UNIGE

PhD in Mathematics XXXIII Cycle

UNIVERSITÀ DEGLI STUDI
DI GENOVA

PhD Thesis

Customer Satisfaction: Data Driven Analysis

Supervisors

Prof. EMANUELA SASSO

PhD. ANNA CODISPOTI

Candidate

VALERIO PARODI

December 2021

Summary

The Project is based on a data analysis of a particular Company in order to increase the brand perception over clients.

To delimit the domain, we start with some highlights on customer experience and customer satisfaction.

The object of the study is the customer satisfaction measured by Net Promoter Score. The goal of the project is to understand which segments of the company have an impact on the brand perception, in particular on the NPS.

The available data are comment forms filled by clients: some questions are multiple choice questions, while others are free text questions.

Starting from this awareness, our work devotes an entire chapter to theory and techniques useful for our purposes. In particular, we want to give a solid background on free text analysis. We start from intuitive processes to reach sophisticated algorithms to achieve the goal.

The initial point is a classification from plain text: text preprocessing, feature extraction from text and linear model for sentiment analysis. Before starting with a language modelling, we introduce a simple deep learning for text classification. Then we describe N-gram language model and neural language model.

After the sentiment analysis, the main goal of free text analysis is the topic extraction. In order to achieve this goal we need to model the meaning of the words. This means studying word and sentence embeddings.Distributional semantic, explicit and implicit matrix factorization, Word2vec, word analogies are just some of the technique detailed.

Topic modelling means navigate through text collection and find the meaning of a particular sentence or paragraph or an entire document. The world of topic

modelling is an ensable of different approaches and algorithms. We chose to analyse PLSA and, in particular, LDA (Latend Dirichlet Allocation).

Another important part related to the knowledge necessary to tackle the problem is related to predictive models and feature engineering. An entire chapter is devoted to the feature importance and regression trees.

The last part of the document shows the project in detail, starting from real data. We divide the project in two parts.

The first analysis is based on multiple choice questions. The objective we have is to understand which question has more impact on the question related to NPS. After that, we want to estimate the variation on the NPS supposing to be able to change the result of a particular question (a particular outlet) through a business action. For this purpose we create a NPS-simulator: starting from a shift of some customer opinions, on a particular outlet, due to a business action, we want to estimate the NPS variation.

The second analysis is based on free text questions. Using methods, already seen, we want to extract sentiment from customers opinion and topics not considered in the multiple choice questions.

What we want to achieve is a consolidation of knowledge, through data-driven techniques, of what the company is already studying (multiple choice questions), but above all we want to understand what are the strengths to focus on or the weaknesses to work on that until now are not known (free text questions).

Acknowledgements

In addition to thanking my supervisors, I would like to remember my colleague Filippo Rossi with whom I collaborated on all the projects along the path, Silvio Lugaro for supporting me on a part of the work and Idil Ismiguzel for helping me write the thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

The Problem: Customer Satisfaction

1.1 Problem Statement

Although companies have the customer at the center of their interest, they do not focus on what the customer experiences throughout the duration of the relationship with the company. This type of study needs to be supported by data and models that are able to extract information from it. Within companies, data is sometimes collected, but most of the time it is not processed to collect information. *Bain Company* analyzed 362 companies with their customers. Those who have defined the experience as optimal are only 8%. On the contrary, 80% of companies think that their customers are totally satisfied. From this point of view, companies can have great margins for improvement.

The landscape in which companies are living is truly varied. The customer has a variety of proposals never seen until recently. These proposals are on the most diverse channels. What today can be a strength is to create a simple, integrated and personalized solution, where the company is increasingly specific to the individual customer. In addition, thanks to globalization, what guides a customer in their choice is the same whether the customer is near or far away. The methodologies are highly scalable.

Many companies now have a large amount of customer data at their disposal. This is because over the years it has been desired to verify customer satisfaction. In most cases, however, these data have not undergone any kind of proactive analysis. Within companies, it is still not entirely clear that measuring customer satisfaction does not help us understand how to improve it. Customer satisfaction is created as

a process of the interaction of each individual experience. Experience that starts from the first moment in which the customer knows the company, up to the moment in which he expresses his opinion. The judgment is expressed at the conclusion of the company-customer relationship. The company must realize that the customer experience is not just the result of the messages that the brand wants to convey.

It is essential that every process, every moment of customer-company interaction, must be managed in the best way.

1.1.1 Customer Experience

We can define the customer experience as any type of interaction that occurs between the customer and the company. Contact can be defined as direct or indirect. In both cases, the user forms, step by step, what will define their general satisfaction. Direct contact is usually created when the customer purchases a product/service: here customer establish a relationship. Indirect contact involves a greater effort on the part of the company. More effort is often required to be able to interact with the customer. This interaction must ensure that the client is not passive to the stimulus, but it is necessary to establish an active and proactive relationship.

Depending on the focus, the company must implement different strategies. Surely those who face a business-to-customer market must adopt different techniques from companies aiming at a business-to-business. For example, the latter cannot only target the end customer, that is, who will benefit from the good at the end. These companies must also be concerned about the experience of intermediary companies: they deal with different types of people who have different interests. The strategies to improve the user experience need to be diversified.

whatever the figure studied (company or customer), when we collect the data it is necessary to define key points. Experience data is collected in the so-called *touch points*. These represent any type of contact between company and customer, company and third party or customer and third party. The customer experience is then described as a series of touch points and is called *customer corridor*. Each touch point must be designed differently depending on the target customer. For example, a family with few resources and little time will need fast and integrated solutions, while an elderly person with a lot of time and a good amount of resources will need a very different experience.

Depending on the types of touch points, it is certainly useful to define priorities. All points of contact are not the same, but above all they will not have the

same importance. The touch points defined by a service will certainly be more important if the company offers a service as its main product. One of the most important points of contact is certainly the point of contact that leads the customer to a subsequent interaction. For example, the 1-click order for Amazon.

Customer satisfaction is created when there is a gap between expectations and experience within each touch point. For any kind of evaluation, people, sometimes even unconsciously, compare each new experience with previous experiences. Customer expectations are based on previous experiences with other companies or with the company itself and consequently satisfaction is defined. Expectations are also affected by market conditions, competitors and a person's purchasing power. A company, even if it has stable credibility, must understand that the customer is still predisposed to disappointment.

Below I want to propose an example to clarify the reasoning made in the previous sentences. *Gilead Sciences* is a good example to understand that it is easy to achieve failure if you do not want to understand the importance of experience and the component of expectations and dissatisfaction. This company wants to launch a new AIDS drug. *Gilead* launches a new drug after proving to be more beneficial than those on the market. As soon as it is introduced the company realizes that sales to new patients grow according to expectations, while old customers tend not to change their habits. For HIV / AIDS patients, changing medications means eliminating a relationship of trust in the hope of uncertain improvement. After a careful analysis of the customer, the company found that HIV-positive patients are much more interested in the potential adverse effects of a new drug than in its alleged superior efficacy. With this new awareness, the company has decided to emphasize in its marketing the lower incidence of serious side effects of the new drug. In addition, it has decided to direct the product towards doctors more inclined to innovations. As a result of these efforts to understand the customer the market share of the company's main competitor has dropped by 33%.

1.1.2 The importance of analyzing

Often companies, especially their CEOs, make choices in order to gather information and increase customer satisfaction. They try to analyze it and quantify it, but they often do not adequately appreciate the results that some analysis tools can give.

Within companies, there are certainly expensive software to manage the relationship with customers. For this reason, some CEOs think that customer satisfaction analysis is superfluous. It is necessary to make people understand the real value of this type of analysis. The problem lies not in the lack of data, but in an ignorance

of the importance of the data. Below we try to highlight the differences that exist between a CRM and an analysis of the customer experience. The CRM collects all the information that a company knows about a particular customer, such as his purchases from the company, returns, requests. The data we want to collect on the customer experience is more subjective. A CRM collects information after the fact has occurred, while a CEM (customer experience management) collects an immediate customer report after the customer company interaction. Customer experience and customer relationships differ on several aspects. Chief among these are object, timing, monitoring, audience and purpose. Moreover, companies often do not fully take customer satisfaction into account. They think they have already studied sufficiently and do not understand the difference with the customer experience that always requires new research.

It certainly makes a lot of difference whether the company's CEO have had experiences in contact with customers or have not had contact. The first usually act with more regard to the customer experience. First of all, he focuses on internal choices based on customer satisfaction and then begins to take an interest in competitors. On the other hand, who has never had a role in contact with the customer often considers all this effort as superfluous or even as a waste of resources. He also thinks that all of this is solely the responsibility of sales, marketing or customer service.

Often within companies it is stated that their business is customer-driven. Few times, however, there are data that confirm this. This can be considered by some to be a contradiction. Let us assume that the data begins to enter the company. From here the first problems begin. Surely every customer has their own needs. Often there are totally discordant requests. What choices will the company have to make? In addition, the company cannot always afford certain actions. In this scenario, many leaders who are used to working with costs and revenues (certainty) are not prepared to work in compromise and divergence. Executives rightly hesitate to make decisions if the data is ambiguous. This is precisely where the entire study of data analysis, statistics and artificial intelligence comes to support. With these techniques we try to reliably quantify the main needs, the critical points on which to intervene and the strengths on which to invest. Starting a decade ago, most companies started massive data collection. The data is collected electronically and describes the user experience. It is now necessary to start analyzing them and extract useful information. Individual responses can be analyzed and aggregated in real time in order to monitor possible problems and even solve them in real time and in an automated way.

One of the biggest obstacles limiting the growth of data analysis is the difficulty in understanding the process of producing the results. Often being able to interpret means giving up precision. However, models are emerging in which we try to reach acceptable compromises for those who work on the business side. An example to be able to gain interpretability and partially stem the problem of lower precision in comparison of complex Machine Learning algorithms. In [1] an interesting approach is illustrated. By training various complex machine learning models with strong predictive capabilities and averaging estimated probabilities, a robust model can be obtained. The latter can be used to supervised the training of a simpler decision tree model. At this point the most influential features can be extracted without losing much of the accuracy. Another interesting aspect is that a model of this type can therefore be used in production systems being extremely simple and fast.

1.1.3 Competition and Customer Satisfaction

Nowadays, companies find themselves in an increasingly fierce market, they always and in every way try to differentiate themselves from competitors in order to maintain a relationship with their customers. This particular customer interest has created a paradigm shift from transactional marketing to relationship marketing [2, 3], many articles consider satisfaction as the essential principle for customer retention and customer satisfaction has moved to the head of relationship marketing approaches [4, 5]. In this scenario it is easy to imagine how companies can invest huge resources with the aim of increasing customer satisfaction. Customer satisfaction costs account for the majority of a company's annual [6] marketing budget. The costs make up about 50% of the total marketing costs [7]. An interesting aspect to note is that many companies have begun to evaluate customer satisfaction compared to their financial results. A sector that is very interested in customer satisfaction is certainly the tourism one. Indeed, in tourism sector, various studies highlight the importance of customer satisfaction in terms of the performance of a company [8, 9].

1.1.4 Net Promoter Score

In recent years, companies find themselves in increasingly competitive markets. They try in every way to meet customer needs to attract new people and retain those who have already had experience with the company. There are two types of customer communication. The first method of communication is of the direct type. Here the company creates advertising campaigns, promotions in stores, public relations activities. The second method of company-customer interaction is interactive, it requires feedback from the customer. Usually this type has more

limited tools, but without doubt we get very valuable information about customer feedback. For example, a voluntary feedback system can be implemented after the use of a service, structured surveys can be created to be submitted to loyal customers. With this methodology, the customer can highlight their needs to the company and the company is more aware of the customer's experience. All this can certainly direct the company towards targeted actions to improve customer experience and satisfaction.

Business consultant Fred Reicheld [10, 11] gives us a single question as the best and sufficient measure of customer satisfaction.

How likely is it that you would recommend [brand or company X] to a friend or colleague?

and they can answer by choosing a number between 0 to 10, with 0 that means *not at all likely*, 5 means *neutral* and 10 means *extremely likely*. The answers are aggregated into a single summary value, the **Net-Promoter Score** (**NPS**). A company's Net-Promoter Score is the difference between the number of customers that answers 9 or 10 (called **promoters**) and the number of customers that answer a value between 0 and 6 (called **detractors**). Respondents on values 7 and 8 are called **neutrals**'.

$$NPS = \frac{\#Promoters - \#Detractot}{\#Promoter + \#Neutral + \#Detractor}$$

Reichheld argues that the Net-Promoter question is all a company needs to know if it wants to have an overview of the satisfaction of its customers. We may also add a follow-up question to understand the reasons for the answer [10]. The author and his collaborators believe that a person's propensity to recommend the company to friends is the main indicator for understanding a customer's loyalty. NPS helps companies understand their relationship with the customer. This is the *ultimate question* [11], *the only number you need to grow* [10] to see the company grow.

In recent years, the analysis of customer satisfaction using machine learning techniques has been very successful. There are many examples that have created value within this domain. In [12] the authors approached the NPS prediction problem as a multiclass classification problem starting from different KPIs. For this specific problem, different types of algorithms have been considered: CNN, ANN, logistic regression, SVM, random forest, naïve Bayes, decision trees and k-NN. The peculiarity of the project is to evaluate the best performing algorithms

for a specific problem through the confusion matrix. Unfortunately we know that latter can only be used for a binary classification, while NPS needs a multiclass classification. The authors therefore proposed a method that reduces the size of a multiclass confusion matrix based on the class grouping process. The results of the method include the definition of a ROC, AUC for a reduced confusion matrix. These show us that a CNN and ANN models surpass the other models, although the random forest occupies an important position. In fact, we will see in the next chapters that the choice of the model for the thesis project will fall precisely on the random forest: we lose some precision in exchange for interpretability.

1.2 Research objective

In the following section, taking in consideration the problem statement, we want to introduce the goal of the project introducing the available data.

Immediately after, we summarize the strengths and weaknesses of the type of data taken into consideration.

1.2.1 The goal

The customer should be the fulcrum of the business decisions in a company. Understanding customer experience and acting in the direction of customer satisfaction means increase company revenues. The purpose of this project is to better understand the customer, collect data and analyze it in order to have a data-driven and more objective approach in choosing business actions.

As we have said in the previous sections we use NPS to summarize the client perception of the brand. The customer position in relation with NPS is the summary of an experiences that includes different interactions client-brand: we call it *touch point*. Each of this *touch points* can influence the overall brand perception (NPS in our case) in different way. They can have:

- positive influence,
- negative influence,
- no influence.

This leverage exist with both positive than negative singular experience. Not necessarely a positive answer on a *touch point* will bring a positive influence on NPS. Furthermore, even if we suppose to know the type of leverage, we do not know which weight will have on the overall perception.

In this work, what we are looking for is to understand which outlets influence NPS. The goal is to measure the leverage of a singular business action on Net Promoter Score.

1.2.2 Available data

The communication channel customer-company, and not only company-customer, plays a role of extreme importance if we want to increase the perception of a brand. There are different way to collect data from customers. In this work we consider as unique way of communication a comment form. The latter is compiled at the end of the complete experience with the brand. It is administered in electronic format and it is composed by 90 multiple choice questions, 3 free text questions and a single question which contributes to the creation of NPS.

$Q1_{Multiple\ Chioce}$
$Q2_{Multiple\ Chioce}$
...
$Q90_{Multiple\ Chioce}$
$Q1_{Free\ Text}$
$Q2_{Free\ Text}$
$Q3_{Free\ Text}$
Q_{NPS}

Figure 1.1: Comment Form

Starting from multiple choice questions and free text questions we want to catch the value of single questions in relation with NPS.

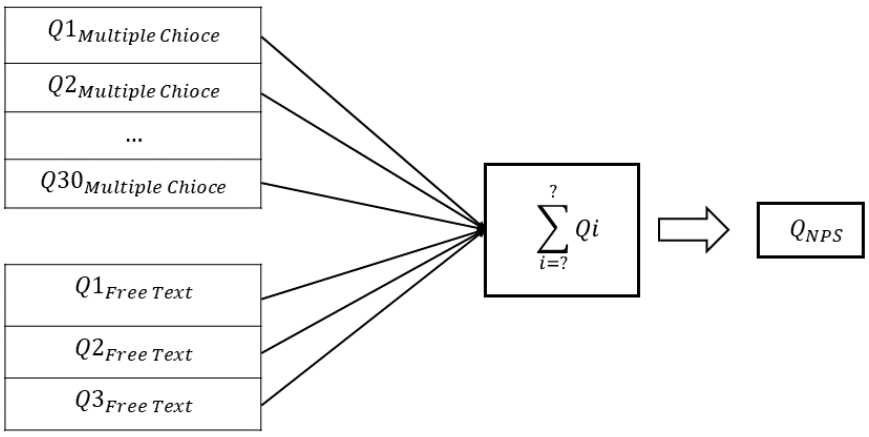

Figure 1.2: Questions and NPS

We have 15.000 comment form available to be analyzed, in particular we have 1.350.000 multiple-choice questions and 45.000 free-text questions. The comment forms are distributed over a period of two years, include an range of ages between 18 and 90 years and the gender is distributed almost homogeneously.

In the following paragraphs we will analyze in detail the structure of the questions.

1.2.3 Multiple-choice questions

Multiple choice questions are the basis for creating a questionnaire. First of all because they are easy to read and quick to answer. Then they are also simpler in the data analysis phase. These types of questions have their own tabular structure which makes a first study extremely fast. All this is possible thanks to the extremely rigid structure. The data that is collected, however, is limited by the options that are provided to those who respond. This necessarily leads to introducing some bias into what we are trying to find.

It is true that this type of questionnaire is very simple to interpret. But it is not so easy to understand what kind of question to ask. There are several types. For example, the first step is to understand whether to leave the possibility of selecting more than one answer.

Suppose we leave the possibility of selecting only one answer from a set of choices. This type is the most used in the questionnaires. Now we can insert binary answers, or answers with a rating or even answers with scales.

We must definitely not forget those questions to which more than one answer can be selected. Depending on what we are looking for, this type of question may be the best. Multiple choice questions have a slightly different purpose than single choice questions. For example we can ask *What is your favorite bar?*, or we can ask *Which of the following bars do you like?*

In our comment form we use just single-answer questions as in 4.12. with a nominal scale from 1 to 10. Where 1 means *I am not satisfied at all* while 10 means*I am absolutely satisfied*.

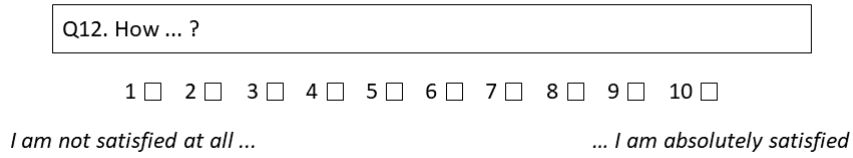

Figure 1.3: Multiple choice question

Multiple choice questions are certainly the ones that have been most successful. Below we list a number of advantages that have some relevance for both those who answer the questions and those who analyze the data of the answers.

- It is certainly easier for the respondent to select an answer than to write. The faster the questionnaire is, the more responses we can collect.
- We can have answers without errors, therefore a clean data to be analyzed.
- Multiple choices help the interviewee to understand how he should respond. We may need answers with different levels of details.
- The interviewee provides deeper levels of detail because he is guided.

We can say that multiple choice questions almost always represent most of the questions within the survey. These are useful in a huge range of situations.

In the next section we will see that where the limit of this type of question begins, free text questions come in help. These questions that have their strengths and weaknesses.

1.2.4 Free-text questions

At the end of the questionnaires it is very common to find an open question to try to retrieve information that had not been thought of. Unfortunately, however, it

is equally common that these questions are not analyzed. This happens because it is much more complicated to analyze these types of questions than multiple choice questions. The project that will be illustrated within the thesis also aims to analyze the free text questions. These questions are important mainly for two reasons: there is no baias brought by those who choose multiple answers and it is possible to analyze issues that were not thought of during the drafting of the questions.

The free text answers can reveal to us strengths and weaknesses that we could not think of. In addition they could bring some answers that it is not possible to define for example with a *Likert Scale*. The customer has the opportunity to express more freely his opinion and satisfaction with the services received.

The importance and potential of free-text questions have always been recognized, but little used due to the complexity of the analysis, especially when the number of questionnaires to be analyzed becomes relevant and difficult to interpret by people.

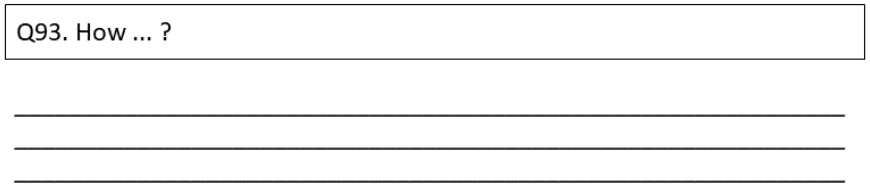

Figure 1.4: Free text question

In the comment form collected for our work, we have three free text question. In particular one question is about the brand overall perception and general suggestions. The others want to discover two *touch points* of customer experience not yet well explored and known.

We have highlighted many advantages of this type of questions, but the difficulties we will encounter when we are going to analyze the data to extract information should not be overlooked. For this reason, the following chapter will be entirely devoted to the study of free-text analysis techniques.

Many project are developed in order to understand meaning in free text and free text associated to NPS score. As one example, in [13] a new type of classifier that is able to manage verbatim text is explained. None of the best performing classifiers can be used for applications such as text classification to obtain NPS because we need a model that uses the ordering information of the classes. In this

paper, the authors use a new methodology called Ordinal Hyperplane Loss Network (OHPLnet) designed specifically for data with ordinal classes. They created a neural network with a loss function to estimate a non-linear mapping of the data, in a space that not only separates the classes but also maintains the order of the classes.

Chapter 2

Free Text Analysis as Background

In this chapter we will talk about NLP: natural language processing. NLP is a set of techniques useful for managing textual analysis.

Processes that use NLP are everywhere. We can think of search engines, automated translations, suggestions in emails or automated spam management. These techniques are also widely used nowadays in dialogue systems called chatbots. Chatbots can be useful for browsing a site, for assistance on a banking platform, for paying an electricity bill or even for managing home automation. These systems are designed to automatically manage a human-like conversation.

In our case, with this technology, we can understand what a person wants the moment they fill out a free text comment form.

The methods of natural language processing can be divided into three broad categories: deterministic methods, traditional learning methods and deep learning methods. The first method includes all those methodologies that define exact rules for analyzing texts, for example regular expressions. The latter method, on the other hand, is becoming more and more popular because it is able to achieve surprising results and the hardware technology that is emerging allows the management of these software at low cost. For the rest of this chapter, basic knowledge of machine learning and deep learning is considered to be acquired.

The first method highlighted, that is the deterministic method, involves defining all the rules that can be inserted in a text. From lexical rules to grammar rules. There must be close cooperation between a linguist and a software developer. It is

certainly a time-consuming, very complex and not very effective process. A little precision and a little generalization can be highlighted.

The second approach is based on automated learning. Here it is necessary to have a good amount of data where it is possible to train and learn a whole series of undefined rules. The needs in this case are to define the right model, choose the most suitable parameters and, after the training phase, apply the model.

The idea just described is also the same for deep learning techniques. In this case the models used are deep neural networks according to the needs. The data that feeds these models are words. As we will see later each word will be encoded in a vector. These vectors are built with a particular technique suitable for understanding the meaning of words. These neural networks are often very complex and have a high amount of parameters to estimate, but they work exceptionally in a wide variety of problems

It is also good not to forget the deterministic methods, used until recently as the main methods. This is because in some particular problem they may be useful rather than complex methods both from a computational point of view and from an interpretation point of view. In addition, these are the basis of the most modern techniques.

