University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Text Mining and Machine Learning techniques for job reports analysis

Supervisor
Bruno Scarpa
University of Padova

Master Candidate
Luca Dal Zotto

Matriculation N°
1236343

Academic Year
2020-2021

To my family,
for its constant support and encouragement.

Abstract

The Big Data Revolution gave a boost to the introduction of innovative techniques with solid mathematical and statistical foundations in numerous fields. This kind of solutions have demonstrated how effective and efficient they can be for the data analysis, producing useful insights and leading to an increase in productivity. In order to perform well in a large variety of problems, many specialized methods have been developed, taking into consideration the context, the purpose of the analysis and, most importantly, the available data. The inclusion of textual data represents one of the most complex challenges, since for a computer program it is difficult to understand the dependence between words in a sentence and the concealed information, due to the elaborated structure of the human language. However, its growing importance and diffusion have motivated many studies, which were able to produce some powerful techniques suitable for different scenarios. Machine Learning is one of the approaches that have caught the interest of many researchers, thanks to its flexibility and its wide range of applications. After having focused on the theoretical fundamentals of text mining and learning methods, this thesis work reports an example of application of these tools, represented by the project faced during the master's degree internship period. A remarkable aspect is that here the complete workflow is described, starting from the data collection phase, to the implementation and evaluation of the models, which are then ready to be used in practice.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AWS	Amazon Web Services
IT	Information Technology
VPN	Virtual Private Network
iSA	integrated Sentiment Analysis
BoW	Bag of Words
TF-IDF	Term Frequency – Inverse Document Frequency
LDA	Latent Dirichlet Allocation
ML	Machine Learning
SVM	Support Vector Machine
RBF	Radial Basis Function
KNN	K-Nearest Neighbors
IG	Information Gain
DL	Deep Learning
FFNN	Feed Forward Neural Network
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory
PCA	Principal Component Analysis
t-SNE	t-distributed Stochastic Neighbor Embedding
KL	Kullback–Leibler
HR	Human Resources
DBA	DataBase Administrator

NLTK	Natural Language ToolKit
MCC	Matthews correlation coefficient
ReLU	Rectified Linear Unit
ELU	Exponential Linear Unit
GELU	Gaussian Error Linear Unit
SGD	Stochastic Gradient Descent
RMSprop	Root Mean Squared Propagation

1 Introduction

"The computer is incredibly fast, accurate, and stupid. Man is incredibly slow, inaccurate, and brilliant. The marriage of the two is a force beyond calculation."

Leo Cherne

The technological progress of recent years led to the introduction of computer science in a wide variety of fields. Even those activities that few decades ago seemed to have nothing to do with technology have discovered how it can actually be an effective and efficient tool. In general, its contribution can be noticed in an improvement both in terms of quantity and quality.

One of the ways in which technology can operate is by means of data: they are obtained measuring a certain phenomenon or collected if already produced. Then they are elaborated in order to have a computer abstraction, and, finally, some output data are produced, suggesting an action or providing some relevant insights about them. Since it represents the key ingredient of this new wealth-generating trend, data is also defined as the new oil (Bhageshpur, 2019), with the difference that it can arise everywhere.

The increasing complexity of available data and the rapid growth of the world-wide web have pushed even further this insurgent process, which has been given the name "Big Data Revolution". The web has become the largest publicly accessible data source in the word, making a wide range of opportunities available to billions of people. In this context, an example of a relevant data generation process is represented by social media, where users continuously upload new materials, giving an additional contribution to the whole data flow.

In order to exploit this huge and ever-growing amount of data, an elaboration is needed. Only in this way the hidden information can be revealed and used. Clearly, specific approaches have to be adopted depending on the problem at handle. The purpose of the study, the available tools and the kind of data itself determine how the analysis should be conducted. For this reason, a cooperation between the sensitivity proper of humans and the great potentiality of computers is fundamental: a machine cannot substitute the choices made by a person in order to tailor the analysis and interpret it.

In Chapter 2, it is presented the project carried out during the internship period spent at Miriade Srl (Miriade, 2021), an information technology company based in Italy. The activity perfectly represents an example of analysis that, starting from a data collection phase, leads to the production of an effective and productive tool, able to solve a useful task. In more details, its purpose is to automatically classify the job reports that are daily inserted by each employee, detecting those that present a mistake in the selected job identifier. This task is currently conducted manually, through a time-consuming procedure.

One of the most common data types is the textual one. Indeed, this is the case for one of the most relevant variables to analyze for the project here reported. For its incredibly large number of possible applications and for the complexity of the human language, the analysis of this data format requires particular attention and the usage of specific tools. In Chapter 3, the way a text should be treated is explained from a theoretical perspective, also showing some possible model approaches.

Chapter 4 is also methodological. It gathers the explanations of all the techniques that have been considered for the analysis. The usage of learning methods was the approach chosen to build a model based on the information extracted from the data at handle. More in details, in that chapter, some classifiers and some deep learning architectures are presented, describing their mathematical foundations and also explaining their capabilities. Furthermore, two dimensionality reduction approaches which have been involved in some additional experiments are described.

In Chapter 5, the preliminary stages of the analysis are reported. The first one is the exploration of the available dataset. This phase is fundamental, since its purpose is to understand more deeply the data at handle and to choose how to solve the task, determining the subsequent steps. The following phase is indeed the pre-processing one, which consists in elaborating the data, transforming it in a more functional format and making the information it encloses more explicit and workable.

Then, in Chapter 6, it is described in details how the models have been implemented, showing all the experiments conducted. Particular emphasis is given to the role of textual variables and to some personalized learning architectures. Finally, the results obtained with all the different approaches considered are listed and compared.

To conclude, some final considerations and remarks are reported in Chapter 7.

2
Presentation of the internship project

The project discussed in this thesis work has been carried out during a company internship. The purpose of this chapter is to provide an introduction to the problem, first of all, giving an overview of the company, useful to understand the context in which the analysis has been performed. Then, the specific process at issue is described in details, showing how the data has been generated and collected. Most importantly, it is explained why an intervention is required and what is the final aim of the project, along with the motivations behind it. In this way, it is possible to discover which are the main characteristics of the problem at handle, and therefore, choose the proper techniques and methods for the analysis.

2.1 The company and the internship

As part of the master's degree in Data Science, students have to carry out an internship at a company, industry or research center, with the purpose of building up new skills, getting in touch with the daily working environment and attitude of a company, possibly developing awareness towards the market and clients' needs (University of Padua, 2021).

The thesis project here described was faced during the internship period at Miriade Srl (Figure 2.1). It is an IT consulting company based in Italy, with 20 years of experience in the field of Data Engineering, Data Science and Cloud Strategy. Its headquarter is in Thiene (VI) but it also has some offices in Padova and Mestre (VE).

MIRIADE
PLAY THE CHANGE

Figure 2.1: Miriade Srl logo.

Miriade belongs to the same group as two other companies: Autentica Srl and Filanda11 Srl. The former is the soul of the project. Indeed, it is a technological hub in which IT, service design, marketing and management solutions converge. Instead, Filanda11 represents a partner for the digital evolution, offering techniques of design thinking aiming at improving client business (Autentica Group, 2021).

Miriade, for its part, offers expertise and assist clients (mostly in North-Eastern Italy) both in the private and public sector with a complete and tailor-made service to meet the customer's specific needs. It is organized in three areas:

1. Data & Analytics: here, many different aspects regarding data management and data analysis are considered. Some sub-areas are Artificial Intelligence and Machine Learning, Business Intelligence, DataBase Administration, DataOps and Data Integration;

2. Applications: the purpose of this area is to develop web applications exploiting innovative technologies, such as Amazon Web Services (AWS) products;

3. Infrastructure & Operations: as the name says, it focuses on the management of customers' computer infrastructures, under different aspects such as Cloud technologies, Cybersecurity and groundbreaking methodologies, such as DevOps (a combination of software development and IT operations leading to a better productivity, Amazon AWS documentation, 2021).

The internship started on 6th April and it lasted six months. During this period, two different activities have been carried out: approximately half of the time was meant to provide a job formation inside the area Data & Analytics, by practicing with the technical tools used by the company, while, for the remaining three months, the thesis project that is discussed herein have been faced.

2.2 Job reports

Now, let's see more in details the objective of the internship project. At the end of each working day, all the employees of Miriade's group have to compile a job report, recording inside a web-based application all the activities faced during the day. The goal is to provide a way both for monitoring the activities of each resource and for the automatic generation of invoices. This means that what is inserted is used for presence recording and for producing bills at the end of each month. This tool has been implemented inside the company intranet by means of a bot web portal, which is used also for other purposes, such as vacation requests and other kind of permissions.

The compiling procedure of the job report is straightforward. The first step is to visit the web page of the company bot, and log in using as credential the corporate e-mail address and a password. As a result, the interface shown in Figure 2.2 appears.

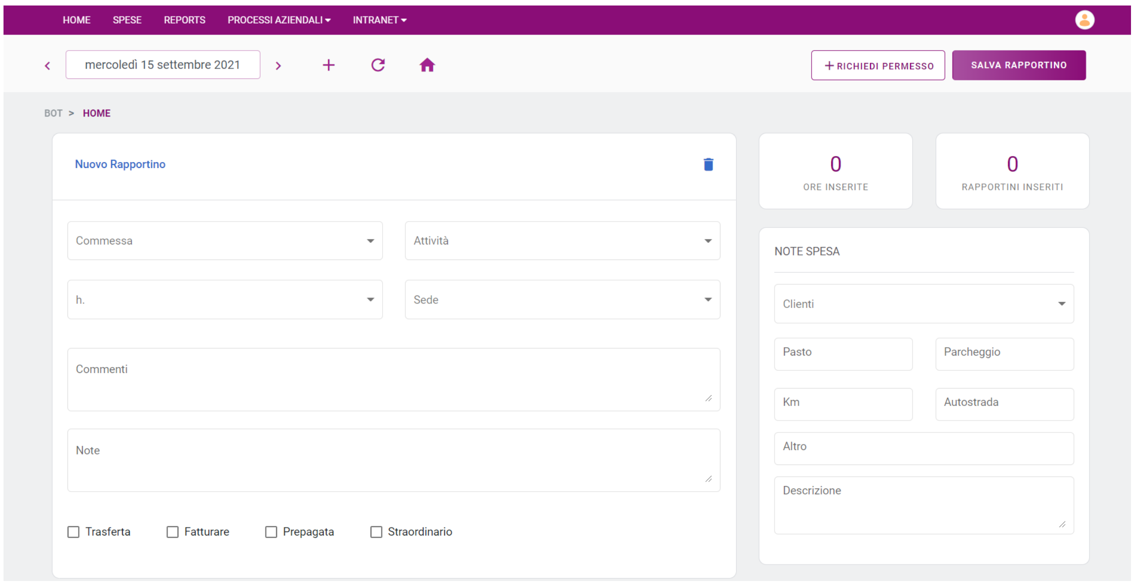


Figure 2.2: The interface where job reports are inserted.

The first field to compile is the one relative to the project name (in the figure *commessa*). After having clicked on it, a drop-down menu pops up, showing a list of alternatives. Each possible identifier is made of two parts: a prefix, which can be the name of a customer or "MIR" in case of internal activities, and a suffix corresponding to the generic activity at issue. For example, the activities regarding the analysis reported in this thesis and the job formation carried out during the internship had as job identifier "MIR - FORMAZIONE", since they did not involve any customer, and their general purpose was to provide an instruction. As it will be shown later, each employee has different options to compile this field, mostly depending on the area they belong to. For instance, a database administrator would not share the choices available for a developer.

More in general, there are three types of job identifiers for customer activities:

1. closed projects: those having a fixed number of working days, agreed with the customer;
2. open projects: those without a specific deadline, but only with a pre-arranged tariff for each day;
3. project organized in work packages: those paid when a target result is reached, independently of the number of days required.

Secondly, it is necessary to select from another drop-down menu an activity, which gives more specific indications about the kind of work faced by the resources. In this case, the available choices depend on the project previously selected. For internal projects, there are many options, instead, for customer activities, in general the default one is consultancy, which comes along with some alternatives, such as overtime. In case this latter option is selected, it is also necessary to flag the corresponding box and to add the starting and ending hours of the activities.

Then, the workplace must be specified and also how much time the activity required. Note that this must be a multiple of 30 minutes. These are also closed-ended questions.

A field extremely relevant to this project is the "comments" one. Here the employee has to write down the complete description of the technical activities he has been carrying out. This is an open question field that has to be compiled by scratch, as the subsequent one, "note". The difference among these two is that the former is visible also to the customer, while the latter only internally. Therefore, in the note field, for instance one can insert some ticket identifiers which are meaningless for the customers but crucially important for internal references. The note field is also used to provide additional information, possibly also doubts about the project name and activities selected, aimed at those who will check the job reports.

As shown in Figure 2.3, when some particular project names are selected, a progress bar appears at the bottom of the page, representing how much time has elapsed since the beginning of that project in proportion to the agreed total time (open projects are excluded). Furthermore, on the top right corner of the page, a number indicates the remaining hours inside a green icon, which becomes red if the planned deadline has already occurred. Moreover, an automatic check is performed over the textual filed and if a ticket code is specified, the associated issue is shown.

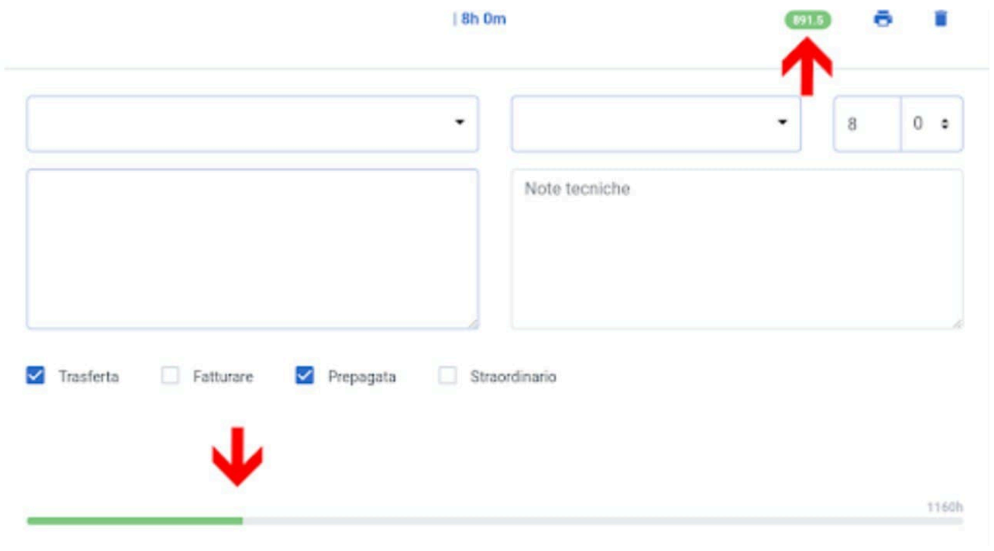


Figure 2.3: Example of a progress bar in a job report.

To conclude there are some additional flags: some of them, regarding billing aspects, are automatically compiled. A separate flag is used to indicate whether the activity has been conducted at the customer workplace or not. They can be manually modified by adding or removing the check mark from the corresponding box.

At the end of each month, project managers check all the job reports inserted in that month, paying particular attention to the project name used, comparing it with the description provided. If the choice is correct, then the job report is approved and it can be effectively used for billing purposes, while if the employee's insertion is wrong, the job report is rejected. In this case, the project manager may provide a correction or send a message to the employee who inserted it for further explanations.

Considering that the company has around 90 employees, the total number of job reports which have to be controlled by the project managers each month is around some thousands (one employee can insert more than one job report each day, if working for several activities). As a consequence, this requires a significant amount of time for a tedious activity. For this reason, an automation of the inspection process would be very useful, considerably reducing the time needed. This is the motivation behind this project. The idea is to insert a tool inside the company intranet which verifies the correctness of job reports, highlighting those which are likely to be wrong. If the tool reaches reliable results, then project managers could check only a small fraction of them, leaving the bulk of the work to an algorithm.

2.3 Data retrieval

The very first step to take in order to start the analysis is to retrieve the dataset. For more than one year, the details present in all the inserted job reports have been collected. Most importantly, each record has been associated with a label, indicating if it had been accepted or refused by the project managers during the review process. Once a job report is checked, a new sample is saved into the dataset, whose size is thus continuously growing.

Therefore, at the moment in which the analysis began (April 2021), a sufficiently large ground truth was available. Since it is stored in a Postgres database (Postresql, 2021), to download an updated version, one can use the platform pgAdmin (pgAdmin, 2021). The dataset can be accessed only from the company network, so, it is required to connect to a Virtual Private Network (VPN) if attempting to access from a place different than the office.

To retrieve the data the first time, it is necessary to create a dedicated server, with the connection pointing to the bot host (which is based on an AWS service). Once established the connection for the first time and having found the correct schema which owns the data, it is possible to query the database using SQL language. At this point, with a simple select query, the dataset can be materialized and downloaded.

3 Text mining techniques

Data comes in a variety of forms and each of them requires specific techniques in order to extract information and make them valuable. The presentation provided in the previous chapter has brought out the main characteristics of the problem. Now the discussion focuses on the methods that can be exploited for handling textual data. They can be originated from a wide range of sources, such as social media, books, web sites, newspapers, marketing surveys, historical archives and many others. In this project, this is the case for the comments inserted in the job reports by the employees, in order to provide a description of the activities performed during the working day. The information carried by this data type is often very implicit and requires an advanced understanding of the context (Lebart et al., 1998). For these reasons, it has been object of many studies and researches. Nowadays, a number of disciplines deal with large text sets requiring both text management and analysis, making it one of the fields with state-of-the-art techniques in quick progress (Ceron et al., 2017).

First of all, let's introduce some definitions. In order to perform a statistical analysis, more than a single text or document is necessary. To refer to a large and structured set of texts, the term *corpus* is used. In the case of this thesis project, it consists in the set of all the descriptions contained in the job reports.

The nature of a *corpus* is determined by the kind of written resources it comes from, and also by the topic it regards. For instance, to analyse the text present in books, a different approach should be adopted compared to the analysis of the comments parsed from a social media. The reason is that in the first case, the language can be very articulated and with terms coming from a large vocabulary, while, in the second case, sentences can be shorter, and, in general, they are more schematic. Besides the differences that can be found from a syntactic perspective, also different semantic levels require specific approaches.

Another aspect to consider is the linguistic register: depending on the objective of a piece of text, the way of communication changes accordingly. For example, advertisements have to be clear and direct, while, on the opposite side, there are situations where language should be as much formal as possible, such as for contracts with legal value.

Finally, the same words may assume different meanings depending on contexts and on the people reading them. The interpretability is probably one of the most challenging aspects to deal with when analysing a text especially with computer-based tools.

So, how is it possible to make order to all this chaos? Are there some milestones on which the analysis can be based? How is it possible to exploit statistics and computer science in such a changeable and articulated scenario? The following sections provide an answer to these questions.

3.1 Text analysis and its principles

The key aspect to understand how a certain text should be interpreted is the application context. A first broad area of applications is that of scientific researches. Indeed, text mining has successfully been exploited to support scientific discovery in fields such as bioinformatics or social sciences. Furthermore, another research field which has gained importance also thanks to the analysis involving social media is the one aiming at determining ideas communicated through text. In this context, it is important to distinguish between sentiment and opinion analysis. The former has to deal with the estimation of the intensity (positive/negative) of a feeling (sentiment), while the latter involves the assessment of the motivations behind this feeling (why positive? why negative?) as well as the identification of the topics (Ceron et al., 2017). A closely related term is opinion mining, which indicates the process of discovering the positive or negative attributes connected to a list of keywords, analysing their distributions and extracting the underlining opinion for each keyword (Kushal et al., 2003). In some cases (e.g., Liu, 1997), the term sentiment analysis is used to replace text mining to indicate the area of study concerning opinions, emotions and sentiments of people with respect to a certain topic.

Secondly, text mining may be used for government or even military needs, such as national security or intelligence purposes. Indeed, terrorism, trafficking, smuggling and other forms of crime are using more and more new technologies for their own activities. Many tools have been developed to fight them, and, in particular, text mining allows to monitor and to analyse the content of online plain text sources such as suspected web sites, blogs and emails (Zanasi, 2009).

A third context which can be identified is the one regarding business application. This can include, for instance, the analyses asked by a company for the customer relationship management (Au et al., 2003). Furthermore, textual analysis can be engaged also to enhance the productivity, improving certain internal processes. The project described in this thesis falls into this category, since it exploits text mining techniques in order to improve the job reports' verification process.

Once having identified the area of application, the most suitable way of approaching the problem should be chosen. What is true in every scenario is that relying completely on brute force techniques, which automatically compute a number of metrics, is a wrong approach. For example, counting the frequency or simply the presence of a certain term in a document may be misleading, since that word itself can assume different nuances. This limitation of existing methods based only on computer science literature has been highlighted and condemned by Hopkins and King (2010). They said that since their only aim consists in correctly classifying documents into a given set of categories, then, these methods often produce bias estimates of these category proportions. Furthermore, they showed how, developing methods for analysing textual data that optimize social science goals directly, they managed to considerably outperform standard computer science methods developed for a different purpose, with the advantage of requiring no modelling assumptions, and being pretty fast.

To sum up, many researches agree on the fact that a type of analysis that is able to combine the accuracy of quantitative estimation over Big Data with a more qualitative approach, which allows to dig deeper in the data, becomes necessary.

In order to use and evaluate quantitative methods for content analysis, Grimmer and Stuart presented four principles of automated content analysis methods (Grimmer and Stewart, 2013). An important aspect is that these principles are valid in general, not only for the text analysis scenario, even though here they are adjusted with reference to this specific case.

Principle 1: All Quantitative Models of Language Are Wrong—But Some Are Useful

This first principle recalls George Box's aphorism "All models are wrong, but some are useful" (Box, 1979), and applies it in the text analysis context. In this scenario, it is possible to claim that the data generation process for any text is purely a mystery: given a sentence with a certain dependency structure, the inclusion of new words can drastically change its meaning. Think, for example, to the so-called functional words (articles, prepositions, adverbs, etc.): on their own, they do not have any special meaning, but they characterize the sentence. For instance, the use of a disjunctive conjunction may entirely flip the message conveyed. Therefore, it is necessary to distinguish between quantity and quality: it is true that a very frequent term may identify the topic of a certain text, but the sentence's interpretation may be determined by rare words. In addition to that, there are also other factors that change the mining of a sentence which cannot be found easily by an automated technique. This is the case of irony, metaphors and so on.

Principle 2: Quantitative Methods Augment Humans, Not Replace Them

Automated content analysis methods cannot do the whole work autonomously. For this reason, the human role is still fundamental, and cannot be replaced. It is still the researcher that guides the whole process, making modelling decision and interpreting the solution of the models. What is true is that computers can amplify human abilities, providing a powerful tool. In few words, humans need computers to improve their analysis, but computers need humans to operate productively, otherwise their brute force would be worthless. Therefore, cooperation is the key, and the final aim is to identify the best way to use both humans and automated methods for analysing texts.

Principle 3: There Is No Globally Best Method for Automated Text Analysis

The variety of possible contexts and the number of facets a text can assume precludes the possibility of having a single method suitable for all scenarios. As previously anticipated, the way newspaper articles and social media posts are analysed cannot be the same. However, the difference in terms of datasets is not the only reason preventing the existence of a global best method. Indeed, another factor is that on the same dataset, many different research questions may be asked and multiple quantity of interest could be sought-after. For instance, possible goals can be to identify the words that allow to discriminate between classes, understand the topic of a text, extract a very specific answer or provide a small summary. Therefore, many different models or families of models should be explored in order to identify the one that best suits the particular case.

Principle 4: Validate, Validate, Validate

Related to the previous principle, it is common to find differences also between text/documents belonging to the same case study. So, the use of one or the other when estimating the model will lead to different results. To understand which model is better, it is crucial for researchers to validate them, using a separate dataset. This point will be described more in details in section 4.1. For the moment, let's just underline the fact that a model that apparently performs well for certain documents, may lead to poor results on others, and vice versa.

3.2 Building variables from raw text

3.2.1 Text cleaning

The data collection phase, which is the one that comprehend all the steps necessary to retrieve a dataset, usually returns something in a human readable format. Such a dataset needs to be transformed in a way that it can be fed to a computer program. This procedure, called pre-processing, can be articulated in many stages, and highly depends on the input data type, and on the purpose of the analysis. Textual data make no exception. Actually, the need of such preliminary step for this kind of data is evident: which information can a computer extract from a written sentence? How can it be encoded into a meaningful format for an algorithm? Which encoding procedure should be followed to produce numeric variables?

