data distribution will continue to change in the next years, the estimated errors will be less and less reliable as the years will pass: to overcome this problem, the models have to be trained and validated on new data in the future.

6.8 Metrics

We classify some variables with Machine Learning models, whereas we extract other variables with a regex-based algorithm. The metrics we use to evaluate these two types of predictions are different.

6.8.1 Classifications

The most common metric in classification problems is the *accuracy*, which simply is the fraction of correct predictions. However, accuracy is not a good metric for unbalanced problems: a predictor biased to the more common class will have high accuracy.

Precision and *recall* are metrics that give a better idea of the quality of a model in unbalanced classifications. The *F1 score* is a metric that summarizes precision and recall with their harmonic mean.

For multi-class classification problems, we can define the *Macro-F1 score* as the average of F1 scores calculated for each class.

Binary classifications metrics

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i)$$

$$Precision = \frac{\sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i = 1)}{\sum_{i=1}^{N} \mathbb{1}(f(x_i) = 1)}$$

$$Recall = \frac{\sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i = 1)}{\sum_{i=1}^{N} \mathbb{1}(1 = y_i)}$$

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recall}$$

Multi-class classifications metrics

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i)$$

$$Precision_c = \frac{\sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i = c)}{\sum_{i=1}^{N} \mathbb{1}(f(x_i) = c)}$$

$$Recall_c = \frac{\sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i = c)}{\sum_{i=1}^{N} \mathbb{1}(c = y_i)}$$

$$F1_c = 2\frac{Precision_c \cdot Recall_c}{Precision_c + Recall_c}$$

$$MacroF1 = \frac{1}{N}\sum_{c=0}^{C-1} F1_c$$

6.8.2 "Find number" variables

The regex-based algorithm that tries to extract the value of the variables from the text, is not always capable of extracting a value.

We indicate the percentage of records in which the algorithm pulls out a value as *Extracted*:

$$Extracted = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(f(x_i) \neq null)$$

We are interested in measuring the accuracy restricted to records where a value was extracted:

$$Accuracy_{Extracted} = \frac{1}{N \cdot Extracted} \sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i)$$

assuming that $\forall i$: $y_i \neq null$.

We are also interested in measuring the fraction of correct predictions over the total number of records, we call these metric *Hit*:

$$Hit = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(f(x_i) = y_i)$$

which is equivalent to the *accuracy* in classification problems if we consider wrong every non-prediction.

Note that $Hit \leq Extracted$, and $Hit \leq Accuracy_{Extracted}$.

Chapter 7

Experiments

In the first part of the experiments, there are the results for the variables we decided to classify: we compare different machine learning and deep learning models. We also show insights on what features are important for the classifications.

In the second part of the experiments, we present the results obtained on the "find-number" variables with the regex-based algorithm.

7.1 Classifications

There are one binary variable, *Tipo-T*, and five multi-class variables: *Grading*, *Stadio-N*, *Stadio-T*, *Sentinella Asportati*, *Sentinella Positivi*. In section **??** we described and motivated the simplifications that were made.

We try to classify the records using: Decision Trees, Random Forests, XGBoost, Support Vector Machines, Neural Networks (both Multi-Layer-Perceptron and Transformer).

Since there was no much difference between *bag-of-words* and *TF-IDF* data formats, in the presented results the non-deep-learning models use the

Accuracy					
	Grading	**Stadio N**	**Stadio T**	**Sentinella Asportati**	**Sentinella Positivi**
Decision Tree	92.3%	95.3%	89.6%	75.6%	87.6%
Random Forest	**94.8%**	**97.6%**	97.0%	83.5%	**92.2%**
XGBoost	94.2%	97.2%	**97.6%**	**84.6%**	90.7%
SVM	93.4%	96.8%	94.4%	83.9%	91.1%
MLP	91.6%	93.1%	94.0%	71.7%	86.8%
Transformer	94.0%	93.4%	94.6%	70.1%	86.8%
num classes	3	5	5	4	3

Table 7.1: Accuracies on the test set for the multi-class classification variables.