The first section 2.1 focuses on text classification. Classifying the text means starting from the words to insert a certain text into a category. This aspect is certainly one of the simplest, but at the same time it solves a great deal of current problems. We can define if an email is to be put in spam, we can understand if a comment is positive or negative, we can understand the sentiment that is expressed in that comment, we can redirect a generic request to the right department in a company. All this can be manage in an automated way starting from training data.

In the second section 2.2 we look at the text from a new point of view. If before we looked at the text as a set of words, now we look at it as a sequence of words. A first technique to achieve this is through linguistic modeling. We calculate the probability that a word can follow a given word. This approach support automated text generation. For example, if we develop an algorithm for the translation, starting from a French sentence we have to generate a text in English.

In the section 2.4 we look for a new representation of the words. We try to represent words in such a way that they can be managed by a mathematical model and in such a way that it can be possible to assign a precise meaning. Better to

say, let us try to create a way to define a similarity between words. We define a word as a vector of approximately 300 size. Words with similar meanings will have similar vectors in terms of cosine similarity. For example *bread* and *focaccia* will be very similar.

In the 2.5 section we see the topic modeling. We analyze different types of models that derive topics from a given text. For example, we can classify different types of text starting from a large amount of data: it is possible to reorganize and hierarchize something that would otherwise be confusing and difficult to interpret.

2.1 Text Classification Problem

This section presents an overview of text classification problems and a set of possible techniques for solving them. The classic example is sentiment analysis. Here we want to understand if a comment is positive or negative. We could also make a more detailed classification by entering a score from 1 to 5. For a human being, distinguishing whether a sentence is positive or negative is absolutely simple and intuitive. For example, *the restaurant is very nice, very kind and helpful service* is definitely a positive sentence, while *the restaurant was far from my expectations, the courses were cold* it is easy to guess it is a negative evaluation. This is all very complicated for a computer. In the following lines we will see some possible solutions. In order to achieve this goal we will start with text preprocessing.

2.1.1 Data Preparation

Let us start by giving a definition of *text*. We can certainly think of the text as an ordered set of words. A text can also be seen as a set of characters, a set of sentences or a set of paragraphs. For our study we consider the text as a whole, a sequence of words. So we consider the word as the smallest element that makes up our data.

We take the word as a basic element because within a text it is easy to isolate it. This is true for example for the Italian language where words are split by a space. This approach is much more difficult in German where some words are written close together without spaces, or in Chinese, where even here the ideograms identify several words or a word is composed of several ideograms.

Suppose we work in a language such as Italian or English where each word is isolated from the others by spaces. The first step is to divide the text into its basic components: the words. This process is called **Tokenization** and each component is called a token. A token therefore in our case will be a word, an element on which

there will then be a semantic processing.

Valerio's sister eats an apple a day!
Valerio's | sister | eats | an | apple | a | day!

We can count seven tokens. A first question arises immediately. The last token includes an exclamation mark. This does not change the meaning of the word. However, the two tokens are still different because they are written differently. We are therefore introducing a new rule. The subdivision is not done only with space but also with punctuation.

Valerio's | sister | eats | an | apple | a | day | !

The problems are not over. There is an apostrophe and *s* that we would like to be separated, in order to be able to isolate single words. At this point it is necessary to insert a series of rules that take into account the grammar of the language we are analyzing. Below we can see a subdivision, or rather a tokenization, which respects what we want.

Valerio | '| s | sister | eat | a | apple | a | day | !

Once the subdivision into words has been carried out, we must standardize the different declensions or conjugations we are considering. We can say that we want to normalize the tokens. For example we are looking for the same form of two words like *mouse* and *mice*. **Normalization** is a technique that allows us to bring two different words with the same meaning to the same form.

Mouse → Mouse
Mice → Mouse

The process just seen can be approached with two different techniques. The first is **Stemming**, that is, we remove or replace the suffixes to get to the root of the word. The root is called *stem*. To obtain this result, we reference to a series of typical rules for each language.

Wolf → Wolf
Wolves → Wolv
Cats → Cat
Feet → Feet
Foot → Foot

The second technique is **Lemmatization**, which is the process of obtaining the basic form of the word. To achieve this, it is necessary to use a dictionary, also in this case typical for each language. The dictionary form is called *lemma*.

$$\text{Wolf} \rightarrow \text{Wolf}$$
$$\text{Wolves} \rightarrow \text{Wolf}$$
$$\text{Cats} \rightarrow \text{Cat}$$
$$\text{Feet} \rightarrow \text{Foot}$$
$$\text{Foot} \rightarrow \text{Foot}$$

To sum up, we see a text as a set of tokens. These tokens are words even though we can think of them as characters or as whole sentences. From these we extract the significant part through stemming and lemmatization.

In the following sections we will transform the extracted and processed tokens into inputs for our model.

2.1.2 From text to Features

In this section we highlight a possible technique for transforming text into useful variables for a predictive model. We can more precisely say that we transform tokens into features. The method we see now is called **bag of words**. This technique takes into account the presence of a certain word and counts how many times it is repeated. With this method we search for words that can be useful, for example, to understand the sentiment of a comment (*good* or *bad*). Let us start with an example. Below we list several parts of sentences extrapolated from reviews. *good travel*, *not a good travel*, *did not like it*. At this point we can create a matrix of the words. Each different token we encounter creates a new column of the matrix. Each sentence is a row of the matrix. For the first sentence we will put 1 under the *good* column and in the *travel* column. In the columns relating to words not present in the particular comment we insert the value 0. We thus begin to create a mathematical structure starting from the text. This is a first technique for taking into account words and individual sentences. This technique is called **text vectorization** because starting from a dataset of sentences we create a series of vectors passing through the tokens. It should be borne in mind that this matrix has a very large size and is very sparse, that is, it contains many zeros. The problems associated with this type of representation are many. One of the most important is the loss of order between words: different phrases correspond to the same type of representation. In fact *bag of words* is the name of this representation. A solution to this problem is to create new columns of the matrix. These new columns represent pairs of tokens, triplets of tokens, or other combinations of tokens. This technique is called **n-gram**

	good	**travel**	**did**	**not**	**like**	**a**
good travel	1	1	0	0	0	0
not a good travel	1	1	0	1	0	1
did not like	0	0	1	1	1	0

Table 2.1: Bag of Words

extraction. *1-gram* stands for token, *2-gram* stands for a pair of tokens, and so on. In this way, however, we are going to further increase the size and sparsity of the matrix. A further problem is the scale of the counters. A normalization would therefore be necessary. The technique highlighted above allows us to take into

	good travel	**travel**	**did not**	**a**	...
good travel	1	1	0	0	...
not a good travel	1	1	0	1	...
did not like	0	0	1	0	...

Table 2.2: n-grams

account the order of the words. The number of features at this point is very high. For example, if our dataset contained 50000 words and we also wanted to take *3-grams* into account, we would have a matrix of about $50000 + 50000^2 + 50000^3$. There are techniques to decrease dimension without impacting the final result. For example, we could choose to delete some *n-grams* based on the frequency with which they appear in the dataset. We can eliminate high frequency or low frequency *n-grams*. High or low in reference to the mean. High-frequency *n-grams* are usually articles, prepositions, etc. which are not useful for prediction purposes, but only for grammatical purposes. These words are called *stop-words*. If we think about low frequency *n-grams* we could find for example typos. It is important that these words are eliminated because they could be an obstacle to the predictive process and partially invalidate the result. Unlike the high frequency values, it is useful to eliminate these latter values in order to improve the final result.

Once we have deleted *stop-words* or *typos*, what remains are the medium frequency *n-grams*. These are the values that bring information about the text. The filtration just done is not very dense, so we still have to deal with a lot of *n-grams*. It is necessary to decide which of these can bring the most information. We need to understand which values are discriminating to understand the meaning of the sentence. We can understand that lower frequency *n-gram* can be more discriminating for capturing the problem of a specific sentence. To carry this idea forward, some definitions are needed. **Term Frequency**, which we abbreviate as **TF**, is defined

as the frequency of a certain **t** term, i.e. a token or a generic **n-gram** present in a document *d*. The different metrics are shown in the 2.3 table. The metric we

Weighting scheme	**TF weight**
Binary	0,1
Row count	$f_{t,d}$
Term frequency	$f_{t,d}/\sum_{t' \in d} f_{t',d}$
Log normalization	$1 + \log(f_{t,d})$

Table 2.3: Term frequency

see first is a binary type metric. The value varies between 1 and 0 depending on whether *n-gram* is present or absent. If we want to refine the metric we can count the number of times a term appears in the text. What we use to carry out our study is a normalized count and we create a probability distribution. We count all the terms in the document and normalize them to have the integral (in this case the sum) equal to 1. A further step can be done by introducing logarithmic normalization.

A further definition that we must know is the **inverse document frequency** abbreviated with **IDF**. With this value we want to quantify the documents of the corpus that contain a specific term. Changing the point of view, we can think about the frequency of the documents and we can take the number of documents in which the term is present and divide it by the number of documents. What interests us is to evaluate the frequency of the inverse document and then we exchange the values and take into account the logarithm. We thus obtain the following formula.

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Where the total cardinality of the documents in the corpus is N, D is the corpus, t the term and d the document.

Now we can define what interests us: **TF-IDF**. We multiply the term frequency by the inverse document frequency of a given term in all documents in the corpus.

$$tfidf(t, d, D) = tf(t, d)\dot{i}df(t, D)$$

This value is very useful because what interests us is a high frequency of terms within the document which simultaneously corresponds to a low frequency of the same terms in the corpus. *TF-IDF* allows us to understand this when it takes on a high value.

Starting from *bag of words*, we substitute a new metric for the counters: *TF-IDF*. A further step that needs to be done is the normalization on each row to create a probability distribution that we will see will be very useful in the following sections. For example we can use the L_2 norm. Returning to the example, in the 2.4 table we can see what we have achieved. To sum up, we can say that we have

	good travel	**travel**	**did not**	**a**	...
good travel	1	1	0	0	...
not a good travel	1	1	0	1	...
did not like	0	0	1	0	...

Table 2.4: TF-IDF

created a mathematical structure starting from free text through *bag of words*, using the counter. To preserve the word order we added *n-grams*. We have replaced the counters with *TF-IDF* to get better results in the predictive models that we will see later.

2.1.3 Linear Classification Models

With the knowledge gained in the previous section we tackle the problem of free text classification. As an example, let us continue to work on sentiment analysis. Therefore the input data are reviews. A review is usually accompanied by a numerical rating: usually a number of stars. We try to make a classification that is not limited only to positive or negative, but we try to assign a score from 1 to 10. We can assume that 7,8,9 and 10 correspond to a positive comment and below 5 the comment is defined as negative. To tackle this problem we use reviews of hotels, restaurants or generic locations because it is very easy to create an online dataset. This dataset needs free text associated with a numerical evaluation. A fair amount of data is required for the creation of such models, that is, a large amount of reviews possibly coming from many people, many different locations. Otherwise we could create a model with a lot of overfit.

One of the simplest and most interpretable models is **logistic regression**. We have text as input data. The text is preprocessed to create *n-grams*, a *bag of word* representation is created by entering *TF-IDF* values. Now let us try to create a forecast starting from the features created. We can use a *sigmoid activation function*. It is easy to train, handles sparse matrices very well and above all it is easy to interpret. The main limitation is that it only handles linear relationships.

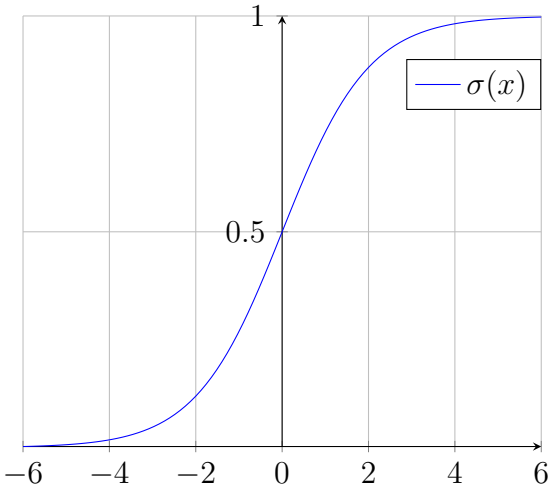

Figure 2.1: Sigmoid function

This model is easy to interpret because its operations is absolutely transparent. If we start from the graph 2.1 we can see that a combination of features close to 0 will return values close to 0.5, so we cannot know if the comment is positive or negative. If we move away from zero upwards, the probability that the comment is positive increases. Moving away from zero towards the top means that our combination of features produces positive values ever closer to 1. Conversely, we move away from 0.5 when my combination of features produces negative values close to -1. This means that comments are very likely to be negative. Each feature has its own particular weight. If the weight is negative, the corresponding *n-gram* has a negative meaning, vice versa a *n-gram* is associated with a positive weight which has a positive meaning.

When we train the model we can see that if we add the *2-grams* to the *1-grams* we have an improvement in the results obtained. If we want to refine the outputs we must certainly keep increasing the *n-grams* and take into account all the suggestions seen previously. More could be done. We could work on the type of tokenization, we could filter with different types of frequencies, we could take into account the symbols (emoji, punctuation, etc.).

Surely the result obtained can be improved by varying the model used. Depending on the type of data, it may be useful to use one model rather than another. Some examples could be *SVM* or *Naive Bayes* or any other model able of handling sparse features. Additional models that can be used refer to *deep learning* techniques. In this case we might not use *bag of words*.

2.2 Deep Learning for Classification

This section focuses on an introduction to deep learning techniques applied to the text. Before starting it is important to remember how a text can be seen. In the previous pages we have chosen to see the text as a set of words, more precisely as a sequence of words or tokens. We have transposed a sentence into a matrix called *bag of words*. What we do now is slightly different. On each line there is no longer a sentence but a single word. In correspondence with the column with the same word we insert the value 1. This technique is called **one-hot-encoded**. Each row corresponds to a row of zeros with a single value equal to 1. This technique also produces a very sparse matrix.

2.2.1 Convolutional Neural Networks

This section focuses on the study of neural networks to extract information from the text. In particular, let us see how to use convolutional neural networks to understand the meaning of a text. The first step is to build the model input data. We no longer use a sparse text representation such as *bag of words*, but we create a new type of representation. This representation is dense and dimensionally much smaller than the previous one, in most cases the size will be around 300. One such representation is **word2vec**. We do not study this representation in detail now because we will see it in the section 2.4. This new matrix is created using unsupervised algorithms. Before starting the discussion on neural networks, it is however important to know an important property of *word2vec*. Two words that are very similar to each other have two vector representations where vectors tend to be collinear. In particular, these vectors tend to have a high cosine similarity, the cosine of their difference approaches zero. This representation is extremely efficient when we apply neural networks as predictors.

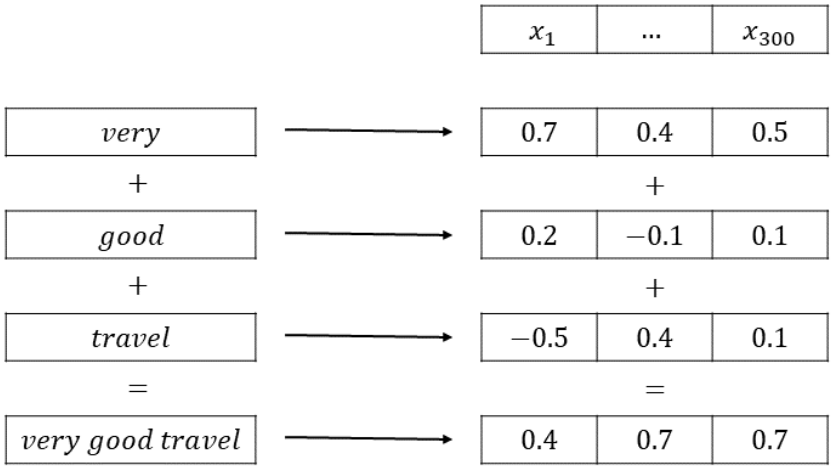

Figure 2.2: Classification with *word2vec*

Starting from the input data just described we can create a neural network. Let us start with an example. In figure 2.3 are two sentences to analyze. *Cat sitting there* and *dog resting here*. For each word we have a 300 dimensional vector representation. Now we can apply a neural network on each sentence, or rather on each matrix. A problem immediately arises: it is not possible to take into account the order, we cannot create *n-grams*. With the technique seen in the previous sections we should have added a column for each *n-grams*. With the current representation of the text it is not possible. An effective solution is to analyze not only single words, but also word pairs, word triplets, etc. In order to do this, we introduce a filter, or rather a window of gradually larger size, based on the number of words we want to analyze at the same time. This window scrolls through the sentence and is called **sliding window**. We can see the technique rendered graphically in figure 2.3.

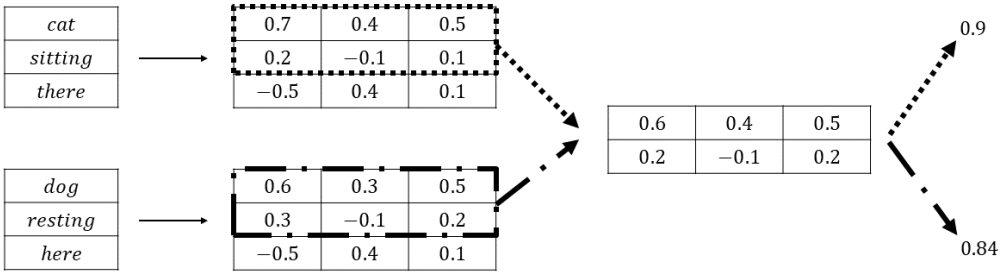

Figure 2.3: CNN on text

The neural network we want to create simultaneously takes into account the words we want to analyze. Once we have selected the portion of the matrix that interests us, we apply a convolution filter of that size ($300 \times n$ of *n-grams*, in this example 300×2).

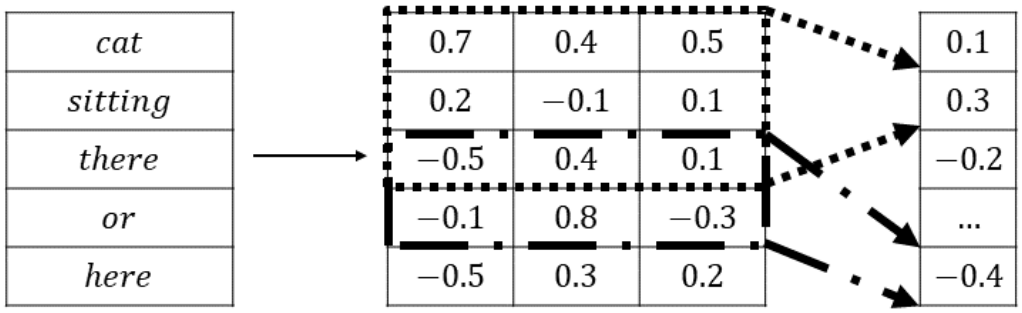

Figure 2.4: CNN and more than one sliding window

To give an interpretation to the application of a convolutional filter we can start from the example. Suppose the entries of the convolution matrix are only very close to the values of the matrix corresponding to the phrase *cats sitting*. So when we apply that filter to the vector representation of this sentence we will have a high activation value because the convolutional filter and the sentence are very similar. If we apply the same filter to the *dog resting* phrase, we have a high activation value because the two phrases are similar. This technique works very well precisely because we have chosen a *w2v* representation type.

In order to go into more detail *word2vec* works very well in our case due to the fact that similar words are very similar in terms of cosine distance. We must remember that cosine distance is similar to the dot product, but convolution is

nothing more than a set of dot products. Returning to the example, it is easy to understand why the two sentences *dog resting* and *cat sitting* are very similar to each other after passing through the convolution filter. More precisely they have a similar activation value. Having a common convolution filter and the vector representations of words being very similar, the scalar product is almost the same. Now it may be interesting not to dwell on 2 filters, but to create wider filters to take into account the relationship that can exist between words that are not necessarily close.

We then create *3-grams*, *4-grams* or any other *n-grams*. When we consider larger relations we do not have the problem that the matrix grows exponentially as in *bag-of-words*. In fact, before we had to increase the columns of the matrix, now we just need to insert a single additional convolution filter with a different size. The array remains 300 in size.

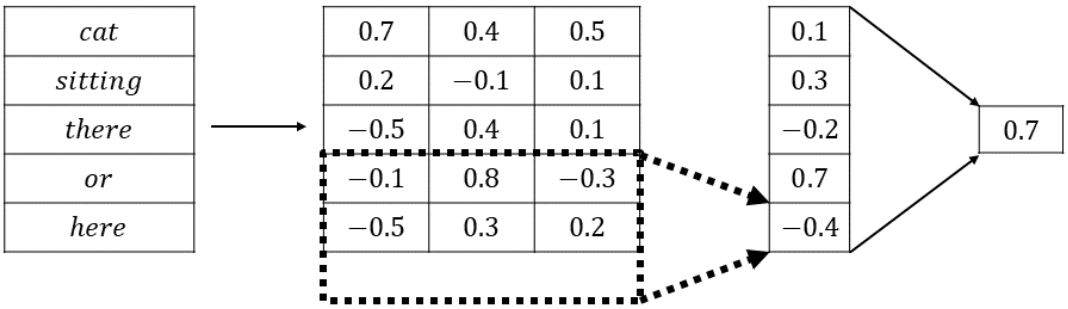

Figure 2.5: CNN *3-grams*

Being a convolutional neural network we can imagine that a single filter is not sufficient to extrapolate all the necessary information. We need more convolutional filters to extract different meanings in relation to the context. Convolutional filters are widely used for image analysis. In this context, the filter moves in every direction to cover the entire image. In the case of the text, however, the filter moves only from the beginning to the end of the sentence to take into account the relationships that may exist in the succession of the sentence. For this reason the filters we use are called *1D convolutions*: they only move in one direction, along the direction of time. If we take the example *cat sitting there or here*, we can consider a sliding window of size 3. In this case we add a padding before the first word and after the last word to have the output size equal to the input dimension. This technique has a purely computational purpose. We do a convolution with the first filter and we get 0.1; if we then slide the filter along the vertical we get the following output value: $0.3; 0.2; 0.7; -0.4$. In the image 2.5 we can see the technique

used graphically expressed. The only negative aspect of this technique is that the size of the input is equal to the size of the output, so if we start from two sentences that are not the same length, we get outputs of different sizes.

In order to solve this problem we need to make a quite strong assumption. We are not interested in knowing where the output appears in the sentence, but it is sufficient to know if it appears within the text under examination. So now, starting from the output, we can take the maximum activation value and discard all the others. We are interested only in this value because it is the portion of text that is most representative of the meaning, or rather it is the most similar to the meaning that the convolution filter carries. As a result of the convolution we get a single value. This process is called **max pooling**. This is a typical image process. Here we translate it for the texts along the time axis. Summary: we start from a vectorized text, make convolutions with a filter that scrolls vertically and at the end we take the maximum activation value to obtain the output.

Let us now define a typical architecture for textual analysis. We choose three types of convolutional filters: dimension three, four and five. This means that we take *3-gram*, *4-gram* and *5-gram* into consideration. For each dimension we train 100 different filters in order to obtain information of different types. As a result we will have an output of 300 size. This vector obtained is nothing more than another way to store the information coming from the text. Now we can apply any type of neural network to make predictions. An example would be applying *multi-layer positron* after applying *dense layers*. What remains to be done is to train the model to get a classification or a regression.

In the papers [14] and [15] two methodologies are compared. The first is based on a *naive-bayes* model applied to *bag-of-words* and the second on a *1D convolutions* model. We can see a precision that rise from 86.3% to 90%.

Finally, after creating a CNN from pre-trained *word2vec*, we need to train the neural network. To achieve the goal we can use *back propagation* both to find the convolutional filters and to train the last part of the neural network. Each model must be trained on specific data to capture the characteristics of the texts to be studied.

2.3 Language modeling

This section is about *NLP* tasks. We are going to discuss about language models. In the second part we will talk about models that work with sequences of words.

