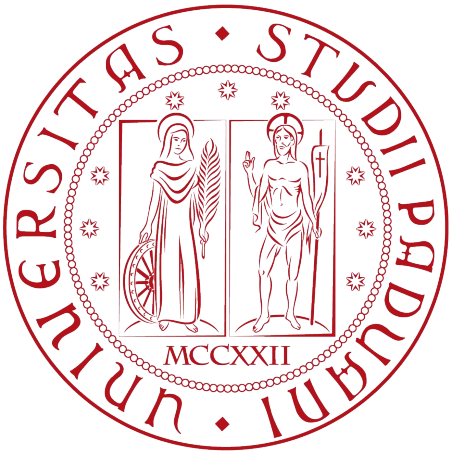
University of Padua

Department of Mathematics "Tullio Levi-Civita"

Master's Degree in Computer Science



Sentiment analysis in context: Investigating the use of BERT and other techniques for ChatBot improvement

Simone Innocente
2020585

Supervisor

Prof. Giovanni Da San Martino

Academic Year 2022-2023

Dedicato a mio padre, per quello che non hai mai avuto la possibilità di realizzare, ma per il quale mi hai sempre spronato.
Grazie Papà.

Abstract

In an increasingly digitized world, where large amounts of data are generated daily, its efficient analysis has become more and more stringent.
Natural Language Processing (NLP) offers a solution by exploiting the power of artificial intelligence to process texts, to understand their content and to perform specific tasks.
The thesis is based on an internship at Pat Srl, a company devoted to create solutions to support digital innovation, process automation, and service quality with the ultimate goal of improving leadership and customer satisfaction.
The primary objective of this thesis is to develop a sentiment analysis model in order to improve the customer experience for clients using the ChatBot system created by the company itself.
This task has gained significant attention in recent years as it can be applied to different fields, including social media monitoring, market research, brand monitoring or customer experience and feedback analysis.
Following a careful analysis of the available data, a comprehensive evaluation of various models was conducted. Notably, BERT, a large language model that has provided promising results in several NLP tasks, emerged among all.
Different approaches utilizing the BERT models were explored, such as the fine-tuning modality or the architectural structure. Moreover, some preprocessing steps of the data were emphasized and studied, due to the particular nature of the sentiment analysis task.
During the course of the internship, the dataset underwent revisions aimed to mitigate the problem of inaccurate predictions. Additionally, techniques for data balancing were tested and evaluated, enhancing the overall quality of the analysis.
Another important aspect of this project involved the deployment of the model. In a business environment, it is essential to carefully consider and balance resources before transitioning to production.
The model distribution was carried out using specific tools, such as Docker and Kubernetes. These specialized technologies played a pivotal role in ensuring efficient and seamless deployment.

"Siamo pennelli, le emozioni, i colori ed anche i tratti più neri fanno parte di bellissimi disegni."

— Mezzosangue

Acknowledgments

In primis ringrazio il mio relatore Prof. Giovanni Da San Martino per le indicazioni, i consigli e l'appoggio durante la stesura di questo documento.

Tutto il mio percorso lo devo alla mia famiglia; grazie per il supporto che mai è mancato, tutti i miei traguardi sono stati resi tali sopratutto grazie a voi.

Ringrazio tutti i miei più cari amici, in particolare Yoada, Elia e Manuel, che fin dalle superiori mi hanno sempre spronato a dare il meglio e sempre aiutato nei momenti più duri. Voglio ringraziare i miei compagni di università, Federico e Alberto, questo percorso è stato incredibile soprattutto grazie a voi.

Infine voglio ringraziare me stesso, per non aver mai mollato, aver creduto nelle mie capacità, per tutto il duro lavoro svolto. È stato un percorso lungo, ricco di stimoli, impegni, soddisfazioni personali, amicizie, tutte cose che porterò con me, perchè ora inizia il bello: inizia una nuova parte della mia vita.

Padova, Luglio 2023 Simone Innocente

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, in a corporate setting, continuous connection with customers is becoming increasingly important. Improving client relationship management leads to a better quality of services and customer satisfaction.
However, often is not possible for companies to offer always-available customer service. In order to simulate this kind of continuity, keep interaction with customers, provide services to the clients and help them find information, the use of ChatBot is crucial.
There are several benefits that this tool brings: it can operate 24/7 and give an answer immediately, saving time for both the customer and the company.
They can enhance the customer experience, due to the fact that they are personalizable, and the answers can be customized for different clients.
One of the major points that bring ChatBot to be so popular nowadays is that they increase efficiency. In particular, they can handle some ordinary tasks that can be easily solved. Chatbots are able to significantly reduce requests going to human operators, by automating tasks such as answering FAQs or booking appointments allowing the support team to focus on managing more intricate queries. [81]
Moreover, companies can hold to an enormous quantity of data in order to power customer experiences and marketing strategies. One way to exploit this amount of information is artificial intelligence (AI).
AI techniques have been widely used for different applications: time series prediction, text classification and other related tasks to improve customer experience.
In particular, when we are talking about ChatBot, AI services are based on Natural Language Processing, which is the field of artificial intelligence which goal is to make computers understand natural language and it can be very helpful for ChatBot improvements.
This thesis will delve into an analysis of data that has undergone revisions due to its inconsistency. Additionally, an approach will be explained, where another dataset is exploited to counter the imbalanced nature of the dataset. Ultimately, the definition of a sentiment analysis model, with a specific focus on comprehensive study and analysis of the BERT language model.