Macro F1					
	Grading	**Stadio N**	**Stadio T**	**Sentinella Asportati**	**Sentinella Positivi**
Decision Tree	91.5%	92.3%	84.5%	66.3%	75.5%
Random Forest	**94.1%**	**96.5%**	93.3%	73.1%	80.4%
XGBoost	93.5%	95.9%	**94.4%**	**77.2%**	81.2%
SVM	92.5%	95.3%	89.8%	75.4%	**82.0%**
MLP	90.6%	87.7%	82.8%	62.6%	69.0%
Transformer	93.3%	89.7%	91.6%	64.3%	72.8%
num classes	3	5	5	4	3

Table 7.2: Macro F1 on the test set for the multi-class classification variables.

bag-of-words representation, which is simpler.

For the multi-class variables, we present the results in terms of *accuracy* in Table 7.1 and in terms of *Macro F1* in Table 7.2. For the binary variable, we evaluate the *accuracy*, *precision*, *recall* and *F1* metrics (Table 7.3).

To investigate what tokens do the algorithms use to classify the records, we control the importance of the features given by random forest (Table 7.4 and 7.5) and the learned decision trees (Figure 7.1 and 7.2).

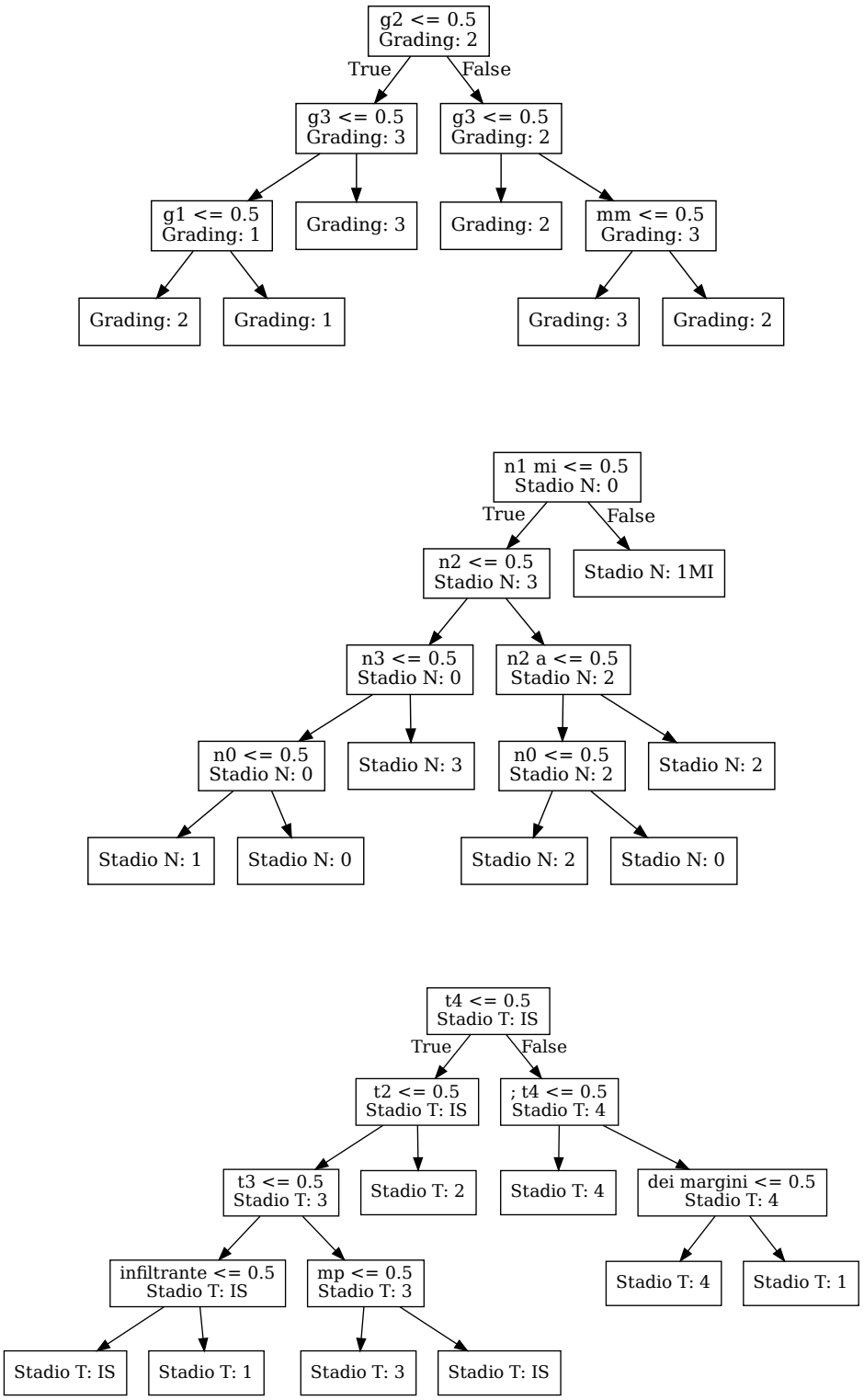

Figure 7.1: Decision Trees for "grading" (top), "stadio_N" (center) and "stadio_T" (bottom). Token <= 0.5 means that the token is not present in record of the patient. "Variable: class" indicates the prediction.