2.3.1 Language Models

In order to understand what to create a language model means, let us start with an example. As humans, when we see an house, we are certainly able to complete the missing part of the phrase *This is the ....* This is because we have accumulated experience from dialogues and books. It cannot be the same for a mathematical model, for a software. To overcome this problem, we begin to define probability estimates for the following words.

$$p(house \mid This\ is\ the) = \ ?$$

We can find it from available data.

The language models we are creating are everywhere around us. From translating text into other languages, to automatic text corrector, to suggestions for completing messages. All these features have text generation as a common factor. This means that they estimate the probability that a word can follow a set of given words. This set consists of a few words up to a long sequence of sentences.

Let $w = (w_1, w_2, \ldots, w_k)$ be a sequence of words and $p(w)$ the probability of a text we want to estimate. In order to construct this probability, we create a dependency sequence where a word is conditioned by all its previous words.

$$p(w) = p(w_1)p(w_2|w_1)\ldots p(w_k|w_1, \ldots, w_{k-1})$$

This methodology becomes very complex and time-consuming. Estimating these probabilities is very complicated. For this reason we use the Markov assumption which tells us that estimating the probability of a word given all the previous words is equivalent to estimating a word considering only the previous $n-1$ words. This process must be iterate for all words in the text.

$$p(w_i|w_1 \ldots w_{i-1}) = p(w_i|w_{i-n+1} \ldots w_{i-1})$$

In this formula we can see the Markov hypothesis introduced. The formula simplifies. If we want to consider the *2-gram* model, we need to set $n = 2$.

$$p(w) = p(w_1)p(w_2|w_1)\ldots p(w_k|w_{k-1})$$

In order to generalize, we could introduce *bigram language model*.

$$p(w) = \prod_{i=1}^{k+1} p(w_i|w_{i-1})$$

where

$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)}{\sum_{w_i} c(w_{i-1}w_i)} = \frac{c(w_{i-1}w_i)}{c(w_{i-1})}$$

and $c(w_{i-1}w_i)$ is the number of times we see w_{i-1} followed by w_i.

These values can only be estimated from training data. In the following pages we will see in detail how to derive the estimates. Once we have found the estimates we need to train a model, test it and apply it. We will see that some problems will arise and we will define some possible solutions.

2.3.2 Language Modeling and Neural Networks

In this section we see the same problems already analyzed but we use different techniques. So far we have seen *NLP* patterns, now let us use neural networks. In addition we see a way of representing the text always using neural networks.

One problem we are continuing to have is representation. We have to solve two critical issues: having a smaller dimension matrix with less sparsity and having a similar vector representation for similar words (for synonyms or words with very close meaning). The matrix we are studying now consists of, at least, as many columns as words in a dictionary. In reality many more if we consider the *n-grams*. To solve these two problems at the same time we can study **distributed representations**. These techniques allow us to have a much smaller vector representation (between 300 and 1000 approximately), where vectors are dense and where similar words have similar representation. The concept of similar representation could have different meanings. Below we will see the details.

To achieve this goal, it is necessary to define a probabilistic model of the data. Below we illustrate one of the best known and most used models [16]. This is also one of the first models on the study of language that use neural networks.

Let $w_1, \ldots, w_T$ be a sequence of word $w_t \in V$, let V vocabulary, very large but finite set. Here a model:

$$f(w_t, \ldots, w_{t-n+1}) = P(w_t | w_1^{t-1})$$

The only constraint is that $\forall w_1^{t-1}$

$$\sum_{i=1}^{|V|} f(i, w_t, \ldots, w_{t-n+1}) = 1$$

with $f > 0$.

We define a C function that from a value $w_i \in V$ goes to a real vector $C(i) \in \mathbb{R}^m$. This is a $|V| \times m$ array of free parameters. Where the *distributed feature vectors*

associated with each word in the vocabulary are represented. We can now define f as follows.

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$$

The C function takes feature vectors as parameters. These, as mentioned above, are defined by the matrix C with dimension $|V| \times m$. The line, i, that is the word i, corresponds to the vector $C(i)$. The g function is defined by a neural network with ω parameters. We can now say that the set of total parameters is $\Theta = (C, \omega)$. Now we need to find Θ by maximizing the following log-likelihood function. This maximization is done on the training data.

$$L = \frac{1}{T} \sum_t \log f(w_t, w_{t-1}, \ldots, w_{t-n+1}; \Theta) + R(\Theta)$$

where $R(\Theta)$ is a regularization term.

Like all neural networks, except in the simple perceptron, there are hidden and therefore hardly interpretable layers. It starts with the initial mapping of the text, passes through the neural network with a certain number of hidden layers to get to the output. There are therefore two hidden non-linear levels. For this reason the neural network is composed of an output function *softmax*, which ensures positive probabilities whose sum is 1. Below we see its formulation.

$$P(w_t | w_{t-1}, ...w_{t-n+1}) = \frac{\exp y_{w_t}}{\sum_i \exp y_i}$$

Let y_i be the un-normalized log-probabilities referring to the word i. This is calculated as follows.

$$y = b + Wx + U \tanh (d + Hx)$$

where the hyperbolic tangent tanh is applied value by value, W could be zero (no direct connections), and x is the word features layer activation vector.

$$x = [C(w_{t-1}), \ldots, C(w_{t-n+1})]^T$$

We can note that $\Theta = (b, d, W, U, H, C)$.

For the determination of all parameters of the neural network we use stochastic gradient ascent. This involves the following update rule on each item of the training data.

$$\Theta \leftarrow \Theta + \frac{\delta \log P(w_t | w_{t-1}, ...w_{t-n+1})}{\delta \Theta}$$

This type of representation solves the problems highlighted at the beginning of the section. Therefore we reduce the size of the text representation matrix and identify similar words with similar vectors.

2.4 Word Meaningful Representation

This part focuses on finding a mathematical representation that allows us to give meaning to words. We will see that giving a meaning to words will mean assigning similar vectors to words with similar meaning. We give meaning to words, but we can also give meaning to a sentence or an entire paragraph.

2.4.1 Words and Semantics

We can start with a sentence to illustrate the problem. We can ask a *barman* if we can have *coffee or cappuccino*. We would like to teach the model to understand that these two words are very similar between them by introducing a mathematical representation. To obtain the result we use the **distributional semantics**. The idea behind this modeling is the count of words that **co-occur**. This means that we highlight whenever two words are in the same sliding-window, for example of size 10. This type of counting must be done on training data that has similar characteristics to the data we want to analyze.

The associations we are looking for can be of two different types. Let us analyze a new example. *Bee* and *honey* are definitely related, ie we often find them in the same sliding-window. This type of relationship is called *syntagmatic associates*. This type of relationship is profoundly different from the relationship of the previous example. Before, in fact, the two words could be used in the sentence leaving the meaning unchanged. So we can define *coffee, cappuccino* as a first order relation, also called **paradigmatic parallels**. This kind of relationship is what we would like to find. We therefore construct vectors that have similar representations if the relation of the corresponding words is paradigmatic. In addition we are looking for a representation that is not sparse and with a dimension that is not too big.

Let us now try to understand how we can define the metrics to make these measurements. As we have seen in the previous sections, in a text we can find words that can only cause errors. An example are *stop words*. It is necessary to find a metric to penalize the latter. One technique used is **Pointwise Mutual Information** (PMI). This metric allows us to define whether two words (u, v) are randomly co-occurented or not $(n_{u,v})$.

$$PMI = \log \frac{p(u,v)}{p(u)p(v)} = \log \frac{n_{u,v}}{n_u n_v} \tag{2.1}$$

From the formula 2.1 it is clear to us to understand that uninformative words are penalized. The formula defines the ratio of the joint probability of words to the product of the probabilities of individual words. So if the words were independent

we would get 1. This means that if two words recur together too often we can identify them and penalize their score because it is clear that they are words or pairs of words that do not carry information.

In this formula we find a logarithm because it helps us to manage the situations in which we find 0. We can find this value when two words never appear in the same sliding window. The logarithm would take very low values, so one solution would be to take the maximum value between PMI and 0. We thus introduce a new quantity: **positive Pointwise Mutual Information**.

$$pPMI = \max(0, PMI)$$

To summarize we can say that this metric takes into account the words close to a given word. So we are starting to assign meaning to words based on context.

We have thus obtained a *bag of words* with a metric other than a pure count. However, we have not yet reached the point of being able to define a similarity metric between vectors. We still have very sparse vectors.

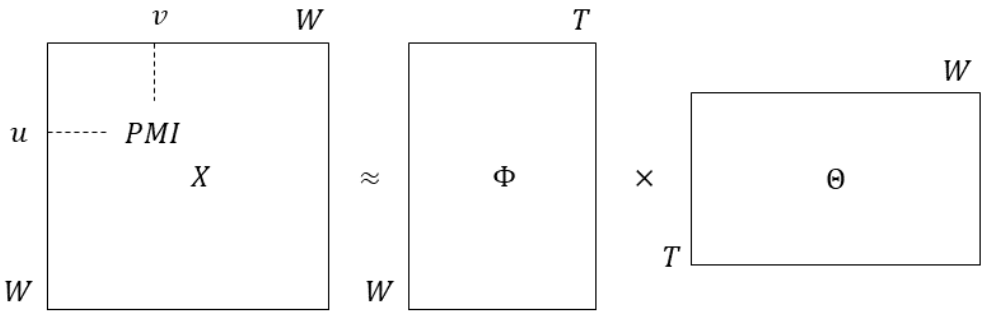

Figure 2.6: Vector Space Models of Semantics

In the previous figure we can see a factorization of the matrix consisting of the vectors just defined. The two factorization matrices are useful for lowering the size. We try to lower the value K. We usually carry it around 300.

The techniques for arriving at the low-dimensional matrix are multiple. In the next section we will look at a widely used method. Now what interests us is to use the V matrix rather than the X matrix. We thus have a dense, low-dimensional set of vectors [17].

This choice leads us to obtain the desired result. That is, we get similar representations for similar words. By similar representation we mean collinearity, or similar cosine difference. This was all created by sliding a window over the text. So, a similarity has been defined based on the words that most often appear next to each other. A context has been defined that depends only on words. There are other techniques to get meaning from words. One of them take grammatical structures into consideration. An example is the *Dependency-Based-Embeddings* pattern [18]. Here the reference context is also defined on the basis of grammatical structures. In this case the matrix is not square but we can create a matrix in which there are the words on the rows and the contexts on the columns. Contexts have their own particular dictionary. This type of model should not be overlooked because it would certainly help us to better understand which relationships are casual and which are not.

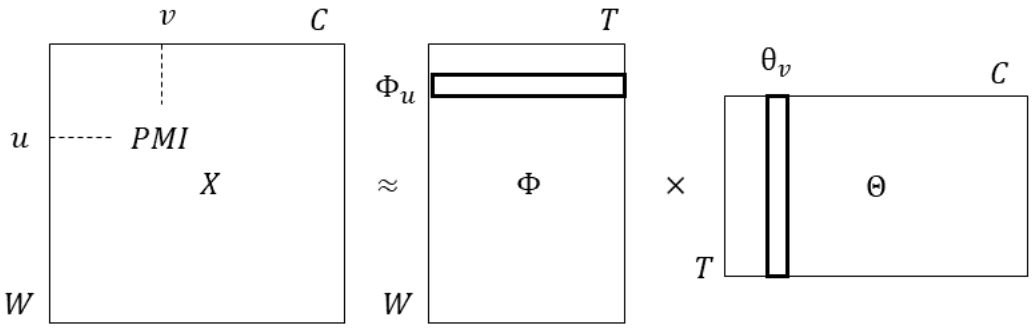

Figure 2.7: Vector Space Models of Semantics with contest

Within our studio, however, we will not use contexts but will create models starting from the co-occurrence of words.

2.4.2 Matrix Factorization

This section will allow us to understand how it is possible to factorize a matrix. There are many ways. Let us start by studying the **Singular Value Decomposition** (*SVD*). This technique is based on the fact that each X matrix can be written through other U, Σ, V^T matrices.

$$X = U \times \Sigma \times V^T$$

It is important to note that those values on the diagonal are sorted in descending order. The number of these values on the diagonal is the number of non-zero

eigenvalues of $X^T X$. So it is related to the rank of that matrix.

A variant of the SVD is **Truncated-SVD**. This method arises from the awareness that the eigenvalues are ordered. We decide to approximate the matrix keeping only the first k components.

$$\hat{X} = U_k \times \Sigma_k \times V_k^T$$

$\hat{X}$ becomes the best approximation of rank k for X, using *Frobenius* norm.

$$\left\| X - \hat{X} \right\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij} - \hat{x}_{ij})^2}$$

In the previous section, we saw a factorization into two matrices instead of three.

$$X \approx \Phi \times \Theta$$

Using an intuition we can transform as follow.

$$\Phi = U_k \times \Sigma_k, \Theta = V_k^T$$

Another way should be splitting diagonal matrix in two.

$$\Phi = U_k \times \sqrt{\Sigma_k}, \Theta = U_k \times \sqrt{\Sigma_k}$$

SVD will help us to find Φ and Θ matrices.
What we have obtained is a factorization of the original matrix through the matrices Φ and Θ. The vectors Φ_u and Θ_v define the embedding we were looking for. SVD allows us to pass from a sparse matrix filled with PMI values to two dense matrices that take into account the same information as the starting matrix. We can define the model thus constructed as distributive semantics.

As already highlighted, there are many other techniques to solve the problem defined within the section. One of these is the **Global Vector** (*GloVe*) [19] model. The results that can be obtained are on average better. The characteristic of the model is that a weighting function f is introduced which must respect some constraints:

- $f(0) = 0$. If f is continuous, it should come to zero as $x \to 0$ such that $\lim_{x \to 0} f(x) \log^2 x$ is finite.
- $f(x)$ non-decreasing so that rare co-occurrences are not overweighted.
- $f(x)$ should be small for large values of x, so that frequent co-occurrences are not overweighted.

Many functions could satisfy these properties, but we restrict the field to a family of functions that can be parameterized as follows.

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

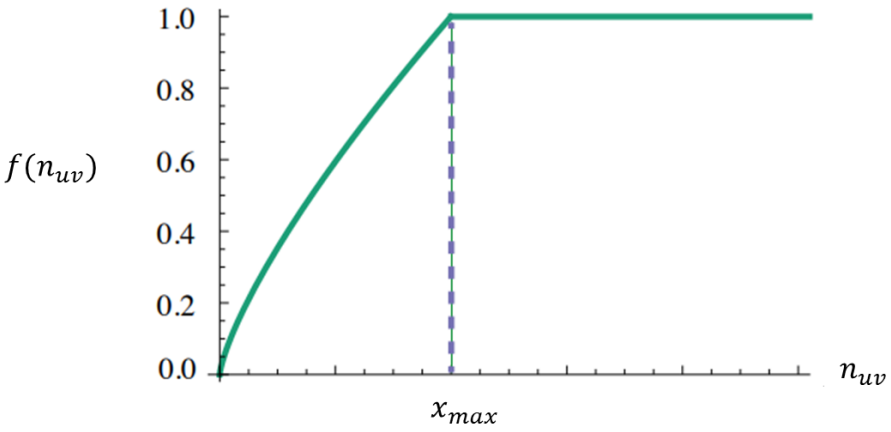

Figure 2.8: Weighting function ($\alpha = 3/4, x_{max} = 100$)

From a certain value of x onwards the function f no longer grows. This is a positive aspect for us because it means that words that are too recurrent, such as stop words, will not be given more importance than they do.

If we want to calculate Φ and Θ matrices, *GloVe* model minimize the loss function L.

$$L = \sum_{u \in W} \sum_{v \in W} f(n_{uv})(\langle \phi_u, \theta_v \rangle + b_u + b'_u - \log n_{uv})^2$$

Where $\log n_{uv}$ is our regional matrix. b corresponds to the bias to allow the model to have flexibility in learning from the data during the training phase. To minimize the Loss function we can use the stochastic gradient descent, as in most learning methods. The training phase consists of two phases. The first phase, called forward, considers each element of the matrix as training data. The next phase, called backward, involves updating the parameters based on the result obtained in the previous phase. Let us go through the process. At the end of the training we have

the ϕ_u and θ_v vectors which are used as word embedding. The starting vectors ϕ_u and θ_v were those associated to the matrix constructed with the PMI.

Factorization is not the only tool to create a word embedding with the characteristics we want. For example, we can rely on the idea that a word acquires a certain meaning based on the context in which it is found, or on the group of the words it is close to. We have already analyzed this concept, but this time we see it to obtain a different result. So let us try to define a probability. In particular we look for the probability of the context given a main word. The context can be understood as a set of words found within a fixed window around the main word. Below we define the probability of having a set of words (i.e. a context, a meaning) given a main word.

$$p(w_{i-h}, \dots w_{i+h}|w_i) = \prod_{-h \leq k \leq h, k \neq 0} p(w_{i+k}|w_i)$$

To define these probabilities, we can see the following formula and we see that it is *softmax*.

$$p(u|v) = \frac{\exp\langle \phi_u, \theta_v \rangle}{\sum_{u' \in W} \langle \phi_u, \theta_v \rangle}$$

Softmax is useful because it normalizes all data and allows us to define a probability distribution. Products between ϕ_u and θ_v define some kind of similarity between words. The model now needs to be trained and this is usually done by maximizing the following formula.

$$L = \sum_{u \in W} \sum_{v \in W} n_{uv} \log p(u|v)$$

The set of words analyzed is usually very large. This fact unfortunately makes the model unattractive because softmax is computationally slow. However, depending on the needs it can still be useful. The above technique is called **skip-gram**.

We can rewrite the whole procedure.

$$\sum_{u \in W} \sum_{v \in W} \underbrace{n_{uv} \log \sigma(\langle \phi_u, \theta_v \rangle)}_{\text{positive example p}} + \underbrace{k \mathbb{E}_{\bar{v}} \log \sigma(-\langle \phi_u, \theta_{\bar{v}} \rangle)}_{\text{negative example n}} \to \max_{\phi_u, \theta_v}$$

This is a technique to stem the computational cost problem due to the large amount of data that must pass through the sigmoid. Looking at the formula we note that the sum has two components: *positive example* and *negative example*. Examples are events where two words u and v appear side by side. This proximity can be casual or informative. If the proximity is random we want to structure a model that says *no*. Otherwise we want to get a *yes*, because we want to carry forward the information on the meaning that a word carries with it.

Here we apply a sigmoid that gives a binary answer (yes, no). Remaining in the formula we say that k is the number of samples, $k_{\bar{v}}$ are the sampled words for a main word u. This model is widely used and it is called **skip-gram negative sampling**.

In the model just studied, the factorization process is essential. We can see that the best value for minimizing the loss is the *shifted PMI* ($sPMI$). This can be verified by making the derivative of the loss function.

$$sPMI = \log \frac{n_{uv} n}{n_u n_v} - \log k$$

Where k is the number of negative examples. $sPMI$ becomes just PMI minus the logarithm of k. This is a recently found result ([20]). In the same article it is said that the model can be seen as an implicit factorization of $sPMI$ matrix. So, starting from different paths, we can say that we have arrived at a structure very similar to the SVD. The difference lies only in the fact that we do not consider PMI but $sPMI$.

2.4.3 Word and Document to Vector

This section focuses on text vectorization. Let us see two types of techniques. The first transforms words into vectors (**word2vec**), the second transforms documents into vectors (**doc2vec**).

The first model has two variants. One based on *bag-of-words*. This means that we define the probability that a word is present starting from the words of the context, that is, from the close words.

$$p(w_i | w_{i-h}, \ldots, w_{i+h})$$

Another technique is exactly the opposite, that is, we define the probability of obtaining a set of words (i.e. the context), given a keyword. This process is named and inspired by *skip-gram*.

$$p(w_{i-h}, \ldots, w_{i+h} | w_i)$$

As already seen above, Softmax is computationally very slow if we work with a large amount of data. One way around this is definitely *negative sampling*. This technique must be used reminding us that our main purpose is to define similar vectors for similar words. Similar vectors, i.e. vectors whose cosine difference is small. A first approach could be to rely on linguistic tables that define similar words, or rather define a similarity between pairs of words. This can be seen in table 2.5.

u	**v**	**Human**	$\cos(\phi_u, \phi_v)$
Lion	Lion	1	...
Media	Radio	0.74	...
Lion	Cat	0.73	...
Train	Car	0.63	...
...	...	...	...

Table 2.5: Similar words, similar vectors

For example, in the table we can see that two equal words correspond to the maximum similarity value and for two similar words there is a still high score. The model we want to create cannot be based on this type of tables due to the large amount of data and context differences depending on the reference domain. The limitations of this approach are many. However, it is possible to make comparisons between subgroups of model words and reference tables. For example we can use the Spearman correlation coefficient. In this way we try to verify how well our model is in agreement with the linguistic tables created directly by human judgment. This methodology can be used to evaluate the performance of the model created.

Now let us suppose we have vectors for the words.

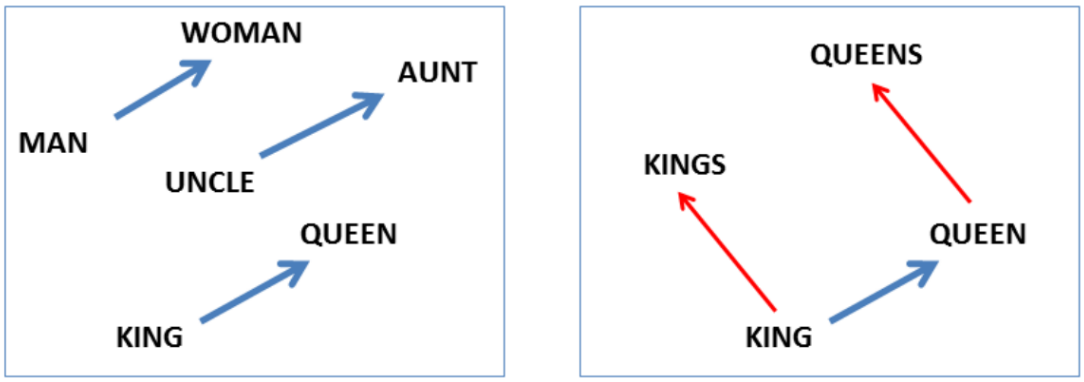

Figure 2.9: Word2Vec

In the image we can also see how it is possible to do operations between vectors and this thing translates into compositions of the meaning of words. For example, the vector *king* minus the vector *man* plus the vector *woman* produces a vector very similar to the vector corresponding to the word *queen*. We can say that the relations of meaning are maintained even when we compose two words, that is

when we compose two or more vectors.

$$a : a' = b : b' \quad (man : woman = king : ?)$$

Another example to understand the concept can be seen in Rome minus Italy plus France. The result is a vector very similar to Paris.

$$\cos(b - a + a' = x) \rightarrow \max_x$$

The construction of this model has been very successful following a recent article [21]. The structure created has also aroused interest in cognition science [22] where it is called **relational similarity**, although until now we have been working with **attributional similarity**.

So far we have described what we want to achieve. Let us see now how the different techniques can lead us to such a result. Or rather let us see how the different models perform ([23]).

win	Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk
2	PPMI	.732	**.699**	.744	.654
	SVD	.772	.671	**.777**	.647
	SGNS	**.789**	.675	.773	**.661**
	GloVe	.720	.605	.728	.606
5	PPMI	.732	**.706**	.738	**.668**
	SVD	.764	.679	**.776**	.639
	SGNS	**.772**	.690	.772	.663
	GloVe	.745	.617	.746	.631

Figure 2.10: Word analogies task: models evaluation

Let us summarize what we have achieved up to this point. It starts with a large and sparse matrix where the element is the PMI. We apply SVD to get a lower size and dense vectors. To achieve this goal we can also choose to apply *skip-gram negative sampling* and then *GLoVe*. In this case we factorize, we speed up the computation with a weighted loss of information.