Data Cleaning is the process of identifying the incorrect, incomplete, inaccurate, irrelevant or missing parts of the data and then modifying, replacing or deleting them according to the necessity. This is a crucial phase because it reduces as much as possible the noise inevitably present in the data, limiting the distortion that would be generated in the models. Some examples of elements that should be removed or modified are now listed:

- punctuation characters, such as "?", ";", "!" do not carry any information in a sentence, so, they could be removed from it. The only case they may be useful is when measuring the intensity of a sentiment;
- uppercase words and lowercase words clearly have the same meaning, but a computer sees them as different objects. A case normalization is the most obvious solution: in general, all words are converted in lowercase characters;
- stop words are the elements of a text used just for the sake of correct sentence formation, but most of the times, without carrying any additional meaning. This is the case of

articles or prepositions. Most of the times, these are the words that are more frequent in a text, but clearly, they do not provide any predictive power for the majority of the tasks. However, some exceptions may occur. This is the case of negative adverbs such as "no" or "not", which can flip the meaning of a sentence. For this reason, stop words should be treated with some precautions, depending on the problem at handle;

- URLs, email addresses, links increase the sentence complexity. Instead of simply dropping them, a more elaborated strategy could be to extract the keywords from them and remove all other parts. For example, an easy option consists in keeping only the domain name of a web address;

- emoji and other graphical parts should be discarded or encoded into words. Indeed, they can be useful in identifying the real feeling of the author of an online post.

After having filtered out some parts of the sentence, the computer should also be able to understand that different declensions of the same word should not be treated differently. In practise, each word should be reduced to its core root or base form, getting rid of prefixes, suffixes, declinations and conjugations. Stemming and Lemmatization are both techniques used for this purpose. According to the Collins Dictionary (Collins English Dictionary, 2021), Lemmatization is defined as the linguistics process of grouping together the inflected forms of a word for analysis as a single item. Instead, Stemming is the process of reducing inflected words to their word stem, base, or root form (Wikipedia, 2021). The critical difference is that a lemmatizer uses context and dictionaries to help discover, for example, that "good" is the base form of "better" and "best". The stemmer is a much cruder algorithm, but considerably faster. The performance does not seem to matter to most applications, so the majority of applied research uses stemming (Grimmer and Stewart, 2013).

After this stage of the pre-processing procedure, each sentence is reduced into a list of stems, that hopefully maintains most of the original information intact. This is what the so-called bag of words (BoW) model does. The name reflects the nature of its output: a simplified representation consisting in a set of (pre-processed) words, without order. This last characteristic shows a limitation of this easy approach: the information regarding the position of the word in the sentence is lost. A partial solution is represented by the usage of pair of words (bigrams) or even triads of words (trigrams) instead of single ones (unigrams). In this way, it is still possible to exploit local dependencies inside a sentence. In what follows, the word stem is used to indicate any generic n-gram, that is the atomic data to analyse.

3.2.2 Encoding techniques

The final step to conduct before building a model is the encoding into a more appropriate and functional format. The most straightforward approach consists in counting the frequency of each stem in all sentences belonging to the *corpus*. The resulting vector is then used as rows of a matrix, the document-term matrix. Its generic entry in position (i,j) represents the frequency of the j-th stem in the i-th document.

Note that a poor data cleaning may lead to a very large number of columns (too many useless stems are kept), so this kind of problem can be easily detected at this stage. What turns out to be true is that, in general, the total number of stems should be no more that 300 or 500 (Ceron et al., 2017). However, this number is influenced by many factors, such as the number of documents in the *corpus* and the linguistic register adopted. Indeed, it can happen that for a moderately large volume of documents without a particularly specialized vocabulary, this matrix may reach between three thousand and five thousand features or terms (Grimmer and Stewart, 2013). However, it is also true that almost all entries will be zero, a condition called sparsity, from which it can be possible to take advantage for storing data more efficiently.

A more advanced approach is TF-IDF, that is shorthand for Term Frequency – Inverse Document Frequency. Its first component, term frequency (TF), is basically the result of the BoW representations. It is computed as the ratio between the number of times a term t appears in a document d and the number of terms in that document.

$$\mathrm{TF}(t,d) = \frac{\text{number of times term } t \text{ appears in document } d}{\text{number of terms in document } d}.$$

So, given a specific document, it gives an indication about how important a stem is by looking at how frequently it appears in the document. However, giving importance to a word according to the number of times it appears may be misleading, and the term itself may not have the predictive power expected. Indeed, if that word is very frequent in all documents, then it is not useful to discriminate between categories, for example.

This explains why another factor, the inverse document frequency (IDF), is considered. Given a stem t, its inverse document frequency is defined as follows:

$$\mathrm{IDF}(t) = \log\left(\frac{\text{total number of documents}}{\text{number of documents containing stem } t}\right).$$

Then, the product between these two quantities (TF and IDF) is computed and the TF-IDF of term t within document d is obtained. In order to have high TF-IDF in a document, a term must appear a lot of times in that document and, at the same time, it must be rare in the other ones. Therefore, such a term is more likely to be characteristic of that document, and so, it may have a significant predictive power.

3.3 Different model approaches

Once the feature matrix is available, it is finally possible to build a textual analysis model. The most common use of content analysis methods in political science is assigning texts to categories. Different families of methods can be identified, but, to make order, it is possible to distinguish between scenarios where the categories are unknown and where categories are known. In the first case, many techniques can be found in turn: two famous approaches are automatic clustering and topic models. On the other hand, examples of possible methods to adopt when categories are known are those based on ontological dictionaries or other supervised methods. In this section, an overview of these approaches is given.

3.3.1 Clustering

Cluster analysis consists in rearranging data samples into homogeneous groups according to some notion of distance. In particular, it is necessary to define a function, taking as input two objects, that satisfies these properties: it returns zero if the two input objects are the same one, it is non-negative, symmetric and the triangular inequality holds. The idea is that close points in the feature space are similar, and therefore should be grouped together.

The most famous example of clustering algorithm is represented by k-means. It is based on the expectation-maximization approach. After having specified k, the desired number of clusters, k samples (the initial centroids) are picked randomly. Then, the following two steps are repeated until convergence or up to a maximum number of iterations:

1. each point is assigned to its closest centroid;

2. the centroids are re-defined by computing the mean of each cluster.

The random initialization can possibly lead to local minima, so it is good practice to repeat the process with different initializations.

3.3.2 Topic Models

Another family of methods is the one of Topic models (Blei, 2012). The goal is to discover the main themes faced in a collection of documents, making it possible to organize them accordingly. Latent Dirichlet Allocation (LDA) is one of the simplest topic models. It assumes that each text is a mixture of topics, which are distributed in the *corpus* according to the Dirichlet distribution. Indeed, a topic is formally defined as a distribution over a fixed vocabulary. The fundamental hypothesis is that during the data generation process, once a topic is selected, a set of stems is sampled conditionally on that topic.

By reversing this process, it would be possible to access the topics discussed in a given text based on the stems present in it. It is important to highlight the fact that this algorithm has no information about the subject of the documents, and no keywords or topic labels are required. The hidden structure of the documents is computed in a completely unsupervised fashion.

3.3.3 Supervised methods

Ontological Dictionaries

Ontological Dictionaries are perhaps the most intuitive method. It is based on the usage of keywords' frequency to classify a document. Suppose that the goal of a textual analysis is to understand whether a film is good or not according to spectators' reviews. In this case, a dictionary would be a list of words, each with an associated score, indicating the positive or negative feelings related to it. For simplicity, suppose that this score can be either 1 or -1. If the scores of all keywords in a document are summed up, it results in a continuous measure reflecting the customers' opinion. Then, by indicating one or more thresholds, each document can be assigned to a specific category. For instance, if the total score is positive, then, the film can be assigned to the good movies class, otherwise to the bad one.

The way this kind of methods work is very straightforward, simply relying on counting keywords and their corresponding score. The most difficult part of the whole procedure is the selection of the dictionary. Usually, finding the separating words is relatively easy. Furthermore, there exists a variety of off-the-shelf dictionaries providing keywords for a number of categories (Bradley and Lang, 1999). However, sometimes the problem at handle has a too specialized linguistic register that existing dictionaries may lead to poor results. In this case, a tailor-made dictionary has to be generated from scratch by hand, and this operation may require a significant amount of time and resources.

Furthermore, ontological dictionaries are effective to perform discriminative tasks among two or more categories only when words taken individually carry on their own a specific meaning useful for the classification purpose. In other terms, there must be some keywords representative for each class. A limitation connected to this aspect is that these methods are not able to handle irony, negations or other complex textual forms.

The procedure through which a semantic meaning is given to a text is also called tagging. When it occurs exploiting ontological dictionaries, the expression automatic tagging is used, while if it is human coding behind keywords extraction and classification, it is called human tagging. In both cases, a preliminary knowledge is necessary: dictionaries have to be defined in advance, and humans should have already seen similar documents and should know the meaning of all words, thanks to past experiences. For this reason, these methods are supervised. Tagging is just one example of this paradigm, possibly the easier one.

The approach selected to face the project reported in this thesis is the usage of classification techniques. The reasons are that clustering or topic models are not appropriate for the discriminative task considered. Furthermore, as it will be shown later, the data exploration phase revealed that even the usage of ontological dictionaries would not be suitable in this case, since the task cannot be reduced to finding keywords with a certain score reflecting the class of belonging. Therefore, the best option was represented by classification techniques, some of which are described in details in the next chapter.

4 Learning Methods

4.1 Classification techniques

The task at handle consists in classifying data samples into two different categories. After having defined the aim of the analysis, it is necessary to start introducing the mathematical formalism. The performance of a computer program should be evaluated in an objective way by leveraging a performance measure. Clearly, it is strictly connected to the task: the discrete-valued output proper of a classification problem implies the utilization of different metrics with respect to the real-valued output of a regression one. Also, the other way around deserves attention: very often, there are multiple possible performance measures for a single task, but some of them may lead to misinterpretation of the results. For these reasons, the best choice is to consider more than one metric, in order to have an evaluation under different perspectives.

Furthermore, one crucial point must be clarified: how can a computer program learn? To answer this question, a premise is needed. In general, two learning phases can be identified: a training phase, in which the model is defined and trained (in a way that will be described later) on a certain dataset and an evaluation phase, in which predictions on new data samples are computed.

When the "right answer" is available for each sample in the data, the expression supervised learning is used. Classification tasks fall under this paradigm. To start using some mathematical expressions, let $\boldsymbol{x}_i$ be the feature vector of the i^{th} samples. On the other hand, the corresponding target value is denoted by y_i. If the variables describing each sample are all numerical, then $\boldsymbol{x_i} \in \mathbb{R}^k$, where k is the number of variables. In general, the feature vectors take values in $\mathcal{X}$, a subset of $\mathbb{R}^k$. Instead, the set of possible values for y_i is denoted by $\mathcal{Y}$. In case of the classification problem, $\mathcal{Y}$ is a discrete set containing the different class labels. For instance, if dealing with a binary classification problem, the two classes can be denoted by the labels 0 and 1, and therefore, $\mathcal{Y} = \{0, 1\}$.

With this formulation, the learning procedure of a supervised model consists in finding out the rule f that associates a vector in $\mathcal{X}$ to its target value in $\mathcal{Y}$. Clearly, f is unknown, and in general, there are no hypothesis on its nature.

To sum up, this is an high-level description of the procedure followed by any supervised learning algorithm: the goal of the learning phase of such a model consists in finding the best function h approximating the unknown target function f

$$h \approx f : \mathcal{X} \to \mathcal{Y},$$

exploiting a labeled training set. The candidates h belong to a hypothesis space $\mathcal{H}$, which depends on the chosen algorithm itself. Indeed, the majority of the models make some assumptions about the nature of the target function, besides imposing a preferential ordering on the hypothesis space. In technical terms, this goes under the name of inductive bias (desJardins and Gordon, 1995; Mitchell, 1980).

In the next subsections, the key aspects of some supervised Machine Learning models are presented. However, before dealing with that, some other relevant observations connected with the evaluation of a model should be explained. The aim of the learning procedure is to model the process that have generated the data. It may happen that one can perfectly describe a certain dataset but, at the same time, achieve terrible performances on new data. From a theoretical perspective, defining the correct model complexity is a complex issue. In general, a high complexity model could start fitting also the noise around training data, including possible outliers. To indicate this phenomenon, the term overfitting is used. This is the reason for using a separate set of previously unseen data at testing time, allowing an unbiased evaluation of the model's generalization capacity. On the other hand, a complementary problem exists: an excessively easy model may fail to approximate the underling process, leading to bad results both in training and testing sets. This latter phenomenon is called underfitting. In Figure 4.1, an underfitting, an optimal and an overfitting model are shown for the case of a classification task.

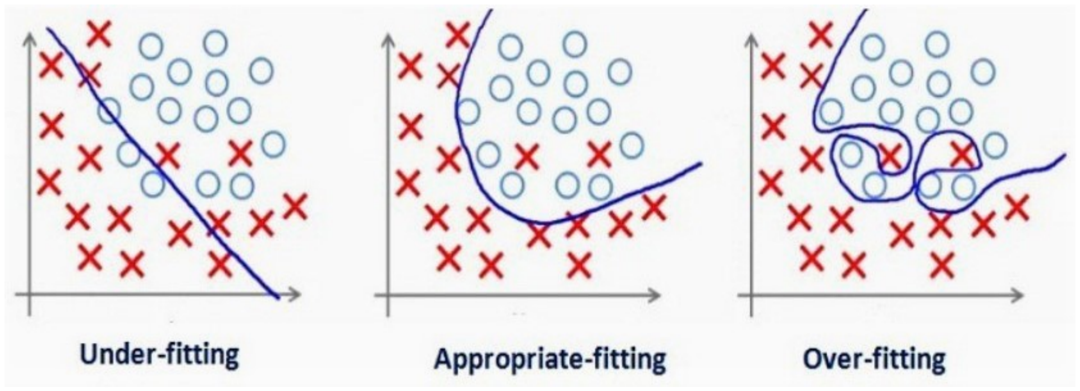


Figure 4.1: Example of an underfitting (left), optimal (middle), and overfitting (right) model for a two dimensional classification problem. In the Figure, samples are represented using red crosses or light blue circles according to their class label, while the blue line represents the decision boundary (source: Kumar, 2015).

These considerations are strictly connected with the bias-variance trade-off. They are two components of the error of a model which limit its generalization capacity. In particular, the bias error is the one implied by wrong assumptions preventing the model from learning the relevant relationships between features and target outputs. Instead, high variance means that there is an excessive sensitivity to small fluctuations, which leads to fitting even the random noise present in the training set. Combining these definitions with the previous one, it is clear that high bias may result in underfitting, while high variance can cause overfitting.

After having chosen a class of models, it is possible to make it more suitable for the specific problem at issue by specifying the values of the tuning parameters. With respect to the standard parameters, which are those that change according to the learning procedure of the algorithm, these do not change during training. Their optimal values cannot be determined a priori, but there are some strategies that can be followed to find them. The most used are:

- Grid Search: it consists in choosing a list of candidate values for each tuning parameter and then trying each possible combination. Since this approach requires training a model an exponential (in the number of parameters) number of times, it may be computational prohibitive. A possible counter measure is first to consider a coarse grid of possible values, then, to identify the hyperspace region that seems to provide better results and finally carry out a finer search in that region;

- Random Search: Grid Search may be too costly, also adopting the multiple step approach just described. Instead of limiting the candidate values for each tuning parameter, the selection is made when choosing their combinations: only a fraction of available combinations is picked randomly, and not all the possible ones. This approach is particularly effective when some tuning parameters are more relevant than others. For instance, let's consider the bi-dimensional case shown in Figure 4.2. This example assumes that only one tuning parameter is relevant in terms of predictive improvement, but clearly, this is not known a priori. As shown, nine trials are completed for both approaches, but with grid search, these trials only test three distinct values for the relevant tuning parameter, while, with random search, all nine trails explore distinct candidates, so, in this case, the latter approach is able to outperform grid search. However, in some situations, it may clearly fail to find the optimal combination due to its stochastic nature (Worcester, 1999).

In addition to these two approaches, it is worth to mention that sometimes guided procedures are adopted, such as stepwise searches or similar.

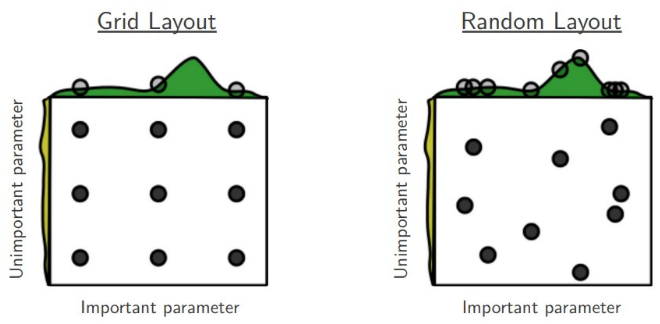


Figure 4.2: Comparison between grid and random search with only one relevant tuning parameter. The profiles shown around the square represent the quality of the model as a function of one tuning parameter (keeping the other fixed) (source: Bergstra and Bengio, 2012).

4.1.1 Logistic Regression

Let's now introduce the easiest way to automatically classify a data sample: recall that each recording is characterized by its feature vector $\boldsymbol{x_i} \in \mathbb{R}^k$ and the corresponding target label y_i. A simple model can compute the scalar product

$$h_\theta(\boldsymbol{x_i}) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x_i},$$

where $\boldsymbol{\theta}$ is the parameter vector, or weight vector, having the same dimension as $\boldsymbol{x_i}$. Its value has to be estimated through learning. Up to now, this is the procedure followed by the linear regression algorithm. In case of a binary classification task, it is necessary to introduce a decision rule: if $h_\theta(\boldsymbol{x})$ is larger than a certain threshold θ_0, than the model predicts class 1 as output, otherwise class 0.

Making a step forward, after calculating the scalar product between the feature vector and the weights, a link function can be computed. One of the most famous is the Logistic function:

$$g(z) = \frac{1}{1 + e^{-z}}.$$

The model obtained in this way is called Logistic Regression. An important property is that its output

$$h_\theta(\boldsymbol{x_i}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x_i}}}$$

can be interpreted as the probability of the i^{th} sample to belong to class 1, because the logistic function assumes values in the interval $[0, 1]$.

During the training phase, the values of the parameter vector $\boldsymbol{\theta}$ are estimated. To do so, a loss function have to be defined. Essentially, it is an expression that measures the predictions quality by comparing them with the target values. An example of loss functions used for classification problems is the Cross Entropy loss, defined as:

$$J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} \mathrm{y}_i \cdot \log\left(h_\theta\left(\boldsymbol{x}_i\right)\right) + \left(1 - \mathrm{y}_i\right) \cdot \log\left(1 - h_\theta\left(\boldsymbol{x}_i\right)\right).$$

This expression is nothing but the negative log-likelihood. In order to understand how it penalizes wrong predictions, assume, for instance, that $y_i = 1$. In this case, the only term which survives is $\log\left(h_\theta\left(\boldsymbol{x}_i\right)\right)$. Since $h_\theta(\boldsymbol{x_i}) = \mathrm{P}(\mathrm{y_i} = 1 \mid \boldsymbol{x_i}; \boldsymbol{\theta})$, if the model assigns high probability of $\boldsymbol{x_i}$ belonging to class 1, then $h_\theta\left(\boldsymbol{x}_i\right)$ is close to 1, so $\log\left(h_\theta\left(\boldsymbol{x}_i\right)\right)$ is close to 0, therefore, the contribution of the i^{th} sample is close to 0. On the other hand, if the model prediction is wrong, then $h_\theta\left(\boldsymbol{x}_i\right)$ is close to 0, so $\log\left(h_\theta\left(\boldsymbol{x}_i\right)\right)$ tends to $-\infty$, therefore, the contribution of the i^{th} sample tends to ∞, increasing the loss function.

Then, it is all about solving an optimization problem, minimizing the loss function with respect to the parameter $\boldsymbol{\theta}$. One critical observation is that using the Cross Entropy loss function, a convex optimization problem is obtained. Convexity is a desired property in optimization theory, because, under certain hypothesis, it guarantees polynomial-time solutions for many classes of problems (Nesterov and Nemirovskii, 1994), while without this hypothesis, convergence may not be guaranteed or be NP-hard.

Now that all the ingredients have been introduced, the learning procedure should be defined. The problem to solve is:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^k} \mathrm{J}(\boldsymbol{\theta}),$$

so, it is an unconstrained minimization problem of a convex function. In this context, Fisher's scoring algorithm is a common choice to quickly minimize $\mathrm{J}(\boldsymbol{\theta})$. It is a modification of the Newton-Raphson method which exploits a local quadratic approximation of the log likelihood (Longford, 1987). Other alternatives are Gradient Descent or one of its variants.

The loss function may be modified by including a regularization term. Its purpose is to improve the generalization performance of the model, avoiding overfitting. Essentially, this regularization term penalizes large values of the parameter. Its mathematical expression is:

$$\frac{\lambda}{2}\|\boldsymbol{\theta}\|^2 = \frac{\lambda}{2}\sum_{\mathrm{j}=1}^{\mathrm{k}} \theta^{\mathrm{j}2},$$

where λ is the regularization coefficient and θ^{j}, with $j = 1,\ldots,P$, is the j-th component of the parameter vector. While the value of $\boldsymbol{\theta}$ is learned during training, λ is fixed throughout learning.

For completeness, here it is reported the gradient descent update rule for the regularized version (Bishop, 2016):

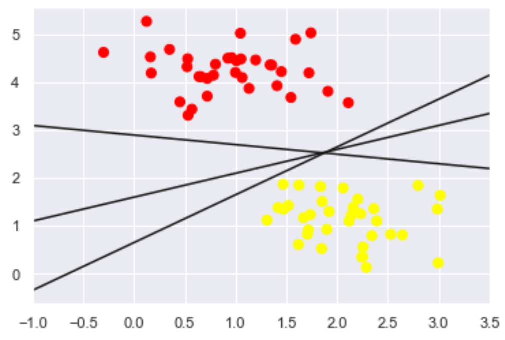
$$\theta^{\mathrm{j}} = \theta^{\mathrm{j}} - \eta\left[\frac{1}{n}\sum_{\mathrm{i}=1}^{n}\left(h_\theta\left(\boldsymbol{x}_{\mathrm{i}}\right) - \mathrm{y}_{\mathrm{i}}\right)\mathrm{x}_{\mathrm{i}}^{\mathrm{j}} + \frac{\lambda}{n}\theta^{\mathrm{j}}\right].$$

4.1.2 Support Vector Machine

At the beginning of the previous section, it has been shown that a very easy classifier can be obtained computing the scalar product between an input vector $\boldsymbol{x_i}$ and the parameter vector $\boldsymbol{\theta}$, and comparing the results with a fixed threshold θ_0. In other words, during training, the aim is finding the values of $\boldsymbol{\theta}$ and θ_0 solving the following system:

$$\begin{aligned} &\boldsymbol{\theta}^\top \boldsymbol{x_i} - \theta_0 \geqslant 0 \quad \text{if} \quad y_i = 1, \\ &\boldsymbol{\theta}^\top \boldsymbol{x_i} - \theta_0 < 0 \quad \text{if} \quad y_i = 0. \qquad (i = 1,\ldots,n) \end{aligned} \tag{4.1}$$

From a geometrical perspective, it is equivalent to finding the hyperplane separating $A = \{\boldsymbol{x_i} : (\boldsymbol{x_i}, y_i) \in T, y_i = 1\}$ and $B = \{\boldsymbol{x_i} : (\boldsymbol{x_i}, y_i) \in T, y_i = 0\}$, where T is the training set. For the moment, let's assume that A and B are linearly separable. In this case, it is clear that an infinite number of pairs ($\boldsymbol{\theta}$,θ_0) exists (equivalently, there exists an infinite number of separating hyperplanes). In Figure 4.3, a simple dataset is represented along with 3 possible hyperplanes satisfying (4.1).


(a) *Example of three separating hyperplanes.*

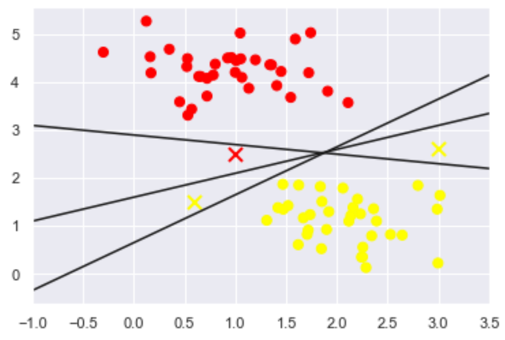

(b) *New data samples represented by crosses.*

Figure 4.3: Possible solution to the system (4.1) (adapted from: VanderPlasp, 2016).

At first sight, the presence of multiple solutions may not seem a problem since all three hyperplanes perfectly separate all data samples. However, when a classification model is considered, the final aim is not to perfectly classify training sample, but being able to make correct predictions on new unseen data. The fact is that once a solution to (4.1) has been found, it gives no guarantees on generalization capacity. On the right-hand side of Figure 4.3, it is possible to see that two of the three drawn hyperplanes cannot classify correctly new samples. In a similar situation, a wise choice of the loss function can be very helpful: for example, the presence of outliers or high leverage points may lead to different results when using a quadratic loss function instead of a linear one.