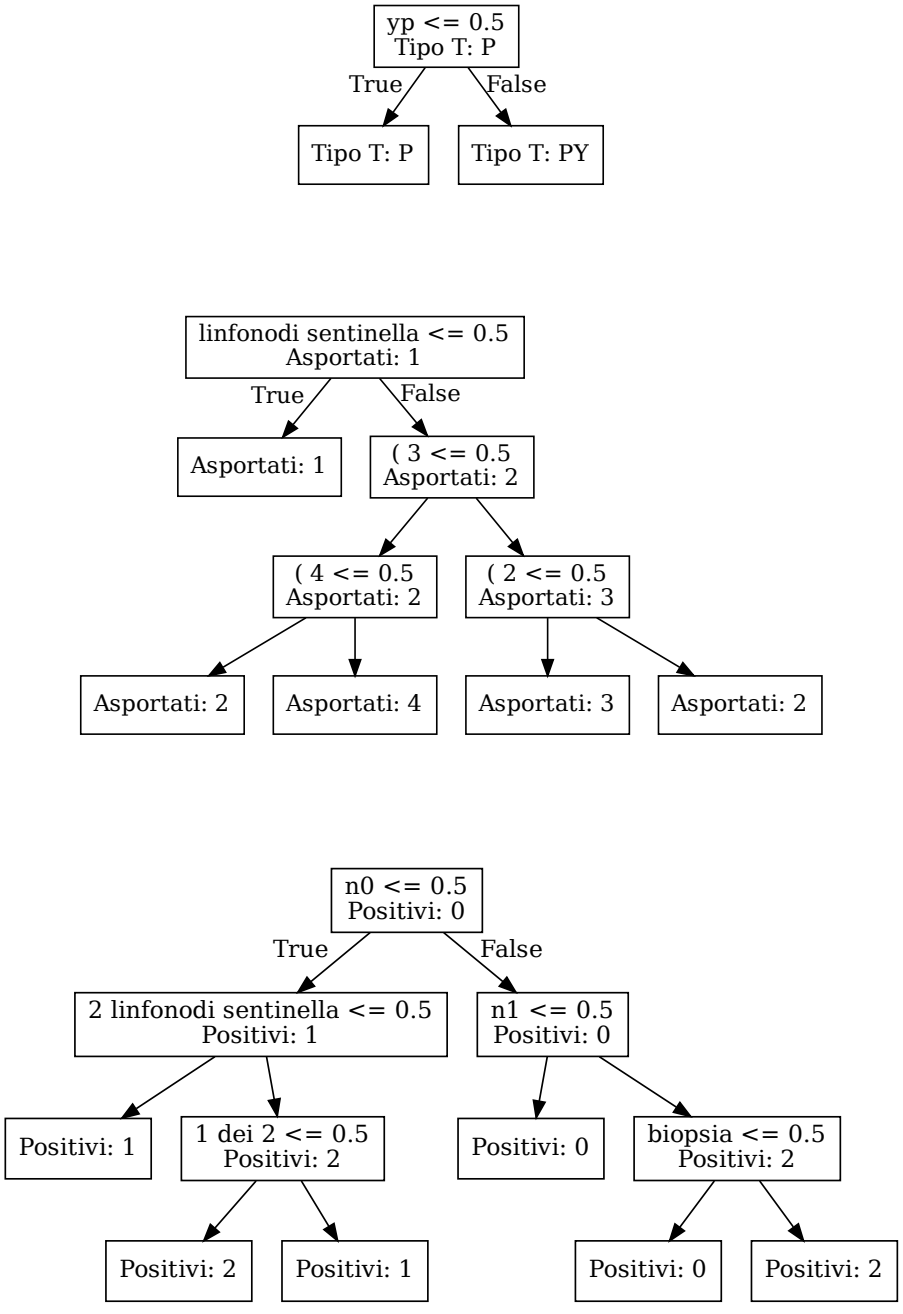

Figure 7.2: Decision Trees for "tipo_T" (top), "numero_sentinella_asportati" (center) and "numero_sentinella_positivi" (bottom). Trees of numero sentinella variables have been simplified to be better displayed. Token <= 0.5 means that the token is not present in record of the patient. "Variable: class" indicates the prediction.

	Accuracy	**Precision**	**Recall**	**F1**
Decision Tree	96.5%	48.2%	70.2%	57.1%
Random Forest	96.5%	47.8%	**77.2%**	**59.1%**
XGBoost	96.4%	46.1%	61.4%	52.6%
SVM	96.4%	47.1%	71.9%	56.9%
MLP	**96.9%**	**53.1%**	45.6%	49.1%
Transformer	95.6%	40.2%	68.4%	50.6%

Table 7.3: Metrics on the test set for *Tipo T* binary variable.

	Grading	**Stadio N**	**Stadio T**
1	g2	n1	t2
2	g3	n0	p t2
3	g1	n1 a	p t1
4	nos g3	n2 a	t1
5	(g2	n2	ptis
6	scarsamente differenziato	p n1	t1 c
7	scarsamente	p n1 a	p t1 c
8	infiltrante nos g3	n3 a	: p t2
9	(g3	n0 (	t2 ,
10	moderatamente differenziato	n3	infiltrante
11	g1 . invasione	n1 mi	: ptis
12	g1 (	p n0	p t4
13	g1 .	. infiltrazione	: p t1
14	g2)	(i -	t4
15	moderatamente	p n1 mi	. p t2