There are many techniques to implement factorization: we have seen four of them. It can be seen that although SVD is an outdated technique, it does not perform worse than the other techniques. This type of evaluation can be done on

the basis of linguistic tables introduced in the previous lines.

Let us now mention the transformation of documents into vectors. So far we have only turned words into vectors. The technique is **paragraph2vec** or **doc2vec**. These two names represent the same model but the first term derives from the paper that first introduced it [24], while the second name comes from the implemented software. The idea behind the model is the same as that behind *word2vec*, where sentences or entire documents are considered instead of words. Here, the approach can be double. Find the context from the document or find the document from the context.

Starting from a d document we calculate the probability of having a document starting from the context: **Distributed memory**, DM.

$$DM = p(w_i|w_{i-h} \dots w_{i+h}, d)$$

Or starting from the context, we find the probability of the document: **Distributed Bag Of Words**, $DBOW$.

$$DBOW = p(w_{i-h} \dots w_{i+h}|d)$$

The latter model is the transposition of the *skip-gram model*, where words are replaced by documents.

Doc2vec is often used to search for similar documents, that is, to understand whether two documents represent the same concept or not.

2.4.4 Vectors and Operations

In this section we go into the details of the calculations we can do to check if two words are similar.

First of all, remember that *Word2vec* is a technique that uses an unsupervised training phase. We create vectors with useful characteristics to grasp the meaning of words and the relationships between them. If we define b as the word *king*, we subtract a (*man*) from it and add a' (*woman*), the result is x (*queen*).

In the following formula we can observe how the closest vector is calculated.

$$x = \arg \max_{x' \notin \{a,a',b\}} \cos(b - a + a', x')$$

What we want to do is to minimize the cosine distance, that is, to maximize the cosine similarity. When we carry out operations, we must remember to exclude from possible candidates the vectors (words) that have been taken into consideration for the operation. In this case, the words *king*, *man* and *woman* are not considered.

2.4.5 Characters, Words and Phrases

It is not always useful to have vector representations of words. Depending on the problem we have to solve, it may be necessary to create vectors that represent subgroups of the word (characters) or groups of words (sentences). In this section we study techniques to not stop only at words but also to represent characters or entire sentences

As a first step, let us try to go into more detail. Starting from the words we go deeper, that is, we work on the level of characters. Some languages have a rich morphology and therefore we can try to value them. For example, some words change their meaning depending on the prefix or suffix they carry with them. The latter give us essential information to understand the meaning of a word. However, not all prefixes or suffixes take on particular importance if we need the meaning. Starting from this awareness, a recent article [25] proposes the following idea. We create two groups of words: in the first we put the words that can undergo major changes in meaning and in the other group the remaining words. We create a distances, we create a space in which we bring some words together and push others away. These shifts can be created with different metrics and constraints, based on the knowledge we have a priori about the language.

Sometimes it may happen that we do not have this type of information a priori. So it is not possible to create the word space ordered as we want. In this case we can create *n-grams* as already seen: instead of words we work on single characters. An example is proposed by Facebook [26] with **FastText**. A sentence is seen as a succession of characters. Blanks also represents characters. Also in this case there are different techniques to arrive at dense and reduced-dimensional vectors. One of these can be *skip-gram negative sampling* ($SGNS$), the use of which has already been seen in the previous sections starting from different representations. At this point we need to represent the similarity (sim) between two words (u, v). A word is no longer made up of a vector, but is made up of a set of vectors. In order to be used in the model we therefore consider the sum of the vectors (G_v) which represents a single word.

$$\textbf{SGNS: } sim(u,v) = \langle \phi_u, \theta_v \rangle \qquad \textbf{FastText: } sim(u,v) = \sum_{g \in G_v} \langle \phi_u, \theta_g \rangle$$

This type of technique is only useful if we work with a morphologically rich language.

After the characters we can decrease the granularity and work with words. We have already spoken extensively about this case. By further decreasing the granularity we come to the sentences. These can be seen as a set of sentences. The first idea

that can come to mind is to represent a sentence vector from word vectors. So the vector representation of a sentence is the average of the vectors that represent the words. Words can be represented with a *Word2Vec* pattern. The average could then be chosen as a weighted average on the $TF - IDF$ values. This technique yields very interesting results.

Another approach is to see sentences as a composition of sub-sentences. Let us work on the similarity between words and phrases (s). We define the word-sentence pair as a positive example if the word is present in the sentence, otherwise we define a negative example if the word is present in another sentence and the two sentences are not similar. This similarity (sim) is defined below.

$$sim(u, s) = \frac{1}{|G_s|} \sum_{g \in G_s} \langle \phi_u, \theta_g \rangle$$

Unions are just groups of words, or rather *n-grams* present in a given sentence. This technique is the same as *FastText*, but where the characters are replaced by groups of words and where we consider the average rather than the sum. This model is called, **Sentence2Vec** [27].

In the previous lines we have seen that the techniques are very similar even if we break down the text at different levels. An idea immediately arises: finding a model that can manage everything in the same way. Or rather, we might be interested in looking for a model that can work simultaneously with different granularities in the text. We could always tell about *n-grams* where these could be either sets of words, sets of characters, or sets of sentences. We see a first attempt with a technique called **StarSpace** ([28]). This model is very interesting for some types of problems. For example, in movie recommendation systems, people, tags, movie descriptions, etc. could be taken into consideration at the same time. It would be possible to have a complete information collected in a single model.

Before concluding the topic, let us give a hint to the deep learning methodologies to tackle these problems, or to represent the text, in particular the sentences. The three main methodologies, from the oldest to the most recent, are: recurrent neural networks (RNN), *convolutional neural networks* and *recursive neural networks*. All these techniques use hierarchical structures of representation, that is, they use syntax to grasp the meaning of words.

An example of a model that works with RNNs is the **skip-thought vectors** [29]. The idea is to predict a sentence starting from a given text. An RNN encodes a sentence, thus we obtain a hidden representation. After a training phase, from representation we foresee the next sentence. The interesting result is that

the hidden representation is just what we want to represent the meaning of the sentence. This technique is similar to a technique often used in images and it is called **encoder-decoder**.

2.5 Representation by Topics

This chapter can be considered as a continuation of the previous one because we are looking for a representation for a sentence, a paragraph, a text in general. Now let us try to change the point of view and find a representation for a text. We focus only on sentences or groups of sentences. We will come to associate labels with portions of text. These techniques are called *topic modeling*.

2.5.1 Topics in Texts

The idea behind this new type of textual representation is to find the topics that describe the text. Or rather, find the associations that exist between words or groups of words with a particular topic. For example the topic *food*, can be associated with the words *cake, milk, pasta*. What we are looking for is nothing more than an association between topic and words that describe it. We will see that this can be mathematically formalized through probability distributions. In particular, starting from a set of words, we can first extract n_{wd}, ie the number of times a word w appears in each d document. At this point we are looking for two probability distributions. The first corresponds to the probabilities of the words given the topic t.

$$\phi_{wt} = p(w|t)$$

and the second is the probability of topics given documents.

$$\theta_{td} = p(t|d)$$

The definition above is precisely the mathematical definition of topic. The topic is therefore only a probability distribution given a set of words.

This type of model is very useful for understanding what a text has to tell us, what someone wants to convey through a text. In this way we try to understand the topics covered and we can also classify the documents according to our needs. In real life these models are widely used. We can find them to classify emails, to analyze comments on social networks, to solve problems that are highlighted by customers of a company through questionnaires, to create chatbots and much more.

Let us see one of the first models proposed that is based on the idea just studied. This is called **probabilistic latent semantic analysis**, *PLSA* and was published

in 1999 by Hofmann [30]. Its particularity is to be elementary and to analyze the combinations of topics to predict the words in the texts. Below we see the probability distribution introduced.

$$p(w|d) = \sum_{t \in T} p(w|t,d)p(t|d) = \sum_{t \in T} p(w|t)p(t|d) \tag{2.2}$$

where T is the set of all topics t and in the first equality we need to remember the law of total probability, thas is

$$p(w) = \sum_{t \in T} p(w|t)p(t)$$

Now, we don't take into account documents, that is we suppose that the probability of word given the topic doesn't depend on the document. We need to add conditional independence assumption.

$$p(w|t,d) = p(w|t)$$

The model we want to train does not start from known example data. This falls into the category of unsupervised algorithms. Probability distributions are created starting from a text of which we only know the words of which it is composed, we know nothing about the topics. The idea behind this technique is the generation of topics starting from the words and at the same time the generation of the next word starting from a topic. We start from the first word of the text, we assign it a topic, we continue in the same way with the second and so on until the end of the text. The assignments are random. At this point we can make some counts and find the probability distributions we are looking for ϕ_{wt} and θ_{td}. To obtain distributions that are not random but that carry information we must train the model through an iterative process that we will see in the following section.

$$p(w|d) = \sum_{t \in T} p(w|t)p(t|d) = \sum_{t \in T} \phi_{wt}\theta_{td}$$

ϕ_{wt} is a matrix representing the probability distributions of the given words in the topics while θ_{td} the probability distributions of the topics in the documents. The distributions are on the columns in both cases. We thus obtain a matrix representation. We will see that latter will be very useful for determining the distributions from the data.

In the next section we will figure out how to train the model.

2.5.2 Training with EM

In the previous section we defined and studied a model that extracts topics from a text, where the text is seen as a set of words. In particular, the model is composed

of two probability distributions ϕ_{wt} and θ_{td}.

Once the structure has been defined, it is necessary to move on to the training phase, or rather to the phase of researching the optimal parameters. What we want to achieve is log-likelihood maximization.

$$\log \prod_{d \in D} p(d) \prod_{w \in D} p(w|d)^{n_{dw}}$$

Before looking for the parameters that maximize the function, let us make some changes.

$$arg \max_{\Phi,\Theta} (\sum_{d \in D} \sum_{w \in D} n_{dw} \log \sum_{t \in T} \phi_{wt} \theta_{td})$$

The first obstacle concerns the logarithm of the product. If we apply the logarithm we get the sum of the logarithms. We can then note that the probability of the document is not affected by the parameters we are looking for, so we eliminate it. We use what we have introduced with our model, so in the last part of the equation we replace the probability distribution of the words in the text with the topics.

We insert the following contraints that define the model. We remember in fact that we are working with probability distributions. So all distributions must have zero sum.

$$\phi_{wt} \geq 0 \qquad \theta_{td} \geq 0$$

$$\sum_{w \in W} \phi_{wt} = 1 \qquad \sum_{t \in T} \theta_{td} \geq = 1$$

At this point, however, we do not have a problem that is easy to solve. Maximizing is quite complicated. A frequently used technique when we have the logarithm for the sum is **EM-algorithm**.

E-step and *M-step* are the two main moments of the training phase. In the *E-step* we define the probability distributions as follows.

$$p(t|d,w) = \frac{p(w,t|d)}{p(w|d)} = \frac{p(w|t)p(t|d)}{p(w|d)} = \frac{\phi_{wt}\theta_{td}}{\sum_{s \in T} \phi_{ws}\theta_{sd}}$$

Bayes-Rule gives me the first equality and *Product-Rule* gives me the conclusion.

In the *M-step* we update the parameters. Starting from the knowledge provided in the previous step, we make the changes as follows.

$$\phi_{wt} = \frac{n_{wt}}{\sum_w n_{wt}} \qquad n_{wt} = \sum_d n_{dw} p(t|d,w)$$

$$\theta_{td} = \frac{n_{td}}{\sum_t n_{td}} \qquad n_{td} = \sum_w n_{dw} p(t|d, w)$$

To complete the training and reach a satisfactory result, it is necessary to iterate *E-step* and *M-step* several times on the same data. First we look at all data, then we calculate the probability distributions and at the end we update the parameters. This process must be repeated until it converges.

The model we have just seen is an unsupervised algorithm. We start from known data on which we do not know a result and in which there are hidden variables that we intend to define. The EM model is very powerful and widely used in cases of this type.

2.5.3 The world of Topic Algorithms

The world of topic modeling is very vast. We have focused on PLSA but there are a large number of different techniques. Each technique has its advantages and its drawbacks. There is no better model than another but there is always a model that is better suited to the type of problem we want to solve and the type of data we want to deal with.

An architecture that originates from the PLSA and is widely used is the **Latent Dirichlet Allocation** [31]. Starting from the model just studied, *Dirichlet priors Dir* are added for the ϕ and θ parameters.

$$Dir(\phi_t|\beta) = \frac{\Gamma(\beta_0)}{\prod_w \Gamma(\beta_w)} \prod_w \phi_{\beta_{wt}}^{\beta_w - 1} \qquad \beta_0 = \sum_w \beta_w, \beta_t > 0$$

where $\phi_t = (\phi_{wt})_{w \in W}$ $\theta_d = (\phi_{td})_{t \in T}$.

The real difference with the PLSA is that parameters that were previously fixed values now become probability distributions. The model output, that is *posterior distribution*, has a higher number of parameters than PLSA. This technique mainly uses two ways of training: *Variational Bayes* and *Gibbs Sampling*.

We now introduce a class of specific techniques for different types of problems and data based on **probabilistic graphical models** [32] and Bayesian inference.

One of the most famous is certainly the **Hierarchical topic model** [33]. This develops from the Latent Dirichlet Algorithm. The idea behind this model is the hierarchy. We try to create an order of importance among the topics. We create main topics and sub topics. Each main topic will be composed in turn of a series of subtopics.

Dynamic topic models [34, 35] is an architecture based on the concept of dynamism. The topics created are designed to be able to change over time. It can be seen that the probability of the topic changes. This technique is very powerful if study information flows such as news or whenever trends change over time.

One problem we can certainly study is the management of multilingual texts. It is certainly complicated to manage texts that are presented each time in a different language but from which we want to obtain information of the same type. For this there are **Multilingual Topic Templates**. We define same topics for different languages. The language used in the texts does not have to be important. The training of these models is usually done on parallel documents, or documents that are the same but written in different languages.

So far we have defined a new model for each problem we want to solve. The idea arises spontaneously of wanting to find a model that can manage all the problems in a general way. A first attempt is the **Additive Regularization for Topic Models** [36]. This model also arises from the PLSA structure. To the latter are added regularizers. We use the same log-likelihood as the PLSA with additional parameters. These parameters are useful to have a certain freedom regarding the issue we have to solve. The basic idea is to add more topics. For example this can be useful when we need to hierarchize them. For the training phase we can use *EM algorithm*. The *E-step* remains unchanged from the PLSA. While the *M-step* needs a little modification. It is necessary to introduce the derivatives of the regularizers.

Still with the idea of finding a general model, we introduce the **Multimodal argument models** [37]. Not only we want to analyze texts with different characteristics or want to solve different problems, sometimes it would be useful to introduce data with different characteristics from the text. Examples are metadata, authors, dates, categories, etc. The starting idea is the same as the previous model, but the new intuition lies in weighing the probability distributions. We assign a probability to each different modality and we define a weight. For example, the probability distributions of words can be studied in a text, but also the probability distributions of the authors. Each of these distributions will have a weight to contribute to the final probability distribution. This model is very useful in several cases. For example, we can find topic trends for a certain author or for a certain date. We can say that this architecture is the most general, which tries to incorporate different problems and data different from the text.

Chapter 3

Features and Forecasting

In order to develop a solution to understand the behavior of NPS related to the business actions on customer satisfactions, this chapter analyse the theoretical foundations of NPS-simulator developed in this project.

The first section is about the preprocessing of the data: from elimination of redundant features up to the creation of new ones. The second part deals with forecasting from numerical data. In particular, we focus on decision trees and random forests.

3.1 Feature Importance

Before starting to create predictive models on the collected data, it is necessary to reorganize what we are looking at. In particular, we can say that the first step is to do Feature engineering. A feature is a property of what we are looking at, of what we want to describe. A data set is made up of a set of observations which are composed of a set of features. The latter will be the variables of the predictive model that we are going to build. Not all models are the same; we must decide which model is best for a certain type of problem and for a certain type of observation. To achieve the goal, feature engineering can help us understand hidden trends in our data. In addition, this can be useful for improving the predictive accuracy of the chosen models by adding or removing features.

In the following sections we will study different types of techniques to learn how to select the features that bring greater value to our model. The first step is surely knowing how to choose whether a feature brings information or not. We must first understand what it represents and if it is useful for our purpose. Then we have to understand if it carries information, for example by looking at whether there

is variance between the data: if there is no variance in data then the feature is not discriminating to obtain the result. If a variable is constant across all data then it carries no information for prediction. The features we want to use must not be random with respect to the output variable. Finally, they must not contain redundant information. If one variable contains a length in meters and another contains a length in km, then one of them can be discarded.

Before starting to create or filter features from a mathematical point of view, it is essential to introduce knowledge of the domain into the techniques used. This fact is invaluable to the success of the model. It is important to remember that the reliability of the final predictive model is as good as the input data is good, therefore the data processed through feature engineering.

3.1.1 Unsupervised Learning

The first step in creating new features starts by looking for if any relationships exist between the data. A relationship can be considered the knowledge that allows us to identify the values of a variable starting from the values of another variable. There are several techniques to achieve this goal. Most of these can be grouped into two cluster. The first is *Supervised learning*, it uses a priori knowledge about an initial dataset. From here it generalizes to predict future responses from different inputs. The second group is called *Unsupervised Learning*. Unfortunately, we do not have a priori knowledge here, but there are techniques that create automated learning: in particular, models or groupings of data are identified and then they are used in a second phase of learning.

In unsupervised learning there is never an output variable on which to compare the values of the results obtained. This technique works with the idea of creating groupings of input data. These techniques are really very interesting when we are faced with a dataset that we do not know and where we do not know type of information it may contain. With this type of technique, the second step is to give meaning to each cluster created. In this case knowledge of the domain could be essential.

Unsupervised learning, however, always focuses on creating groups of data, but these can often be difficult to interpret.

To summarize, what we try to do is to work with features. The latter help us to identify the relationships that exist between our data. These relationships are then essential to create accurate predictive models. The creation of these features can take place in several ways. First we must use our knowledge of the domain,

then we can use unsupervized learning techniques to create clusters within the data.

In next sections, we will see some of the different techniques that can be used to find clusters in data.

3.1.2 Clustering

The most used methods to search unknown groupings in the dataset are certainly the clustering methods. These techniques use iterative processes to bring together the elements that are most similar to each other.

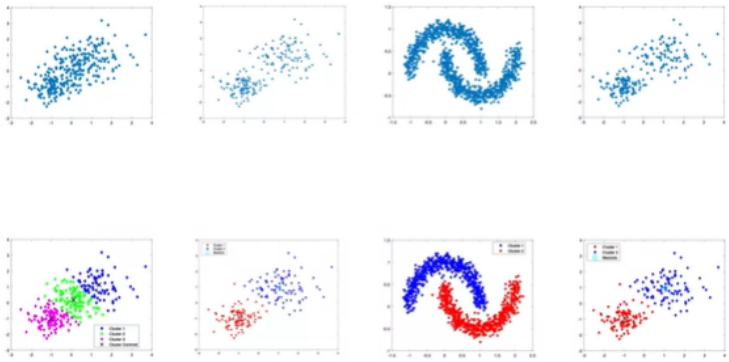

Figure 3.1: Clustering

A classic example of an unsupervides learning algorithm is **k-means**. Let us now try to analyze how it works to understand what is the idea behind this type of algorithm. The first step is to choose a priori the number of k clusters we want to find. This k identifies the k of k-means. The second step is to define k centers called *centroid* taken between the elements of the set. There are several techniques for choosing centers optimally, but we now assume that they are chosen randomly. Now, each remaining point is assigned to a centroid, or rather, to a group. The group is chosen on the basis of the similarity that exists between a single element and the group, usually on the basis of proximity. At this point the centroids are updated, choosing as the new point the average between the elements of each set. Then we reanalyze each element to see if it needs to be moved to another group. This happens because the distances to the centroids may have changed. All these operations are repeated until the minimum of the sum of the distances between all points and centroids of their respective clusters is reached. A clasterization of the input elements is thus obtained.

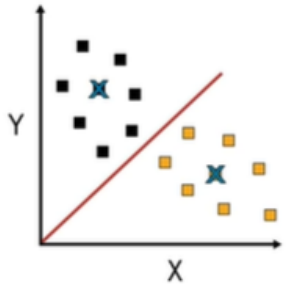

Figure 3.2: k-means

In figure 3.2 we can see a classification carried out by a k-means algorithm. There exist many algorithms that allow us to do this type of clustering. Each leads to a different type of clustering depending on the type of data, one algorithm can be more suitable than another. In the following sections we will see *k-means, k-medoids, spectral clustering, and Gaussian mixture models*. The method just seen is one of the most widespread and simplest. It works well for creating circular clusterings.

These algorithms are usually the first ones used to understand the data. The results of these algorithms can be very useful to then insert an additional predictive algorithm or to make comparisons between models. Recalling what we have seen, we can say that the input points are grouped according to their distance from the centroids, based on the minimum possible distance of all elements from the centroid. The distance chosen is usually the Euclidean distance. However, there are a large number of possible measures that can be used. The choice of distance depends on the type of data used and the result we want to obtain.

An example of a measure could be the median. A certain type of data is very suitable for this measure. In this case the algorithm is called **k-medoids**. The structure of k-medoid is very similar to k-means. The median is used to eliminate the impact that an outlier can give. In fact, if there are many outliers in the data, the average is not the right measure because it is too sensitive. A feature of this algorithm is that the centroid is always one of the elements of the input data.

The two techniques just highlighted work very well when the data can be divided into well-defined clusters that can be separated by a straight line. These fail to bring value if we are faced with data like in figure 3.3.

In the figure we can clearly see three clusters that cannot be separated by a

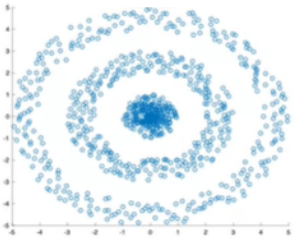

Figure 3.3: Data to cluster

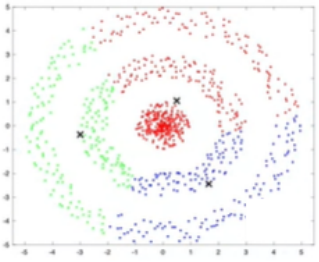

Figure 3.4: Clustering with k-means

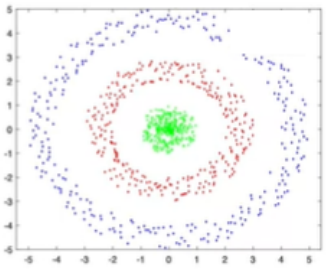

Figure 3.5: Spectral clustering

straight line, but wrap around each other. The k-means and k-medoids algorithms cannot find this subdivision. It is necessary to introduce a new algorithm called **Spectral clustering**. This considers the distance of each point to its neighbors and assigns groups based on similarity.

Another problem arises when we have overlapping clusters. In this case we have to use **Gaussian mixture models**. In this algorithm we introduce the probability to determine the membership cluster. A k number of clusters is chosen and consequently k Gaussian distributions associated with the data are created. Each point has a certain probability of belonging to a distribution (ie to a cluster) or to another. Eventually a point will belong to the cluster in which it has the highest probability. To simplify the representation, we can think of distributions as ellipses. If the variance is the same for all distributions then the ellipse turns into a circle.

To sum up, clustering algorithms allow us to identify hidden patterns or groupings data. The best algorithm we can use depends on our data.

3.1.3 Features Selection

In the following paragraph, the main objective is to eliminate unnecessary or superfluous information, keeping only what is necessary. We try to eliminate those features that do not lead. All this brings many advantages. Surely the model we are going to build will be more easily interpretable and its computational cost will be very lower. Below we see the main quantitative methods to evaluate the features.

Main methods for features selection are: *wrapper, embedded* and *filter*.