A possible solution is represented by Support Vector Machines, which is a very important class of machine learning tools. The aim becomes determining at training time which is the best hyperplane. First of all, an objective measure is required to compare the candidates. This role is played by the separation margin, defined as:

Definition: the separation margin ρ of the hyperplane $H = \left\{\boldsymbol{x} \in \mathbb{R}^k : \boldsymbol{\theta}^T \boldsymbol{x} + \theta_0 = 0\right\}$ is the minimum distance between samples in the training set $T = A \cup B$ and the hyperplane H. Its value is given by:

$$\rho(\boldsymbol{\theta}, \theta_0) = \min_{\boldsymbol{x_i} \in A \cup B} \left\{ \frac{\left|\boldsymbol{\theta}^T \boldsymbol{x_i} + \theta_0\right|}{\|\boldsymbol{\theta}\|} \right\}.$$

In Figure 4.4, a graphical representation of the separation margin for the 3 previously seen hyperplanes is shown.

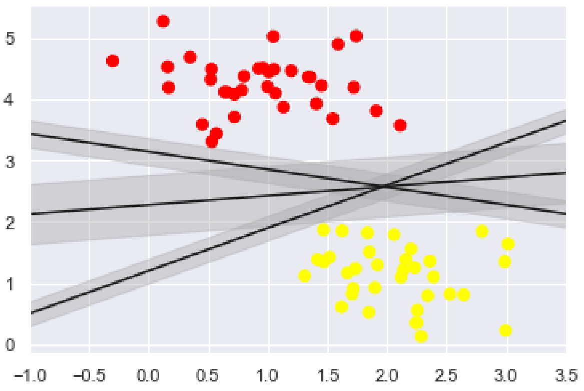


Figure 4.4: Adding the separation margin to the hyperplanes (adapted from: VanderPlasp, 2016).

Note that the separation margin of a hyperplane is determined by a limited amount of data samples, to be precise, the closest to the hyperplane itself. These points are the so-called support vectors, which give the name to the algorithm class.

Intuitively, the hyperplane with the higher separation margin can better classify new samples. Indeed, some statistical learning results (Vapnik, 1998) have shown that such a hyperplane guarantees the best generalization capacity. Furthermore, it is possible to prove its existence and uniqueness (Vapnik, 1995).

According to what have just been said, determining the optimal hyperplane is equivalent to solving the following problem:

$$\begin{aligned} \max_{\boldsymbol{\theta},\theta_0} \quad & \rho(\boldsymbol{\theta}, \theta_0) \\ s.t. \quad & \boldsymbol{\theta}^T \boldsymbol{x_i} + \theta_0 \geq 1, \quad \boldsymbol{x_i} \in A, \\ & \boldsymbol{\theta}^T \boldsymbol{x_i} + \theta_0 \leq -1, \quad \boldsymbol{x_i} \in B. \end{aligned} \tag{4.2}$$

Note that the constrains are much more restrictive than those shown in problem (4.1): the separating hyperplane is forced to be far enough from all training samples. For this reason, SVM is said to be a large margin classifier.

It can be proved that problem (4.2) is equivalent to a convex quadratic minimization problem, having as objective function $\frac{1}{2}||\boldsymbol{\theta}||^2$. This can be solved exploiting the method of Lagrange multipliers and considering the corresponding Wolf dual problem (Mangasarian, 1994; Grippo and Sciandrone, 2003).

Dataset not linearly separable: kernel methods

The analysis of SVM seen until now, was carried out under the restricted hypothesis of a linearly separable dataset. This is an ideal situation that is quite rare in reality. Therefore, it is necessary to relax this hypothesis. The first step towards a more common scenario is the case in which the dataset is still separable, but not linearly. In this case, the solution is represented by the kernel methods. The idea is to project data samples into a higher dimensional space, where the dataset allows a linear separation. By calling ϕ this projection function, it is possible to define:

Definition: the kernel function K is defined by the scalar product:

$$K\left(\boldsymbol{x_i}, \boldsymbol{x_j}\right) = \phi\left(\boldsymbol{x_i}\right)^t \phi\left(\boldsymbol{x_j}\right),$$

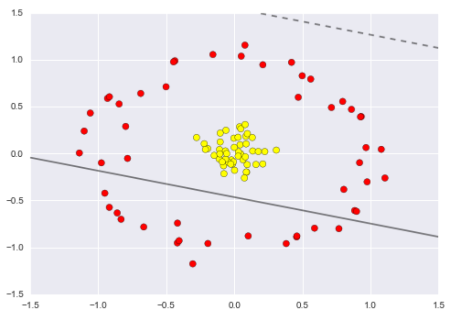
where $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are two generic samples.

Depending on the specific case at handle, a different kernel function can be adopted (therefore, it will be a key tuning parameter for the model). The most common options are:

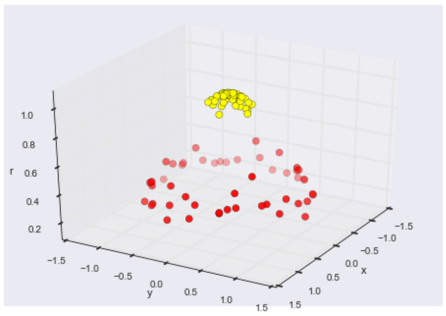
- Linear Kernel:
  $K\left(x_i, x_j\right) = x_i^t x_j$;
- Polynomial Kernel:
  $K\left(x_i, x_j\right) = \left(\gamma x_i^t x_j + r\right)^d, \gamma > 0,\ r \geq 0,\ d \in \mathbb{N},\ d \geq 1$;
- Gaussian Kernel, also known as Radial Basis Function, RBF:
  $K\left(x_i, x_j\right) = \exp\left(-\gamma \left\|x_i - x_j\right\|^2\right), \gamma > 0$;
- Laplacian Kernel:
  $K\left(x_i, x_j\right) = \exp\left(-\gamma \left\|x_i - x_j\right\|\right), \gamma > 0$;
- Sigmoid Kernel:
  $\tanh\left(\gamma \cdot \boldsymbol{x_i}^T \boldsymbol{x_j} + r\right), \gamma > 0.$

In Figure 4.5, it is possible to compare the effectiveness of the Gaussian kernel against the linear one for a non-linearly separable dataset.

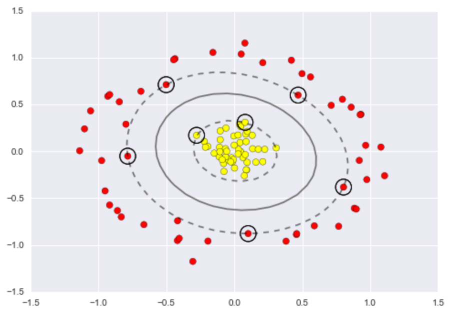
The positive constant γ, which appears in all kernel function with the only exception of the linear one, is another important tuning parameter. In abstract terms, it defines the intensity of the kernel function: for small values of γ, the model behaves like linear SVM, while for high values of γ, the model becomes more influenced by the support vectors, as can be seen in Figure 4.6.



(a) *Linear Kernel*

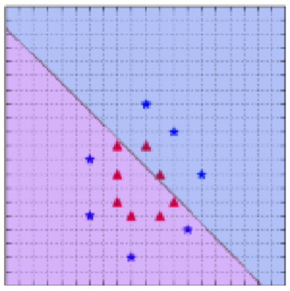


(b) *Dataset projection using Gaussian Kernel*

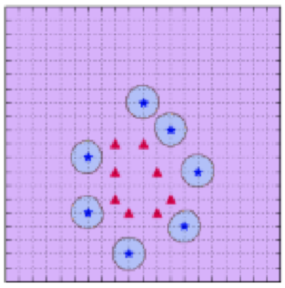


(c) *Decision boundary with Gaussian Kernel and relative support vectors.*

Figure 4.5: Implementation of Linear Kernel and Gaussian Kernel with a non-linearly separable dataset (adapted from: VanderPlasp, 2016).



(a) $\gamma = 10^{-5}$



(b) $\gamma = 2$

Figure 4.6: The decision boundary implied by values of γ too small or too large.

Dataset not separable: Soft-margin SVM

In what follows, it will be discussed the more generic (and common) situation, where the dataset is not separable, even after the projection in a higher dimensional space given by kernel methods. To handle this scenario, it is possible to allow some data points to violate the separation margin of a small amount, tolerating classification errors on training samples. In this formulation, called soft-margin SVM, the positive variables ξ_i, with $i = 1, \dots, n$ are introduced and the separability request is relaxed as follows:

$$\begin{aligned} &\boldsymbol{\theta}^\top \boldsymbol{x_i} + \theta_0 \geq 1 - \xi_i, \quad \boldsymbol{x_i} \in A, \\ &\boldsymbol{\theta}^\top \boldsymbol{x_i} + \theta_0 \leq -1 + \xi_i, \quad \boldsymbol{x_i} \in B. \quad (i = 1, \dots, n) \end{aligned} \tag{4.3}$$

Note that for the samples on the boundary (the support vectors), the following holds:

$$y_i \left(\boldsymbol{\theta}^\top \boldsymbol{x_i} + \theta_0 \right) = 1,$$

therefore, the quantity ξ_i measures how much the data sample $(\boldsymbol{x_i}, y_i)$ can violate the margin. As a consequence, the quantity $\sum_{i=1}^n \xi_i$ has to be minimized. By specifying a penalization coefficient $C > 0$ (which will weight training errors), the quantity to be minimized can be modified as follows:

$$\begin{array}{ll} \min_{\boldsymbol{\theta}, \theta_0, \xi} & \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t.} & \boldsymbol{\theta}^\top \boldsymbol{x_i} + \theta_0 \geq 1 - \xi_i, \quad \boldsymbol{x_i} \in A, \\ & \boldsymbol{\theta}^\top \boldsymbol{x_i} + \theta_0 \leq -1 + \xi_i, \quad \boldsymbol{x_i} \in B. \quad (i = 1, \dots, n) \end{array} \tag{4.4}$$

Recalling that the first addend of the objective function concerns the thickness of the margin, while the second addend regards possible violations, it is clear the strategy behind the choice of the tuning parameter C: for large values, the training accuracy improves and a tighter margin is used, with the risk of overfitting for excessively large values. Instead, for small C, the dimension of the margin is maximized, but more margin violations are allowed. This trade-off is even more explicit from Figure 4.7.

4.1.3 K-Nearest Neighbors

All models seen so far require a training phase in which the parameters are computed through an optimization procedure, until certain conditions (like a loss function value smaller than a fixed threshold) are satisfied. An important limitation of this parametric approach is that the chosen model, for its intrinsic hypothesis (inductive bias), might be a poor one for the distribution that generated the data. In opposition to this approach, a different class of models that do not make any assumption about the shape of the decision boundary has been introduced: the non-parametric models.

K-Nearest Neighbors (KNN) is one of them. Indeed, during the training phase, this method

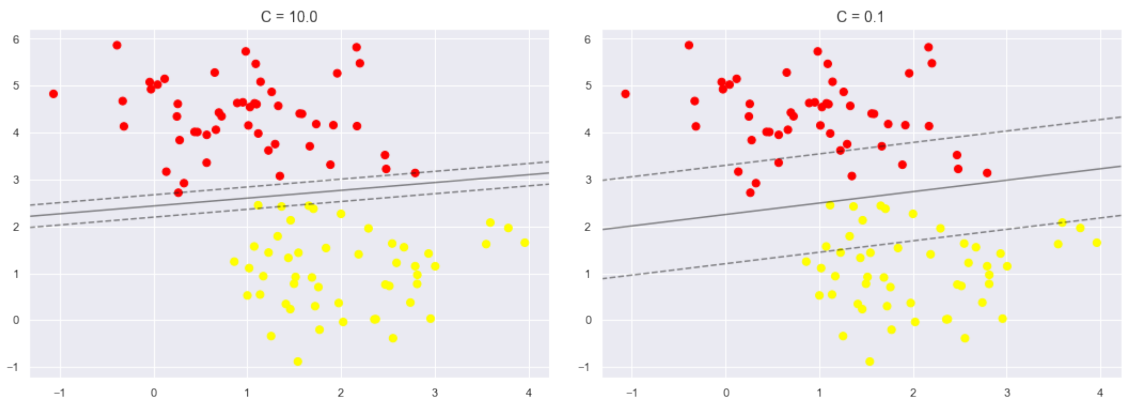


Figure 4.7: Effect of tuning parameter C on the final model result: with smaller C, the model considers an higher separation margin, and the final decision boundary has a slope slightly larger than the one obtained with larger C (adapted from: VanderPlasp, 2016).

does not estimate any parameter. Instead, the learning procedure consists in simply storing training points. All the work is done at prediction time: to classify a new data point, the k closest training samples are identified, and their class distribution is analyzed. The idea is that samples close in the feature space are similar, and therefore, they should belong to the same class. So, the new sample is labeled with the most frequent class present in its neighbors.

As said, this model does not have any parameter to estimate. However, it has different tuning parameters that need to be chosen. One of them is the distance metric, which influences the decision boundary. However, the most relevant tuning parameter of this method is k, the number of training points to consider for each prediction. It controls the degree of smoothing of the decision boundary, so that small k produces many small regions, whereas large k leads to fewer larger regions. For this reason, the optimal value of k is related to the trade-off variance-bias: small values of k imply high variance and low bias, while large values lead to low variance (smoother prediction) and high bias (the local structure of the true class distribution may not be captured). In Figure 4.8, the decision boundaries obtained with different values of k are shown for a two-dimensional dataset.

Intuitively, more training points are available, the better the feature space is described, and so the generalization capacity is expected to improve. This intuition is confirmed by an interesting property:

Property In the limit $n \rightarrow \infty$, where n is the training set size, the nearest-neighbor (k=1) classifier has an error never more than twice the minimum achievable error rate of an optimal classifier, i.e., one that uses the true class distributions (Cover and Hart, 1967).

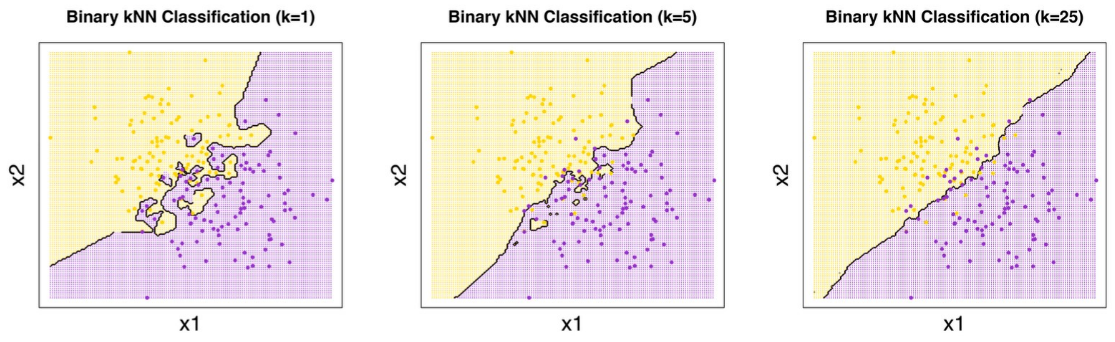


Figure 4.8: Decision boundaries obtained with k = 1, k = 5 and k = 25 (source: De Wilde, 2012).

However, large training sets require a huge amount of memory simply to store them. In addition to this, recall that each prediction requires the computations of n distances to find the k closest points, and this could become computationally prohibitive. In other words, the prediction time is the real bottleneck of this algorithm. To limit this issue, some tree-based search structures can be introduced, in order to find (approximate) near neighbors without doing an exhaustive search of the dataset (Bishop, 2016).

4.1.4 Classification Tree and Random Forest

A classification tree is another non-parametric, supervised learning algorithm, which classifies data in an very intuitive and self-explainable way (Gareth et al., 2015). This tool is commonly used in operations research, specifically in decision analysis, but it is also very popular in the machine learning field. The idea behind a decision tree consists in recursively partitioning a dataset according to the values assumed by the variables. The order in which the variables are picked and the thresholds used to make the partition define the model structure. A data split works differently for continuous and categorical variables. For the first case, some value ranges are identified using certain thresholds, while for the second case, all possible values have to be partitioned into two sets, and the sample in analysis follows one branch or the other depending on its value for the considered variable. Once certain conditions are satisfied, splits no longer occur. At prediction time, each new data sample undergoes the sequence of splits until reaching an end. Then, the labels of all training samples following the same route are taken into account and the model output for the new sample is computed following a majority voting scheme.

A model built following this procedure allows an explicit graphical representation formed by a set of nodes connected to each other through some edges. This creates the characteristic tree structure which gives the name to the algorithm class.

Each node can have at most one incoming edge and possibly multiple outgoing edges. There are three kinds of nodes:

- root node: the one that causes the first split, that each data sample has to undergo. It is the only one having only outgoing edges;
- leaf nodes: those that do not have outgoing edge, since they terminate the branch route determining the sample classification. For this reason, they are also called terminal or decision nodes;
- internal nodes: those having exactly one incoming edge and at least two outgoing edges.

With the only exception of the leaf nodes, all the others subdivide the dataset into two or more subsets and the criterion used is shown inside the node. An example of a decision tree is shown in Figure 4.9.

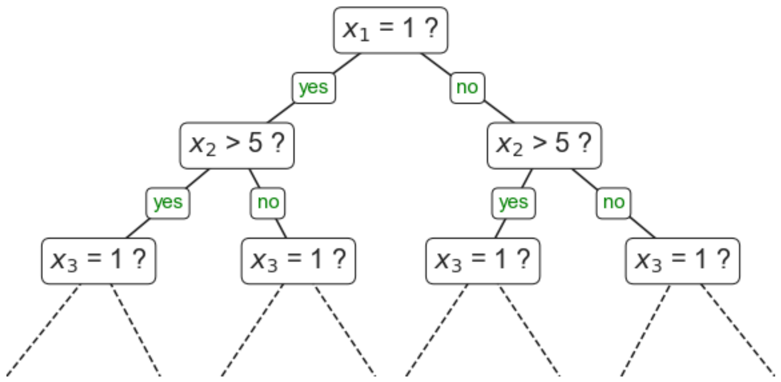


Figure 4.9: Example of a decision tree.

In order to implement the algorithm, the criterion used to select the variable at each split step should be chosen. Before the first split, the dataset consists in all training samples, so the initial class distribution is the training distribution. Therefore, the initial set is heterogeneous. Instead, in a leaf node, the aim is to have all or most of the samples reaching it with the same label, obtaining a set as much homogeneous as possible. An index commonly used to measure the degree of homogeneity (or randomness) of a certain set S is the entropy. It is defined as:

$$Entropy(S) = \sum_{c \in C} -p(c) \cdot \log_2 (p(c)),$$

where C is the set of classes in S, and $p(c)$ is the fraction of elements in S belonging to class c. Since $p(c)$ assumes values between 0 and 1, $\log_2(p(c))$ will be negative. Then, this is multiplied by $-p(c)$, so it follows that all addends of the sum are positives or equal to zero, in case of a perfectly homogeneous set (if all samples in S belong to class $\tilde{c}$, then $p(c) = 0$ for $c \neq \tilde{c}$, while $\log_2(p(c)) = 0$ for $c = \tilde{c}$, so, in both cases, the product is equal to 0).

The last ingredient to introduce is the Information Gain (IG), another quantity coming from information theory that measures the variation of entropy implied by the subdivision of the set S according to attribute a. It is defined as:

$$IG(S, a) = Entropy(S) - Entropy(S \mid a) = Entropy(S) - \sum_{T \in \mathcal{T}} p(T) Entropy(T),$$

where $\mathcal{T}$ is the list of subsets obtained after the data division, and $Entropy(S \mid a)$ is their entropy computed as a weighted sum. Indeed, $p(T)$ indicates the fraction of data samples of S that after the split belongs to the subset T. The rationale is the following: at each step, the information gain is computed for each attribute not yet used in the tree construction, and the one leading to the largest information gain (equivalently, the largest entropy decrease) is selected to split the set at the current iteration.

An alternative to the entropy index is the Gini index, which is defined as:

$$Gini(S) = 1 - \sum_{c \in C} p(c)^2.$$

As before, if S mainly has elements belonging to the same class $\tilde{c}$, then $p(c) = 0$ if $c \neq \tilde{c}$, while $p(c) = 1$ if $c = \tilde{c}$; as a consequence, Gini index will be equal to 0. Instead, if S has a more heterogeneous nature, Gini index will be larger.

In general, the advantages offered by a decision tree are:

- straightforward interpretation and visualization of the model;
- it is a non-parametric model, so it can adapt also to dataset not linearly separable;
- ability to handle both categorical and numerical variables, requiring little data preparation (Gareth et al., 2015);
- often good results on large datasets, in general, requiring a reasonably low amount of time (Liberman, 2017).

On the other hand, even decision trees do have some limitations. Since at each step, the attribute that guarantees the best information gain is chosen, it is a greedy algorithm. In practice, this kind of approaches may fail to yield the global optimum (Ben-Gal et al., 2014). In addition to this, it is difficult to identify the correct depth of the tree, i.e., the number of splits. Indeed, too large trees may overfit the training set, while smaller ones can fail to handle subtle distinctions in data. Following the Occam's Razor principle, the smallest tree able to fit the observations should be kept. To do so in practice, a number of tuning parameters related to the variance-bias trade-off have to be optimized, such as:

- the maximum depth of the tree;
- the minimum number of elements in a node to allow further splits;
- the maximum number of leaf nodes;
- the minimum information gain to allow the split of a node.

Furthermore, some random splits may be considered during training, or different information gain criteria may be tried.

Bagging and Boosting

To overcome the limitations of unstable models, such as decision trees, one possibility is to combine the predictions obtained by various methods. Bagging and boosting are two possible approaches. The former consists in the random extraction of B subsets from the training set and fitting to each of them a model. Then a new classifier can be defined by considering the B outputs computed by the fitted models and taking the most frequent value, or instead computing the average value and comparing it with a threshold (possibly different from 0.5). This classification procedure is called bootstrap aggregating, from which the abbreviated term bagging is derived. Since the output of many classifiers can be interpreted as the probability of belonging to a certain class, in a more general formulation, bagging works by averaging these probabilities (Azzalini and Scarpa, 2012).

Boosting takes a further step forward: instead of selecting randomly the B subsets to which the preliminary models are fitted, the most misclassified observations in early stages are given a higher probability of being selected in these subsets. To make this possible, the preliminary models cannot be trained in parallel anymore, but sequentially, and at the end of each iteration, the probability of each sample to be selected is updated accordingly to its classification error. In this way, more emphasis is given to those samples that create more problems. Both bagging and boosting procedures have been studied theoretically, proving statistical properties that justify their excellent empirical performance (Friedman et al., 2012). Without going into the details, the main factor is the reduction of the variance of the model, which mitigates the overfitting risk without requiring pruning (Hartshorn, 2016).

Random Forest

Instead of training a model in different subsets of the training dataset, a complementary approach consists in selecting randomly different subsets of the explanatory variables. In the case of decision trees, the model obtained with this procedure is called random forest. This term has later been adopted to indicate any classifier obtained as a combination of a set of decision trees, for instance using bagging or boosting.

A random forest model inherits all the tuning parameters of decision trees, along with new ones, such as:

- the number of trees in the forest;
- the number of variables to be selected for the training of each tree;
- if bagging or boosting are used, the size of each subset of samples.

Note that with respect to other methods of model combination, random forests have some interesting advantages: their predictive capacity is comparable, for instance, to that of boosting and in some cases it is even better. However, random forests are much faster since the dimensionality of the problem is significantly reduced, and so its complexity is lower too. This results from the fact that every single tree is based on fewer variables and the computational burden is therefore lower. It is also relatively simple to build an algorithm that, taking advantage of parallel computing, can further accelerate the random forest procedure (Azzalini and Scarpa, 2012).

4.1.5 Gradient Boosting

Gradient boosting is a powerful algorithm based on trees and boosting. Its origin may be found in the original idea of Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function (Breiman, 1997).