16	ii]	i -)	t2 , p
17	g2 (	p n2	duttale in
18	g3)	. infiltrazione neoplastica	p
19	g3 .	n1 mi (	carcinoma duttale in
20	, scarsamente	p n0 (	p t2 ,
21	, moderatamente differenziato	infiltrazione	duttale in situ
22	(g1	(i	t4 b
23	- g1 (	n0 (i	t1 b
24	: ii	micrometastasi	t2 p
25	cribriforme g1	p n3	edizione : ptis
26	%	mi	t3
27	tubulare g1	mi (	situ della
28	g3 (	(micrometastasi	p t1 b
29	scarsamente differenziato (	p n2 a	t2 ;
30	g3 multifocale	p n3 a	cerb
31	(g1)	neoplastica di	(p t2
32	iii]	, p n1	p t3
33	g1)	linfonodi	p t4 b
34	stologico : ii	linfonodi ascellari	situ
35	: ii]	sn)	duttale infiltrante
36	(g2)	sentinella (micrometastasi	ki67
37	(g3)	infiltrazione neoplastica di	carcinoma duttale infiltrante
38	: iii	i -	capezzolo
39	duttale infiltrante	capezzolo	p t2 p
40	ben differenziato	coinvolgente	microcalcificazioni . b5

Table 7.4: Top-40 most important features for prediction of variables “grading”, “stadio_N”, “stadio_T” with Random Forests. Remind that in preprocessing step we replace words like “grado 1” with “g1”, and we put spaces around characters like “n2”.