Wrapper methods choose the features to be selected based on the result obtained

from the different models. By evaluating different models with different combinations of features, these techniques select the group of featuers that perform better.

Embedded methods are an integral part of the models themselves. More importance is given to features that bring more information and less weight is given to those features that are not very informative. These two techniques have in common the fact that they both study the predictive model before making a selection of features. In particular, they select the best set of features according to the model used.

Filter methods do not take into account the predictive model in which the features will then be used. In this case we study statistical relationships between feature and feature and between feature and result. This technique can be used with a wrapper technique, before an embedded technique.

In the following, we can see some techniques, *variance thresholding*, *covariance and correlation* and *statistical tests for feature independence*.

The first technique we study is called **variance thresholding**. This model studies how variations in the input data impact variations in the output data. It can be easily understood that we are interested in those features in which a large variation in the input data corresponds to a large variation in the output data. A first element that we want to study is the variance σ^2 which we can see calculated below.

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Where x_i corresponds to each value and $\bar{x}$ to its mean.

If all the elements are very close to the mean, we can see the variance value very low. The greater the variance, the more the elements are located at a distance from the average. Before using this quantity to compare the different features it is important to remember that the variance is affected by the reference scale. For example, if we consider two identical measurements expressed one in grams and one in kilograms, we find the variance completely different. It is therefore necessary to re-scale the variance before any kind of comparison. At this point a first selection of the features can be made by eliminating those features that have a variance below a certain threshold. A very low variance indicates little information, on the contrary from a high variance we expect a lot of information. At this point it is necessary to choose the threshold. It is definitely not an easy choice. What we are looking for is to divide a group of features that have a high variance and to eliminate those variables that have a drastic decline. In practice, we can graphically represent all the variance values in descending order and set the threshold at the

point where a very sharp elbow is created. However, it is an empirical technique, so it is better to be cautious by keeping a low threshold and use other techniques to refine the selection. One of these metrics can be covariance.

Covariance is the mean of the product of two value. The first value is the distance between each predictor and the mean of all predictors. The second value is the distance between each response and the mean of all responses.

$$Cov(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

Unlike variance, covariance can take on both positive and negative values. The closer the covariance between two variables is to zero, the weaker the relationship between them. In our case, we study the relationship between an input data feature and the output variable. This means that these features carry little information to create the output. We can note that within the covariance formula it is important to take into account the sign of the input with respect to the sign of the output. When variables are not in a relationship, then the signs appear randomly positive and negative. Taking this into account we understand that the average will bring these values close to zero due to cancellation. Then the covariance will be close to zero. We can easily understand that this type of formula takes into account a type of correlation linked to the linear dependence of the variables. Unfortunately, the covariance also has the particularity of being influenced by the scale. It is therefore necessary to scale the covariance with respect to the standard deviation in order to make comparisons.

Knowing covariance, we can define **Pearson correlation coefficient** r.

$$r(x, y) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

This value determines which features bring a high information content to the output and which contribute weakly to its formation. Therefore the coefficient is used in order to select the interesting features. Pearson coefficient has good behavior on linear dependencies. If we are looking for nonlinear dependencies it is better to use **Spearman** and **Kendall correlation**.

All the indices highlighted so far perform well to identify monotony. To identify more complex dependencies we can use other techniques.

I want to highlight a statistical technique to select the most important features. This is called **statistical tests for feature independence**. Let us take two

features at a time and show if they are independent. To achieve this goal we calculate the p-value to prove that two distributions are independent. Now we choose the most suitable statistical test: the choice depends on input variable and output variable. Finally we need to define a *alpha value* probability threshold. If p-value is greater than the threshold, the feature considered is defined independent of the output. For this reason it will be deleted. If p-value is below the threshold then the feature considered is defined important to create the output.

3.1.4 Principal Components

In the previous section we figured out how to select the most informative features for a predictive model. We have seen that keeping only the necessary information is beneficial to the model. In particular, until now we have seen techniques linked to variance that we can summarize as techniques that select features with high variance and discard those with low variance.

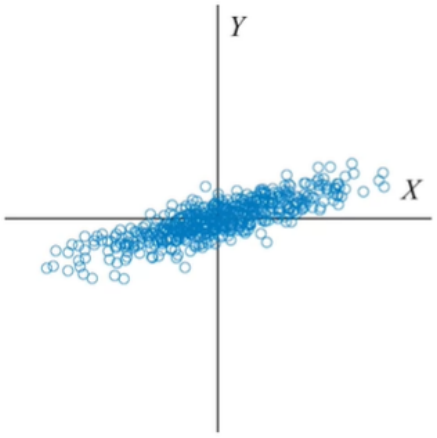

Figure 3.6: PCA: high variance feature

We now introduce a new technique to try to maximize the variance we can capture. If until now we have only selected the features with greater variance, now we create a new set of features so that we maximize the variance and minimize the number of features needed. These methods are called **principal component analysis** or **PCA**. We always start from the original set. For simplicity we consider a small set of points (see figure (3.7). We eliminate Y using the variance threshold technique. We obtain a single feature. The number of features obtained is called dimensionality. If we want to give a geometric interpretation, we are projecting two-dimensional data onto a single dimension.

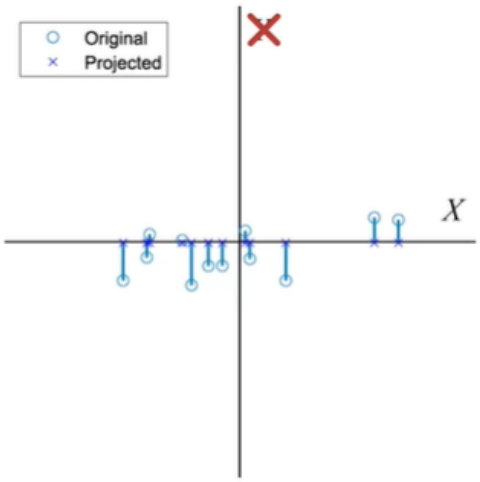

Figure 3.7: PCA: projection on a component

Dimensionality reduction techniques are widely used in data analysis. In particular in automated processes and machine learning. This approach has a lot of advantages. The first is certainly the efficiency of the predictive models that start from these processed data. However, an equally important aspect should not be overlooked: the interpretability of the data and the result. There are not only advantages. A negative aspect is certainly the error that we create when we carry out the projection on a dimensionally smaller space. Returning to the example we can see that the error made by projecting onto the Y axis would have been greater than the error made by projecting onto the X axis. If we look at the figure 3.8 we can see that we cannot minimize the error by projecting it onto the X axis or projecting it onto the Y axis.

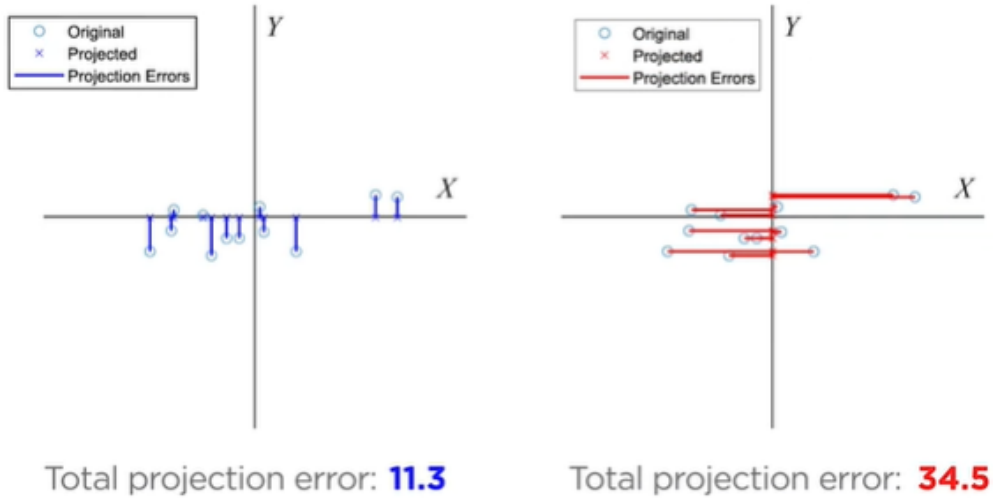

Figure 3.8: PCA: projection on two components

There is another type of projection that allows us to minimize the error. We define a new axis that we call the first **principal component** of the set of features. By determining the axis that minimizes the error, we realize that this has a further advantage. We can see that not only the error is minimized but the variance is also maximized: this is exactly what we are looking for. Remaining in the example, we can define this new $P1$ axis as the new feature created from the starting set.

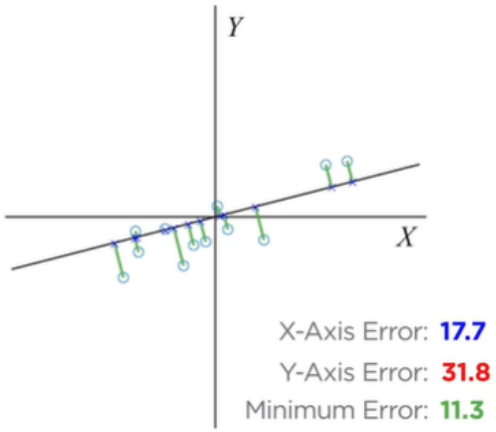

Figure 3.9: PCA: new dimension

We therefore eliminate X and Y and continue our feature determination process with only $P1$. We obtain a dimensionality equal to 1. At this point it is necessary

to determine the new values as a function of the new principal axis. These are determined by projecting the values on $P1$ and measuring the distance with the sign from the origin of each projection. The second main component $P2$ is defined perpendicular to the first. This is necessary to maximize the variance not yet taken into account.

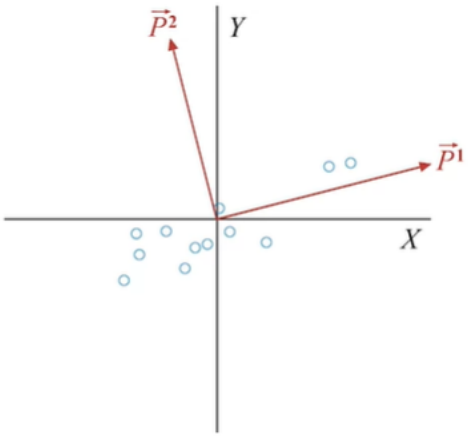

Figure 3.10: PCA: new dimensions

We often find ourselves working on a number of features that far exceeds 2. For this reason, below we can see a technique used for the choice of components beyond the second. The first step remains unchanged, ie the first component is determined as in the previous lines. To determine the second component we define all the components orthogonal to $P1$ and among these we choose the one that maximizes the variance. The third component is chosen from the orthogonal components a $P1$ and $P2$ such that it maximizes the variance. For the subsequent components the process is iterative. The process ends when a last and only component remains orthogonal to all the other components. This will be the final component. The result is a new set of orthogonal axes rotated with respect to the starting ones (in order to maximize the variance).

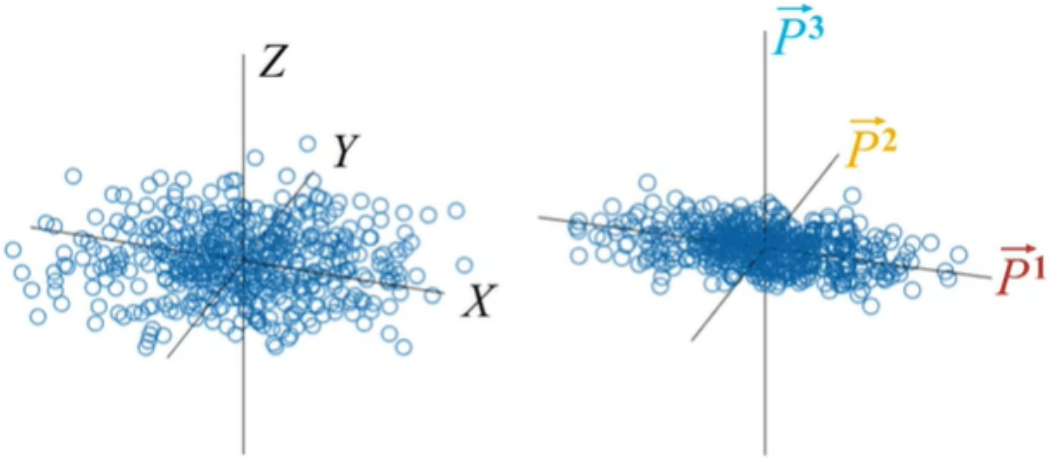

Figure 3.11: PCA: more than 2 dimensions

The next step is to reduce the dimensionality, that is to understand how to reduce the number of principal components. Below we see a possible technique. First of all it is necessary to rescaling all elements to obtain comparable scales. The second step is to place the components in the origin: this means rescaling all the features with mean equal to zero. At this point we are ready to reduce the dimensionality. On each new principal component we calculate the variance and fix a threshold variance for which we believe there may be information above it. All components that have variance above this threshold are kept, while all others are eliminated. If we remember, we have created the main components in order of variance, so the components will be eliminated starting from the last.

3.2 Tree-base Methods

Tree-base Methods, in particular, Regression trees are part of those models that fall into the category of artificial intelligence, particularly in machine learning. Although they are fully part of it, we can say that they are looking for something more. These not only content them with finding techniques to produce more and more accurate results, but give great importance to the interpretability of the result. Where the majority of machine learning algorithms are black boxes, , tree-based methods are driven by the goal of producing understandable results, on mostly any kind of data.

Below I propose a historical-bibliographic path to better understand how we arrived at the models illustrated in the next sections.

- Morgan and Sonquist [38] first developed a tree-based method called automatic

interaction detector (AID) for handling multi-variate non-additive effects on survey data.

- Based on AID technical improvements and software were developed in the following years by several authors [39, 40, 41, 42].
- Breiman [43, 44], Friedman [45, 46] and Quinlan [47, 48] were certainly the main figures who simultaneously and independently proposed very close algorithms to tree-based models.

Decision trees (and all tree-based methods) have been very successful over time. The factors of this attractiveness are many, but the main ones are the following.

- Tree-based methods can handle complex relations between inputs and outputs without a priori assumptions. They are non-parametric;
- Tree-based methods can manage ordered or categorical variables, or a mix of both;
- Tree-based methods are robust to noisy variables because implement feature selection;
- Tree-based methods can handle outliers or errors in labels;
- Tree-based methods are very easily interpretable.

Tree-based methods , in particular decision trees are at the basis of many state-of-the-art algorithms, including neural network and deep learning, where they are used as building blocks for composing larger models.

3.2.1 Definition of Decision Trees

Tree-based method are very simple if we analyse it geometrically. The idea behind the model is just a recursively partition. Starting from the input space $\mathcal{X}$ we have to approximate a partition, assigning constant prediction values $\hat{y} \in \mathcal{Y}$ to all objects $x \in \mathcal{X}$. In order to go into more detail, we need some definitions:

- A **tree** is a graph $G = (V, E)$ in which any two vertices are connected by exactly one path
- A **rooted tree** is a tree in which one of the nodes are the root. In our study, we consider that a rooted tree is a directed graph, where all edges are directed away from the root.
- If there exists an edge from t_1 to t_2 ($(t_1, t_2) \in E$) then node t_1 is called **parent** of node t_2 and node t_2 is called **child** of node t_1.

- In a rooted tree, a node is called **internal** if it has one or more children and **terminal** if it has no children. Terminal nodes are also called **leaves**.

- A **binary tree** is a rooted tree where all internal nodes have exactly two children.

Starting from the previous definitions, we can define a **decision tree** as a rooted tree. Usually this will be binary, but a design with multiple children is not ruled out. We can define the model as $\Phi : \mathcal{X} \to \mathcal{Y}$. Each node t of the tree is a subspace of the starting space $\mathcal{X}_t \subseteq \mathcal{X}$, reminding us that the node t_0 is equivalent to the space itself $\mathcal{X}$. Within each internal node t we create an ad hoc **split** s_t, where s_t belongs to a set of questions Q which lead to a choice of division. The questions in Q are like $x \in \mathcal{X}_A$*?*. In particular, the space $\mathcal{X}_t$ will be split in subspaces that represent its children. Two subspaces $\mathcal{X}_t \cap \mathcal{X}_A$ and $\mathcal{X}_t \cap (\mathcal{X} \setminus \mathcal{X}_A)$ are thus created: the first is the left child of t and the other in the right child of t. The leaves of the tree are the output of our model: $\hat{y}_t \in \mathcal{Y}$. This model is suitable for both regression and classification problems. If Φ is a classification tree, then $\hat{y}_t \in \{c_1, \ldots, c_J\}$ while if Φ is a regression tree, then $\hat{y}_t \in \mathbb{R}$. To better understand the idea depicted in the previous lines see algorithm 1.

Algorithm 1 Prediction of the output value $\hat{y} = \Phi(x)$ in a decision tree.

```
1: procedure PREDICT(Φ, x)
2:     t = t_0
3:     while t is not a terminal node do
4:         t = the child node t' of t such that x ∈ X'_t
5:     end while
6:     return ŷ_t
7: end procedure
```

Let us see an example, 3.12 shows a decision tree Φ composed by five nodes with the input space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 = [0; 1] \times [0; 1]$. We are solving a classification problem where $Y = c_1, c_2$.

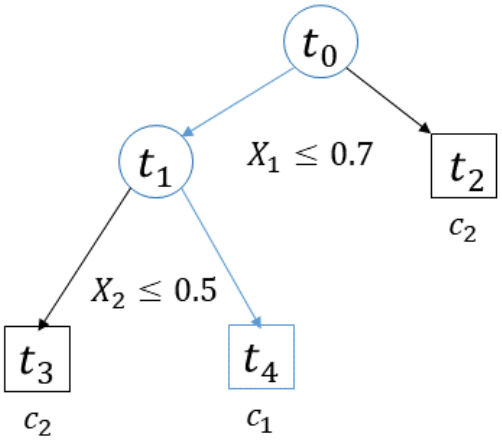

Figure 3.12: Decision Tree

t_0 represents the root node therefore $\mathcal{X}_{t_0} = \mathcal{X}$, that is $\mathcal{X}_{t_0}$ in the input space. All splits are binary. The first split is $X_1 \leq 0.7$, who allow us to divide the input space in two disjoint spaces $X_{t_1} \cup X_{t_2}$. The left child t_1 in the collection of all input element x such that $x \leq 0.7$, while the right child t_2 it is made up of all the elements x such that $x_1 > 0.7$.

In the same way t_1 is devided by the split $X_2 \leq 0.5$ which split $\mathcal{X}_{t_1}$ into two disjoint subsets $\mathcal{X}_{t_3} \cup \mathcal{X}_{t_4}$. The left child t_3 in the collection of all input element x such that $x \leq 0.5$, while the right child t_4 it is made up of all the elements x such that $x_1 > 0.5$.

The leaves of the tree t_2, t_3 and t_4 are labeled with an output value $\hat{y}_t$. They represent a partition of $\mathcal{X}$, where $\mathcal{X}_{c_k}^{\Phi}$ is the union of the subspaces $\mathcal{X}_t$ of all terminal nodes t such that $\hat{y}_t = c_k$. In particular, $\mathcal{X}_{c_1}^{\Phi} = \mathcal{X}_{t_4}$ while $\mathcal{X}_{c_2}^{\Phi} = \mathcal{X}_{t_2} \cup \mathcal{X}_{t_3}$.

This type of partition creates classes where inside we can find elements that are much more homogeneous with each other than they could be in the starting space. In particular, at the level of leaves $((\mathcal{X}_{t_3} \cup \mathcal{X}_{t_4}) \cup \mathcal{X}_{t_2})$ elements present in different nodes tend to be very different from each other. In the following sections we will see that this type of split leads to a partition made of rectangles in which the final prediction is made by propagating the instances through the nodes of the tree.

Looking at the graph theory, we can say that decision trees belong to a larger family of methods known as **induction graphs** [49]. In the models we are studying,

the structure is a directed acyclic and this allows the nodes to be divided and recombined.

3.2.2 Impurity Measure

Taking a metric as a reference point [50], let us define a measure such that the smaller this is, the better the predictions of each node t is. We call it impurity measure $i(t)$. The predictions are $\hat{y}_t(x)$ for all $x \in \mathcal{L}_t$, where $\mathcal{L}_t$ is the subset of all training sample $(x, y) \in \mathcal{L}$ such that $x \in \mathcal{X}_t$. As usual we start from the root node which contains all the elements of the learning set $\mathcal{L}$. The idea is to create decision trees that, following the iterative divisions, can have more and more pure nodes. Each node is divided into the two smaller nodes, that is into smaller subsets of $\mathcal{L}$, and for these nodes we look for an optimal split s so that the elements of the downstream nodes can be more homogeneous (purer) than the starting node. In all this search it must also be taken into account that we are looking for the smallest possible tree that can have a good generalization. The first step we take is to locally maximize the decrease in the impurity measure of the child nodes. Below we define this measure (**impurity decrease**) by remembering that we are making a binary split $s \in Q$ dividing node t into a left node t_L and a right node t_R

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R) \tag{3.1}$$

where p_L and p_R is the ratio $\frac{\mathcal{N}_{tL}}{\mathcal{N}_t}$ and $\frac{\mathcal{N}_{tR}}{\mathcal{N}_t}$ of learning samples from $\mathcal{L}_t$ going to t_L and t_R and where N_t is the size of $\mathcal{L}_t$. in the algorithm 2 we can see a formation of the general procedure described above. The Algorithm 2 is developed with binary splits.

Within algorithm 2 there are still some important points to discuss: finding good splits and knowing when to stop splitting.

3.2.3 Building Decision Tree for Regression

If we set a stopping criterion (3.2.4) and say that t is the leaf node, what we now need to do is classify t by assigning it a value of $\hat{y}_t$. The latter is the value that we consider output.

We now consider the node t as a model Φ defined on $\mathcal{X}_t \times \mathcal{Y}$ and we emphasize that minimizing the global error of the tree is equivalent to minimizing every single local value in the leaf nodes of each simplistic model, in fact

$$\begin{aligned} Err(\Phi) &= \mathbb{E}_{X,Y}\{L(Y, \Phi(X))\} \\ &= \sum_{t \in \tilde{\Phi}} P(X \in \mathcal{X}_t)\mathbb{E}_{X,Y|t}\{L(Y, \hat{y}_t)\} \end{aligned} \tag{3.2}$$

Algorithm 2 Greedy induction of a binary decision tree.

procedure BUILDDECISIONTREE($\mathcal{L}$)
    Create a decision tree Φ with root node t_0
    Create an empty stack S of open nodes $(t, \mathcal{L}_t)$
    S.PUSH($(t_0, \mathcal{L})$)
    while S is not empty **do**
        $t, \mathcal{L}_t = S$.POP()
        if the stopping criterion is met for t **then**
            $\hat{y}_t =$ some constant value
        else
            Find the split on that maximizes impurity decrease

$$s^* = arg \max_{s \in Q} \Delta i(s, t)$$

            Partition $\mathcal{L}_t$ into $\mathcal{L}_{t_L} \cup \mathcal{L}_{t_R}$ according to s^*
            Create the left child node t_L of t
            Create the right child node t_R of t
            S.PUSH($(t_R, \mathcal{L}_{t_R})$)
            S.PUSH($(t_L, \mathcal{L}_{t_L})$)
        end if
    end while
    return Φ
end procedure

where $\tilde{\Phi}$ is the set of leaves nodes in Φ and where the inner expectation (the joint expectation of X and Y is $\forall i$ such that $x_i \in \mathcal{X}_t$) is the local generalization error of the tree at node t. In this formula, a tree which minimizes $Err(\Phi)$ is a tree which minimizes the inner expectation at the terminal nodes. Looking for the best $\hat{y}_t$ in leaf nodes therefore means defining the best possible tree.