For simplicity, let's introduce this algorithm in the regression scenario. The training set is composed of n pairs of the kind $(\boldsymbol{x_i}, y_i)$ and the goal is to find a good estimate $\hat{y_i} = f_1(\boldsymbol{x_i})$ of y_i by minimizing the mean squared error. Here with f_1, the first model estimation is indicated. Suppose that the model is imperfect, so there are some error terms, the residuals, different from zero, namely:

$$r_i = y_i - f_1(\boldsymbol{x_i}).$$

In order to improve the prediction, the boosting approach described previously retrains the model on different subsets of the training set by giving more emphasis to the samples more difficult to classify. With gradient boosting, something similar is done, but with a different approach: a new estimator, usually a regression tree h_1 is fitted to the residuals:

$$h_1(\boldsymbol{x_i}) \approx y_i - f_1(\boldsymbol{x_i}).$$

Then, h is added to the original model, so that the new prediction becomes:

$$\hat{y_i} = f_2(\boldsymbol{x_i}) = f_1(\boldsymbol{x_i}) + h_1(\boldsymbol{x_i}).$$

The role of h_1 is to compensate the problems of f_1, so the new model is expected to be better than the original one. At this point, the reasoning can be repeated with respect to this model f_2 fitting a new tree h_2 on the new residuals, to further improve the prediction, which becomes:

$$f_3(\boldsymbol{x_i}) = f_2(\boldsymbol{x_i}) + h_2(\boldsymbol{x_i}).$$

This procedure can be repeated M times, where at each stage

$$f_{m+1}(\boldsymbol{x_i}) = f_m(\boldsymbol{x_i}) + h_m(\boldsymbol{x_i}).$$

To come full circle, let's see how this model can be generalized for classification problem, and how it is related to gradient descent. Since the quantity to minimize is the quadratic loss function $L(y, f(\boldsymbol{x})) = \frac{1}{n}\left(y - f(\boldsymbol{x})\right)^2$, let's compute its derivative with respect to a generic prediction f:

$$\frac{\partial L}{\partial f} = -\frac{2}{n}(y - f(\boldsymbol{x})).$$

And here, a key observation follows: residuals are proportional equivalent to the negative gradients:

$$-g\left(x_i\right)=-\left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right] \propto y_i-f\left(x_i\right).$$

Therefore, gradient boosting can be generalized to other problems, adopting lost functions different from the quadratic one, by fitting at each step h_m to the negative gradients.

As anticipated, gradient boosting is typically used with decision trees as additional estimators. For this special case, a modification was proposed by Friedman, improving the quality of fit (Friedman, 1999). The resulting algorithm is the following:

1. Initialize the model with a constant value:

$$f_0(\boldsymbol{x})=\arg\min_{\gamma} \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \gamma\right).$$

2. For $m=1, \ldots, M$:

   - Compute the pseudo-residual [*]:

$$\tilde{r}_{im}=-g\left(\boldsymbol{x_i}\right)=-\left[\frac{\partial L\left(y_i, f\left(\boldsymbol{x_i}\right)\right)}{\partial f\left(\boldsymbol{x_i}\right)}\right]_{f(\boldsymbol{x})=f_{m-1}(\boldsymbol{x})}, \text{for } i=1, \ldots, n.$$

   - Fit a decision tree $h_m(\boldsymbol{x})$ to pseudo-residuals, giving terminal regions (leaf nodes) $R_{jm}, j=1,2,\ldots,J_m$. Also, introduce the indicator function of these regions, defined as:

$$\mathbf{1}_{R_{jm}}(x)=\begin{cases}1 & \text{if } x \in R_{jm} \\ 0 & \text{if } x \notin R_{jm}.\end{cases}$$

   - Compute the multipliers γ_{jm} with $j=1,2,\ldots,J_m$, by minimizing the loss function inside each region R_{jm}:

$$\gamma_{jm}=\arg\min \sum_{\boldsymbol{x_i} \in R_{jm}} L\left(y_i, f_{m-1}\left(\boldsymbol{x_i}\right)+\gamma\right).$$

   These are the values that will be used in each of the regions identified by the tree. The residuals belonging to the same region are expected to be similar, therefore,

[*]the term pseudo-residual is used because, as shown before, the anti-gradient is equal up to a multiplicative constant to the residuals; besides, this relationship holds only for the quadratic loss.

the same value is used for them, but instead of keeping the value provided as output by the tree h_m, the previous minimization is performed. This is an additional modification proposed by Friedman.

- Update the model according to the rule:

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(\boldsymbol{x}).$$

3. Return the final output f_M.

As in other models, also in Gradient Boosting a regularization technique can be introduced to reduce the risk of overfitting. This is achieved by shrinkage, which modifies the update rule as follows:

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(\boldsymbol{x}),$$

where ν is the learning rate, with values between 0 and 1.

4.2 Deep Learning tools

In this section, the main aspects of a deep learning model are described. Let's start with the architecture of a single artificial neuron, which is the computational unit of a neural network (Bishop, 2016). Basically, it takes in input a feature vector $\boldsymbol{x_i}$ and multiplies it with a parameter (or weight) vector $\boldsymbol{\theta}$. Usually, often a bias term θ_0 is included in the computation. For ease of notation, it is possible to redefine the input vector adding a dummy component: $\boldsymbol{x} = [x_0, \boldsymbol{x}] = [1, \boldsymbol{x}]$. The same is applied to the weight vector, which becomes $\boldsymbol{\theta} = [\theta_0, \boldsymbol{\theta}]$. To the result of this scalar product, called activation, a differentiable, nonlinear activation function f is applied, therefore the output will be:

$$h_\theta(\boldsymbol{x}) = f\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}\right).$$

A possible choice for f is the Heaviside step function:

$$f(t) = \begin{cases} 1 & \text{se } t \geqslant 0 \\ 0 & \text{se } t < 0. \end{cases}$$

In this case, the artificial neuron computation resembles that of a simple linear classifier. Instead, if the logistic function is adopted, it is like performing a logistic regression. Things become more interesting when more neurons are strong together, forming a neural network. Over the years, many architectures have been introduced for different purposes. In what follows, two of them are described more in details: a standard feed forward fully connected neural network and a recurrent neural network.

4.2.1 Feed forward Neural Networks

In a neural network, neurons are organized in layers: each unit in the input layer takes the feature vector and performs the computation explained before producing a scalar output, called activation. Then, the neurons in the subsequent layer do the same, but taking as input the results of the previous layer. The final result of the network is produced by the last layer, named output layer, while those in the middle are called hidden layers. At this point, it is clear that the number of layers and the number of neurons in each layer are tuning parameters which define the complexity of the network, and, along with the activation functions, determine its structure and computations. For example, for binary classification problems, a single logistic sigmoid output can be used.

Let's consider a simple example of a network taking input vectors with only 3 variables, one hidden layer with 3 neurons and an output layer with one unit. If the activation of unit i in layer l is indicated with $a_i^{(l)}$, and with $\theta_{ab}^{(l)}$ the entry in position $[a, b]$ of the weight matrix controlling function mapping from layer l and layer $l + 1$, the output of the hidden layer are:

$$z_j^{(2)} = f(a_j^{(2)}) = f\left(\theta_{j0}^{(1)} x_0 + \theta_{j1}^{(1)} x_1 + \theta_{j2}^{(1)} x_2 + \theta_{j3}^{(1)} x_3\right), \text{with } j = 1, 2, 3. \qquad (4.5)$$

For generalization purposes, it is convenient to introduce the notation $z_i^{(1)} = x_i$. Therefore, 4.5 becomes:

$$z_j^{(2)} = f(a_j^{(2)}) = f\left(\theta_{j0}^{(1)} z_0^{(1)} + \theta_{j1}^{(1)} z_1^{(1)} + \theta_{j2}^{(1)} z_2^{(1)} + \theta_{j3}^{(1)} z_3^{(1)}\right), \text{with } j = 1, 2, 3. \qquad (4.6)$$

Regarding the final output, let's consider first a simple linear model in which the outputs are linear combinations of the hidden activations:

$$h_\theta(\boldsymbol{x}) = z_1^{(3)} = a_1^{(3)} = \theta_{10}^{(2)} z_0^{(2)} + \theta_{11}^{(2)} z_1^{(2)} + \theta_{12}^{(2)} z_2^{(2)} + \theta_{13}^{(2)} z_3^{(2)}, \qquad (4.7)$$

where $z_0^{(2)} = 1$ and $\theta_{10}^{(2)}$ is the bias term of the second layer.

This output computation is also referred to as forward propagation. This term is in opposition with the one used for defining the training algorithm: backpropagation. Clearly, all weights of a network should be updated. However, only the final output is known (for a supervised model), so the error can be computed explicitly only for the last layer. Therefore, this error has to be propagated backwards to the units that contributed for the corresponding output, and this is done by weighing according to the weight matrix value.

More formally, the first step of the backpropagation algorithm consists in computing the error gradient. This is the reason why activation functions have to be differentiable. Exploiting the chain rule, the generic component of the loss function gradient can be written as:

$$\frac{\partial L_n(\boldsymbol{\theta})}{\partial \theta_{ij}^{(l)}} = \frac{\partial L_n(\boldsymbol{\theta})}{\partial a_i^{(l+1)}} \frac{\partial a_i^{(l+1)}}{\partial \theta_{ij}^{(l)}} = \delta_i^{(l+1)} \frac{\partial a_i^{(l+1)}}{\partial \theta_{ij}^{(l)}}, \qquad (4.8)$$

where the quantity $\delta_i^{(l)}$, representing the "error contribution" of unit i in layer l, has been introduced. Considering the expressions 4.5 and 4.6, the second partial derivative can be easily computed:

$$\frac{\partial a_i^{(l+1)}}{\partial \theta_{ij}^{(l)}} = x_j = z_j^{(l)}. \qquad (4.9)$$

Instead, the value of delta is more complex to be determined, requiring the application of the chain rule for the hidden layer units, taking into account inter-layer connections (Bishop, 2016).

Backpropagation stops when the first hidden layer is reached, and delta values are not computed for input nodes, as these do not need to adjust the weights.

4.2.2 Recurrent Neural Networks

For the machine learning models described in the previous section and for feed forward neural networks, the order in which training samples are taken is not considered. However, there are some data generating processes for which each recording is strongly correlated with previous ones, so, the order of the samples represents an additional information, and, often, it is crucially important. For example, this is the case of sequential data, such as audio signals, text, motion recordings and so on. In situations like these, the input is no more a static vector but rather a time series of ordered vectors.

In the artificial neural network scenario, the idea is to store past information in the hidden layers. This is made possible by the presence of feedback connections. In practical terms, when the values of the hidden neurons are computed for sample at time t, also the hidden activations relative to the previous sample are considered. Then, the activations of these new hidden units are copied to a context layer, through feedback connections, for use in subsequent steps. An architecture exploiting this idea is called Recurrent neural network (RNN).

A recurrent neural network can effectively learn short-term temporal dependencies. However, learning long-term dependencies is practically very hard. Intuitively, this happens because at each temporal step, a multiplication with the weight matrix is performed, updating the context layer, so the contribute to the context layer of early samples is exponentially lower than the contribution of the last ones. Indeed, with a standard RNN, the long-term gradients which are backpropagated can tend to zero. The phenomenon behind it is called gradient vanishing (Calin, 2020; Hochreiter et al., 2001).

To overcome this issue, Long-Short Term Memory (LSTM) networks have been proposed. The aim of this network is to be able to identify which input information should be used for the current iteration, how much past knowledge should be used and how much information should be kept in memory and propagated to the future. This is achieved by means of a set of units, each with three *gates*. These gates regulate the flow of information into and out of the cell and each of them have its own set of weights.

In Figure 4.10, a schematic representation of the data flow in a LSTM cell is shown.

Now, a formal mathematical analysis of the gates is made. At each step, a LSTM unit takes as input the current data sample x_t, and the previous cell state C_{t-1} and hidden state h_{t-1}. A key point for understanding how a LSTM cell is implemented, is the usage of sigmoids and hyperbolic tangent functions. The former takes values between 0 and 1, and, thanks to this property, it can be used to index the elements that should be discarded (those that will be multiplied by values close to 0) or retained (those multiplied by larger values), making a selection. Instead, the latter is used as activation to add information to the current cell state. This difference is crucially important for gate modelling.

The first gate encountered when starting a new iteration is the forget gate. Its purpose con-

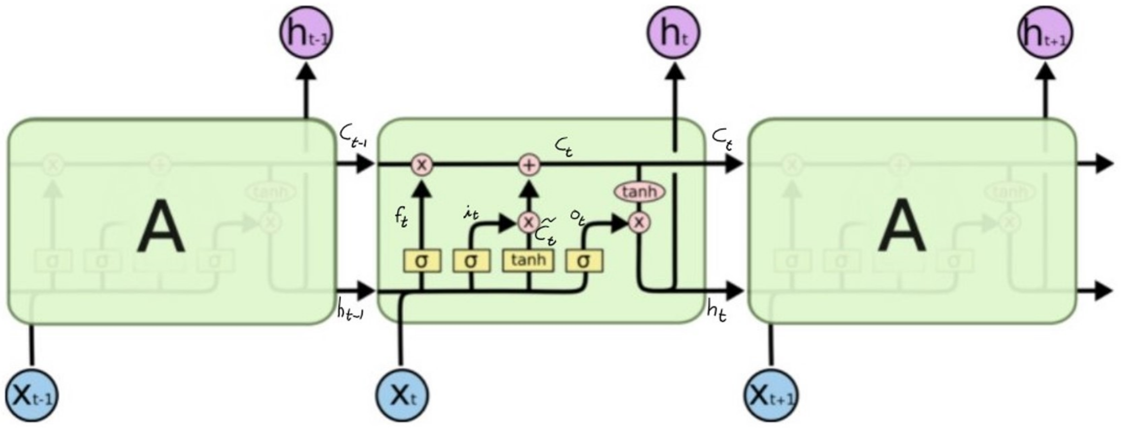


Figure 4.10: The repeating module in a LSTM (adapted from: Christopher, 2015).

sists in selecting which information is going to be thrown away from the previous cell state. In practice, it computes the multiplication between its weight matrix W_f and the concatenation of current input x_t and past hidden state h_{t-1}. A bias term b_f is then added to the results, and finally the sigmoid function is applied:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right).$$

The resulting vector will be used as selector on the cell state component to keep or forget.

Secondly, there is the input gate. This determines what new information is going to be extracted from the current input. The gate output is computed as before, but with different parameters:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right).$$

Again, this vector only identifies the components to be saved. The actual values that are going to be added, denoted by $\tilde{C}_t$, are given by the following expression:

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right).$$

Here the hyperbolic tangent function is used.

At this point, the cell update can be computed: this is obtained by summing selected past information (obtained by multiplying the forget gate with the previous cell state) and selected current information (obtained by multiplying the input gate with $\tilde{C}_t$):

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t.$$

Finally, the output gate is implemented using an expression similar to those of previous gates:

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right),$$

and it is used to determine the new hidden state, that is a filtered version of the cell state:

$$h_t = o_t \cdot \tanh\left(C_t\right).$$

4.3 Dimensionality reduction approaches

After a data preparation phase, each sample is represented by a feature vector $\boldsymbol{x}$ that can be used as input for the classification models. An important question is how each component of this vector contributes in the training phase. Intuitively, just a relatively small amount of these features has a crucial role, while others may be useless, representing even an obstacle for the classifier. In order to overcome this problem, it is possible to use some specific statistical learning or machine learning techniques which are able to identify a minimal subset of features that describes well the variance of the dataset and also provides good generalization capacity. This process goes under the name of feature selection.

Other methods achieve a dimensionality reduction by transforming, instead of selecting, the existing variables. Some advanced algorithms build specifically for this purpose are unsupervised, so they analyze unlabeled data finding an intrinsic structure in them.

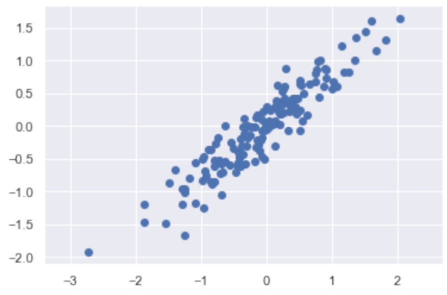
The advantages of similar methods can be many, depending on the problem at handle. A dimensionality reduction can improve predictive power by removing useless information, reduce the computational complexity of a problem, perform a lossy data compression, enable data visualization, and so on.

In the following subsections, two examples are described.

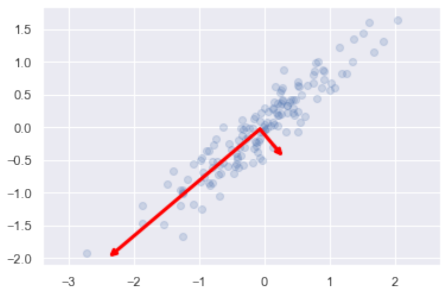
4.3.1 Principal Component Analysis

Principal component analysis (PCA) is an unsupervised method that can be used for dimensionality reduction purposes. By way of example, let's consider the two-dimensional dataset in Figure 4.11. It's clear that between the abscissa and ordinate variables a linear dependence exists.

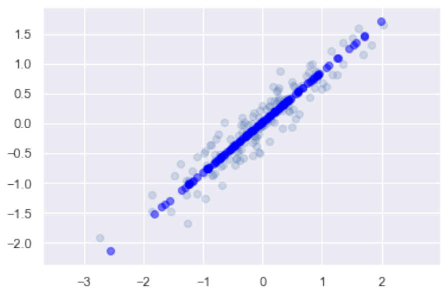
From a linear algebra perspective, PCA looks for a new basis, which is a linear combination of the original one, that better expresses the dataset at handle, so, it aims at finding the directions with a higher signal to noise ratio. In other words, it identifies the subspace such that the orthogonal projection onto it maximizes the variance of the projected points. In the example in the Figure, it is pretty evident that by considering just one component, the main data behavior is kept. Therefore, it is possible to orthogonally project the samples along that axis to obtain a dimensionality reduction.



(a) *Input Data*



(b) *Principal Components*



(c) *Projection into the first component*

Figure 4.11: PCA for dimensionality reduction (adapted from: VanderPlasp, 2016).

In more details, PCA computes the covariance matrix, which presents the variances on the diagonal elements, and the covariances in the off-diagonal elements. Then, since the quantity to minimize is the redundancy, measured by the covariance, and the one to maximize is the signal power, measured by the variance, ideally, the covariance matrix should be diagonal. So, the problem becomes finding a linear transformation, represented by matrix P, leading to the new base $Y = PX$, such that the covariance matrix of Y is diagonal, and with the diagonal terms rank-ordered according to the variance associated with the corresponding dimension. All this is possible thanks to the eigenvector decomposition theorem, from which it follows that taking the eigenvectors of the covariance matrix of X as rows of P, the desired properties listed before hold.

4.3.2 t-distributed Stochastic Neighbor Embedding

One limiting assumption of PCA is linearity, which frames the problem as a change of basis. An example of non-linear dimensionality reduction is t-distributed stochastic neighbor embedding (t-SNE). It is a statistical method mainly used for visualization purposes, which applies to each data point a two or three-dimensional map. Like PCA, it is unsupervised and non-parametric.

As first step, a probability distribution over pairs of data samples is constructed in such a way that similar recordings are assigned higher probability. Given two samples $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, let's define

$$p_{j|i} = \frac{\exp\left(- \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k\neq i} \exp\left(- \left\|\boldsymbol{x}_i - \boldsymbol{x}_k\right\|^2 / 2\sigma_i^2\right)}.$$

The interpretation has been explained by Van der Maaten and Hinton (2008): "the similarity of data point $\boldsymbol{x_j}$ to data point $\boldsymbol{x_i}$ is the conditional probability, $p_{j|i}$, that $\boldsymbol{x_i}$ would pick $\boldsymbol{x_j}$ as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at $\boldsymbol{x_i}$". From this, it is possible to define:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}.$$

Then, a second probability distribution is defined in the lower-dimensional map. Let's denote with $\boldsymbol{y_i}$ and $\boldsymbol{y_j}$ two points in this space and define:

$$q_{ij} = \frac{\left(1 + \left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|^2\right)^{-1}}{\sum_k \sum_{l\neq k} \left(1 + \left\|\boldsymbol{y}_k - \boldsymbol{y}_l\right\|^2\right)^{-1}}.$$

Finally, t-SNE determines the locations of the points in the low dimensional space by minimizing, using gradient descent, the Kullback–Leibler divergence of the first distribution P to the second one Q:

$$\mathrm{KL}(P\|Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

5 Data exploration and pre-processing

5.1 Importing and cleaning the dataset

Once having retrieved the data from the company database as described in Section 2.3, the dataset has been loaded. However, some formatting issues have been encountered, namely:

1. empty recordings;
2. split recordings;
3. wrong column type (e.g. numeric columns interpreted as string variables).

The third point is a direct consequence of the second one, because a recording divided in multiple rows implies data misalignment with values inserted in the wrong columns. In order to understand the cause for the recordings' divisions, some additional queries were performed against the original Postgres database and the results were compared with the dataset already imported. Most of the time, the reason could be identified in the presence of particular characters in the textual variables and also in some extremely long textual values. Therefore, the code necessary to correct these issues was implemented, with the aim of restoring as much data as possible.

Then, an overview of the correctly formatted dataset was given. The first thing that has been notice was the large number of missing values for three variables (two of them were completely empty). Probably, they were used in previous versions of the application or have just been generated by mistakes. Another variable with a large percentage of missing values (more than 80%) was the variable `note`. Since it may contain relevant information for the classification task, it has been merged with the other textual variable, `workdesc`. This was a reasonable choice for two reasons: first of all, they both provide a description of the activity, with the only difference that what is inserted into the `note` field is visible only internally and not to the customer. Secondly, from a data analysis point of view, they present the same kind of data, so they have to be processed in the same way. Once merged, the original variable `note` was dropped along with the other two empty ones.

The missing values analysis was followed by a first exploration of the available variables, with the aim of finding out if there were others not helpful for the task. In fact, this was the case for three of them, which have been removed:

- one indicating the unit of measure of the time variable, always equal to a fixed value;
- a variable representing the kind of application used to save the job report: even in this case, all recordings had the same value, "web";
- a boolean variable always equal to "False", with the exception of a couple of samples.

Before proceeding with the data exploration, some other preliminary steps have been performed in order to obtain a clean and consistent dataset. Without going into details, some duplicated recordings with particular flags have been removed, since not useful for this project. Up to this point, the indications given by the company employees who knew the meaning of the dataset and how it has been generated played a central role.

At the end of this procedure, each row of the dataset corresponded to a specific recording. However, a job report could be present multiple times, in the case it was rejected and then reinserted with corrections. Since the aim of the project is to create a model able to discriminate between correct and wrong insertions, a possible choice is that if a job report is present multiple times, then, only one wrong copy should be kept.

Finally, the dataset was split into training, validation and test set with a proportion of 60%, 20% and 20% of the samples respectively. Note that stratified sampling has been used, so in all sets there is the same target distribution, which is important in case of unbalanced dataset.

In the next sections, the variables in the dataset obtained after these preliminary operations are analyzed individually in details. In order to have a complete overview, in Table 5.1, they are listed along with a very brief description.

Variable name	Description
`tipo_update`	indicates if the job report has been accepted or not
`jobid`	identifier of the project
`jobtaskid`	identifier of the activity
`resid`	identifier of the resource who inserted the job report
`jobtaskdt`	day in which the activity in the job report was carried out
`data_ins`	date and time of the insertion
`qty`	number of hours indicated in the job report
`custid`	identifier of the customer the resource worked for
`sede`	where the activity was carried out
`flg_trasferta`	if the activity has been carried out at the customer site
`pay`	if the activity has to be considered for billing purposes
`flg_prepagato`	if the activity is prepaid
`flg_straordinario`	used to indicate an eventual overtime
`workdesc`	textual description of the activity

Table 5.1: List of variables in the original dataset.

5.2 Non textual variables

5.2.1 Response variable

After having filtered the rows of the dataset, a detailed data exploration can start. The total number of recordings is 21802. The response variable, `tipo_update`, has only two possible values, indicating if the job report inserted has been approved or rejected. A crucial observation is that the dataset is highly unbalanced: around 90% of samples are labeled as approved. Since the task is to identify erroneous insertions and these are a small fraction with respect to the correct ones, it is possible to state that it deals with an anomaly detection problem. As a consequence, it is important to implement possible counter measures, such as resampling approaches, and, at the evaluation time, to consider particular metrics, which take into account class frequencies.

5.2.2 Project and activity name

In what follows, all predictive variables are listed, explaining what they represent, the values assumed and how they have been encoded. The first one is `jobid`, the code which identifies the project that the employee has been working on. As explained in Section 2.2, it is the most important parameter of a job report, since it is used for billing purposes. Indeed, the final choice to accept or reject a job report depends mostly on it. The possible values are 334, and even though they are numbers, the variable is categorical: each value is an identifier.

From Figure 5.1, it is possible to get an idea of the distribution of this variable: clearly, it is far from being a uniform one, since some values are present in thousands of recordings, while others have been selected only a few dozens of times.