	Tipo T	**Sentinella Asportati**	**Sentinella Positivi**
1	yp	linfonodi sentinella	n0
2	p	linfonodi sentinella (	2 linfonodi sentinella
3	: yp	nel linfonodo sentinella	n0 (
4	, yp	sentinella (2	n1
5	edizione : yp	linfonodo sentinella (	i -)
6	ypmx	sentinella (3	linfonodo sentinella
7	sentinella	(3	linfonodo sentinella (
8	p t1	nel linfonodo	(i -
9	ki67 (	(2	2 linfonodi
10	cerb (	linfonodi sentinella .	linfonodi sentinella
11	ki67	2 linfonodi sentinella	p n1
12	, ypmx	3 linfonodi	p n0 (
13	neoadiuvante	3 linfonodi sentinella	-)
14	maligna . parametri	linfonodo sentinella	p n0
15	fish	2 linfonodi	i e
16	. yp	linfonodi	i -
17	x	(3)	(i
18	score : 2	(2)	invasione
19	margine	(4	2 dei
20	cerb /	sentinella (1	e ii
21	mm	linfonodo	sentinella (
22	cellule neoplastiche	sentinella (4	1 linfonodo sentinella
23	chemioterapia	intraoperatoria linfonodi	ascellari
24	. parametri	sentinella (	14 g .
25	. parametri biologici	3) .	coinvolgente 1
26	biologici	(4)	1 dei
27	ki67 (clone	sentinella	linfonodi sentinella (
28	centromero	4 linfonodi sentinella	linfonodi ascellari
29	, ypmx .	neoplastica nel linfonodo	i e ii
30	. score	4 linfonodi	1 dei 2
31	segnali	2) .	. infiltrazione
32	color probe	nei linfonodi sentinella	(ck -
33	ptis	biopsia intraoperatoria linfonodi	n1 a
34	5	linfonodo sentinella n	-) (
35	) :	2 - sn	nei 2 linfonodi
36	terzo	dei linfonodi	neoplastica di 1
37	dal quale	nei 3	linfonodo sentinella positivo
38	del tessuto osseo	(3 -	n2
39	chemioterapia neoadiuvante	parametri	micrometastasi
40	margini	sentinella n	ad 1 dei

Table 7.5: Top-40 most important features for prediction of variables "tipo_T", "numero_sentinella_asportati", "numero_sentinella_positivi" with Random Forests.

7.2 Find-number

These variables require finding a number in the text.

We use the approach based on regular expressions that was described in section 6.4.3 to extract these numbers. Note that this approach is not a Machine Learning Algorithm: nevertheless, it still has parameters (such as the dimension of the searching window) that are crucial to obtain good results. We manually chose these parameters evaluating the metrics on the training and validation sets.

In table 7.6 we report the results obtained on the test set, with metrics described in section 6.8.2.

In future works, these results can be used as baseline for Machine Learning algorithms. In section 8.2, we suggest a possible strategy to formulate the extraction of these variables as a learning problem.

	Extracted	**Acc. on extracted**	**Hit**
recettori estrogeni	96.3%	71.3%	68.7%
recettori progestin	96.8%	89.9%	87.0%
mib1	100%	18.2%	18.2%
cerb	96.1%	83.6%	80.4%
ki67	97.7%	81.9%	80.0%
dimensioni	76.9%	73.7%	56.7%

Table 7.6: Metrics on the test set for the "find-number" variables. The poor results for the *mib1* variable can be explained by the fact that it has very few labels: for this reason, the chosen parameters might be good only for the few training and validation examples.

Chapter 8

Conclusions

8.1 What we discovered

In this dataset there are different kinds of variables. Some of them are suitable to be classified, while others require finding numbers in the text. We excluded the *metastasi* variable from the study since it has very few labels, and the few that are present seem to have an unreal distribution: only a handful of patients did not have metastases.

In section 1.3.1 we have formulated a question, to which we have tried to answer through the experiments.

Question: Is it possible to predict variables from these reports?
Short answer: it's possible to have high-accuracy predictions in most cases.
Long answer: the *mib1* variable has very few labels: the quality of predictions is not satisfactory. Variables *dimensioni*, *numero_ sentinella_ asportati*, numero_sentinella_positivi and *tipo_ T* are a bit difficult to predict: we achieved decent results. In the other cases, at least one algorithm has very good performance. Whether these accuracies are enough can be verified only using these models in practice.

8.2 Future works

Possible improvements in future works include predicting also rare values, and predicting the sub-class for the *stadio* variables.

It would be interesting to have a predictor that recognizes if the variable of interest is **written in the report or not**. It must be clarified if the available data is sufficient for this purpose or you need more information.