If we consider L as squared error loss, the inner expectation in Equation 3.2 is minimized by the expectation of Y in t:

$$\begin{aligned} \hat{y}_t^* &= arg\min_{\hat{y}\in\mathcal{Y}} \mathbb{E}_{X,Y|t}\{(Y-\hat{y})^2\} \\ &= \mathbb{E}_{X,Y|t}\{(Y)\} \end{aligned} \tag{3.3}$$

In order to find a solution for the equation, 3.3, we need the probability distribution $P(X,Y)$. But we make an approximation through the local generalization error:

$$\begin{aligned} \hat{y}_t &= arg\min_{\hat{y}\in\mathcal{L}} \frac{1}{N_t} \sum_{x,y\in\mathcal{L}_t} (y-\hat{y})^2 \\ &= \frac{1}{N_t} \sum_{x,y\in\mathcal{L}_t} (y) \end{aligned} \tag{3.4}$$

Where N_t is the number of elements in $\mathcal{L}_t$. Although we are analyzing a regression problem, we can say that the same reasoning are true for a classification problem. We estimate $P(\mathcal{X} \in \mathcal{X}_t)$ through $p(t)$ and approximate the local generalization error with $\frac{1}{N_t}\sum_{x,y\in\mathcal{L}_t}(y-\hat{y})^2$. At this point it can be seen that 3.2 is just the estimate of Φ. Therefore the 3.4 assignment rule minimizes the training error and not the true generalization error. However, it can be seen that the more a leaf node is split, the smaller the estimate $\widehat{Err}^{train}(\Phi)$ becomes.

3.2.4 When to stop

We can certainly say that the deeper the tree, that is the more splits there are in the model, the more accuracy on my training data is. While this is a positive aspect, we need to be careful that the model doesn't fit too much into the training data and then we risk it not working well on all possible data. This is the case in which the model would suffer from the baias of the learning data, that is the phenomenon we call overfitting. Therefore, it is necessary to know how to evaluate the right moment in which the error is minimized for all possible data, that is to find the optimal depth of the model in order to avoid the bias of the learning data. In other words, we are looking for the best rule to stop dividing nodes. We try to accurately formulate line 7 of the 2 algorithm. To achieve the final goal, as a

first step, we analyze all possible stopping criteria without taking into account the overfitting problems. The t node is defined as a terminal node when $\mathcal{L}_t$ can no longer be split, which happens in the following cases:

- When the output values of the elements in $\mathcal{L}_t$ are homogeneous. That is, if $y = y'$ for all $(x, y), (x', y') \in \mathcal{L}_t$.
- When the input variables x_j are locally constant in $\mathcal{L}_t$. That is, if $x_j = x'_j$ for all $(x, y), (x', y') \in \mathcal{L}_t$, for each input variable X_j. It is impossible to divide $\mathcal{L}_t$ into non-empty subsets.

There are many techniques for choosing the stopping criteria, but in most cases these are defined or integrated with heuristics. The partition is stop if $\mathcal{L}_t$ has become too small or if a good enough partition cannot be found. Some examples we see below:

- t is a terminal node if it is composed by less than N_{min} samples. (N_{min} is called *min samples split.*)
- t is a terminal node if the depth of the tree d_t is equal to a threshold d_{max}. (d_{max} is called *max depth.*)
- t is a terminal node if the total decrease in impurity is less than a fixed threshold β. That is, $p(t)\Delta i(s^*, t) < \beta$.
- t is a terminal node if there is no split such that t_L and t_R both has at least N_{leaf} elements. (N_{leaf} is called *min samples leaf.*)

Within all the stopping criteria highlighted above there are some parameters that must be defined in advance (N_{min}, d_{max}, β or N_{leaf}), i.e. they must be chosen in order to find the right compromise between a tree that is too deep and a tree that is too small. As previously highlighted, a too deep tree risks having a too high generalization error due to the overfitting problem. The opposite approach risks not using all the information in $\mathcal{L}$. Finding the correct parameter set is not easy. There are several techniques, especially of a computational type, but the cost in terms of time and resources is much more. In order to achieve this goal, various techniques guided by heuristics are used.

3.2.5 How to Make Splits

Another important criterion necessary for the creation of a decision tree is related to the optimal choice of the type of split $s^* \in Q$ of t. We definitely want to maximize the impurity decrease $\Delta i(s^*, t)$ (Line 10 of Algorithm 2).

Recalling the concept of split s, let us say that the latter divides the starting space of node t into disjoint subspaces, thus creating partitions of the starting space $\mathcal{X}_t$. These partitions are called children of t. A set of subsets is thus created such that each element of the starting space belongs to only one of the child subsets. We can therefore say that $\mathcal{X}_t$ is a disjoint union of subsets.

The starting space $\mathcal{X}_t$ is divided, in general, into a finite number of subsets, called children nodes $t_{i_1}, \ldots, t_{i_n}$ as shown in figure 3.13. In our case we will consider two subsets. Then we can say that s is a binary split and its left and right children are denoted by t_L and t_R.

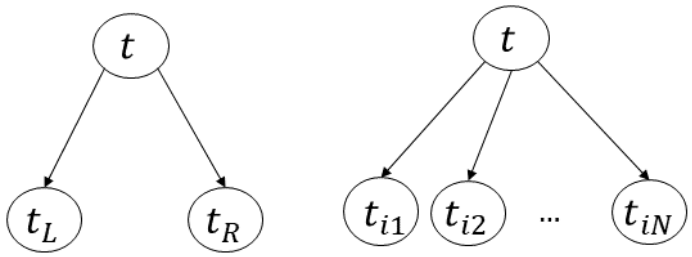

Figure 3.13: Binary split and multiway split

We define $\mathcal{S}$ as the set of all possible splits s that is all possible partitions of $\mathcal{X}_t$. If we want to define the cardinality of $\mathcal{S}$, we can say that the latter is infinitely large if at least one input variable has infinite cardinality. If $\mathcal{X}_t = \mathbb{R}$ there are infinite different partitions. As we said in the previous lines, our case study is restricted to binary partitions, so we can start excluding all those partitions that form an empty subset, infact it would result in $\mathcal{X}_t = \mathcal{S}$. So let us consider the best not empty partition s^* of $\mathcal{X}_t$. This can be approximate as looking for the best sample partition of the $\mathcal{L}_t$ node. Considering the node N_t and assuming that the inputs has distinct values for all samples, the number of partitions of $\mathcal{L}_t$ in k non-empty subsets is given by the Stirling number of the second type [51].

$$S(N_t, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-1} \binom{k}{j} j^{N_t}, \tag{3.5}$$

We thus get $2^{N_t - 1} - 1$ partitions if we only consider binary partitions. The optimal solution for finding the best partition should be to consider all possible partitions. As we can see from the previous formula, the computational cost would be unacceptable. Hypotheses are therefore defined that can lead to simplifications. An induction algorithm is defined such that s^* lives in a family $Q \subseteq \mathcal{S}$ of candidate splits of restricted structure. The set Q of splits is defined as binary splits on a

single variable and resulting in non-empty subsets of $\mathcal{L}_t$:

$$Q = \{s|s \in \bigcup_{j=1}^{p} Q(X_j)\} \tag{3.6}$$

If X_j is an ordered variable in $\mathcal{X}_j$, then binary splits on X_j are all binary non-crossing partitions s_j^v of $\mathcal{X}_j$:

$$Q(X_j) = \{(\{x|x_j \leq v\}, \{x|x_j > v\})|v \in \mathcal{X}_|\} \tag{3.7}$$

Taking into account the partition 3.6 into subsets $Q(X_j)$, the split $s^* \in Q$ is the best split defined on each input variable. It means,

$$s^* = arg \max_{s_j^*, j=1,\ldots,p} \Delta i(s_j^*, t) \tag{3.8}$$

$$s_j^* = arg \max_{s \in Q(X_j)} \Delta i(s, t) \tag{3.9}$$

Following this study, we can say that it all reduces to the definition and calculation of the equation 3.9. The partition defined in 3.7 is the most used partition type, but there are many other possibilities.

Certainly one of the most interesting possibilities is to replace a binary partition with the creation of a set of finite subsets. For example, if $\mathcal{X}_j$ takes into account L different values $b_1, \ldots, b_L$, then the partition on X_j splits $\mathcal{X}_j$ into L child nodes . Therefore, $Q(X_j)$ becomes:

$$Q(X_j) = \{(\{x|x_j = b_l\}|b_l \in \mathcal{X}_j)\} \tag{3.10}$$

3.2.6 Goodness of Partition

In the previous paragraphs we have highlighted several partition techniques. In this section we try to understand how good a particular splitting rules. In particular we study the impurity measures $i(t)$ for evaluating the goodness of partitions.

The technique used in the decision trees, i.e. the creation of successive partitions, reduces as we know the squared error loss on the training set. However, this phenomenon occurs only in classification problems. It rarely happens in regression problems. In this case it only happens when the average values of the output in the child nodes coincide with the average value of the output in t. Therefore the local resubstitution estimate turns out to be suitable for regression problems.

Our study takes in to account regression problems. So, for regression, the impurity

function $i_R(t)$ using local resubstitution estimate defined on the squared error loss is:

$$i_R(t) = \frac{1}{N_t} \sum_{x,y \in \mathcal{L}_t} (y - \hat{y}_t)^2$$

Looking from a different point of view, to better understand what we are saying, we can notice that this criterion corresponds to the variance within the node of the output value in t. Thus s^* is the partition that maximizes variance reduction $\Delta i(s,t)$ in the child nodes.

3.2.7 How to Choose Binary Partition

Until now we have shown the pillars to create an optimal partition rule, now we need to define the algorithm. In fact, we have highlighted Q families of partition rules with the criteria for evaluating their goodness through impurity measure. Knowing that Q is the set of binary split, we have shown that s^* is the best of the best s_j^* binary partitions defined on each input variable. All this leads us to the following algorithm.

Algorithm 3 Find the best split s^* that partitions $\mathcal{L}_t$.

```
1: procedure FindBestSplit(L_t)
2:     Δ = −inf
3:     for j = 1, ..., p do
4:         Find the best binary split s_j* defined on X_j
5:         if Δi(s_j*, t) > Δ then
6:             Δ = Δi(s_j*, t)
7:             s* = s_j*
8:         end if
9:     end for
10:    return s*
11: end procedure
```

3.3 Random Forest

The decision trees analyzed in the previous sections can be used as standalone models or they can be part of a more complex structure. An example is **Random forests**. The latter are formed by a set (called forest) of decision trees starting from a randomized variant of the induction algorithm described in section 3.2.

The decision tree is certainly a good candidate for all those models that are

created from single independent units, in fact they have a low bias and a high variance. This makes them interesting in all mean-related processes. The idea behind Random Forest is to insert random perturbations within the induction procedures of each individual decision tree. The most complex part of this process is being able to insert randomness while maintaining a low bias in the individual trees.

3.3.1 Random Forest algorithms

We can empirically verify that the average of several decision trees leads to a better final result than any tree taken individually. This fact was first noticed by *Kwok* and *Carter* [52]. This conclusion, in the work of the two authors, was reached by selecting from time to time different parts of the random forest and noting that these were comparable to the optimal partitions, one could say almost good equal.

In the works of *Breiman* [53] we can see shown that $\mathbb{E}_{\mathcal{L}}\{\Phi_{\mathcal{L}}\}$ has an expected generalization error less than $\Phi_{\mathcal{L}}$. This author is in fact one of the first to demonstrate that aggregate models produce greater accuracy. Using a technique called Bagging, we want to approximate $\mathbb{E}_{\mathcal{L}}\{\Phi_{\mathcal{L}}\}$ by merging models created with *bootstrap samples* [54] $\mathcal{L}^m$ (for $m = 1, \ldots, M$) taken from the training set $\mathcal{L}$. $\{\mathcal{L}^m\}$ replicates of $\mathcal{L}$, each element consisting of N cases (x, y), thought at random but with replacement from $\mathcal{L}$.

It is important to highlight that even if $|\mathcal{L}| = |\mathcal{L}^m| = N$, 37% of the couples (x, y) from $\mathcal{L}$ are on average missing in the bootstrap replicates. Moreover, as consequence of N extractions with replacement, it will never be selected with probability

$$(1 - \frac{1}{N})^N \simeq \frac{1}{e} \simeq 0.368 \tag{3.11}$$

The Bagging technique is one of the most used methodologies to create combinations of models, not only for decision trees. This procedure has proved to be very effective although in some cases it has the following defect. We start by assuming that the chosen model is not able to manage small changes in the starting set. This means that small changes in the training set can lead to big changes in the model, i.e. we are talking about an unstable model. *Bagging* builds models that are different from one *bootstrap sample* to another so they can take advantage of all the benefits that come with the average. However, it can happen that the learning set $\mathcal{L}$ is small, so subsampling could lead to a bias which, being very large, cannot be compensated by the variance. This inevitably leads to a lower accuracy of the result.

While there is less accuracy, we can certainly empirically show that Bagging often leads to better models. In fact, the latter method, although it is unable to decrease the bias, generates a strong increase in variance due to randomization. There are many different techniques to achieve the best possible result. For example in [52], *Dietterich* and *Kong* suggest randomizing the choice of the best partition of the t node by uniformly choosing one of the 20 best partitions of the node. The authors empirically show in [55] that randomizing in this way gives slightly better results than *Bagging* but only in low noise settings.

Depending on the domain of application, and consequently depending on the constraints, there are several attempts to find the best technique. One example is in the field of handwritten characters recognition. In this case it is easy to imagine that the number p of the input variables is very large. *Amit et al.* [56] define a variant of the algorithm suggesting to search the best partition in each node through a sub-sample of the variables. Let $K \leq p$ (called *max features*) be the number of variables taken into account for each node. Below we propose the variant to the 3 algorithm.

Algorithm 4 Find the best split s^* that partitions $\mathcal{L}_t$, among a random subset of $K \leq p$ input variables.

```
1: procedure FindBestSplitRandom(L_t, k)
2:     Δ = −inf
3:     Draw K random indices j_k from 1, ..., p
4:     for k = 1, ..., K do
5:         Find the best binary split s*_{j_k} defined on X_{j_k}
6:         if Δi(s*_{j_k}, t) > Δ then
7:             Δ = Δi(s*_{j_k}, t)
8:             s* = s*_{j_k}
9:         end if
10:    end for
11:    return s*
12: end procedure
```

This type of randomized algorithm generates different models each time. These are different, but all with a comparable degree of accuracy. This usually leads to a high variance that is managed by the mean, but also to a slightly higher bias. However, it is interesting to note how we can work on the K value to increase or decrease both the bias and the variance. When $K \to 1$ the bias increases but the variance also increases: the work done by the mean becomes effective. Conversely, if $K \to p$ the bias decreases, but also the variance decreases: this only has an advantage on the single tree. Overall it is not advantageous.

A further proposal to create a set of decision trees is always inspired by *Bagging* [50], but it introduces random subsets of [56] variables. *Ho* [57] proposes the Random Subspace (RS), where each decision tree is created on different and random subsets of the input elements. We can empirically verify that this type of technique leads to results comparable to the performance of the state of the art. As previously we note, randomization leads to an increase in bias which, however, can be controlled by the size of the randomly generated subsets.

At this point we try to combine the intuitions brought forward by the authors highlighted above. Breiman [58] combines *Bagging* [53] with the choice of random variables in each [56] node. Thus Random Forests (RF) was born. The combination of these techniques leads to a model with surprising accuracy, but the most interesting aspect lies in the fact that these techniques work for almost any type of problem. We can empirically demonstrate that the model is competitive with boosting and arcing techniques, although the latters work on the bias while the random forest works on the variance. Achieving this goal is certainly not the result of a single person, but we can consider *Breiman* the father of randomized trees. Probably his fame is also due to the fact that he not only provides theoretical conclusions in his articles, but also implements softwares. These softwares are open source and easily interpretable.

In addition, I want to mention Perfect Random Tree Ensembles (PERT) proposed by *Cutler* and *Zhao* [59]. The authors propose to randomly choose both the variables and the discretization threshold. Starting from a node t, X_j is randomly extracted using the 4 where $K = 1$ algorithm. The v cut-point is placed halfway between two random samples that are extracted with the procedure 5.

Algorithm 5 Draw a random split on $_j$ that partitions $\mathcal{L}_t$.

1: **procedure** FINDBESTSPLIT-PERT($\mathcal{L}_t, X_j$)
2: Draw $(x_1, y_1), (x_2, y_2) \in \mathcal{L}_t$ such that $y_1 \neq y_2$
3: Draw α uniformly at random from [0,1]
4: $v = \alpha x_{1,j} + (1-)x_{2,j}$
5: **return** s_j^v
6: **end procedure**

In this type of algorithm we proceed to create partitions until different samples can be taken or until it is no longer possible to make partitions. This method is technically very simple due to the fact that calculation of the impurity measure is not required when creating the partitions. It has been experimentally proved [59] that *PERT* is almost as good as Random Forest. The difference is that the

random trees in PERT are generally larger.

Chapter 4

Projects as Solution

Starting from the awareness of our business goal and the availability of free-text and multiple-choice questions resulting from questionnaires, in this chapter we will cover all steps that allow us to have a detailed knowledge about customer experience and to have mastery over actions that make us possible to increase customer satisfaction.

The first step is to analyze multiple-choice question in relation with *Net Promoter Score*, that is to understand if connections exist between NPS and a particular segment of the business taken into account in the comments forms.

The second step is created to understand the type of relations and the correlations between a business action and NPS passing through the customer perception. In order to achieve this goal we will create an *NPS-Simulator*.

The last part arises from the need to find out new factors that can influence the customer satisfaction and are not taken into consideration in the multiple-choice questionnaire. Starting from this awareness we will analyze free-text questions that are not driven from the company, but where customers have the freedom to express their opinion on any issue.This analysis includes two steps. We need not only to understand which topics are present in the customers' answers, but we also want to know if a topic is mentioned as a problem or as a strength.

4.1 NPS Simulator

In order to create the NPS-simulator we use knowledge elaborated on the sections 3, 3.1 and 3.3. We start from data preprocessing to arrive at new NPS prediction passing through the definition of the model without dwelling on theoretical aspects. However, we will focus on a very important aspect not analyzed in the previous

chapters: training and model evaluation.

4.1.1 The data

Having in mind the idea of understanding the NPS with a data-driven approach, we want to find the data that describes the customer perception of all possible aspects around the company during the entire experience with the brand. The only tabular data available is the collection of comment forms.

Let us start with multiple-choice questions. As we have seen in the first chapter, a singular comment form consists of 90 multiple-choice questions with a nominal scale from 1 to 10. Where 1 means *I am not satisfied at all* while 10 means *I am absolutely satisfied*.

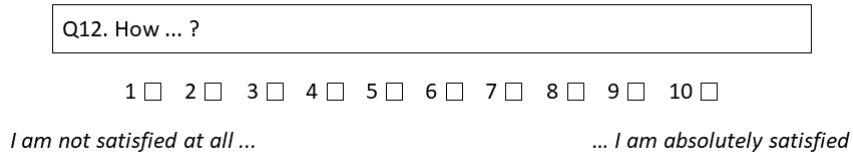

Figure 4.1: Multiple choice question

The client compiles the form after having totally concluded his experience with the company.

We have 15.000 comment form available to be analyzed, in particular we have 1.350.000 multiple-choice questions. The comment-forms are distributed over a period of two years, include a range of ages between 18 and 90 years and the gender is distributed almost homogeneously.

There exists another multiple-choiche question, we call it Q_{NPS}, that in used to calculate the total NPS. Our goal is just to analyze relations between all multiple-choice questions $Q_1, \ldots, Q_{90}$ and Q_{NPS}. These questions $(Q_1, \ldots, Q_{90})$ matches all factors that influence the customer experience, or rather all the factors that business experience suggests influence the customer perception.

4.1.2 Data preprocessing

In the previous section we have highlighted all the available data. Now it is necessary to process and organize it to ensure that it is suitable to be included in a forecast model.

Preprocessing can be summarized in 3 main steps:

- **Data structuring**: starting from all available data we need to reorganize it in order to be suitable for the model. For example we will create tabular data.
- **Data cleaning**: often the data collected contains misleading information like typos or missing values. Therefore it is necessary to adopt techniques that are going to act as filters on values.
- **Feature engineering**: each feature has different impact on what we want to predict. Some can have no impact, others may have an influence that is difficult to understand if not studied by different points of view. Therefore, a manipulation of the features is necessary to obtain the maximum result.

All numerical data we have are 15.000 questionnaires composed by 90 questions answered from 1 to 10 ($Q_1, \ldots, Q_{90}$) and a question Q_{NPS} with answers from 1 to 10. We organize data as a tabular data with 15000 records and 91 features.

	Q_1	...	Q_{90}
1	9	...	1
2	4	...	10
...	...	...	...
15000	5	...	9

Q_{NPS}
9
6
...
6

Figure 4.2: Data Structuring: Tabular Data.

After that, we need to check if all values of the dataset are as we expect: no typos and no missing values. Customer can not insert values different from numbers from 1 to 10 because forms are totally electronic, but not the entire process from client to us is totally automated. Therefore we check all values and delete all numbers different from what we expect. Regarding missing values we fill them with a mean calculated over all values of that particular question (feature). However, if a questionnaire (a dataset record) has more than 40% of the empty questions (features), then this is not taken into consideration and removed from the dataset.

Feature engineering is an important step in preparing our data for predictive modeling. We want to uncover hidden trends in our data for the model to identify. We can improve a models predictive power by applying our domain knowledge

to generate useful features. As we have seen in 3.1 the first step is a statistical analysis based on correlation in order to isolate features that have no influence on what we want to predict and features that are redundant, that is, we delete all features (except one) that have too high correlation between them.

Figure 4.3: Features selection: Correlations

After that, we use a *Principal Component Analysis* (3.1.4) in order to generate a new feature set $(F_1, F_2 \dots)$ that can help us capture the maximum amount of variance with the fewest number of features.

After the previous data preprocessing, final data available for predictive model consists in a dataset of 14.000 records with 30 features $(F_1, \dots, F_{30})$ as predictor and a singular value per record to be predicted (Q_{NPF}).

	F_1	...	F_{30}
1	9	...	1
2	4	...	10
...	...	...	...
14000	4	...	9

Q_{NPS}
9
6
...
7

Figure 4.4: Data After Preprocessing

4.1.3 Predictive Model

in the previous section we have focused on preparing the data to extract as much information as possible and make it as suitable as possible for a predictive model. Now we need to find the best architecture in order to achieve our goal. It consists in three steps.

- **Architecture:** starting from literature, we need to isolate the best model, or more precisely, the best composition of different predictive models that we suppose fit our data and our goal.
- **Suitable Hyperparameters:** once the right architecture has been found, it is necessary to find the right set of parameters for models used.
- **Models Evaluation:** evaluating the forecasts with the exact result is certainly necessary to decide if the work carried out can be acceptable or we need to restart from step one and evaluate another type of architecture.

Starting from architecture, the first algorithm we can test is Linear Regressor, but we have to remember we deal with categorical features: this fact drive us to choose a tree based method. This project is totally focuses on this type of model, more precisely on Random Forest. Another reason pushes us to choose this kind of algorithm is its interpretability, although there exists a variety of models that may also perform better. Interpretability is actually a primary factor, especially because we are talking about a business project.