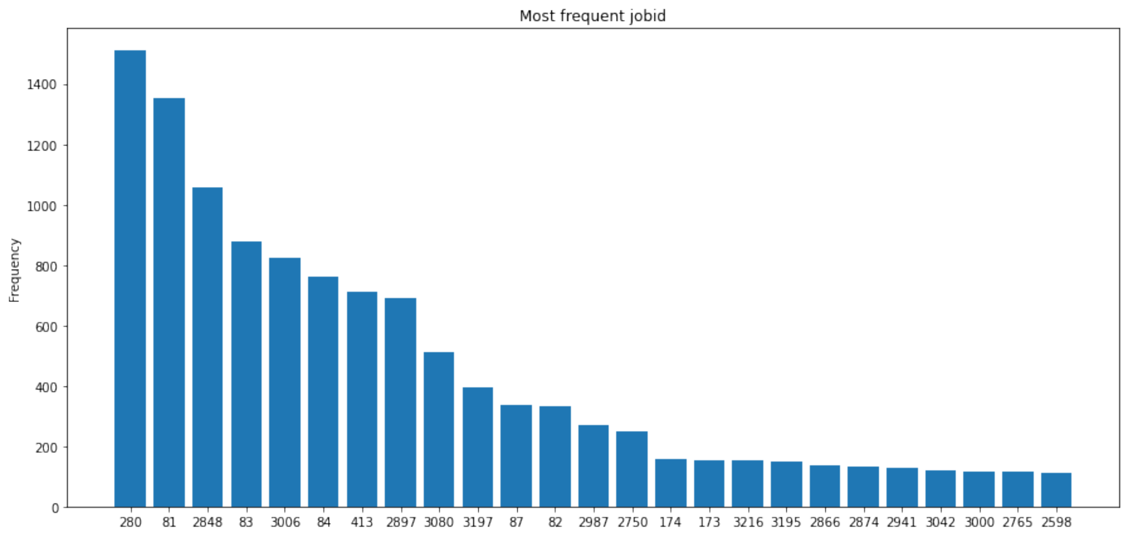


Figure 5.1: Frequency of the 25 most common project identifiers.

The second predictive variable is `jobtaskid`, that is an identifier of the activity selected. The possible values are 620, and, even in this case, some of them are present in a couple of thousands of recordings, while others are very rare.

5.2.3 Resource level features

The employee who inserted the job report is identified by the variable `resid`. There are 97 different values and their frequencies clearly depend on the assumption date. To get a first insight, the error rate of each resource was computed (Figure 5.2).

The percentage of mistakes is pretty high (larger than 25%) for three resources, and it is equal to zero in 25 cases. This suggests that for some employees it is more difficult to identify the correct project name when inserting the job report, while for others it is straightforward. A possible explanation is that some people have access only to a large number of choices, while others do not.

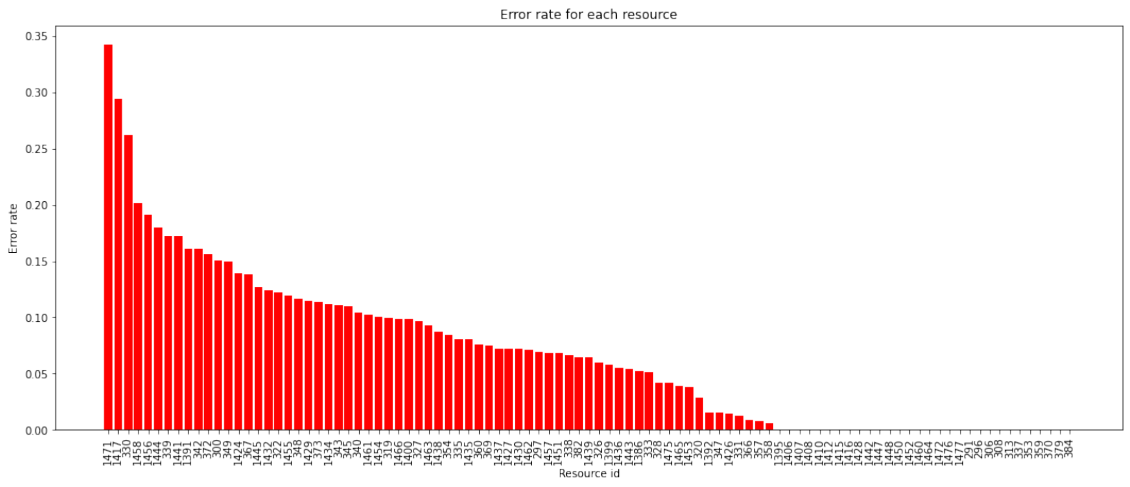


Figure 5.2: Error rate for each resource.

This hypothesis has been confirmed computing the number of distinct project names inserted by each resource. In fact, it was found out that four employees had more than 50 project names, while around 20 people have always selected the same option (Figure 5.3).

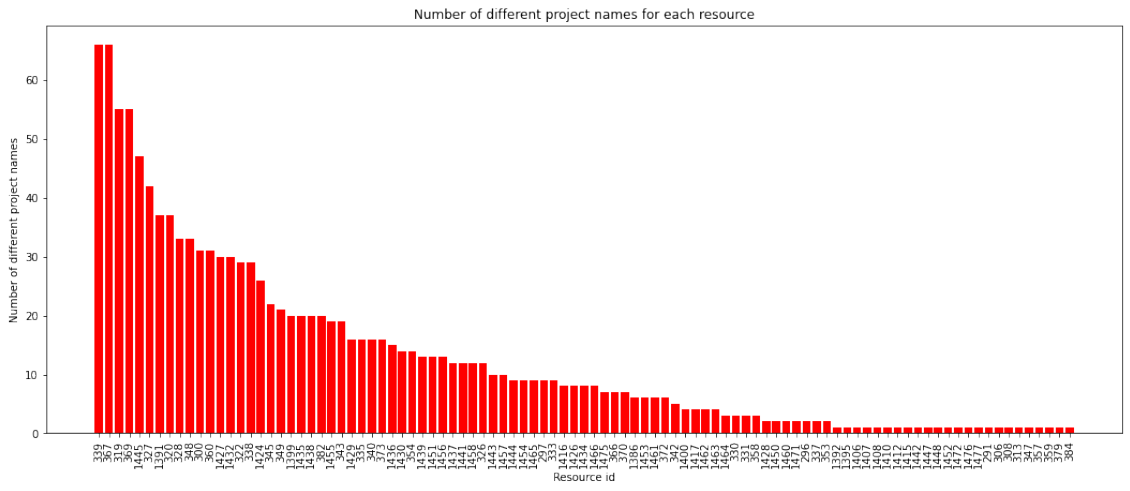


Figure 5.3: Number of distinct project names for each resource.

The analysis of this variable suggests a couple of ideas in view of solving the prediction task. First of all, mistakes are probably less likely to occur among people that have been working in the company for many years, thanks to their experience, while younger employees may have more doubts. Another aspect related to the resources is the area they belong to: if this information was added to the dataset, then the predictive capacity is likely to increase, because each area works at different kind of projects. These pieces of information were retrieved from another company database. The additional dataset has as columns the resource identifier, the date of recruitment and the area of affiliation.

Therefore, it was possible to perform a manual feature engineering phase, adding to the original dataset the variables `area` and `date_of_recruitment`. Before proceeding with the exploration of the other variables, an analysis of these resource-level features has been conducted and their correlation with the response variable has been studied.

For the previously explained reason, the belonging's area could be a possible motivation for the differences in terms of error rate between resources. In Figure 5.4, it is possible to see that the error rate varies from the 0% of the human resources office, up to the 13% of the system administrator area.

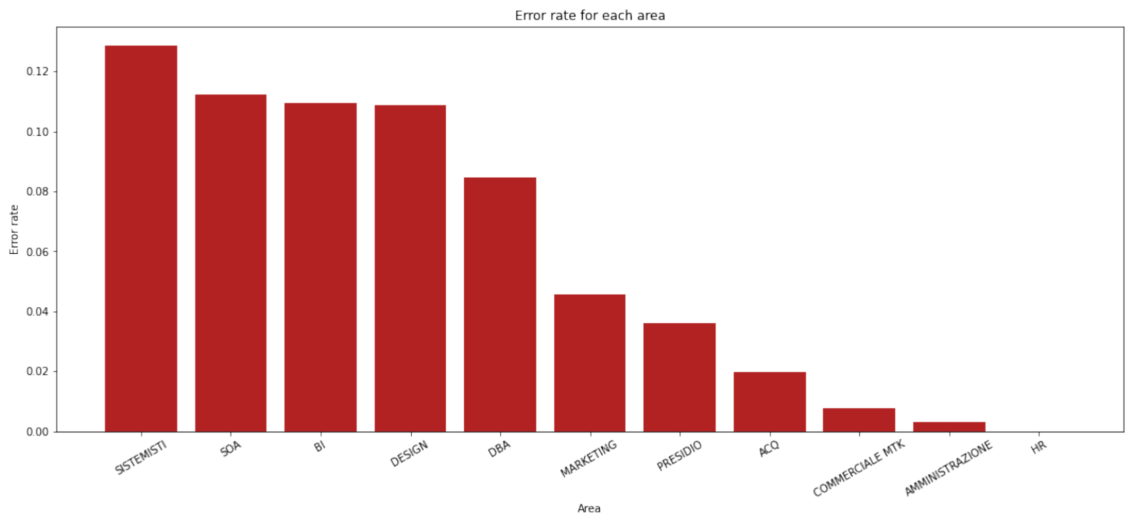


Figure 5.4: Error rate for each area.

It has been computed that the system administrator one is also the area with the largest number of project names (over 169 different identifiers have been used), while for the HR area there is only one option (Figure 5.5). This explains why some resources have high error rates, while others have no mistake at all.

The difference in terms of number of options for each area is even more evident by considering the number of distinct activity values used (Figure 5.6): system administrators and database administrators have around 250 possible values, while for the HR and administration area there are only two options.

The other resource-level feature coming from the additional dataset that has been analyzed

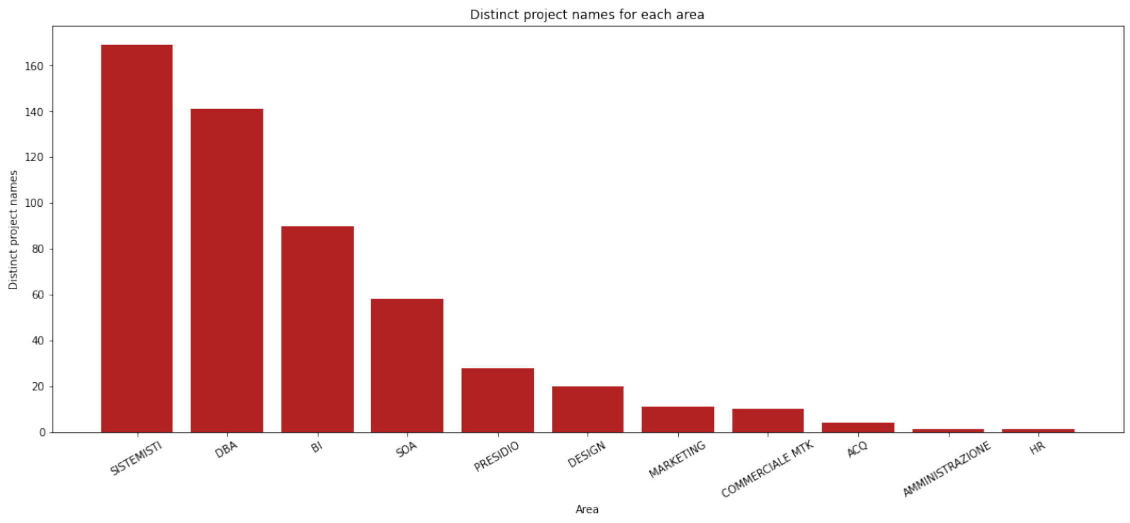


Figure 5.5: Number of distinct project names for each area.

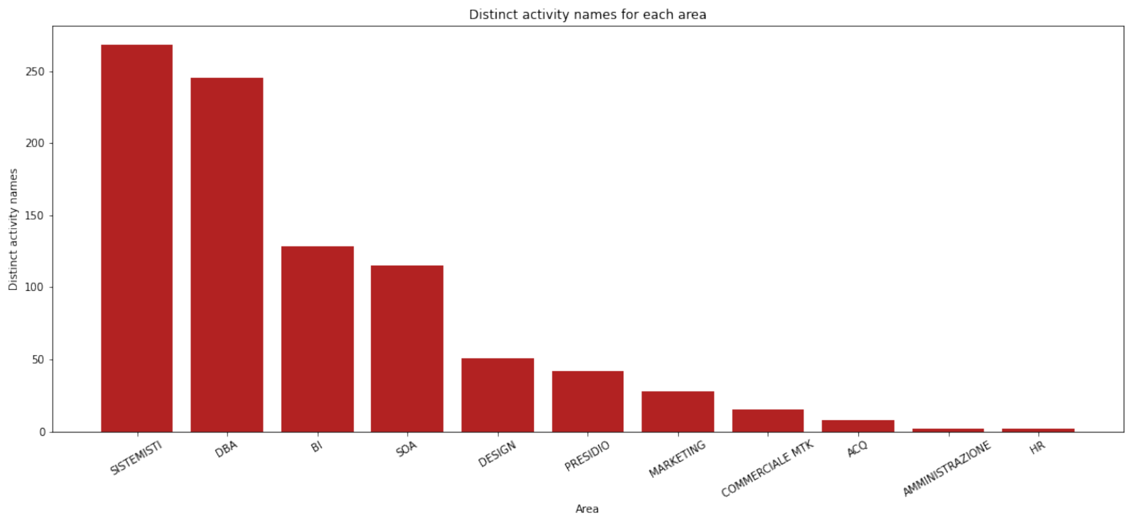


Figure 5.6: Number of distinct activity names for each area.

more in depth is the `date_of_recruitment`. The first sample recorded in the dataset at disposal dates back to March of 2007, while the last one has occurred few weeks before the dataset was retrieved from the company database. For all the employees hired during the same month, the total number of job reports inserted and the total number of mistakes have been computed. In Figure 5.7, the error rate is represented as a function of the assumption date. From this analysis, no particular pattern can be underlined, therefore, the presence of a lower error rate for older employees cannot be concluded.

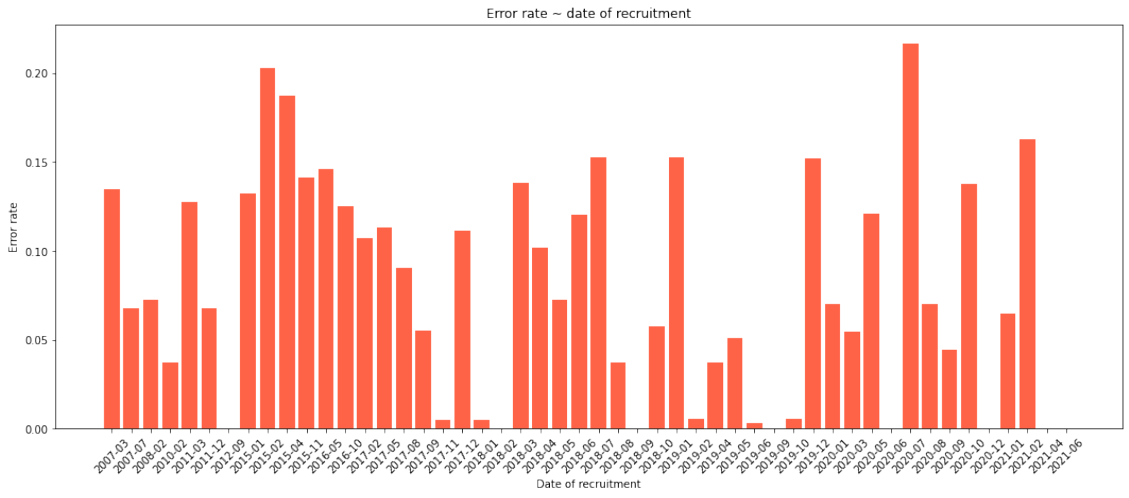


Figure 5.7: Error rate as a function of the assumption date.

5.2.4 Time related variables

Then, the variable `jobtaskdt` was considered. Its values represent the day in which the activity described in the job report was carried out. By parsing this variable, the day and month have been extracted, and a time series analysis has been carried out. In the case of the `month` variable (Figure 5.8), it is not possible to identify the presence of a trend or pattern due to the extremely short time horizon: the number of job reports is significant only during the last ten months.

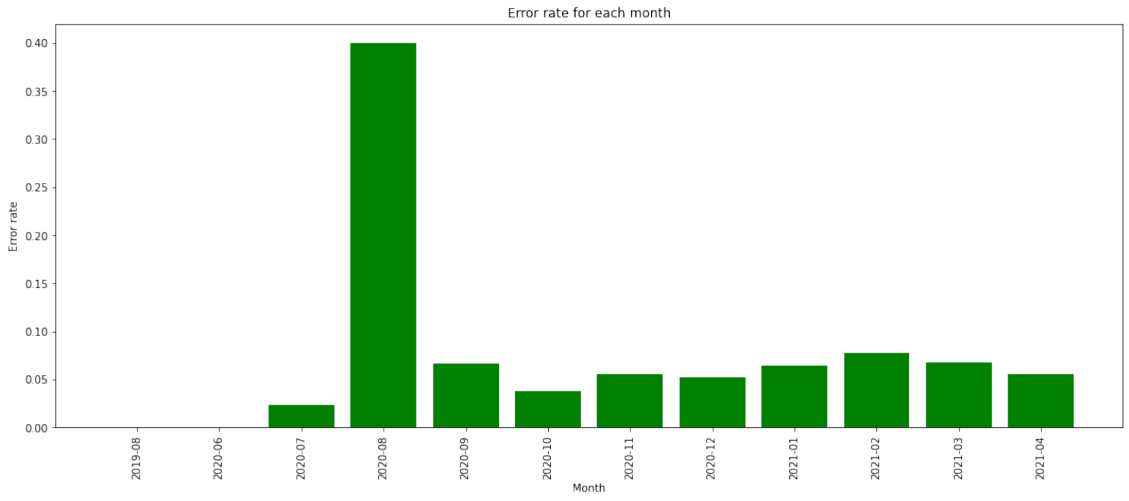


Figure 5.8: Time series of the error rate on a monthly basis.

Also, the time series of the error rate at the daily level seems to be white noise (Figure 5.9). Possibly, during the last 150 days it is possible to identify the presence of a weak seasonality, but it seems not enough marked to justify a further analysis.

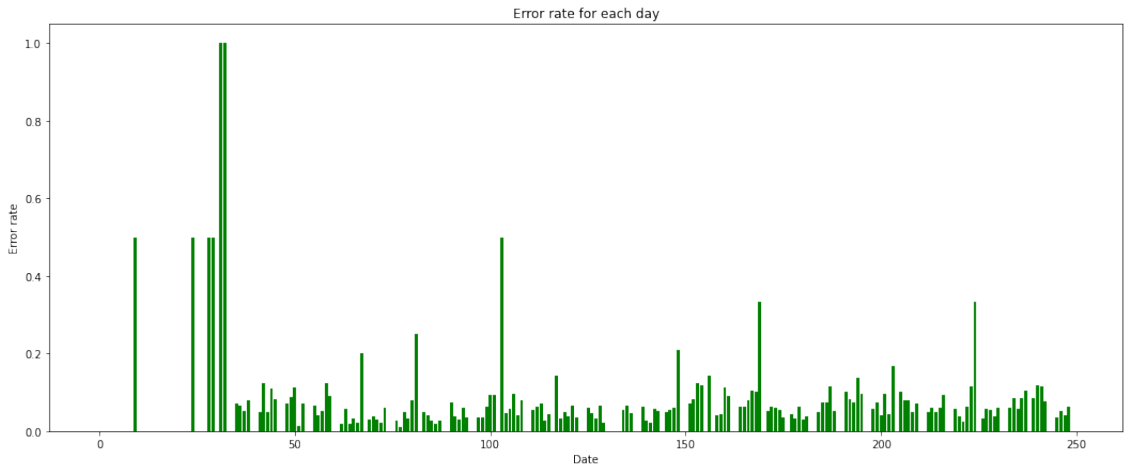
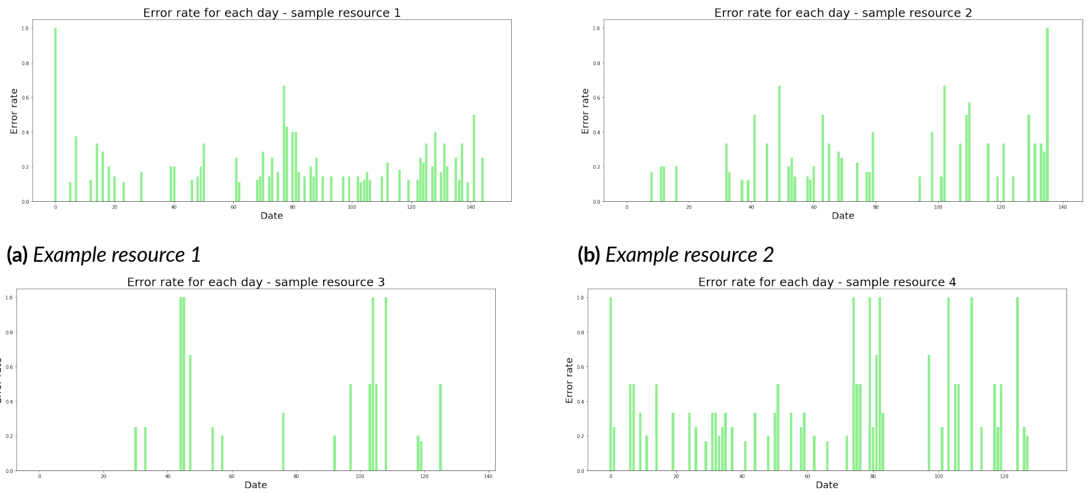


Figure 5.9: Time series of the error rate on a daily level.

This analysis was also carried out from a different perspective: some employees were chosen randomly and, for each of them individually, the error trend was studied. In Figure 5.10, four examples are reported. Even for them, it is not easy to identify any specific trend: in all cases, there is white noise.



(a) *Example resource 1*

(b) *Example resource 2*

(c) *Example resource 3*

(d) *Example resource 4*

Figure 5.10: Trend of the errors for four samples.

Before continuing, another idea concerning time variables deserves to be described. Quite often, job reports are not inserted at the end of the day, but with a delay of a few days. This can be noticed by comparing the variable `jobtaskdt` just analyzed with the variable `data_ins`, which indicates the date and time of the insertion. This suggested to manually create another variable, `delay`, simply computing the difference between the insertion date and the day indicated in the job report. The rationale behind it is that describing the activities of the same day can be easier and therefore more accurate than trying to remember the details of what has been faced one week before. As a consequence, the delay of an insertion may be correlated with the error rate. In Figure 5.11, it is possible to see that the large majority of the job reports was inserted before the end of the same day or within few days.

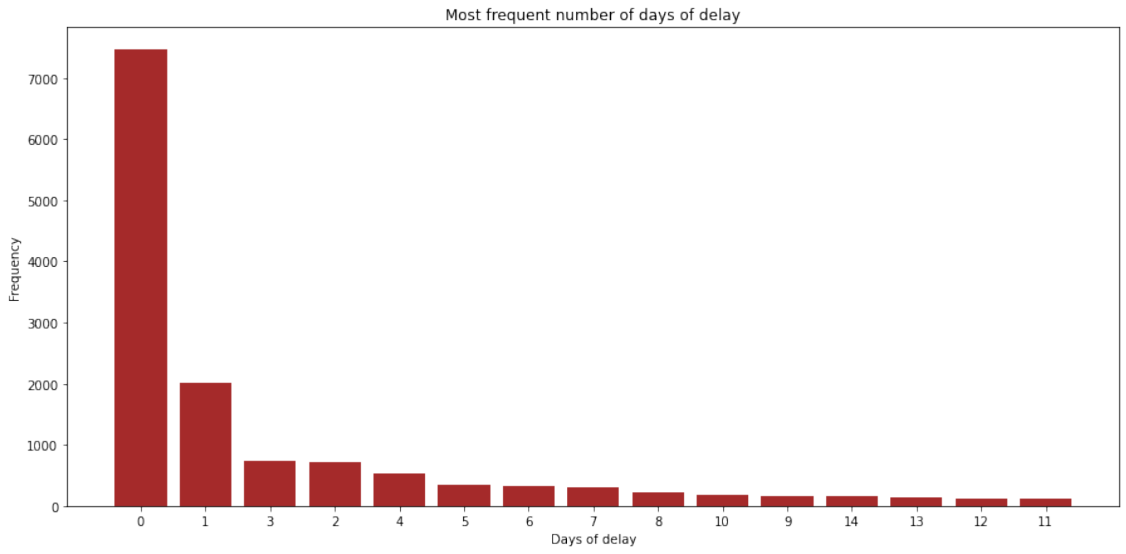


Figure 5.11: Most frequent number of days of delay.