Another thing to try is to obtain data from **cancer registers of other Italian regions** and not only from Tuscany. This would help to more precisely validate the models and to create more reliable models.

Having more data available, it might be worth trying to **add unsupervised tasks** for neural-network-based models and check if that improves single-task metrics.

Compact decision trees give human-interpretable decisions, while ensembles of decision trees provide insights on feature importance: these models obtained good results in our experiments. Future studies can focus on obtaining **interpretable models**, such as optimal decision trees, as we didn't create the models with interpretability as objective.

"Find-number" variables should be tested with some **machine learning models**. A possible strategy might be to find the location of the ground truth in the reports and formulate the problem as a localization/segmentation one. In many cases, there is exactly one appearance of the ground truth in the patient's record, though it can happen that either it is not present or that it shows up multiple times. From Figure 8.1 to 8.6 we show how many times the ground truth appears in the text as a substring, for each "find-number" variable.

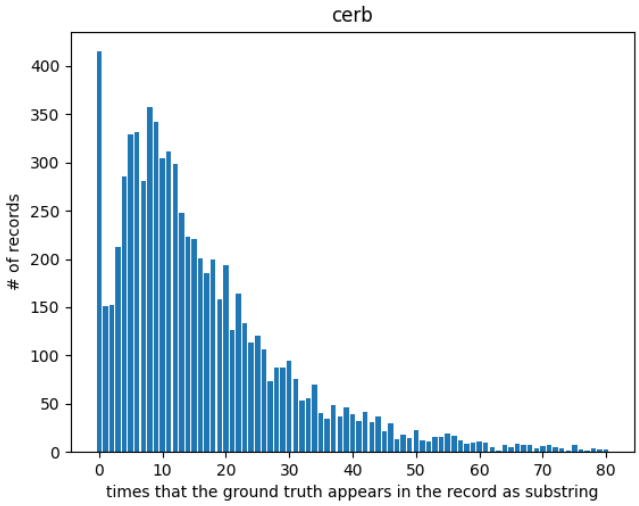

Figure 8.1: The ground truth often appears in the text many times: there is a lot of ambiguity.

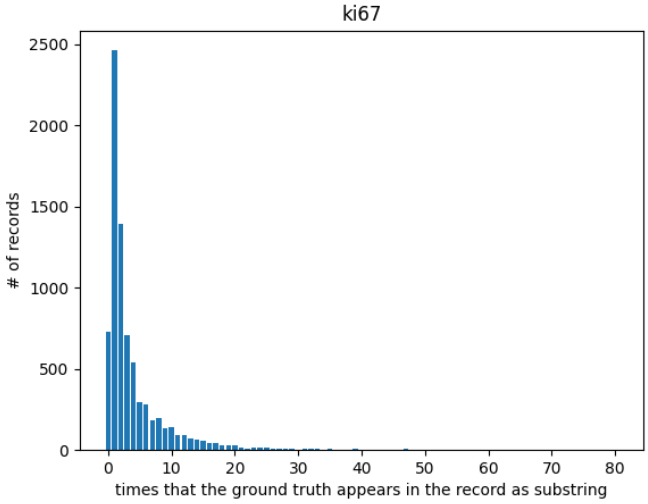

Figure 8.2: Most of the times, the ground truth appears just once in the record. There is some noise, but not so much, it seems possible to determine the position of the ground truth in the text.

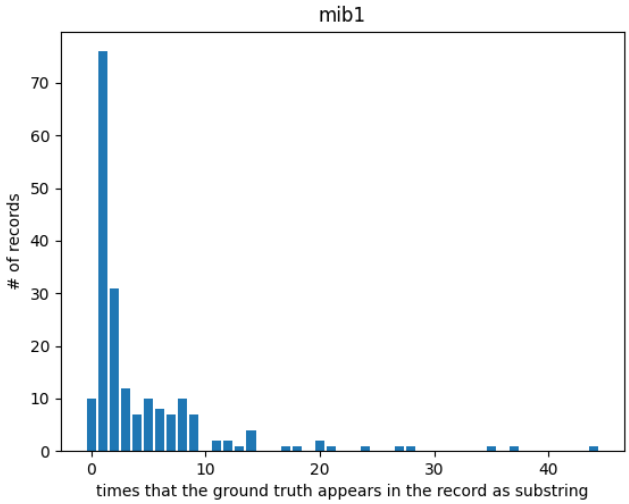

Figure 8.3: Most of the times the ground truth appears just once in the record. There is some noise, but not so much, it seems possible to determine the position of the ground truth in the text.