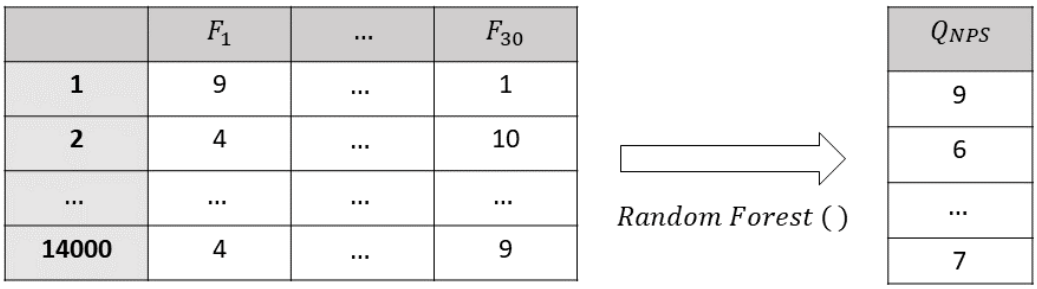

Figure 4.5: Predictive Model

One of the most important steps when we choose to create a well performing model is the choice of hyperparameters that define it. The majority of models have a variable quantity of parameters that have to be chosen at the beginning. Each model has a different optimal set of hyperparameters and this set depends also by the data we have and the type of problem we want to solve. There exist a huge variety of methods to obtain this set, but each method is based on exploration of the hyperspace which the parameters live in. The main two ways to achieve this

goal are: empirically looking for the best combination of parameters and by relying on Bayesian Optimization.

The moment of performance evaluation is the end of the model development. Now we can have an idea of what kind of result we managed to get from predictions. Different evaluation metrics can be created in order to evaluate the outputs of the models selected, but we remember that all of them take into consideration the distance between the real value and the prediction of the model. As we will see in detail in 4.1.4, we are creating a supervised learning method: when we are developing, we can use real values of the output if we need to compare them with the output of the model. These are historical data which the model is built on. Most used metrics are **Mean Relative Error** (mre) and **Mean Absolute Error** (mae)

$$mre = \frac{\sum_{i=o}^{n} \frac{|M(\hat{x}_i) - \hat{y}_i|}{\hat{y}_i}}{n} \tag{4.1}$$

$$mae = \frac{\sum_{i=o}^{n} |M(\hat{x}_i) - \hat{y}_i|}{n} \tag{4.2}$$

where M represent the model and $(\hat{x}_i, \hat{y}_i)$ the singular element of a test set of size n.

4.1.4 Training and Test

To sum up, what we have now is a dataset composed by 14.000 records with 30 features $(F_1, \ldots, F_{30})$ as predictors and a singular value per record to be predicted (Q_{NPF}).

In computer science generalization usually means the ability of an algorithm to be effective across various inputs. For example it means that the Machine Learning models do not diminish their performance if new data from the same distribution are used.

For us, as humans, generalization is natural, since we were born we are used to it. We can definitely recognize a car although we didn't see this brand before. This could be very complicated for an algorithm. We try to solve this problem in Machine Learning models. Proving the algorithm's ability to generalize is an important step of our process. We solve this problem using Cross-Validation (CV).

Cross-validation is one of the most used technique in order to evaluate and test the performances of machine learning models. The reasons of its popularity is definitely

the ease of understanding the result, the ease to implement the algorithm, the high frequency of having low bias. All this allows us to select an appropriate model comparing different choices.

There exist a lot of different ways to cross-validate a model: all of them have a similar structure:

1. Divide the dataset (two parts): training and testing.
2. Train the model on the training set.
3. Validate the model on the test set.
4. Repeat 1-3 steps a couple of times. This number depends on the CV method we are using.

There are a huge number of CV techniques.

Hold-out cross-validation is very common and very simple. We certainly use this technique even without actually knowing its name. The algorithm of hold-out:

1. Divide the dataset into two parts: training set and test set. commonly, 80% of the dataset for training set and 20% for test set but we could choose any splitting that suits us best.
2. Train the model on the training set.
3. Validate on the test set.
4. Save the result of the validation.

Hold-out method is suitable on large datasets and it requires training the model once.

Figure 4.6: Cross Validatio: Hold-Out

Unfortunately, Hold-out has one major drawback. If we consider a dataset where data are not completely distribution-wise, training set and test set could be very

different, or better to say, training set is not a good representative of chosen data for test set. This type of problem cannot be solved with this algorithm, therefore this technique is often considered inappropriate or rather not sufficient to have a good degree of accuracy.

Aware of the previous considerations we use **k-Fold** CV for our project. k-Fold Cross-Validation is an algorithm that minimizes the issues highlighted for hold-out. k-Fold uses a completely different splitting method in order not to be dependent on the perfectly uniform distribution of the data. The algorithm of k-Fold technique:

1. Choose a number of folds k. Commonly, k is 5 or 10 but we can choose any number which is less than the dataset's length.

2. Split the dataset into k equal parts (called folds).

3. Choose $k - 1$ folds which will be the training set. The remaining fold will be the test set.

4. Train the model on the training set. On each iteration of cross-validation, we must train a new model independently of the model trained on the previous iteration.

5. Validate on the test set.

6. Save the result of the validation.

7. Repeat steps 3, 6 k times. Each time use the remaining fold as the test set. In the end, we should have validated the model on every fold that you have.

8. To get the final score average the results that we got on step 6.

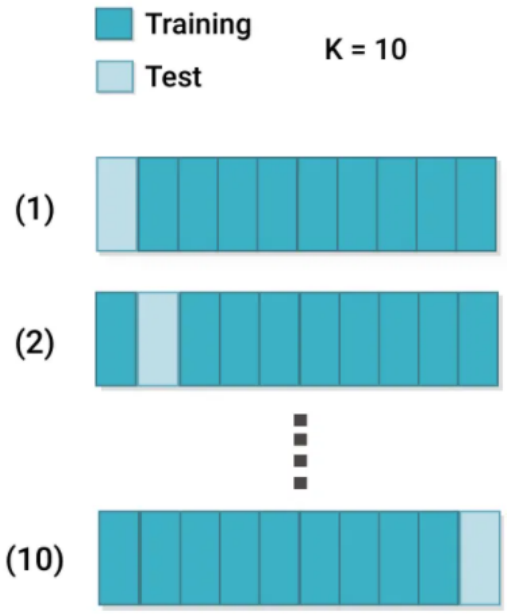

Figure 4.7: Cross Validatio: k-Fold

In general, k-Fold is always better than hold-out: k-Fold performs training and testing different parts of the dataset. Moreover we could make use of another useful step in this technique is to iterate k-Fold methods on different subset of our dataset: combining these scores we can to create an overall score more accurate and robust.

k-Fold method has also a disadvantage. The more we increase k the more expensive and time-consuming the model becomes from the computational point of view.

4.1.5 Simulator and Business

Until now we have created a useful model in order to predict an answer to a particular question Q_{NPS} staring from a comment form $(Q_1, \ldots, Q_{90})$. In particular we are detecting relations between different segments of the business and the brand pereption overall, that is we are predicting overall perception from e sequence of well defined parts of the customer experience.

Let us try to make this tool useful for the business side. In order to do this we create an **NPS-Simulator**.

The idea starts from the needs of evaluate the effects of business actions on

customer satisfaction, in particular on NPS. It is necessary knowing these consequences before the actions are actually put in place.

Starting from a business action, the user must estimate segment of the company influences, what kind of person impacts, how much it impacts and in what percentage. In particular he must know:

- Business segment: the particular question associated (Q_i).
- Target cluster: the action will impact *promoter* or *neutral* or *detractor*.
- Action score: the new score between 1 and 10 associated to Q_i
- Action penetration: the percentage of impacted people among those involved.

Starting from trained model and all data available, we select Q_i, the target cluster and we change the answer score to a percentage of elements of this group that corresponds to the Action penetration. We obtain a new dataset ready to be fed to the predictive architecture described above. We can now have a new Q_{NPS} answers and in consequence a new NPS.

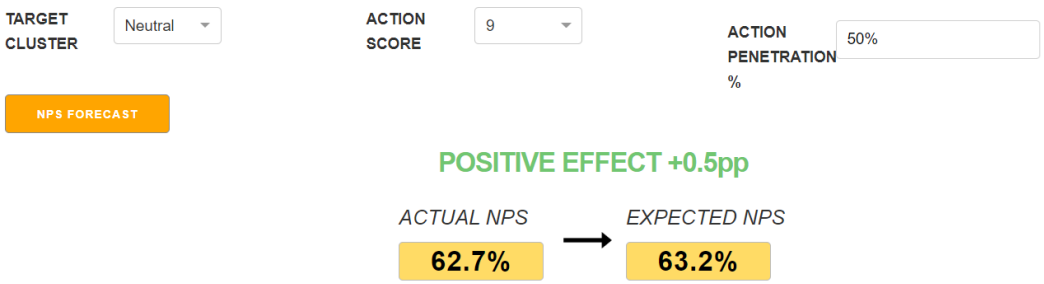

Figure 4.8: NPS Simulator

Moreover we have developed a dashboard that consists of an introductory support section for choosing the optimal parameters to be included in the simulator, see figure 4.9. A detailed analysis of the data available gives the company the possibility to direct its efforts in the direction that gives the possibility to maximize the increase in NPS.

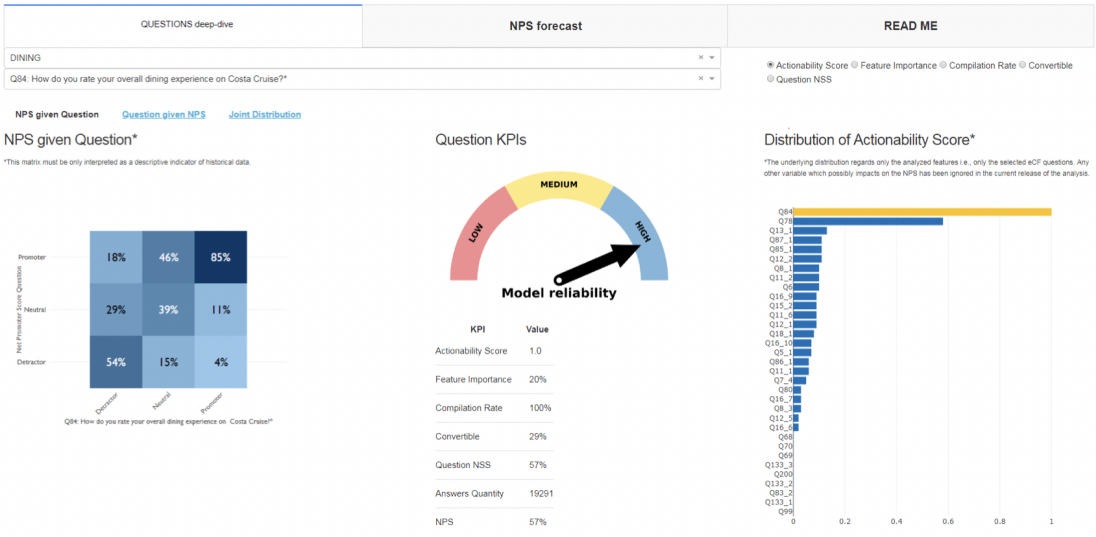

Figure 4.9: NPS Dashboard

Within a company, there are many possible actions to increase customer satisfaction. Even having the possibility of using a simulator, often we do not have a clear idea of the direction to take, or rather, of the sector to be investigated and how to analyze it. In our case, all this translates into the inability to choose the appropriate input parameters to obtain valuable information from the simulator.

In figure 4.9 we can see the presence of three sections. First we need to select the domain of interest (ex: Dining, Hotellerie, ...) and the particular question of the questionnaire (ie the particular sector to be analyzed). The first section helps the user to understand the distribution of customer satisfaction, by relating the question analyzed and the NPS question. In the second section, some KPIs are extracted. They help the user to understand whether working on this particular question can have a significant impact on the NPS. Here the individual KPIs are highlighted and a metric called *actionability-score* is also created. The latter summarizes the impact that this sector can have on the NPS. The third section compares all questions from a particular domain based on the *actionability-score*.

4.1.6 Data Collection Process for Validation

At the end of the test phase, a validation phase is required before publishing the platform into production. Within the company, such a decision-making process was not present before, so there is no historical data necessary for the validation of the model. It is necessary to study a data collection process that takes into account business actions and consequences at the NPS level. Such a process takes a long

time because actions are not carried out on a daily basis. Unfortunately, there was not the time necessary to collect data, but the process was nevertheless designed and implemented.

Looking back at what should happen: the user analyzes the dashboard to understand which outlet to intervene on to increase NPS. Now a business action is decided and it is translated into the input parameters of the model. Before describing the process, a preliminary consideration is necessary. It is true that a business action can be translated into the input parameters of the model, but it cannot be ruled out that this affects other sectors and groups of people not taken into consideration. Therefore, it is not possible to carry out a data collection in which only the input variables, the predicted NPS and the real NPS (NPS_r) are taken into account.

We therefore define two types of predicted NPS:

- Initial predicted NPS, NPS_{pi}, dependent on Business segment, Target cluster, Action score and Action penetration as defined in the previous section.

- Total predicted NPS, NPS_{pt}, dependent on Business segment, Target cluster, Action score and Action penetration, but also dependent on all customers answers after a certain business action.

Two types of errors are defined as follows:

$$err_i = NPS_r - NPS_{pi}$$

$$err_t = NPS_r - NPS_{pt}$$

err_t takes into account only the error that the model, the simulator brings with it, while err_i also takes into account the human error when user try to estimate what type of customer and sector can be impacted by a concrete action.

Below is a dataset that is being created for data collection.

Business Action	Business segment	Target cluster	Action score	Action penetration	NPS_{pi}	NPS_{pt}	NPS_r
Cleaning 3 times...	Dining	Neutral	8	50%	61,4	62,1	62,3
....	...	...	...	...	...	...	...

Figure 4.10: NPS Validation Dataset

4.2 Topics from Free Text

The aim of this section is to extract meaning from free text. The idea behind of this purpose is to find out the themes present in texts, understand what are the points that the writer wants to highlight and convey to the reader.

The steps are mainly three: to process the data by dividing it into elementary parts, to rearrange parts to assign meaning and, finally, to extract the general meaning of the portion of the text starting from the meaning of the individual parts and the relationships between them.

4.2.1 The data

All available data we have is a dataset of comment forms. Each questionnaire is composed not only of multiple choice questions but also of free text questions. It is precisely the latter that are of interest to us at this time.

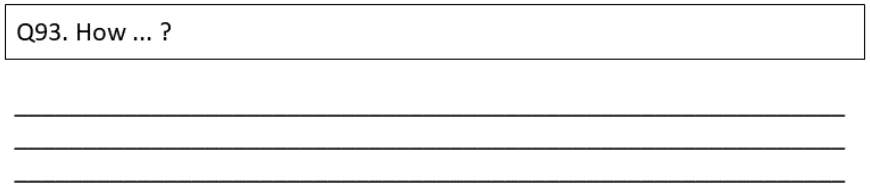

Figure 4.11: Free text question

The client compiles the form after having totally concluded his experience with the company.The customer is totally free to enter any type of information and comment within the answers we consider.

In each questionnaire we have 3 free text questions Q_{91}, Q_{92}, Q_{93}. We have 15.000 comment form available to be analyzed, in particular we have 45000 answers. The comment-forms are distributed over a period of two years, include a range of ages between 18 and 90 years and the gender is distributed almost homogeneously.

It is important to highlight that answers are in many different languages, these are five: Italian, English, French, German, Spanish. So it is to take this aspect into consideration for many aspects related to the result we want to achieve.

4.2.2 Data preprocessing

The first step we have follow is to create a sequence of different elements in each answer: in particular we choose a sequence words. We start like this because is natural to think of a text as a sequence of words and we can think of a word as a meaningful sequence of characters.

Let us start with Tokenization. So we can think of a token as a useful unit for further semantic processing. In the following examples we will consider a token as a word. We also split by punctuation.

There are other steps to improve tokenization, for example we could use rules related to the grammar of the language, such as *'s* in English. However, we cannot go further because we must remember that there are five different languages in our dataset.

The second step is normalization. Just to remember, we want the same token for different forms of the word like *mouse* and *mice*. The **normalization** is the process where we merge two different tokens with same meaning in a single one.

As process of normalizing we use lemmatization. Having as an input a word, we return the base or dictionary form of this word, which is known as the *lemma*.

It is always important to remember that we have more than one language. So, to achieve our goal we need to draw from 5 different dictionaries: English, French, Italian, Spanish and German.

4.2.3 Words embeddings: W2V

This section is focus on on making words mean, we want to reorganize the dictionary in a mathematical way. We represent this meaning by vectors: words with similar meaning have similar vectors. We use a technique called Word2vec.

We consider the dataset of all free text answers. In order to achieve the W2V representation of the words we have the following steps:

- Text vectorization with Pointwise Mutual Information (PMI).
- Matrix factorization with Singular Value Decomposition (SVD).

let us create a matrix of words. We star by a square matrix with the words on the rows and columns. $(n_{u,v})$ represents the number of times two words are present in the same sentence, so we can understand whether two words (u, v) are randomly

co-occurrent or not. In each entry we calculate the PMI as follows:

$$PMI = \log \frac{p(u,v)}{p(u)p(v)} = \log \frac{n_{u,v}}{n_u n_v} \tag{4.3}$$

where (u, v) is a couple of words, $p(u, v)$ is the probability of having u and v in the same answer of the Questionnaire and $p(u)$ is the probability of having u considering all answers. We can translate all simply in count: $n_{u,v}$ is the number of times we find u and v in the same answer and n_u is the number of times we find u in all documents.

	Love	...	Trip
Love	9	...	1
Restaurant	4	...	10
...	...	...	...
Trip	4	...	9

Figure 4.12: Text Vectorization

We use this technique because we want to highlight paradigmatic parallels and penalize stop words or too popular words (2.4.1).

As previously expressed, starting from the idea of associating a vector to each word, we want to calculate how similar two words are, by calculating the difference in their cosine. Starting from the matrix just calculated, we can see that the latter is very long and very sparse. For this reason we use techniques to reduce the size. What we are looking for is a matrix of about 300 rows. We have lots of different options how to do this factorization, but we use a Singular Value Decomposition (SVD) (2.4.2) and we obtain a matrix as in 4.13.

	X_1	...	X_{300}
Love	2	...	4
Restaurant	1	...	5
...	...	...	...
Trip	3	...	3

Figure 4.13: W2V: Matrix Factorization

Computing SVD to the PMI matrix we get somehow, dense and low dimensional vectors (Word2Vec).

4.2.4 Topics extraction: LDA

Starting from some text, we are looking for topics in each document (part of a text). We say that the latter is described with topics present in these documents: we are creating a new representation. . We are given a text collection of words, the counts n_{wd} of how many times every word w occurs in every document d and what we are looking for is two probability distributions. The first is the probability distribution over words for topics t

$$\phi_{wt} = p(w|t)$$

and the second is the probability distribution over topics for documents.

$$\theta_{td} = p(t|d)$$

Topic is just a probability distribution we need.

We use **Latent Dirichlet Allocation** [31] model. This is not very different from PLSA model. It introduce *Dirichlet priors Dir* for ϕ and θ parameters.

$$Dir(\phi_t|\beta) = \frac{\Gamma(\beta_0)}{\prod_w \Gamma(\beta_w)} \prod_w \phi_{\beta_{wt}}^{\beta_w - 1} \qquad \beta_0 = \sum_w \beta_w, \beta_t > 0$$

where $\phi_t = (\phi_{wt})_{w \in W}$ $\theta_d = (\phi_{td})_{t \in T}$.

It is important to note that, using LDA, there is not a set of fixed parameters, but it is necessary to consider distributions of parameters. It can now be understood why as a result we obtain two distributions over parameters. This is *posterior distribution* and it will be also Dirichlet but with some other hyperparameters.

In order to train this algorithm, we use expectation–maximization (EM) algorithm (2.5.2). EM-algorithm is a very useful algorithm when we have observable data and hidden variables. Building our topic model means to repeat *E-step* and *M-step* iteratively. We collect data, we create probability distribution of topics with current parameters; after that we need to update parameters relying on probabilities of topics and then compute this again and again.

Let us remember that when we apply LDA algorithm it is necessary to fix a priori the number of topics we are looking for. So we need to choose the best number of topics in order to maximize the model performance. The most used measure to evaluate this model is *coherence*. Topic Coherence measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic. These measurements help distinguish between topics that are semantically interpretable topics and topics that are artifacts of statistical inference. The goal is to maximize coherence minimizing the number of topics, for example in the figure 4.14 the optimal number of topic is 8.

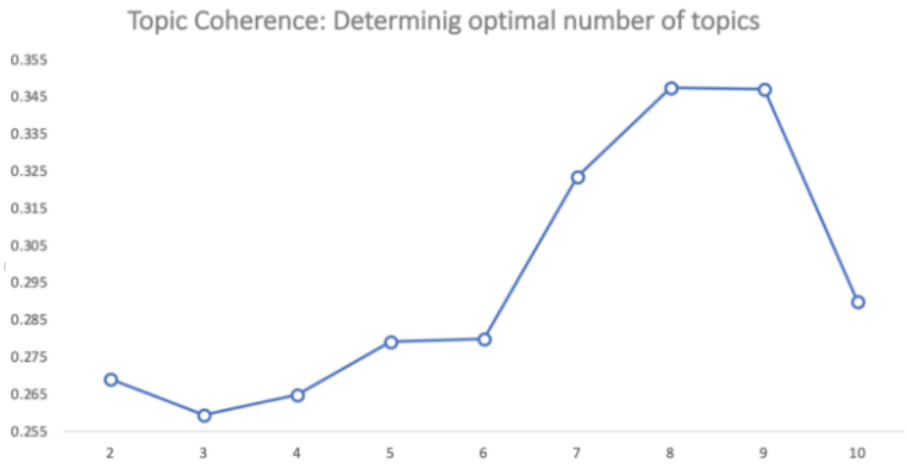

Figure 4.14: LDA Coherence

4.2.5 Zoom in as result

Starting from raw data, or better, from all answers with words vectorized via w2v, the first step is to apply LDA fixing the number of topic to five. This decision is driven by literature and experiments and the awareness that a first subdivision can take place by language, because in texts we have 5 different languages. As we can see in 4.15, the first clustering has 5 topics.

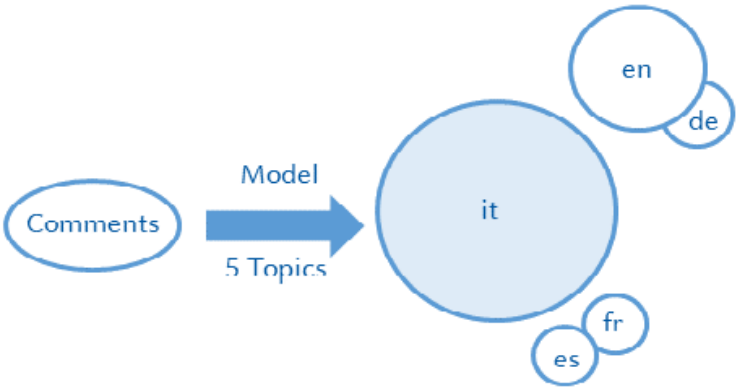

Figure 4.15: LDA: Zoom Languages

once the first five groups have been obtained, we isolate each group and deepen the analysis on each one. In particular, we start from all the Italian texts and apply LDA by choosing the number of topics to 3, see 4.16.

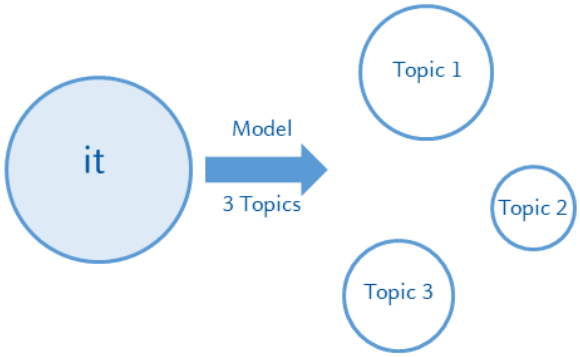

Figure 4.16: LDA: Zoom Topics

This type of methodology is interesting because we can choose how much to go into detail on each topic (see 4.17). In particular, in our case we will implement only one further step.