5.2.5 Other non-textual variables

The variable `qty` represents the number of hours indicated in the job report. As anticipated, it must be a multiple of 30 minutes. By looking at its distribution shown in Figure 5.12, it is possible to notice that 8 hours is the most frequently chosen duration, which makes sense considering that it is the length of a standard working day.

What is more unexpected is that very often, the option one hour or just half an hour are selected. This may happen, for example, when a resource is asked to face a urgent job different

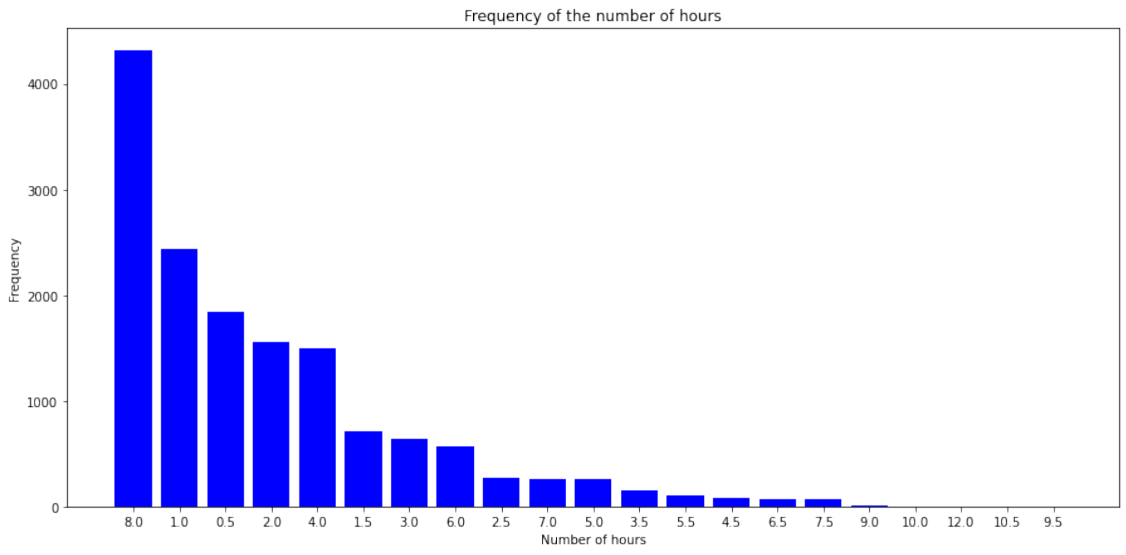


Figure 5.12: Frequency of the selected number of hours.

from his/her planned activity for that day, or simply because the employee have planned to work at different projects and some of them require very little time.

Another available variable is `custid`, that is an identifier of the customer the resource worked for. To be precise, there are some identifiers related to internal activities, and they are the most frequent ones highlighted in Figure 5.13.

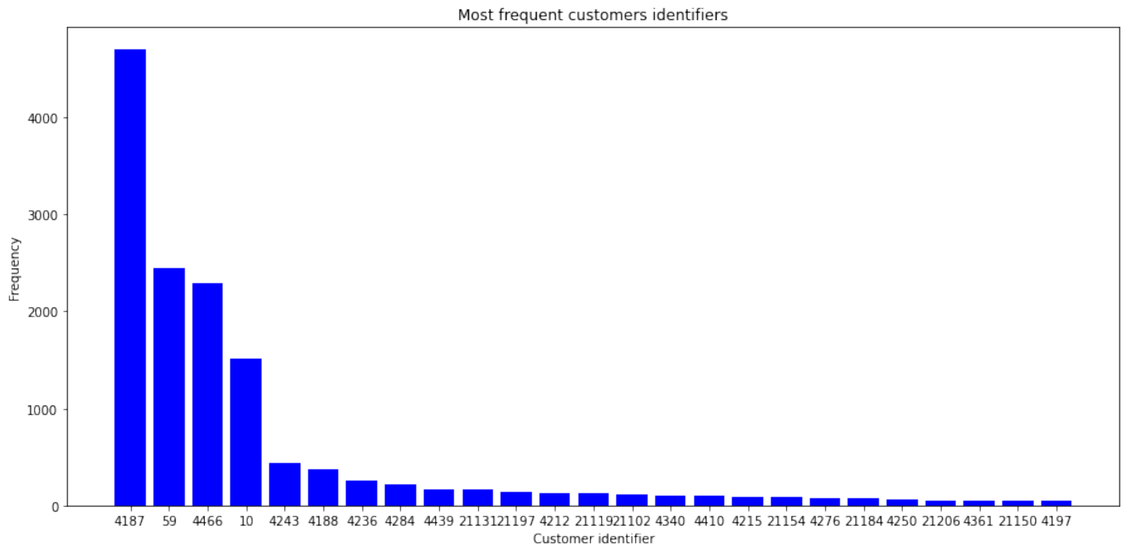


Figure 5.13: Most frequent customer identifiers.

The total number of different customer identifiers is 119, but only 20 of them have been selected a significant number of times (in more than 100 job reports).

Then, there is the variable sede, which indicates where the activity was carried out. There are 9 possible values because, besides the main offices in Padova and Thiene, it is possible to choose the smart working option, presidium Veneto region or other options related to working activities at the customer site. Due to the Coronavirus period, the most frequent option is smart working, followed by Thiene and Padova (Figure 5.14).

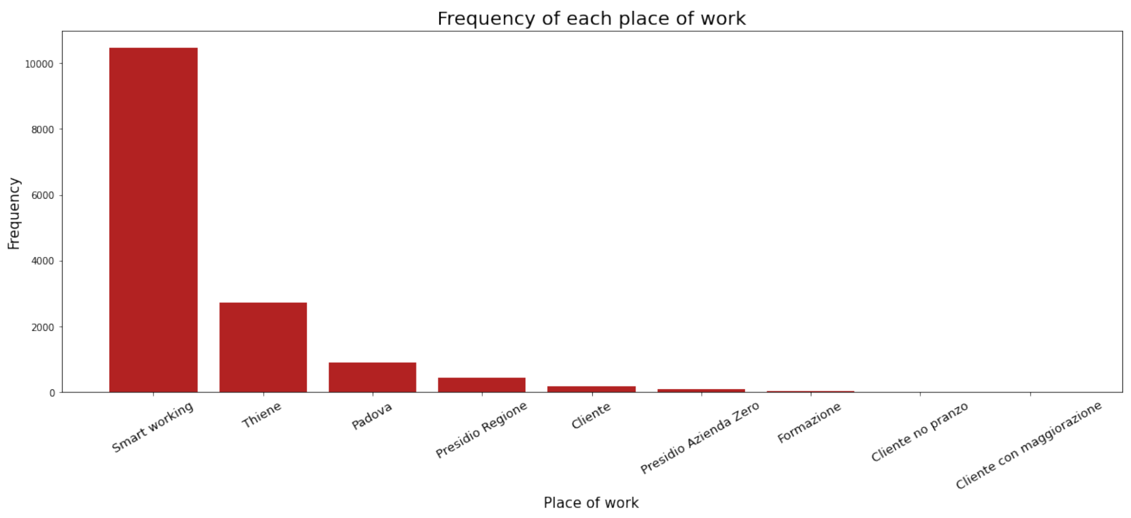


Figure 5.14: Frequency of each work location.

Among the remaining variables there are four boolean flags:

- flg_trasferta: if true, it means the activity has been carried out at the customer site;
- pay: if true, it indicates that the activity has to be considered for billing purposes;
- flg_prepagato: it specifies whether the activity is prepaid;
- flg_straordinario: to indicate an eventual overtime.

Their distribution is the shown in Table 5.2.

In order to deepen the analysis, for the variables qty, sede and the four flags, a bivariate analysis was conducted. In Figure 5.15, the cross tabulation of these variable with the response is represented. The name of the response variable in the figure is tipo_update, while the values assumed are approvato or cambio_commessa, which stand for approved or rejected.

	approved job reports	rejected job reports
`flg_trasferta`	14256	586
`pay`	13490	1352
`flg_prepagato`	8700	6142
`flg_straordinario`	14355	487

Table 5.2: Distribution of the four boolean variables.

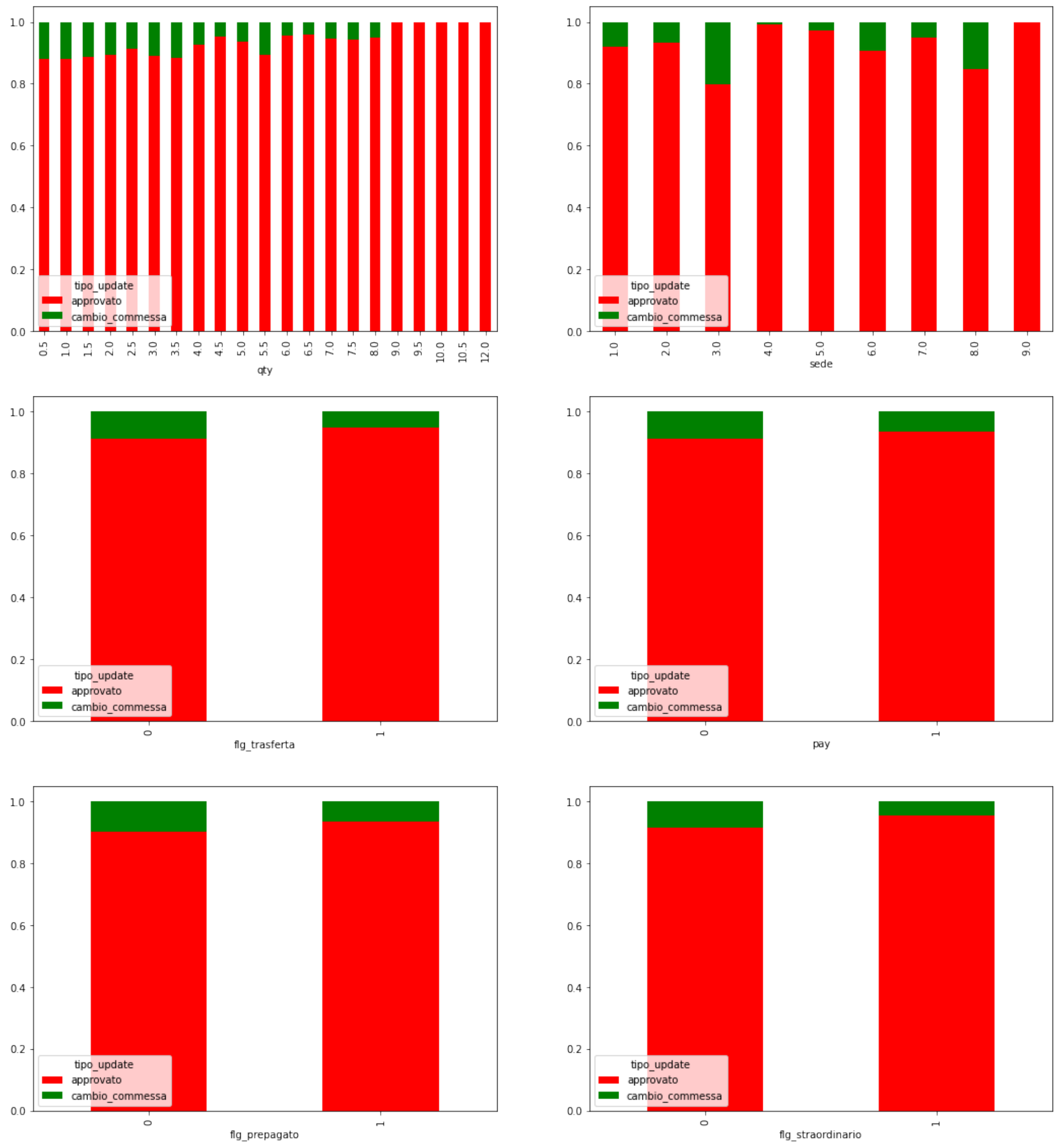


Figure 5.15: Bivariate analysis.

It is possible to see that there are no rejected job reports for values of the variable `qty` larger than 9. However, this is not very useful since they comprehend a limited number of recordings. Looking at the variable `sede`, the percentage of wrong insertions changes depending on the value assumed. The same cannot be concluded for the four flags, which do not seem to have an influence in determining the response value.

5.3 Text Mining

After having merged the variables `comments` and `note`, there is only one column with textual values. This section will explain step by step how this variable containing the description of the working activities has been cleaned and analyzed. First of all, some global statistics of the original corpus have been computed. What emerges is that, on average, the descriptions are quite short (only 8 or 9 words), with some exceptions close to 200 words. The distribution of the description length is shown in Figure 5.16.

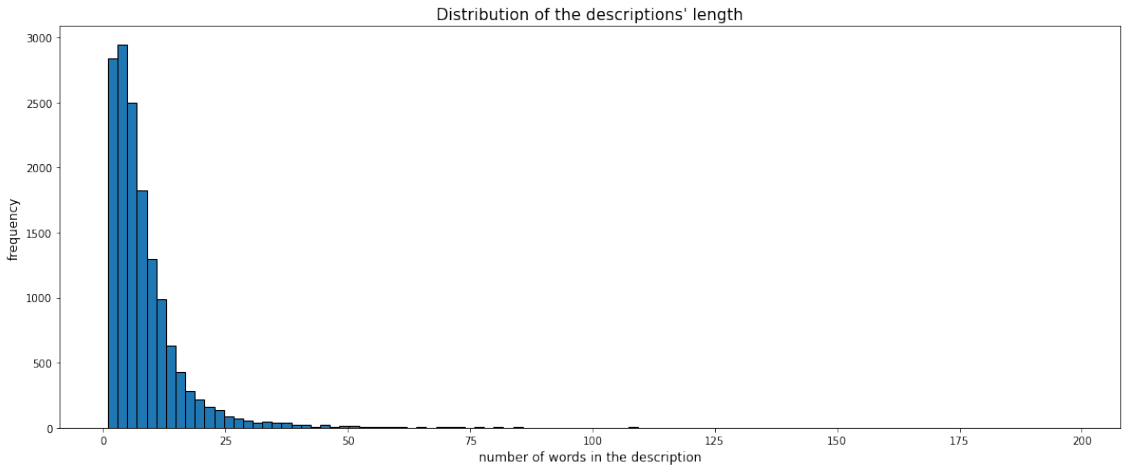


Figure 5.16: Distribution of the description length.

Most of the terms are Italian words, while English ones are a minority. The total number of different words is 14861, which is very large compared to usual corpus. The reason behind it is that there are lots of ticket numbers, which are alphanumeric strings that identify certain works. In addition to that, there is a significant amount of very specific terms that are not shared by many job reports. Finally, it is necessary to take into consideration that these statistics refer to the original, raw corpus. In what follows, it is described how it has been cleaned and reduced to stems, and, after that, a more significant analysis is reported.

5.3.1 Text cleaning

Data cleaning is the process of extracting as much useful information as possible from a certain dataset, removing all those parts that are without value, redundant or deviant for the final aim of the analysis. In the case of textual variable, the sources of noise are several and have been discussed in details in Section 3.2. The first one is stop words: these are those parts of the sentence that are needed to make it clear and sound for a human being but that probably are not functional for the analysis at handle, and therefore must be discarded. For instance, articles and prepositions fall within this category.

Punctuation also do not carry relevant information for this problem. For this reason, a list of characters to be removed has been generated. Other things that have been removed are URLs, e-mail addresses and numbers. Finally, all remaining words have been converted to lowercase.

To check the effectiveness of all these passages, the number of remaining words has been computed at each step:

- removing Italian stop words and the main punctuation marks, the number of terms decreased from 14861 up to 8660, which is quite remarkable: it was already basically halved;
- filtering out also English stop words, the remaining terms were 8609, not a relevant decrease, but this was expected, since English terms were outnumbered;
- removing URLs, only 8528 terms survived;
- discarding also e-mail addresses, the remaining terms were 8402.

Clearly, what reduces most the number of terms was the Italian stop words' removal, while the other phases had a more marginal effect.

The last phase of the cleaning process is the stemming. As previously described, it consists in reducing each word to its root, getting rid of verb conjugation or of terms declination. Different tools provide an implementation of this procedure.

5.3.2 Analysis of cleaned text

Once terminated the cleaning phase, the number of distinct stems was still high: 6674. According to the literature, usually the number of stems is at most 300/500, but some exceptions occur (see Section 3.2). In order to get an explanation about this fact, the cleaned text was analyzed. It was found out that on average a term is present in only 14 samples. This is due to the short length of each description and also to the massive usage of specific terms. As a consequence, there are lots of extremely rare ones, present only in 1 or 2 job reports. For each frequency, the number of terms present with that frequency was computed, and this number is reported on the y-axis of Figure 5.17.

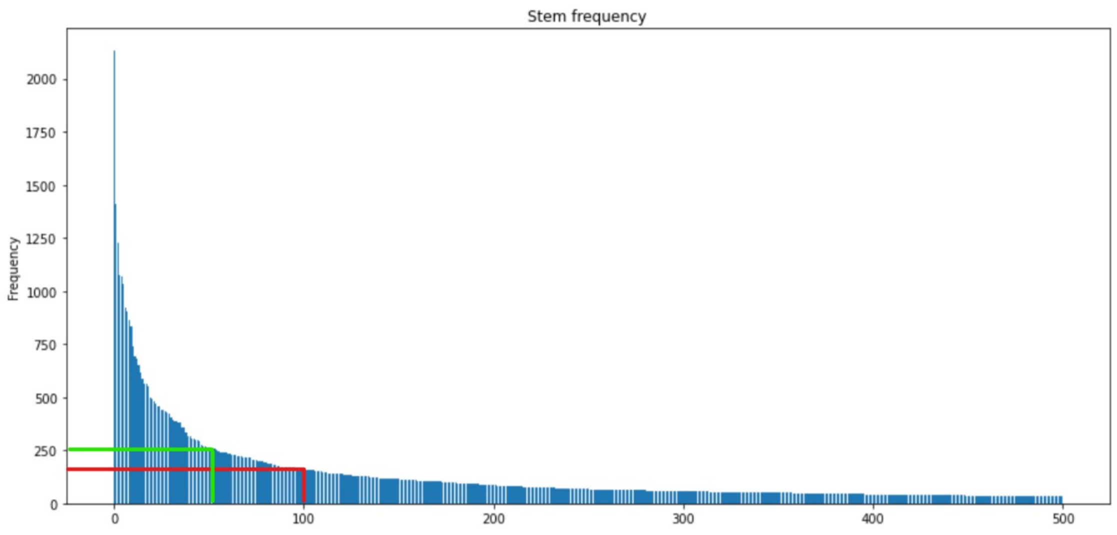


Figure 5.17: Distribution of the stem frequency.

To help with the interpretation of this figure, let's consider these three observations:

- there are 2130 terms that are present in only one job report;
- there are about 250 terms that are present in exactly 50 job reports (green line in the figure);
- there are about 170 terms that are present in exactly 100 job reports (red line in the figure).

Then, the response variable came into play: the main objective of this part of the analysis was to check if the stems were well represented in both classes. It was found out that:

- among the 300 most frequent stems, 282 are present in the `cambio_commessa` class;
- among the 500 most frequent stems, 465 are present in the `cambio_commessa` class;
- among the 750 most frequent stems, 669 are present in the `cambio_commessa` class.

Related to this, the most frequent stems of both classes have been computed, and the first 20 are represented in Figure 5.18. In addition to that, the word clouds are shown in Figure 5.19. From this analysis, it seems that the two classes share some of the most frequent terms, therefore, in order to discriminate between them, it could be better to keep also less frequent terms.

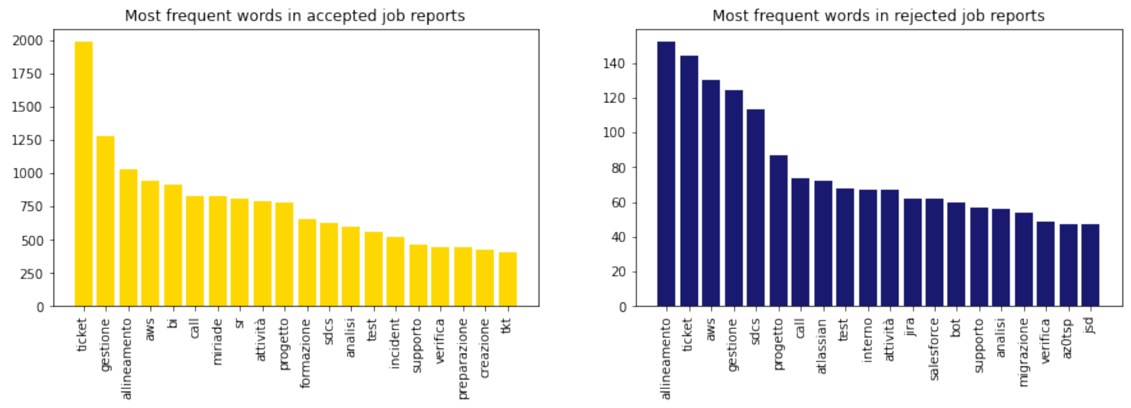


Figure 5.18: Most frequent stems of both classes.

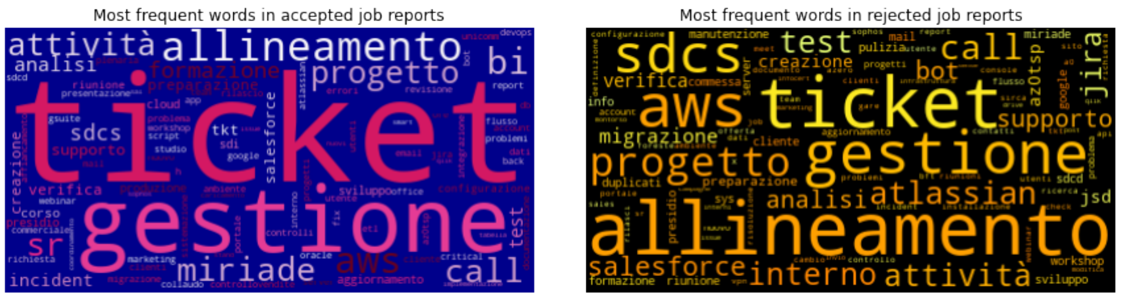


Figure 5.19: Word clouds of the two classes.

To conclude, the number of stems to keep will be one of the most relevant tuning parameters of the models, and, probably, it will play a key role in the trade-off between predictive accuracy and model complexity.

5.4 Other pre-processing steps

The last step before being ready to train a model consists in using the correct encoding for the variables. Regarding the textual variable, a simple method returning a matrix of token counts and an implementation of TF-IDF have been used. Four parameters regarding these methods have been considered:

- the n-gram range, represented by a couple of values. For example, the pair (1,2) means that both unigrams and bigrams are considered, while with (1,1) only unigrams are used;
- the minimum document frequency: the stems with a document frequency lower than this value are discarded;
- the maximum document frequency: the stems with a document frequency higher than this value are discarded;
- the maximum number of terms N that will be considered, meaning that after having applied the two previous thresholds regarding document frequency, only the most frequent N terms will be kept.

Regarding the other variables of the dataset, for the categorical ones, the approach used was one-hot encoding, while for the response variable, simple label encoder was used: essentially, the class of correct job reports was identified with the value 0, while the class of wrong insertions with 1.

6
Experiments and results

This chapter focuses on the way the models described in Chapter 4 have been implemented, optimized and evaluated for the problem at handle. In particular, after having presented the experiments carried out using different classifiers, the deep learning architectures constructed are described in details, highlighting some particular strategies that have been considered. Then, all the results are commented and compared.

6.1 Implementation

As anticipated in the previous chapter, the dataset has been split in training, validation and test set in a stratified fashion, meaning that in all three sets the distribution of the two classes is the same. In Table 6.1, the number of samples in the three sets are reported divided accordingly to the class they belong to.

	approvato	**cambio_commessa**
Label	0	1
Proportion	90.76%	9.24%
Training set	11872	1208
Validation set	3958	403
Test set	3958	403

Table 6.1: Class distribution in the dataset.

In order to limit the problem of class imbalance, a function performing different resampling methods has been implemented. In more details, it takes as input the training set and a parameter which specifies the approach to use. Possible options for this latter are:

- keep the original proportion;
- perform undersampling: keep all of the data in the minority class and randomly select from the majority class a number of samples equal to those of the minority class;
- perform oversampling: keep all the elements in the majority class and replicate those of the minority class until reaching the cardinality of the majority class;
- perform something between the last two approaches: the samples in the majority class are halved, while those of the minority one are replicated until reaching two classes with the same cardinality.

The advantage offered by resampling techniques is that a learning model trained on such a dataset is expected to be less biased. However, these approaches have some limitations: with undersampling, a large amount of data is wasted, while, with oversampling, many replicas are generated and clearly, they are not as good as having additional samples. Indeed, in this way, a very strong distortion is introduced.