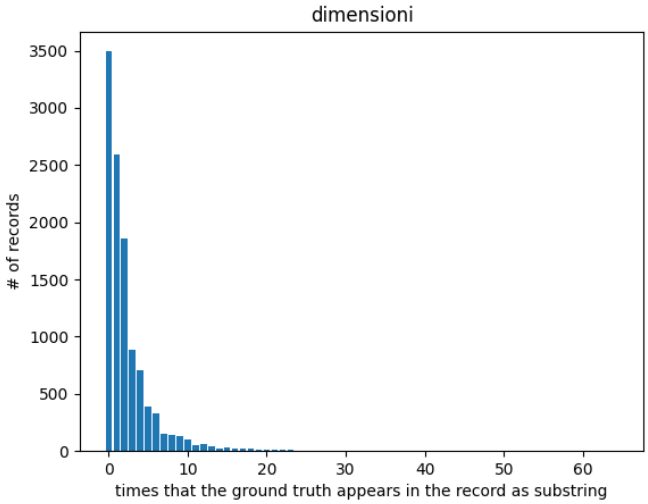

Figure 8.4: Most of the times the ground truth does not appear in the record as an exact substring. This is partially due to the use of centimeters in the text, whereas the labels are always in millimiters.

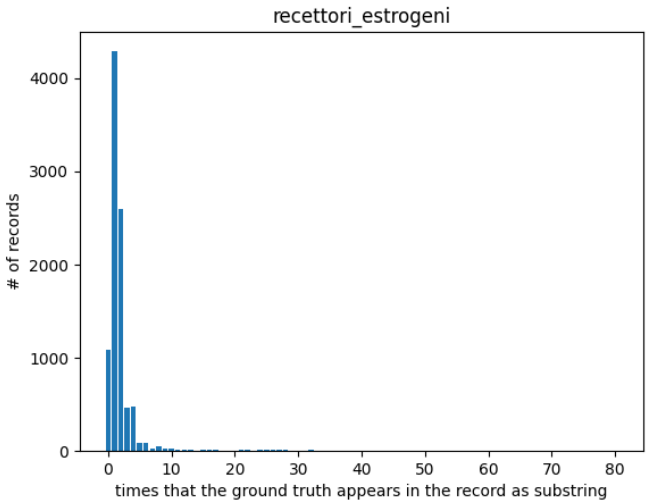

Figure 8.5: Most of the times the ground truth appears just once in the record. There is some noise, but not so much, it seems possible to determine the position of the ground truth in the text.

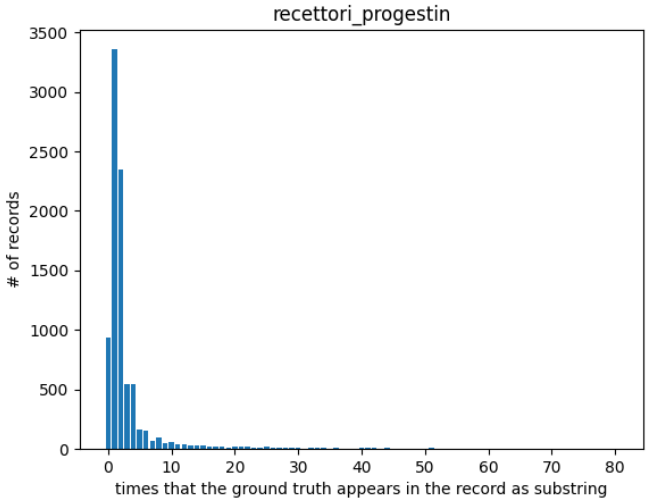

Figure 8.6: Most of the times the ground truth appears just once in the record. There is some noise, but not so much, it seems possible to determine the position of the ground truth in the text.