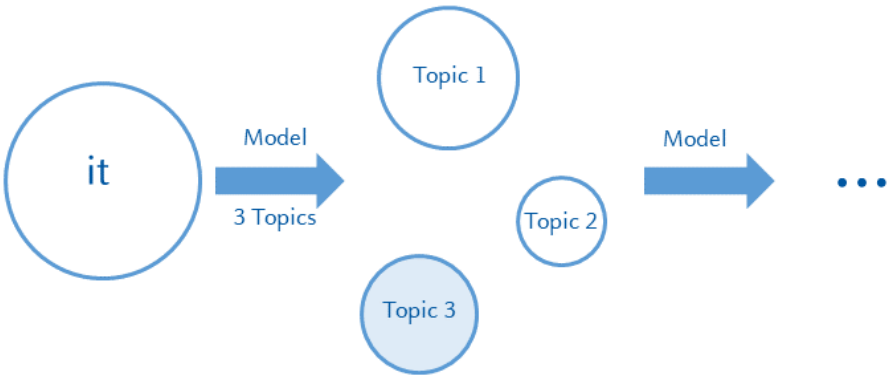

Figure 4.17: LDA: More Zoom

4.3 Sentiment from Free Text

Another information we need to understand is knowing if a particular comment has a positive or negative connotation. This is the first step to achieve the real meaning of a sentence. In this section we will analyze one of the techniques most advanced and utilized: Convolutional Neural Network over time. we will cover the entire process, starting from data collection and preprocessing to get sentiments, passing through model training.

4.3.1 The data

In order to achieve the goal highlighted, we collect the same data studied in 4.2.1. This is a dataset of comment forms, in particular we select free text questions from each questionnaire.

Just to recap, the customer compiles the form after having totally concluded his experience with the company and he is totally free to enter any type of information and comment within the answers we consider.

In each questionnaire we have 3 free text questions Q_{91}, Q_{92}, Q_{93}. We have 15.000 comment form available to be analyzed, in particular we have 45000 answers. The comment-forms are distributed over a period of two years, include a range of ages between 18 and 90 years and the gender is distributed almost homogeneously.

It is important to highlight that answers are in many different languages, these are five: Italian, English, French, German, Spanish.

The data reorganization process in order to make it suitable to be processed by a model is the same used in 4.2.2.

- Tokenization: we consider a token as a word. We also split by punctuation, but we do not use rules related to the grammar of the language because we have five different languages in our dataset.
- Normalization: as process of normalizing we use lemmatization. Having as an input a word, we return the base or dictionary form of this word. It is always important to remember that we have more than one language. So, to achieve our goal we need to draw from 5 different dictionaries: English, French, Italian, Spanish and German.
- Words embeddings: we want to get the meaning of words. We represent this meaning by some vectors, in such a way that similar words have similar vectors in the sense of cosine difference. We use a technique called Word2vec. In order to obtain W2V we follow two steps: text vectorization with Pointwise Mutual Information (PMI) and matrix factorization with Singular Value Decomposition (SVD). Computing SVD to the PMI matrix we get somehow, dense and low dimensional vectors (Word2Vec)

4.3.2 Sentiment Analysis with CNN

Extract a sentiment (positive or negative) from each free text comment is what we are doing in this section. There exist a lot of algorithms, but our approach is creating a neural network over *embeddings*.

In each comment, for each word, we take a row that represents a *word2vec* embedding of length 300. Above the latter we build an algorithm based on neural networks. The first problem we have to solve is the management of *n-grams*. To achieve this purpose, we actually use convolutional filters that have the size (length of the embedding) $\times$ (n of *n-grams*). In particular we use 3 kinds of filters: for 3-grams, for 4-grams and for 5-grams.

A single filter captures a single meaning of the word in relation to the context. What we need is to grasp different meanings depending on the context, but the word is always the same: we manage many *n-grams*. The filters we use are are *1D convolutions* as we just need to move our window in one direction only: over time, that is, we are interested in maintaining the ordering of the words. For purely calculation reasons it is necessary to add some padding so that we can have the same output and input size. What comes out of these metrics then undergoes a convolution. After using the filters over the whole text, what we can do is to select

the *maximum activation value* over all convolutions. This value is considered the result of our convolution and is called *maximum pooling over time*.

We take an input sequence, a convolutional window with size of three by the number of variables of embedding, we convolve with that filter sliding in one direction and then we take the maximum activation: this is our output.

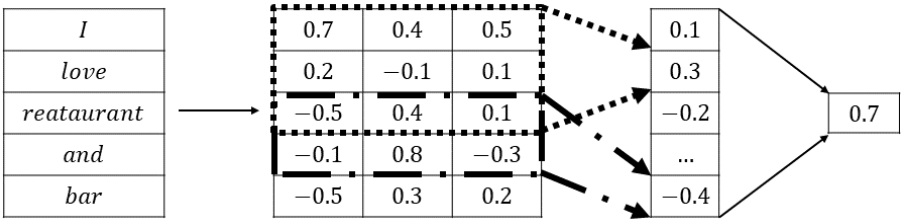

Figure 4.18: CNN Over Time

To sum up, we use the filters of size three, four and five, in order to collect information about three, four and five grams. For each *n-gram* we create 100 filters. It means we have 300 outputs. This output represent an embedding of our input. At this point we want to convert our input sequence into a vector of fixed size: we apply some *dense layers* and, in particular, we apply *multi-layer positron* over those 300 features. The last step is to train the model whether we need to get the sentiment of the comment. This is a binary classification problem or, to better say, a regression problem.

4.3.3 Training for CNN

Our CNN consists of a convolutional layers, a Max Pooling over time layer and a multi-layer positron with Softmax function. We have two classes: positive or negative.

Training a neural network means follow these two phases:

- A forward phase, where the input is processed through the network. During the forward phase, each layer store any data it need for the backward phase. This means that any backward phase must be related and anticipated by a corresponding forward phase.

- A backward phase, where gradients are backpropagated (**backprop**) and weights are updated. During the backward phase, each layer receive a gradient and return a gradient. It receive the gradient of loss (L) with respect to its

outputs (y) ($\frac{\delta L}{\delta y}$) and return the gradient of loss with respect to its inputs (x) ($\frac{\delta L}{\delta x}$).

We start from the end and return towards the beginning, this is backprop.

We remember the definition of **cross-entropy loss**:

$$L = -\ln p$$

where p is the predicted probability. The first thing we need to calculate is the input to the multi-layer positron's backward phase, $\frac{\delta L}{\delta(y_s)}$, where y_s is the output from the multi-layer positron.

$$\frac{\delta L}{\delta(y_s)} = -\frac{1}{p}$$

Before implementing the backward phase, we need to perform the forward phase and store three useful information for backward phase:

- The input's shape before we reduce it.
- The input after we reduce it.
- The values passed in to the multi-layer positron.

The first step is to derive the gradients for the backprop phase. Deriving the input to the Softmax backward phase, we have: $\frac{\delta L}{\delta(y_s)}$.

We need the gradients of loss against weights, biases, and input:

- We use the weights gradient, $\frac{\delta L}{\delta w}$, to update our layer's weights.
- We'll use the biases gradient, $\frac{\delta L}{\delta b}$, to update our layer's biases.
- We'll return the input gradient, $\frac{\delta L}{\delta x}$.

In order to calculate these three gradients, we need to derive three results: the gradients of totals against weights, biases, and input. The equation is:

$$t = w * x + b$$

$$\begin{aligned} \frac{\delta t}{\delta w} &= x \\ \frac{\delta t}{\delta b} &= 1 \\ \frac{\delta t}{\delta x} &= w \end{aligned} \tag{4.4}$$

Joining everything together:

$$\begin{aligned}\frac{\delta L}{\delta w} &= \frac{\delta L}{\delta y} * \frac{\delta y}{\delta t} * \frac{\delta t}{\delta w} \\ \frac{\delta L}{\delta b} &= \frac{\delta L}{\delta y} * \frac{\delta y}{\delta t} * \frac{\delta t}{\delta b} \\ \frac{\delta L}{\delta x} &= \frac{\delta L}{\delta y} * \frac{\delta y}{\delta t} * \frac{\delta t}{\delta x}\end{aligned} \tag{4.5}$$

The only thing we are left is to train multi-layer positron. We update the weights and bias using **Stochastic Gradient Descent** (SGD).

A Max Pooling over time doesn't need to be trained because it doesn't have any weights, it is just necessary to implement a method to calculate gradients.

Convolution is the core layer in a CNN: backpropagation in this layer is the core of training. We start with a 2d array as input to a convolutional layer, in particular with a kind of *bag of words* of our comments. Taking into consideration a loss gradient, we update filter weights. We already know for the $\frac{\delta L}{\delta y}$ conv layer, so we just look for $\frac{\delta y}{\delta filters}$. In order to find the right way to update weights we must be aware that changing a filter's weight affect the convolutional layer's output. Any changes affect the entire output because every output uses every n-gram weight during convolution.

$$\begin{aligned}y(i,j) &= convolve(comment, filter) \\ &= \sum_{h=0}^{m}\sum_{w=0}^{n} comment(i+h, j+k) * filter(h,k)\end{aligned} \tag{4.6}$$

$$\frac{\delta y(i,j)}{\delta filter(h,k)} = image(i+h, j+k) \tag{4.7}$$

We can put it all together to find the loss gradient for specific filter weights:

$$\frac{\delta L}{\delta filter(h,k)} = \sum_i \sum_j \frac{\delta L}{\delta y(i,j)} * \frac{\delta y(i,j)}{\delta filter(h,k)} \tag{4.8}$$

Now we can calculate backpropagation for a convolutional layer. We use our derived equation by iterating over every comment filtered and incrementally building the loss gradients. After updating filters (using Stochastic Gradient Descent), we are ready to train Convolutional Neural Network for a few epochs and, at the end, then test it on a previously isolated test set.

Convolutional Neural Network is a supervise learning technique, so we need for training a dataset where the output is known. In particular we want a dataset of sentences that have sentiment (positive or negative) known to us. We have to realize that we have no suitable data in our database. One way could be to manually cluster a dataset, but the amount of data to be created is too large. Exploring the web we can find a large amount of comments with an associated rating (usually from 1 to 5). So let us scrap online to recover this type of data to create a new dataset. 1 and 2 stars correspond to a negative feeling, 4 and 5 stars to a positive feeling, while the comments with 3 are not collected in our data. We are now ready to use this dataset in the training architecture just presented.

4.4 Use Case

To better understand the goal of the project, below we describe a use case.

In an accommodation facility, the analysis (using Latent Dirichlet Allocation algorithm) of free text comments in the last month highlighted a growing trend in the topic *room-service*. A deeper analysis, using Convolutional Neural Network, reveals that the most widespread sentiment is negative (83.7%).

Within the comment form there is a multiple choice question regarding room service (Q_{12}). It is noted that the score related to the question is 43.2, much lower than the current NPS (64.2). The correlation index is 0.1: there is no important linear relationship between this topic and NPS. It is necessary to evaluate if there exist other types of correlations. For this purpose, the NPS simulator was developed.

The problem is handled by the room service department. The use of the simulator allows us to convert a possible business action into a forecast of variation of the NPS.

We note that the detractors of the question represent 48% of the total population taken in to account, the business action that we want to undertake, according to estimates, leads to 60% of these detractors to be Neutrals. The simulator selects 60% of the records from the dataset that respect the characteristics highlighted above and transforms the evaluation from detractor (evaluation: from 0 to 6) to neutral (evaluetion: 7).

Starting from the new dataset created, the random forest defined in the previous chapters makes a prediction on the Q_{NPS} question for each person. From here the new NPS is composed.

By doing so, we discover that a business operation of this type brings an increase of 0.2pp. NPS goes from 64.2 to 64.4. Now the company must decide whether an impact of this type justifies the action to be done and the related costs to be incurred.

Chapter 5

Conclusions

The object of the study has been the customer satisfaction measured by Net Promoter Score. The aim of the project was to understand which segments of the company have an impact on the brand perception, in particular on the NPS.

We started analysing each company sectors and their impact on NPS. We achieved this goal by studying the relations between customers answers on questions focuses on particular segment highlighted by business.

We developed an NPS-Simulator in order to give a prediction of new NPS starting from a precise business action: the inputs are company segment, target cluster (detractor, neutral or promoter on this segment), action penetration (percentage of impacted customers) and action score. This tool is very useful to evaluate a new business action, but also to gain awareness about customer experience and his relation with brand perception.

This kind of data and this analysis suffers from a bias because highlighted sectors are extract by the company itself. For this reason we also analyzed free-text questions: we tried to understand what are the strengths to focus on or the weaknesses to work on that until now were not known.

Bibliography

[1] Andreas Johnsson Selim Icki Jawwad Ahmed and Jorgen Gustafsson. «On Network Performance Indicators for Network Promoter Score Estimation». In: *Eleventh International Conference on Quality of Multimedia Experience (QoMEX)* (2019) (cit. on p. 5).

[2] C. Gronroos. «Quo Vadis, marketing? Toward a relationship marketing paradigm». In: *Journal of Marketing Management* (1994), 347–360 vol.10 (cit. on p. 5).

[3] N.Sheth and A.Parvatiyar. «The evolution of relationship marketing». In: *International Business Review* (1995), 397–418 vol.4 (cit. on p. 5).

[4] X.Luo and C.Homburg. «Neglected Outcomes of Customer Satisfaction». In: *Journal of Marketing* (2007), pp. 133–149 (cit. on p. 5).

[5] M.S.Krishnan S.Mithas and C.Fornell. «Why Do Customer Relationship Management Applications Affect Customer Satisfaction?» In: *Journal of Marketing* (2005), pp. 201–209 (cit. on p. 5).

[6] Wilson. «Factors for Success in Customer Relationship Management (CRM) Systems». In: *Journal of Marketing Management* (2002), 193–219 vol.18 (cit. on p. 5).

[7] Sisodia Sheth Jagdish and Rajendra. «Feeling the heat». In: *Marketing Management* (1995), vol.4 (cit. on p. 5).

[8] B.Denizci and X.Li. «Linking Marketing Efforts To Financial Outcome: an Exploratory Study in Tourism and Hospitality Contexts». In: *Journal of Hospitality Tourism Research* (2009), 211–226 vol.33 (cit. on p. 5).

[9] W.Vorhies and A.Morgan. «Benchmarking Marketing Capabilities for Sustainable Competitive Advantage». In: *Journal of Marketing* (2005), 80–94 vol.69 (cit. on p. 5).

[10] Fred Reicheld. «The One Number You Need». In: *Harvard Business Review* (Dec. 2003), 81(12), 46–54 (cit. on p. 6).

[11] Fred Reicheld. «The Ultimate Question. Driving Good Profits and True Growth». In: *Harvard Business School Press* (2006) (cit. on p. 6).

[12] Ioannis Markoulidakis, Ioannis Rallis, Ioannis Georgoulas, George Kopsiaftis, Anastasios Doulamis, and Nikolaos Doulamis. «Multiclass Confusion Matrix Reduction Method and Its Application on Net Promoter Score Classification Problem». In: *Technologies* (2021), pp. 9, 4 (cit. on p. 6).

[13] Ying Xie Bob Vanderheyden and Mohan Rachumallu. «Net Promoter Sentiment Classifier Using OHPL-ALL». In: *IEEE International Conference on Big Data (Big Data)* (2019) (cit. on p. 11).

[14] Ronan Collobert Jason Weston Leon Bottou Michael Karlen Koray Kavukcuoglu and Pavel Kuksa. «Natural Language Processing (Almost) from Scratch». In: *Journal of Machine Learning Research* (Dec. 2011), pp. 2493–2537 (cit. on p. 27).

[15] Yoon Kim. «Convolutional Neural Networks for Sentence Classification». In: (Sept. 2014) (cit. on p. 27).

[16] Tomaso Poggio Jaz Kandola Thomas Hofmann and John Shawe-Taylor. «A Neural Probabilistic Language Model». In: *Journal of Machine Learning Research* (Mar. 2003), pp. 1137–1155 (cit. on p. 29).

[17] Peter D. Turney and Patrick Pantel. «From Frequency to Meaning: Vector Space Models of Semantics». In: (Mar. 2010), p. 1003.1141 (cit. on p. 32).

[18] Omer Levy and Yoav Goldberg. «Dependency-Based Word Embeddings». In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (June 2014), pp. 302–308 (cit. on p. 33).

[19] Jeffrey Pennington Richard Socher and Christopher D. Manning. «GloVe: Global Vectors for Word Representation». In: *Stanford University* () (cit. on p. 34).

[20] Omer Levy and Yoav Goldberg. «Neural Word Embedding as Implicit Matrix Factorization». In: *Proceedings of the 27th International Conference on Neural Information Processing Systems* (Dec. 2014), 2177–2185 Vol.2 (cit. on p. 37).

[21] Tomas Mikolov Wen-tau Yih and Geoffrey Zweig. «Linguistic Regularities in Continuous Space Word Representations». In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (June 2013), pp. 746–751 (cit. on p. 39).

[22] Gentner D. «Structure-mapping: A theoretical framework for analogy». In: *Cognitive Science* (Apr. 1983), 155–170 Vol.7 (cit. on p. 39).

[23] Omer Levy Yoav Goldberg and Ido Dagan. «Improving Distributional Similarity with Lessons Learned from Word Embeddings». In: *Transactions of the Association for Computational Linguistics* (June 2015), 211–225 vol. 3 (cit. on p. 39).

[24] Andrew M. Dai Christopher Olah and Quoc V. Le. «Document Embedding with Paragraph Vectors». In: (July 2015) (cit. on p. 40).

[25] Ivan Vulić Nikola Mrkšić Roi Reichart Diarmuid Ó Séaghdha Steve Young and Anna Korhonen. «Morph-fitting: Fine-Tuning Word Vector Spaces with Simple Language-Specific Rules». In: (June 2017) (cit. on p. 41).

[26] Piotr Bojanowski Edouard Grave Armand Joulin and Tomas Mikolov. «Enriching Word Vectors with Subword Information». In: (June 2017) (cit. on p. 41).

[27] Matteo Pagliardini Prakhar Gupta and Martin Jaggi. «Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features». In: (Dec. 2018) (cit. on p. 42).

[28] Ledell Wu Adam Fisch Sumit Chopra Keith Adams Antoine Bordes and Jason Weston. «StarSpace: Embed All The Things!» In: (Nov. 2017) (cit. on p. 42).

[29] Ryan Kiros Yukun Zhu Ruslan Salakhutdinov Richard S. Zemel Antonio Torralba Raquel Urtasun and Sanja Fidler. «Skip-Thought Vectors». In: (June 2015) (cit. on p. 42).

[30] Thomas Hofmann. «Probabilistic Latent Semantic Analysis». In: *Uncertainity in Artificial Intelligence, UAI'99, Stockholm* (1999) (cit. on p. 44).

[31] Asuncion A. Welling M. Smyth P. and Teh Y. W. «On Smoothing and Inference for Topic Models». In: (May 2012) (cit. on pp. 46, 91).

[32] Ali Daud Juanzi Li Lizhu Zhou and Faqir Muhammad. «Knowledge discovery through directed probabilistic topic models: a survey». In: *Frontiers of Computer Science in China* (Jan. 2010), 280–301 vol. 4 (cit. on p. 46).

[33] D. Blei et. al. «Hierarchical Topic Models and the Nested Chinese Restaurant Process». In: (2003) (cit. on p. 46).

[34] David Blei. «Probabilistic Topic Models». In: *Communications of the ACM* (Apr. 2012), 77–84 vol. 55 n. 4 (cit. on p. 47).

[35] Jianwen Zhang Yangqiu Song Changshui Zhang and Shixia Liu. «Evolutionary hierarchical dirichlet processes for multiple correlated time-varying corpora». In: *Proceedings of the 16th ACM SIGKDD international conference* (July 2010), pp. 1079–1088 (cit. on p. 47).

[36] K. Vorontsov and A. Potapenko. «Additive regularization of topic models». In: (2105) (cit. on p. 47).

[37] K. Vorontsov et. al. «BigARTM: Open Source Library for Regularized Multimodal Topic Modeling of Large Collections». In: (2015) (cit. on p. 47).

[38] J. N. Morgan and J. A. Sonquist. «Problems in the analysis of survey data, and a proposal». In: *Journal of the American Statistical Association* (1963), 58(302):415–434 (cit. on p. 60).

[39] A. Sonquist. «Multivariate model building: The validation of a search strategy». In: *Survey Research Center, University of Michigan* (1970) (cit. on p. 61).

[40] Messenger and L. Mandell. «A modal search technique for predictive nominal scale multivariate analysis». In: *Journal of the American statistical association* (1972), 67(340):768–772 (cit. on p. 61).

[41] M. Gillo. Maid. «A honeywell 600 program for an automatised survey analysis». In: *Behavioral Science* (1972), 17:251–252 (cit. on p. 61).

[42] E. L. Baker J. A. Sonquist and J. N. Morgan. «Searching for structure: An approach to analysis of substantial bodies of micro-data and documentation for a computer program». In: *Survey Research Center, University of Michigan Ann Arbor, MI* (1974) (cit. on p. 61).

[43] L. Breiman. «Parsimonious binary classification trees». In: *Preliminary report. Santa Monica, Calif.: Technology Service Corporation* (2005) (cit. on p. 61).

[44] L. Breiman. «Description of chlorine tree development and use». In: *Technical Report, Technology Service Corporation, Santa Monica, CA* (1978) (cit. on p. 61).

[45] J. H. Friedman. «A recursive partitioning decision rule for nonparametric classification». In: *Computers, IEEE Transactions* (1977), 100(4):404–408 (cit. on p. 61).

[46] J. H. Friedman. «A tree-structured approach to nonparametric multiple regression. In Smoothing techniques for curve estimation». In: *Springer* (1979), pp. 5–22 (cit. on p. 61).

[47] J. R. Quinlan. «Discovering rules by induction from large collections of examples. Expert systems in the micro electronic age». In: *Edinburgh University Press* (1979) (cit. on p. 61).

[48] J. R. Quinlan. «Induction of decision trees». In: *Machine learning* (1986), 1(1):81–106 (cit. on p. 61).

[49] D. A. Zighed and R. Rakotomalala. «Graphes d'induction: apprentissage et data mining». In: *Hermes Paris* (2000) (cit. on p. 63).

[50] R. A. Olshen L. Breiman J. H. Friedman and C. J. Stone. «Classification and regression trees». In: (1984) (cit. on pp. 64, 73).

[51] D. E. Knuth. «Two notes on notation». In: *The American Mathematical Monthly* (1992), 99(5):403–422 (cit. on p. 68).

[52] S. W. Kwok and C. Carter. «Multiple decision trees». In: *Fourth Annual Conference on Uncertainty in Artificial Intelligence* (1990), 327–338. North–Holland Publishing Co. (Cit. on pp. 71, 72).

[53] L. Breiman. «Bagging predictors». In: (1994) (cit. on pp. 71, 73).

[54] B. Efron. «Bootstrap methods: another look at the jackknife». In: *The annals of Statistics* (1979), pp. 1–26 (cit. on p. 71).

[55] T. G. Dietterich and E. B. Kong. «Machine learning bias, statistical bias, and statistical variance of decision tree algorithms». In: *ML-95* (1995), p. 255 (cit. on p. 72).

[56] D. Geman Y. Amit and K. Wilder. «Joint induction of shape features and tree classifiers. Pattern Analysis and Machine Intelligence». In: *IEEE* (1997), 19(11):1300–1305 (cit. on pp. 72, 73).

[57] T. K. Ho. «The random subspace method for constructing decision forests». In: *Pattern Analysis and Machine Intelligence, IEEE* (1998), 20(8):832–844 (cit. on p. 73).

[58] L. Breiman. «Random Forests». In: *Machine learning* (2001), 45(1):5–32 (cit. on p. 73).

[59] A. Cutler and G. Zhao. «Pert-perfect random tree ensembles». In: *Computing Science and Statistics* (2001), 33:490–497 (cit. on p. 73).