Regarding the evaluation of the models, the samples of the correct job reports class have been identified as actual negatives, while those of the wrong job reports class as actual positives. To evaluate a supervised model, it is sufficient to compare its predictions with the correct labels. In this way, the confusion matrix can be derived. Its entries are:

- True Positives (TP): observations correctly predicted as positive;
- True Negatives (TN): observations correctly predicted as negative;
- False Positives (FP): observations predicted as positive but which are actually negative;
- False Negatives (FN): observations predicted as negative but which are actually positive.

From the confusion matrix, a number of classification metrics can be derived. In particular, those selected for this project are:

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

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

$$\textit{sensitivity/recall} = \frac{TP}{TP + FN},$$

$$\textit{specificity} = \frac{TN}{TN + FP},$$

$$\textit{balanced accuracy} = \frac{\textit{sensitivity} + \textit{specificity}}{2},$$

$$F_1\textit{-score} = 2 \times \frac{\textit{precision} \times \textit{sensitivity}}{\textit{precision} + \textit{sensitivity}},$$

$$\underset{\textit{(MCC)}}{\textit{Matthews Correlation Coefficient}} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}.$$

Then, a function performing word embedding was written. As anticipated, two approaches have been considered: one returning the matrix of token counts and another returning the matrix of TF-IDF features.

A relevant aspect is that the vectorizers have been fitted only on the training set. Then it has been applied on all the three sets. This is important to avoid a phenomenon called leakage, which consists in the use of information coming from the validation and test set during the training process, causing the evaluation scores to overestimate the model's utility when running in a production environment (Kaufman et al., 2011). As a consequence, the stems present in the validation and test sets but not in the training one will be encoded with the same identifier. This is one of the reasons why, in a production environment, it is important to update the training set quite often and re-build the models.

Now the procedure adopted for training each classifier is described. First of all, the list of tuning parameters has been identified: some of them are model-specific, while others are used for every classifier. This is the case, for instance, of the tuning parameters related to the choice of the resampling approach and those related to the word embedding. In a first phase, well separated values for the numeric tuning parameters have been chosen. As suggested by many studies (e.g., Hsu et al., 2016), exponentially increasing sequences of values have been adopted. Then, in a random search fashion*, some combinations of tuning parameters have been selected, and

*to be precise, the approach used is not exactly a random search, since instead of indicating a range in which some values should be randomly picked, the list of possible candidates was pre-defined.

the corresponding model defined and fitted to the training set. The list of model-specific tuning parameters that have been considered is reported in Table 6.2.

model	**tuning parameter**
Logistic Regression	inverse of the regularization coefficient
	maximum number of iterations
SVM	regularization coefficient
	kernel type
	the kernel parameter γ
KNN	number of neighbors to use
	distance metric to use
Random Forest	number of trees in the forest
	criterion used for the information gain
	maximum depth of the tree
	minimum number of samples required to split an internal node
	minimum information gain to allow the split of a node
Gradient Boosting	number of boosting stages to perform
	the maximum depth of the individual estimators
	fraction of samples to be used

Table 6.2: List of model-specific tuning parameters.

For each model, two configurations of tuning parameters have been selected, according to the corresponding results obtained in the validation set: the one leading to the highest validation accuracy, and the one leading to the highest balanced validation accuracy. The reason is that, with an imbalanced dataset, accuracy tends to reward those models more incline to predict the majority class label, therefore leading to a high number of false negatives. As it will be explained later, for the problem at handle, false negatives are more dangerous than false positives, since they represent wrong job reports that the model classifies as correct. On the other hand, for models with a high balanced accuracy, the number of false negatives is more limited, but the false positives are much more, leading to a higher overall error rate (as expected). Nevertheless, for the moment, the two best alternatives are kept for each model, and the decision will be made after further analysis and comparisons.

This procedure was applied for the dataset with all pre-processed variables but also to a version of it considering only the textual variable. By looking at the results obtained in the second case, it is possible to evaluate the effectiveness of the approach used for cleaning and encoding textual information, besides getting a first idea of the reachable performances. In addition to this, by comparing the results in the two datasets, it is also possible to understand whether the inclusion of the other features is beneficial or not.

Then, also some deep learning architectures have been considered. The aim of this part was to understand whether more sophisticated models were able to reach better results for this specific problem.

The first deep learning model considered was a feed-forward fully connected neural network. In order to have a flexible architecture, many tuning parameters have been considered. Those determining the size of the network are the number of hidden layers, and the number of neurons in each layer. Also, different options have been explored regarding the activation functions applied after each hidden layer. The candidates were the Sigmoid, the Hyperbolic tangent and some related to the ReLU activation (Figure 6.1):

- LeakyReLU: it solves the dying ReLU problem (Maas et al., 2014) adding a small negative slope for inputs smaller than 0;
- ELU (Exponential Linear Unit): it solves the dying ReLU problem and also saturates for large negative values, allowing these values to be 0, leading to sparse activations;
- GELU (Gaussian Error Linear Unit): it is defined as $x\Phi(x)$ where $\Phi(x)$ is the standard Gaussian cumulative distribution function. The resulting shape is similar to the ReLU but it has no first order discontinuity;
- Softplus (or SmoothReLU): it is a smooth convex approximation of the ReLU function.

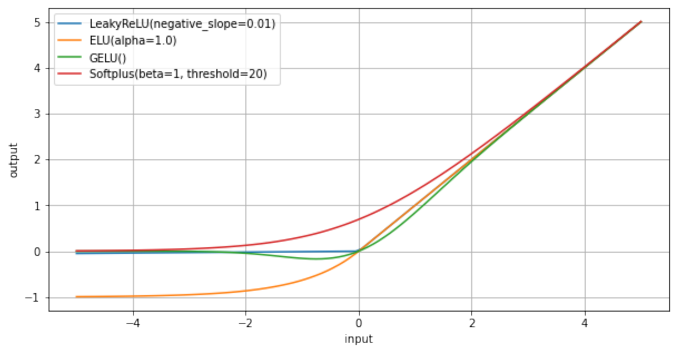


Figure 6.1: Comparison between some activation functions.

The loss function considered was the Cross Entropy, while the optimizers tried were: SGD, RMSprop and Adam. For the first one, a basic implementation without momentum was considered. The advantage of the other two consists in adapting the learning rate individually for each weight depending on the training circumstances. In particular, RMSprop considers a running average of past values of a certain weight to modify the learning rate of that weight, while Adam optimizer takes into consideration also an estimate of the second moments of the gradients.

For what concerns the regularization techniques adopted, dropout layers with a certain probability p after the fully connected layers and the L2 regularization were tried. Actually, the weight decay parameter of the optimizers was modified, but it can be proved that this approach is equivalent to adding an L2 term to the loss function when using SGD if the weight decay factor is reparametrized according to the learning rate (McCaffrey, 2019). Moreover, the effect of batch normalization, a technique which should improve learning stability, was also tested. However, at most one of these three methods at a time was always used, to avoid possible interferences.

To prevent overfitting, an early stopping routine was implemented from scratch. Basically, it interrupts the training loop if the validation loss is not improved for a certain number of successive epochs (patience parameter). However, also a maximum number of epochs was set.

In order to find a satisfactory configuration of the previously mentioned tuning parameters, the same random search approach used for the other classification models was adopted, where the values of each tuning parameter were sampled from a list of candidates. The optimization was done in more than one stage. Indeed, it started by considering a wide range of values for each tuning parameter, and once individuated a region in the parameter-space leading to good results, the focus moved to those values, making other more precise random searches.

For each configuration tested, training and validation loss and accuracy were computed, and the learning curves were plotted to have an additional feedback about the learning process. Even in this case, two architectures were selected for the final evaluation in the test set: the one with the highest accuracy and the one with the highest balanced accuracy in the validation set.

In addition to this, also a Recurrent Neural Network was considered. In this case, the term frequency matrix was not computed, but some LSTM layers were used as feature extractor from the ordered list of words forming the job report descriptions. What is much different with respect to the procedure followed for the FFNN is the pre-processing phase, that is now described.

The dataloader implementation requires input sequences of the same length, but this was not the case. To handle this problem, it is necessary to use padding, which consists in adding empty strings to the shortest descriptions. Note that if all sequences were padded to the same maximum length, a huge number of empty strings would have been introduced, or, alternatively, the longest descriptions should have been truncated. In order to avoid both a data loss and a significant introduction of noise, the following trick was used: the dataset has been sorted according to the number of words in each description, and only afterwards the data batches have been generated. In this way, since the sequences inside the same batch are expected to have similar length, if padding is applied singularly inside each batch, a small number of empty strings would be introduced.

As anticipated, instead of using a word vectorizer, a dynamic representation of each text has been kept. However, a numerical encoding was still necessary. Therefore, a label encoder has been applied, meaning that each description has been converted into a list of numbers.

These batches of encoded descriptions are then passed as input to some LSTM layers. The number of layers and hidden units are new tuning parameters. Since for the problem at handle the architecture is a many-to-one network, only its last hidden state is considered as output.

In this way, a vector representing the high-level features extracted from the recurrent architecture has been created for each input sequence. Then, it has been concatenated with the other variables, and the result has been used as input for some additional fully connected layers.

The rationale behind the other auxiliary functions required for the training procedure (dataloader, training loop, ...) was similar to what have been used for the FFNN, but the implementation was much more complex. Also, the other tuning parameters were the same and the way they have been optimized followed most of the steps previously described. The tuning parameters' values selected at the end of the optimization procedure are reported in the Appendix.

6.2 Results and comparisons

After having retrained the best models in the merged training-validation set, they have been evaluated in the test set. The results are shown in a graphical way in the following figures. In particular:

- Figure 6.2 shows the results obtained for the models with the highest validation accuracy, trained on the dataset only with the textual variable;
- Figure 6.3 shows the results obtained for the models with the highest balanced validation accuracy, trained on the dataset only with the textual variable;
- Figure 6.4 shows the results obtained for the models with the highest validation accuracy, trained on the complete dataset;
- Figure 6.5 shows the results obtained for the models with the highest balanced validation accuracy, trained on the complete dataset.

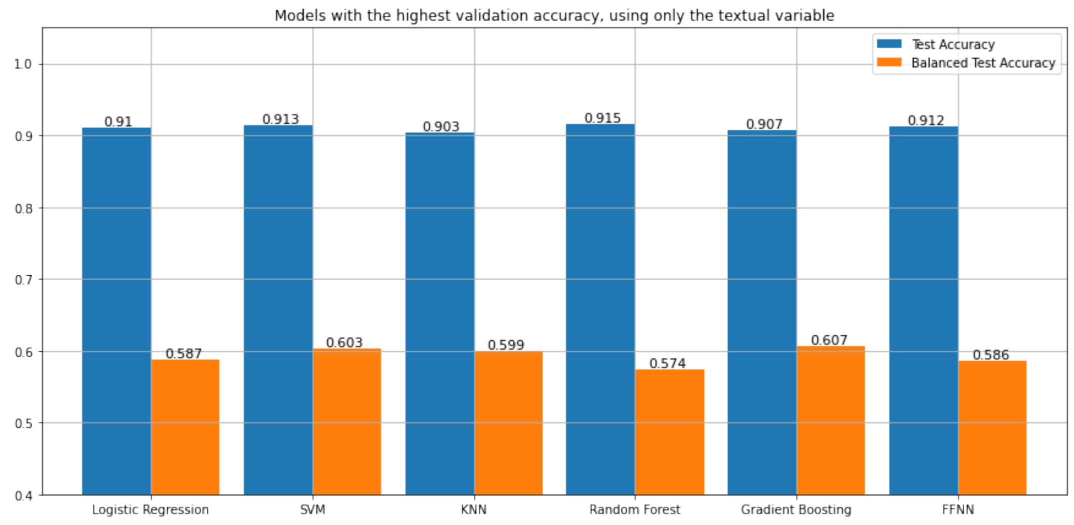


Figure 6.2: Models with the highest validation accuracy, using only the textual variable.

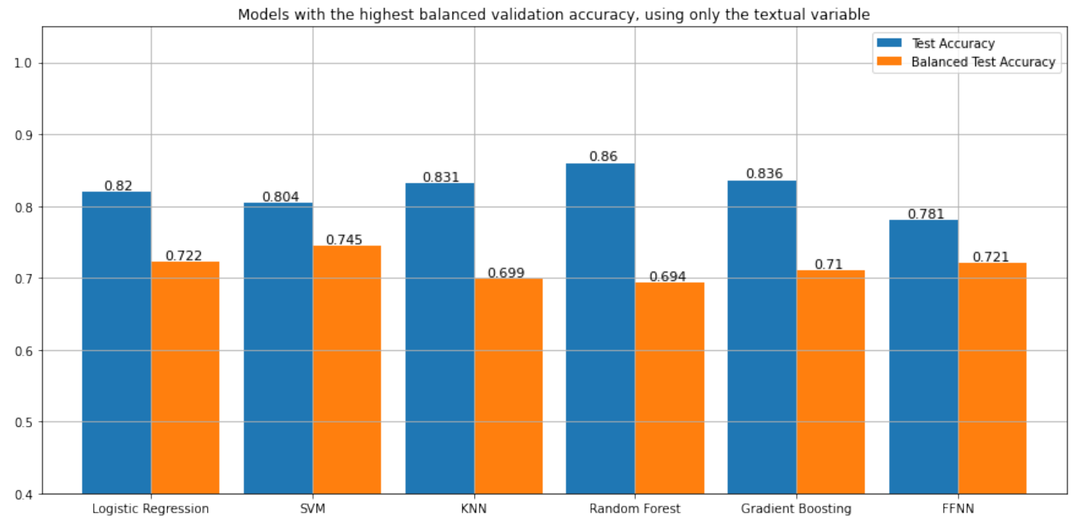


Figure 6.3: Models with the highest balanced validation accuracy, using only the textual variable.

From these results, it is possible to make the following observations:

- considering the results obtained using only the textual variable (Figure 6.2 and Figure 6.3), it is possible to conclude that the models are already able to learn to decently discriminate between the two classes. However, considering the models with the best validation accuracy, from the large number of false negatives and from the balanced accuracy close to 50%, it is clear that there is the tendency to classify the samples with the majority class label. Instead, the models with the best balanced accuracy in the validation set, inevitably have a lower accuracy in the test set;

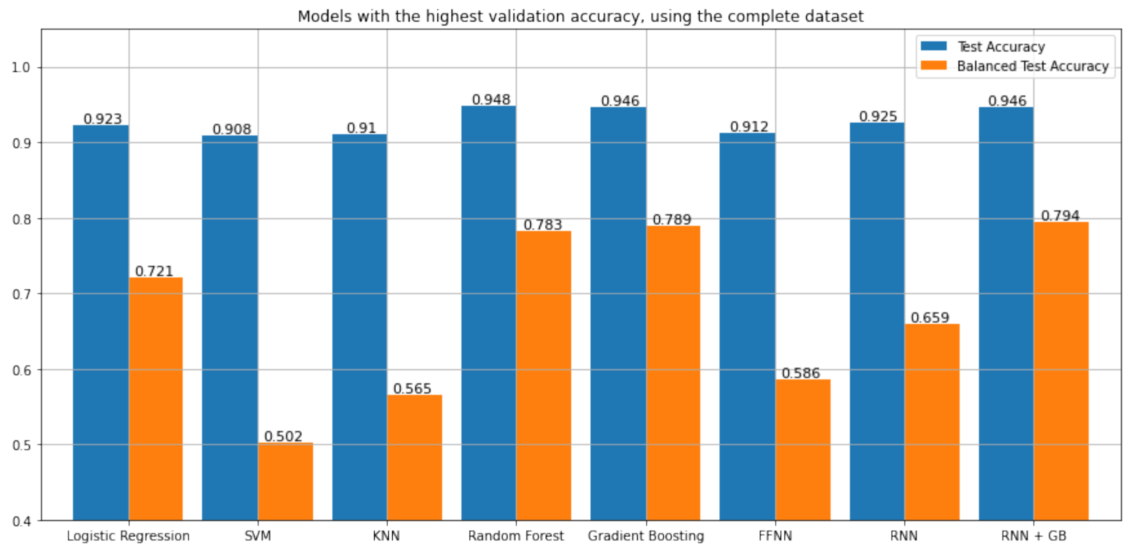


Figure 6.4: Models with the highest validation accuracy, using the complete dataset.

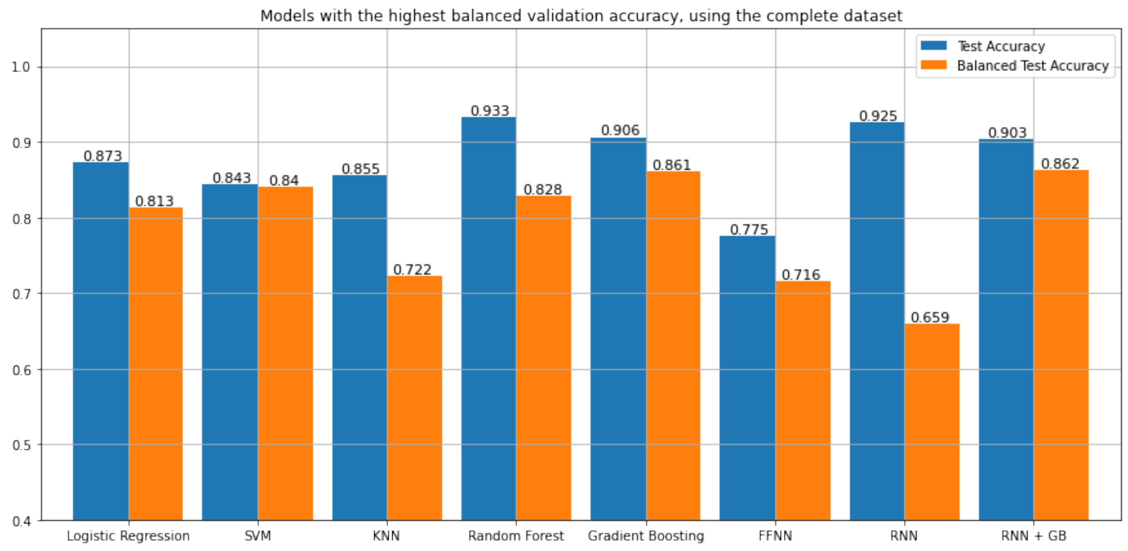


Figure 6.5: Models with the highest balanced validation accuracy, using the complete dataset.

- by comparing Figure 6.2 with Figure 6.4 and Figure 6.3 with Figure 6.5, it is evident that the inclusion of the other available variables bring additional information to the models which are able to exploit it and, therefore, to improve their performances. Indeed, in almost all cases, every classification metric considered shows a significant increment. For this reason, from this point on, all the attention is given to the models trained in the complete dataset.

- making a retrospective analysis on the tuning parameters selected by the optimization procedure (see the Appendix), it emerges that the models with a higher balanced accuracy are those exploiting the previously mentioned resampling techniques, while, in general, those with the higher accuracy consider the original dataset. Therefore, this gives an empirical proof of the effectiveness of these approaches in case of imbalanced datasets;

- it is interesting to notice that Logistic Regression is able to outperform KNN and even the more sophisticated Support Vector Machine, except when considering the models with the highest balanced accuracy. However, this is not completely unexpected, since in many cases, complex models perform worse than easier ones, since their own complexity may not be suitable for all situations, becoming even an obstacle;

- tree-based methods seem to be the most suitable approaches for this specific problem. In particular, Gradient Boosting is able to reach the highest balanced accuracy, 86.1%, and, with another combination of tuning parameters, the second highest accuracy, 94.6%, slightly lower than the one reached by Random Forest;

- the results obtained with the FFNN are not as high as expected. In particular, it is interesting to notice that they are significantly lower than those reached by Gradient Boosting, for instance. In Figure 6.6, the learning curves relative to the network with the highest validation accuracy are shown;

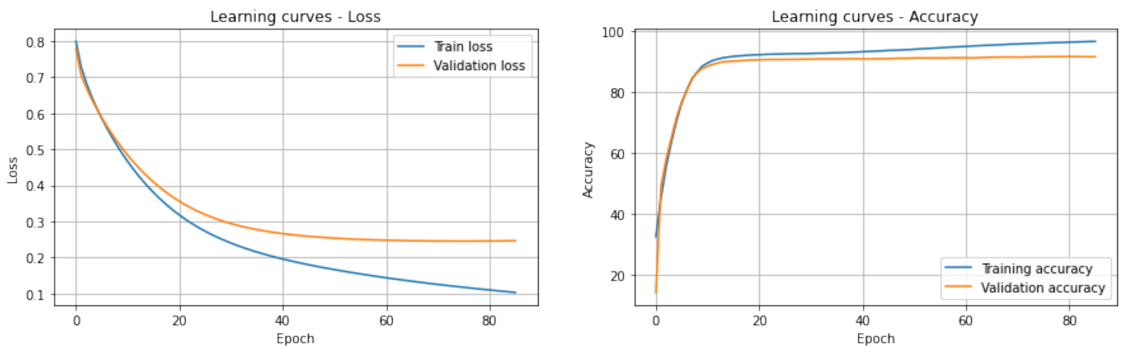


Figure 6.6: Learning curves of the FFNN model with the highest validation accuracy. On the left-hand side, the training and validation loss function values, while on the right-hand side, the training and validation accuracy, both as a function of the number of elapsed training epochs.

- the RNN was able to outperform not only the fully connected architectures, but also some other models such as KNN and SVM. Indeed, the accuracy reached in the test set was 0.925 and the balanced accuracy 0.659. However, Gradient boosting still offers much better results, confirming the fact that the deep learning approach may not be very suitable for this specific problem. This observation is underlined also by the high instability of the training phase observed despite all the measures adopted: indeed, a meticulous fine tuning was necessary to obtain learning convergence;

- a further experiment was carried out: the goal was to discover whether the performances of the best Gradient Boosting models can be improved exploiting the features automatically extracted by the RNN. After having trained and saved the neural network architecture, the description of each sample was propagated forward through the RNN layers, and the results combined with the other available variables. Then, the models have been trained on these new feature vectors. In some cases, this approach managed to improve the performances of Gradient Boosting, but only marginally. The corresponding results are reported in Figure 6.4 and Figure 6.5 with the label "RNN + GB". For ease of reading, they are also shown in Table 6.3.

	accuracy	*balanced accuracy*
Gradient Boosting (highest accuracy)	0.946	0.789
RNN + Gradient Boosting (highest accuracy)	0.946	0.794
Gradient Boosting (highest balanced accuracy)	0.906	0.861
RNN + Gradient Boosting (highest balanced accuracy)	0.903	0.862

Table 6.3: Comparison between Gradient Boosting results with and without the automatic feature extraction approach represented by RNN.

A possible conclusion is that the schematic and short sentences used in the textual descriptions do not justify the adoption of elaborated techniques such as LSTM, which would be more appropriate for sentences with a more articulated structure. For this reason, a much easier approach consisting in computing the term frequency matrix leads to comparable results, with a significantly lower computational complexity.

A more exhaustive list of results and all the classification metrics of the best models are reported in the Appendix.

In order to also have a rough idea of the computational cost of the considered models, both the fitting and predicting running time of the models with the highest validation accuracy were computed and reported in Table 6.4.

However, these results about the computation of the running time should be carefully interpreted. First of all, the values obtained may be inaccurate since for running the models, an online platform was used (see the Appendix), and the computational resources available may change from a runtime to another. Furthermore, there are some tuning parameters which have a relevant impact on the computational cost. In particular, those regarding the document frequency value range for a term to be kept, may lead to a matrix of token counts with a number of columns (stems) varying from few hundreds up to several thousands. For this reason, a model that in general is quick may become very slow, and vice versa.

	Fitting time (s)	Predicting time (s)	Total time (s)
Logistic Regression	295.0	0.5	295.5
SVM	195.0	102.0	297.0
KNN	0.2	100.0	100.2
Random Forest	44.9	3.1	48.0
Gradient Boosting	388.4	1.3	389.7
FFNN	581.1	15.2	596.3
RNN	202.9	16.5	219.4
RNN + GB	452.8	1.6	454.4

Table 6.4: Running time comparison among the versions of the models with the highest validation accuracy. This is done in terms of fitting time, predicting time and total time. The unit of measurement is the second.

Keeping in mind the previous considerations, it is still possible to make some general observations: FFNN is by far the slowest model. It is interesting to notice that the inclusion of some LSTM layers leads to better classification results and reduces the computation time. This can be explained by the fact that, in this second case, a smaller number of fully connected layers with less neurons are enough to achieve good results. As expected, KNN has an extremely small fitting time, since all the work is performed at predicting time. Remember that the training phase consists just in storing the training samples, while it is at fitting time that all pairwise distances are computed. Finally, Random Forest seems to be the fastest model in terms of total time, and this is remarkable considering that it was also one of those leading to the best classification metrics. To conclude, it should be underlined that for the problem at handle it is more important to have a low predicting time, as it will be explained later. The reason is that if the model is run at the moment of the insertion by the company employee, an immediate result is required. Instead, the training time is not a problem.