8.3 Final considerations

We experimented that *Neural Networks*, who achieve state-of-the-art results in many different fields, did not perform better than the other algorithms on this dataset. One of the usual advantages of using *Neural Networks* is that they are capable of learning good representations of the examples. Probably, a bag-of-n-grams representation is enough to describe the record of a patient in this dataset, and Neural Networks are not capable of extracting a better representation. The cause can be the limited amount of data and how these reports are written: there are mainly keywords and numbers.

The good results obtained with *XGBoost* and *Random Forest* algorithms indicate that they could be effectively used in practice. They have margins of error, but they make it possible to carry out statistics on recent data, which wouldn't be possible with the only work of human experts.

We imagine that two cancer registers that proceed at a different speed can coexist: the existing one is slower but more reliable, with records labeled by the experts; the other one can be less reliable but faster, with records labeled by the algorithms.

We hope that the results of this thesis can help those who monitor if people are receiving adequate care for cancer. The feeling is that we are at the beginning of the exploration of artificial intelligence techniques to analyze data in this field. Hopefully, many other works will be done in this direction.

Bibliography

[1] Stefano Martina, Leonardo Ventura, and Paolo Frasconi. Classification of cancer pathology reports: A large-scale comparative study. IEEE J. Biomed. Health Informatics, 24(11):3085–3094, 2020.

[2] World Health Organization. International classification of diseases for oncology (ICD-O) – 3rd edition, 1st revision. 3rd ed. edition, 2013.

[3] Shang Gao, Michael T Young, John X Qiu, Hong-Jun Yoon, James B Christian, Paul A Fearn, Georgia D Tourassi, and Arvind Ramanthan. Hierarchical attention networks for information extraction from cancer pathology reports. Journal of the American Medical Informatics Association, 25(3):321–330, 11 2017.

[4] Shang Gao, John X. Qiu, Mohammed Alawad, Jacob D. Hinkle, Noah Schaefferkoetter, Hong-Jun Yoon, Blair Christian, Paul A. Fearn, Lynne Penberthy, Xiao-Cheng Wu, Linda Coyle, Georgia Tourassi, and Arvind Ramanathan. Classifying cancer pathology reports with hierarchical self-attention networks. Artificial Intelligence in Medicine, 101:101726, 2019.

[5] Associazione Italiana di Oncologia Medica (AIOM). I numeri del cancro in italia, 2020.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[7] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[8] J. R. Quinlan. Induction of decision trees. Mach. Learn., 1(1):81–106, March 1986.

[9] J. Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[10] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. Classification and regression trees. CRC press, 1984.

[11] Leo Breiman. Random forests. 45(1):5–32, October 2001.

[12] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In Machine Learning, pages 273–297, 1995.

[14] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 1958.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9 of Proceedings of Machine Learning Research, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[16] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey, 2015. cite arxiv:1502.05767Comment: 43 pages, 5 figures.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms., 2016. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. CoRR, abs/1706.03762, 2017.

[19] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pretraining help deep learning? J. Mach. Learn. Res., 11:625–660, March 2010.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. CoRR, abs/1810.04805, 2018.

[21] A. Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[22] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(56):1929–1958, 2014.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[26] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ, third edition, 2010.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. `http://www.deeplearningbook.org`.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.

Greetings

I want to thank Professor Paolo Frasconi for all the supervision on how to handle machine learning models and on how to deal with all the decisions of this real-life problem from a statistical perspective. He always gave me interesting hints about how to develop new ideas for this research. And he was decisive in setting up all the work rigorously and scientifically.

I also wish to thank Leonardo Ventura - and his colleagues of ISPRO - without whom this thesis wouldn't have been possible. He explained to me the peculiarities of cancer registers, and how can computer science be helpful in this field.

Many thanks go to Stefano Martina, who told me about what he previously experimented with in his researches, and to Professor Simone Marinai for the help in the final stages of this thesis.

I would like to thank my girlfriend, Giulia, for all the support she showed me during this thesis - and for having always believed in me. She motivated and helped me in all the possible ways, especially when I was full of doubts and concerns.

Special thanks go to my family, who always made sure that nothing missed me, and to all the people who have been close to me in the latest years.

Finally, thanks to all my classmates of the University. It was a pleasure for me studying with them.