6.3 Dimensionality reduction

In this section, some additional experiments involving two dimensionality reduction approaches are reported. The purpose of this part of the analysis was to discover if the implementation of these techniques could reduce the problem complexity, and therefore also the training time, still guaranteeing good results.

The first method tested was Principal Component Analysis. The tuning parameter which allows to regulate the complexity level is the number of components to keep. Only Gradient Boosting was considered as classifier to be implemented in the reduced space. Indeed, the main focus was on how the number of components affects the performances. What emerged was that good performances are achievable with a limited reduction up to 100 or 500 components, while, with a smaller number of components the performances inevitably decrease. In Table 6.5, the results for the models performing better in the validation set are reported.

	number of components	*accuracy*	*balanced accuracy*
PCA + Gradient Boosting (highest accuracy)	500	0.935	0.720
PCA + Gradient Boosting (highest balanced accuracy)	500	0.831	0.849

Table 6.5: Results obtained by Gradient Boosting after the application of Principal Component Analysis.

For these models, it was not possible to conclude a relevant decrease in terms of fitting time. Instead, the total running time in some situations was even higher: for the model reaching the highest validation accuracy, the fitting time was equal to 461 seconds, which is higher compared to the 388 seconds required by Gradient Boosting alone. A possible explanation is that the number of components required to reach acceptable results is still high. Furthermore, on its own PCA requires a certain amount of time, especially during the training phase. Therefore, the advantage of training a classifier in a lower dimensional space is counterbalanced by the computational cost implied by PCA.

The experiments carried out using t-SNE were less significant. Firstly, it was used for visualizing the dataset in a three-dimensional space. In the resulting plot, the samples belonging to the two classes were mixed together. The only possible observation is that some clusters were identified in which only samples of the majority class were present.

Even though it is not the original purpose of t-SNE, also in this case some classifiers were trained on the reduced feature space, but the results were far from good due to the extremely high dimensionality reduction.

6.4 Final tests

After all the described experiments, gradient boosting without the addition of RNN-based feature extraction and without PCA was selected as the best model for the task of classifying job reports. Depending on whether the goal is to limit the number of false positives or of false negatives, one of the two tuning parameters' configurations identified (the one with highest accuracy and the one with the highest balanced accuracy in the validation set) should be considered.

To conclude the analysis, the model has been trained on the whole available dataset and one final test has been performed. To do so, a dataset containing new job reports inserted during the last two months has been retrieved from the company database. In particular, it contains 7159 recordings, but its class distribution is even more imbalanced than the original one. Indeed, 6972 samples belong to the approved class, while only 187 to the minority class. This unexpected change of distribution observed during the last months implies that the values of the metrics measuring the performances of the model are expected to be different compared to those regarding the previously described experiments. Indeed, it was noticed a slight increment of the classification accuracy, and, at the same time, a significant decrease of the balanced accuracy. The precise results are reported in Table 6.6.

	accuracy	*balanced accuracy*
Gradient Boosting (highest accuracy)	0.952	0.648
Gradient Boosting (highest balanced accuracy)	0.864	0.680

Table 6.6: Final results in the new test set.

At this stage, the point is to understand how these models can be exploited in practise. Two possible applications can be found. The first one consists in using the model at the moment of the insertion of the job report by the company employee, producing an immediate feedback regarding its correctness. In order to avoid annoying the employee, the model should be fast and with a low number of false positives. Furthermore, it should be already trained and ready to predict the outcome of a single recording. To make it possible, two objects should be saved and easily loadable: the trained model itself and the vectorizer used for the text encoding at training time.

The second moment in which the model could be applied is during the checking phase. In this case, a reliable model should avoid false negatives as much as possible. Indeed, if this is the case, the project managers assigned to perform the check may focus only on job reports predicted as positive, relying on the model prediction for the correct job reports. At the moment, this action is performed monthly, and this implies two observations: firstly, input data can be organized in batches, secondly, there are no restrictions regarding the speed of the process, that is not asked to be instantaneous.

An aspect to keep in mind is how frequent the model should be updated. This is crucially important because quite often new pieces of data are inserted into a job report, and the model would not know how to treat them. For instance, this happens when a new employee is assumed, when a new customer is introduced or, most importantly, when a new working activity starts, implying the usage of new job identifiers. A first solution consists in retraining the model after the final check performed by the project managers. In this way, once the information about the correct label is added, the test set of a month becomes part of the training set of the following one, which therefore is continually growing. Alternatively, considering that the entire procedure is sped up, the job report inspection might occur more frequently, for example at the end of each week. In this way, the model could be also updated more frequently, possibly improving its reliability.

6.5 Code availability

The code written for the described analysis and some additional produced files are available at https://github.com/DalZottoLuca/Master_thesis-Job_report_analysis.

7 Concluding remarks

This thesis work describes how to handle problems which include textual variables, starting from the theoretical fundamentals of text mining and learning models. Afterwards, an empirical perspective is assumed, reporting all the passages performed for the project faced during the internship. Its final goal was to classify the job reports inserted by the company employees as either correct or wrong: a procedure crucially important for billing purposes.

The analysis of the data generating process was extremely useful in order to obtain a full understanding about the meaning of the variables and the reasons behind certain insertions. With this knowledge, some preliminary steps were completed aiming to correct some formatting issues encountered after the data loading. In addition to that, an extensive exploration of the available variables revealed which ones were not useful for this specific analysis, because containing non-relevant information or just metadata included by the generating process.

The pre-processing phase has been probably the most relevant and delicate one, because of the number of steps necessary to extract useful information for a learning model. Depending on the data type, different actions have been performed. Great emphasis was given to the textual variable containing the activity description: after having showed how the cleaning phase has been conducted, the attention was on the encoding techniques.

Then, some resampling methods have been implemented as countermeasure for class imbalance, and many classifiers have been defined, optimized and evaluated. The model which obtained the best results was Gradient Boosting. Regarding the deep learning architectures, to improve the low performances of the feed forward fully connected neural networks, some

LSTM layers have been used as automatic feature extractors, exploiting the sequentiality of textual data. The resulting architecture managed to sensibly improve the performances of the network. However, Gradient Boosting was still able to produce better results. This fact is remarkable, but at the same time explainable: the complexity of neural networks does not guarantee excellent results in all possible situations. Indeed, sometimes a simple model is able to outperform a more complicated one, when the dataset or the problem at handle does not justify elaborated techniques. For instance, in this case, the textual variable does not present articulated sentences with well-defined dependencies in them, but extremely schematic ones. Therefore, a straightforward frequency-based encoding is enough to reach good results, in addition to being much faster.

Some additional experiments considering dimensionality reduction techniques have been carried out. However, these approaches were not so effective for the problem at handle. Indeed, besides being not able to improve the accuracy of the models, also the execution time did not decrease significantly. Anyway, time did not represent an issue for this problem, since the models should be re-trained only once in a while, to keep them updated about new values which can be assumed by the variables. For instance, it should be reasonably sufficient to plan this activity monthly or weekly.

At the end of the analysis, two models using Gradient Boosting have been selected: one for usage at insertion time, while the other for usage at checking time. This differentiation has been done according to the needs of the two phases. In the former one, it is preferable to have a model providing an higher accuracy, with a limited number of false positives, in order to have a fast preliminary check. Instead, in the latter case, the main concern is to avoid as much as possible false negatives, so that only the insertions predicted as wrong by the model should be checked by hand.

Finally, the code has been predisposed for the putting into production, by making a notebook with all essential steps for tuning and training, to be used for updating the model, and one aimed at computing predictions. Even though the models are not operating yet inside the company intranet, their effectiveness has been further tested on the job reports inserted in the last couple of months, showing satisfactory results.

For what concerns future developments of this analysis, once a larger dataset will be collected, not only the discriminative power of the methods is expected to increase, but also more complex models may be justified. Moreover, a possible extension of this project may consist in trying to construct a model able to suggest how to correct wrong job reports, providing, for example, the proper job identifier to insert.

Appendix A: implementation details

For the analysis, Python language has been chosen. In particular, the code has been written using Google Colaboratory, which is an online, free platform based on Jupyter, an Open Source project. There are at least two motivations behind this choice: first of all, the notebook format has been preferred since it allows to combine executable code and rich text in a single document in order to describe in depth some key passages, along with graphical representations. Besides, in this way, all outputs have been saved. Secondly, Colaboratory represents an alternative to a local code execution, providing some remote resources, including also GPUs, if needed. The only thing required to access and use it is a Google account. Remarkably, it is possible to point Colab to Google Drive, in order to load and save different kinds of files.

In what follows, some relevant aspects of the implementation are listed. For more details, it is possible to access the notebooks on the GitHub page mentioned in Section 6.5.

For handling the dataset and making some computations, the main libraries used were `pandas`, `NumPy` and `SciPy`. For the graphical representations, `matplotlib` and `seaborn` have been used.

Let's start with the tools used for the data exploration and pre-processing. The list of Italian and English stop words have been downloaded from the Python library, `stop_words`. To generate a list of characters to be removed, the library `string` has been exploited. Other operations regarding text cleaning have been completed using some standard Python functions and some utilities of the library `re`. The tool chosen to compute the stems one was `SnowballStemmer` from the library `nltk` (which stands for Natural Language Toolkit). Finally, to plot the wordcloud, the homonymous library was used.

For the encoding of the textual variable, two pre-defined functions of the `Scikit-learn` library have been exploited: `CountVectorizer` and `TfidfVectorizer`.

`Scikit-learn` was also used for implementing the classifiers. Instead, for the neural network architectures, the library `PyTorch` was used. In this second case, the following auxiliary pieces of code have been written before defining the model and the training loop:

- a class representing a dataset, which inherits the `PyTorch` data primitive `Dataset`, used for storing samples;

- a callable class implementing the necessary tensor transformation;
- a dataloader, based on the homonymous `PyTorch` module, which essentially is an iterator providing useful features, such as batching, shuffling and loading data in parallel using multiprocessing workers.

In addition to this, for the RNN models, some utility functions have been used for the additional per-processing steps described in Section 6.1, taken from the `PyTorch` package `rnn_utils`:

- `pad_sequence`;
- `pack_padded_sequence`;
- `pad_packed_sequence`.

Regarding the dimensionality reduction tools (PCA and t-SNE), again the Python library used was `sklearn`.

Finally, in order to save and load the necessary objects needed to run the model in practice, a possible option is to use the Python module `Pickle`. Indeed, it allows to serialize a machine learning algorithm and save this serialized format to a file. Later, this file can be loaded to deserialize the model and use it to make new predictions.

Appendix B: more about the results

In this section some additional details about the results obtained are reported. Firstly, the values selected for the tuning parameters at the end of the optimization procedure are reported. In particular, in Table 7.1, those corresponding to the classifiers reaching the highest validation accuracy are listed, while, in Table 7.2, those corresponding to the classifiers reaching the highest balanced validation accuracy are shown. In both cases, the dataset considered is the complete one (i.e., the one with all variables).

model	**tuning-parameter**	**value**
Logistic Regression	number of stems considered	3725
	n-gram range	$1-3$
	word vectorizer	countVectorized
	resampling approach	none
	inverse of the regularization coefficient	10
	maximum number of iterations	2048
SVM	number of stems considered	250
	n-gram range	$1-1$
	word vectorizer	TfidfVectorizer
	resampling approach	none
	regularization coefficient	0.01
	kernel type	linear
KNN	number of stems considered	2703
	n-gram range	$1-3$
	word vectorizer	countVectorizer
	resampling approach	none
	number of neighbors to use	5
	distance metric to use	minkowski
Random Forest	number of stems considered	500
	n-gram range	$1-2$
	word vectorizer	TfidfVectorizer
	resampling approach	over
	number of trees in the forest	200
	criterion used for the information gain	Gini
	maximum depth of the tree	100
	min number of samples to split an internal node	2
	min information gain to allow the split of a node	0
Gradient Boosting	number of stems considered	2703
	n-gram range	$1-2$
	word vectorizer	countVectorizer
	resampling approach	none
	number of boosting stages to perform	50
	the maximum depth of the individual estimators	50
	fraction of samples to be used	1

Table 7.1: List of tuning parameters and values used by the models with the highest validation accuracy.

model	**tuning-parameter**	**value**
Logistic Regression	number of stems considered	4935
	n-gram range	$1-2$
	word vectorizer	countVectorized
	resampling approach	mid strategy
	inverse of the regularization coefficient	1
	maximum number of iterations	8192
SVM	number of stems considered	2703
	n-gram range	$1-2$
	word vectorizer	TfidfVectorizer
	resampling approach	mid strategy
	regularization coefficient	0.1
	kernel type	linear
KNN	number of stems considered	4935
	n-gram range	$1-3$
	word vectorizer	TfidfVectorizer
	resampling approach	oversampling
	number of neighbors to use	3
	distance metric to use	minkowski
Random Forest	number of stems considered	3725
	n-gram range	$1-2$
	word vectorizer	TfidfVectorizer
	resampling approach	mid strategy
	number of trees in the forest	200
	criterion used for the information gain	Gini
	maximum depth of the tree	200
	min number of samples to split an internal node	2
	min information gain to allow the split of a node	0
Gradient Boosting	number of stems considered	4935
	n-gram range	$1-3$
	word vectorizer	countVectorizer
	resampling approach	mid strategy
	number of boosting stages to perform	100
	the maximum depth of the individual estimators	10
	fraction of samples to be used	1

Table 7.2: List of tuning parameters and values used by the models with the highest balanced validation accuracy.

In Table 7.3, the tuning parameters selected for the best neural network architectures are listed.

model	**tuning-parameter**	**value**
FFNN (highest accuracy)	number of stems considered	4935
	n-gram range	$1-1$
	word vectorizer	countVectorized
	resampling approach	none
	number of hidden layers	2
	hidden neurons	$2048-2048$
	activation	$GELU$
	optimizer	$RMSprop$
	learning rate	$1e-6$
	regularization	none
	number of training epoch	75
FFNN (highest balanced accuracy)	number of stems considered	4935
	n-gram range	$1-1$
	word vectorizer	TfidfVectorized
	resampling approach	mid strategy
	number of hidden layers	2
	hidden neurons	$256-256$
	activation	$GELU$
	optimizer	$RMSprop$
	learning rate	$1e-4$
	regularization	none
	number of training epoch	15
RNN (best architecture)	number of recurrent layers	3
	number of fully connected layers	3
	number of feature in the hidden state of the LSTM	128
	hidden neurons	$1024-1024-1024$
	activation	$GELU$
	optimizer	$Adam$
	learning rate	$1e-4$
	regularization	dropout
	number of training epoch	30

Table 7.3: List of the tuning parameters and values used by the best neural network architectures.

In Table 7.4, the classification metrics of the models reaching the highest validation accuracy on the complete dataset are reported.

	accuracy	*balanced accuracy*	*precision*	*sensitivity*	*specificity*	*MCC*	F_1*-score*
LR	0.923	0.721	0.606	0.474	0.969	0.495	0.532
SVM	0.908	0.502	0.995	0.005	0.995	0.067	0.010
KNN	0.911	0.565	0.564	0.141	0.989	0.251	0.226
RF	0.948	0.783	0.807	0.581	0.986	0.658	0.675
GB	0.946	0.789	0.767	0.596	0.982	0.648	0.67
FFNN	0.912	0.586	0.581	0.186	0.986	0.295	0.282
RNN	0.925	0.659	0.694	0.333	0.985	0.447	0.45
RNN + GB	0.946	0.794	0.763	0.608	0.981	0.653	0.677
PCA + GB	0.935	0.72	0.736	0.457	0.983	0.548	0.677

Table 7.4: Classification metrics of the models with the highest validation accuracy.

In Table 7.5, the classification metrics of the models reaching the highest balanced validation accuracy on the complete dataset are reported.

	accuracy	*balanced accuracy*	*precision*	*sensitivity*	*specificity*	*MCC*	F_1*-score*
LR	0.873	0.813	0.4	0.739	0.887	0.482	0.519
SVM	0.843	0.84	0.352	0.836	0.843	0.476	0.496
KNN	0.855	0.722	0.33	0.558	0.885	0.354	0.415
RF	0.933	0.828	0.624	0.7	0.957	0.624	0.66
GB	0.906	0.861	0.495	0.806	0.916	0.585	0.613
FFNN	0.775	0.716	0.237	0.643	0.789	0.288	0.346
RNN	0.925	0.659	0.694	0.333	0.985	0.447	0.45
RNN + GB	0.903	0.862	0.485	0.811	0.912	0.58	0.607
PCA + GB	0.831	0.849	0.339	0.871	0.827	0.476	0.489

Table 7.5: Classification metrics of the models with the highest balanced validation accuracy.

References

[1] Amazon AWS documentation (2021). Available at: https://aws.amazon.com/it/devops/what-is-devops (Accessed: 20 Nov 2021).

[2] Au T., Ma G. and Li S. (2003), *Applying and Evaluating Models to Predict Customer Attrition Using Data Mining Techniques*. Journal of Comparative International Management.

[3] Autentica Group web page (2021). Available at: https://www.autenticagroup.com (Accessed: 20 Nov 2021).

[4] Azzalini A. and Scarpa B. (2012), *Data Analysis and Data Mining: An Introduction*. Oxford University Press Inc.

[5] Ben-Gal I., Dana A., Shkolnik N. and Singer G. (2014), *Efficient Construction of Decision Trees by the Dual Information Distance Method*. Quality Technology & Quantitative Management. Vol. 11, No. 1, pp. 133-147.

[6] Bergstra J. and Bengio Y. (2012), *Random Search for Hyper-Parameter Optimization*. Journal of Machine Learning Research 13, 281-305.

[7] Bhageshpur K. (2019), *Data Is The New Oil – And That's A Good Thing*. Available at: https://www.forbes.com/sites/forbestechcouncil/2019/11/15/data-is-the-new-oil-and-thats-a-good-thing (Accessed: 14 Sep 2021).

[8] Bishop C. M. (2016), *Pattern Recognition and Machine Learning*. 1st ed. Springer.

[9] Blei D. M. (2012), *Probabilistic topic models*. Communications of the ACM, 55(4).

[10] Box G. E. P. (1979), *Robustness in the strategy of scientific model building*, in Launer R. L., Wilkinson G. N. (eds.), *Robustness in Statistics*, Academic Press.

[11] Bradley M. M. and Lang P. J. (1999), *Affective norms for English words (ANEW): Instruction manual and affective ratings*. Technical Report C-1, The Center for Research in Psychophysiology, University of Florida.

[12] Breiman L. (1997), *Arcing The Edge*. Technical Report 486. Statistics Department, University of California, Berkeley.

[13] Calin O. (2020), *Deep Learning Architectures: A Mathematical Approach*. 1st ed. Springer.

[14] Ceron A., Curini L. and Iacus, S. M. (2017), *Politics and Big Data Nowcasting and Forecasting Elections with Social Media*. 1st ed. Routledge.

[15] Christopher O. (2015), *Understanding LSTM Networks*. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs (Accessed: 26 Nov 2021).

[16] Collins English Dictionary (2021). Available at: https://www.collinsdictionary.com (Accessed: 15 Sep 2021).

[17] Cover T. M. and Hart P. E. (1967), *Nearest neighbor pattern classification*. IEEE transactions on information theory, VOL. IT-13, no. 1.

[18] De Wilde B. (2012), *Classification of Hand-written Digits*. Available at: https://bdewilde.github.io/blog/blogger/2012/10/26/classification-of-hand-written-digits-3/ (Accessed: 14 Sep 2021).

[19] desJardins M. and Gordon D. F. (1995), *Evaluation and Selection of Biases in Machine Learning*. Thomas G. Dietterich.

[20] Friedman J. (1999), *Greedy Function Approximation: A Gradient Boosting Machine*. IMS 1999 Reitz Lecture.

[21] Friedman J., Hastie T. and Tibshirani R. (2000), *Additive logistic regression: A statistical view of boosting (with discussion)*. Annals of Statistics, Vol. 28, No. 2, 337-407.

[22] Gareth J., Witten D., Hastie T. and Tibshirani R. (2015), *An Introduction to Statistical Learning*. Springer.

[23] Grimmer J. and Stewart B. M. (2013), *Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts*. Political Analysis, 21.

[24] Grippo L. and Sciandrone M. (2003), *Metodi di ottimizzazione per le reti neurali*. Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

[25] Hartshorn S. (2016), *Machine Learning With Random Forests And Decision Trees*. Seattle.

[26] Hochreiter S., Bengio Y., Frasconi P. and Schmidhuber J, (2001), *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*. In S. C. Kremer and J. F. Kolen, eds., A Field Guide to Dynamical Recurrent Neural Networks. IEEE press.

[27] Hopkins D. J. and King G. (2010), *A method of automated nonparametric content analysis for social science*. American Journal of Political Science, 54.

[28] Hsu C. W., Chang C. C. and Lin C. J. (2016), *A Practical guide to Support Vector Classification*. Department of Computer Science National Taiwan University, Taiwan.

[29] Kaufman S., Rosset S. and Perlich C. (2011), *Leakage in Data Mining: Formulation, Detection, and Avoidance*. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[30] Kumar A. (2015), *Machine Learning – How to Diagnose Underfitting/Overfitting of Learning Algorithm*. Available at: https://vitalflux.com/machine-learning-diagnose-underfittingoverfitting-learning-algorithm (Accessed: 26 Nov 2021).

[31] Kushal D., Lawrence S. and Pennock D. M. (2003), *Mining the peanut gallery: Opinion extraction and semantic classification of product reviews*. Proceedings of WWW 2003, Budapest, Hungary.

[32] Lebart, L., Salem, A and Berry, L. (1998), *Exploring Textual Data*. 1st ed. Springer.

[33] Liberman N. (2017), *Decision Trees and Random Forests*. Available at: https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991 (Accessed: 26 Nov 2021).

[34] Liu B. (1997), *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. 2nd ed. Springer.

[35] Longford N. T. (1987), *A fast scoring algorithm for maximum likelihood estimation in unbalanced mixed models with nested random effects*. Biometrika, 74.

[36] Maas A. L., Hannun A. Y. and Ng A. Y. (2014), *Rectifier Nonlinearities Improve Neural Network Acoustic Models*. Computer Science Department, Stanford University, CA 94305 USA.

[37] Mangasarian O. L. (1994), *Nonlinear Programming*. SIAM, Philadelphia.

[38] McCaffrey J. D. (2019), *The Difference Between Neural Network L2 Regularization and Weight Decay*. Available at: https://jamesmccaffrey.wordpress.com/2019/05/09/the-difference-between-neural-network-l2-regularization-and-weight-decay/ (Accessed: 26 Nov 2021).

[39] Miriade web page (2021). Available at: https://www.miriade.it (Accessed: 20 Nov 2021).

[40] Mitchell T. M. (1980), *The Need for Biases in Learning Generalizations*. Rutgers University, New Brunswick, NJ.

[41] Nesterov Y. and Nemirovskii A. (1994), *Interior Point Polynomial Methods in Convex Programming*. 1st ed. SIAM.

[42] pgAdmin web page (2021). Available at: https://www.pgadmin.org (Accessed: 20 Nov 2021).

[43] Postresql web page (2021). Available at: https://www.postgresql.org (Accessed: 20 Nov 2021).

[44] University of Padua (2021), *Data Science Internships*. Available at: https://datascience.math.unipd.it/internships (Accessed: 14 Sep 2021).

[45] van der Maaten L. and Hinton G. (2008), *Visualizing Data using t-SNE*. Journal of Machine Learning Research 9.

[46] VanderPlasp J. (2016), *The Python Data Science Handbook*. 1st ed. O'Reilly.

[47] Vapnik V. N. (1995), *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

[48] Vapnik V. N. (1998), *Statistical Learning Theory*. Wiley, New York.

[49] Wikipedia (2021), *Stemming page*. Available at: https://en.wikipedia.org/wiki/Stemming (Accessed: 15 Sep 2021).

[50] Worcester P. (2019), *A Comparison of Grid Search and Randomized Search Using Scikit Learn*. Available at: https://medium.com/@peterworcester_29377/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85 (Accessed: 25 Nov 2021).

[51] Zanasi A. (2009), *Virtual Weapons for Real Wars: Text Mining for National Security*. Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08. Advances in Soft Computing.

Acknowledgments

At the end of this dissertation, I would like to mention all the people, without whom this thesis work would not even exist.

First of all, I would like to express my gratitude towards my Supervisor, Prof. Bruno Scarpa, for his availability and for his precious suggestions, representing a guidance throughout this project.

I would like to thank all the staff of the company Miriade, for their hospitality and for the skills I acquired in the field. In particular, thanks to my Tutor Arianna Bellino who helped me to carry out my research.

A special thank to my family, especially to my parents and my sister for their endless support and for their closeness in the hardest moments.

Last but not least, thanks to my friends for all the light-hearted moments. Thanks to Daniele, Paolo, the high school friends and the university friends met along the way. Finally, thanks also to those who represented for me an inspiration.