1.1 The collaboration with PAT Srl

PAT Srl has been active for over 30 years in the development of applicative and strategic solutions aimed to support major Italian and international companies in the areas of Service Management, Virtual Assistance, Customer Relationship Management and Customer Care.
The goal is to improve customer relationship management with solutions that allow companies to automate business processes.
The company has four locations (Treviso, Milan, Florence and Madrid) with more than 70 collaborators and since 2013 Pat has been part of the first Italian Software House in ICT panorama: Zucchetti Group.
PAT solutions allows improving the accessibility between different areas of a company, by enhancing business processes, especially in fields such as customer service, customer care, service desk, multichannel iteration tools (social, telephone and e-mail) and process automation.
PAT combine experience and technological expertise in machine learning to improve customer experience by optimising multichannel customer interactions.
The solutions that PAT provides to their customers is divided in three different software:

1. **Helpdesk Advanced (HDA)**: it controls services in all organisational areas in which Service Management is one of the prerogatives for successfully, optimally and effectively achieving the company's objectives, operating through high-quality standards. This solution is based on the logic of process automation, supporting the different strategic scenarios of service governance, thanks to the automation and high configurability of IT and Business processes;
2. **Engagent & CX Studio**: is a customer interaction solution that, thanks to artificial intelligence (AI) and bot, enables real-time interaction with customers in order to provide to them the information and virtual assistance they need, anticipating their interest based on their website or mobile application navigation actions.
   CXStudio is the framework that allows to create multichannel or one-to-one multilingual user dialogues and make 24/7 interaction possible. Solutions provided by Engagent and CX Studio allows to improve the customer experience through machine learning technologies.
3. **IC Studio**: is a customer relationship management platform that assists companies in different areas from sales and marketing to customer care and call centers. It helps companies to develop positive companies and profitable business experiences with their customers and partners.

Since 2013, PAT Srl has been part of the first Italian Software House in the ICT panorama: Zucchetti Group.
Moreover, recently, a new initiative is shaping up, the *AI Factory - Innovation Lab* which is a collaborative team comprising engineers and developers from both PAT Srl and Zucchetti, with the goal of implementing AI and machine learning technologies in order to add new innovative features in the PAT solutions.
The development of the solutions described in this thesis was under the guidance of the AI Factory, through regular updates on the state-of-the-art including weekly reports on key metrics and achieved results.

1.2 Thesis Overview

The purpose of this project is to investigate the potential of Artificial Intelligence (AI) and Machine Learning (ML) to enhance customer service and experience in a real-world scenario, Engagent and CX Studio provide the opportunity to do this.
Indeed, the framework's architecture is highly modular and permits the developer to define some plugins called *Adapter*, which can easily integrate external components, such as artificial intelligence models.
The main goal of the thesis is to design, develop and integrate into Engagent and CX Studio, a sentiment analysis model based on users' requests provided in the chatbot.
As written before, the framework allows enhancing the customer experience, providing a chatbot and an entire framework that permits the clients to interact directly with their consumers.
More precisely, when users' necessities are not satisfied by the automatic system, they are redirected to a human operator that sometimes might be busy with another client and thus they are added to a queue.
The idea is to give priority to the customers who are dissatisfied so that their problematics are solved sooner.
A main component to achieve this result involves the examination of the user's emotions, which can be carried out by a sentiment analysis model to understand the feelings expressed by users through the text available in the chat.
Along with CXStudio and Engagent, the sentiment analysis model will be integrated in **Power Text Explorer**, an innovative product developed by AI Factory. It provides a flexible environment for collecting, organizing, integrating, and processing unstructured data from various sources.
It offers a wide range of AI-based analysis which allow users to tag data for further analysis or automatically identify semantically similar text groups. For example, it is possible to automatically create *clusters* of data or identify similar *topics*.

The thesis will explain a preliminary data analysis that has been performed, together with the preprocessing step. This phase is crucial for the correct realisation of the model itself.
A thorough examination has been conducted to determine the most suitable model, by analyzing baseline, state-of-the-art approaches with respect to their respective performance metrics.
Moreover, the adapter has to be developed in order to integrate the AI model with the framework.
The deployment of the model will be in a double mode, with Docker and Kubernetes. The latter is an open-source platform that allows to manage and orchestrate the application deployment in a set of nodes; this brings better scalability and reliability.
Furthermore, a detailed and thorough documentation is provided for every project phase. In this thesis, a more detailed analysis about BERT will be conducted including an evaluation of various architectures, reaching an accuracy value of 88% in the test set.
Moreover, a new approach which exploit additional datasets will be applied to reduce the unbalanced nature of the starting dataset. Although BERT is a large-scale language model, this thesis will showcase the process of fine-tuning it for the specific task of sentiment classification, reducing the gap between theory and real-world application, with a deeper understanding of its effective utilization.

1.3 Document Structure

The thesis document is organized as follows: chapter 2 contains a theoretical description of Natural Language Processing, with a specific focus on text classification based on sentiment analysis.
Chapter 3 explain a detailed data analysis on the dataset used. It incorporates also the preprocessing part and some techniques used for mitigate the problem of unbalanced dataset.
In particular, a more detailed dataset evolution description is provided in figure 1.1.
During the model evaluation part (red) which was followed by the data analysis (blue), was requested to analyse a further dataset in order to send it to a colleague for categorisation.
Afterwards, two dataset reviews were performed (yellow part) due to the inconsistency of the sentiment labels. All these dataset modification will be discussed in chapter 3.

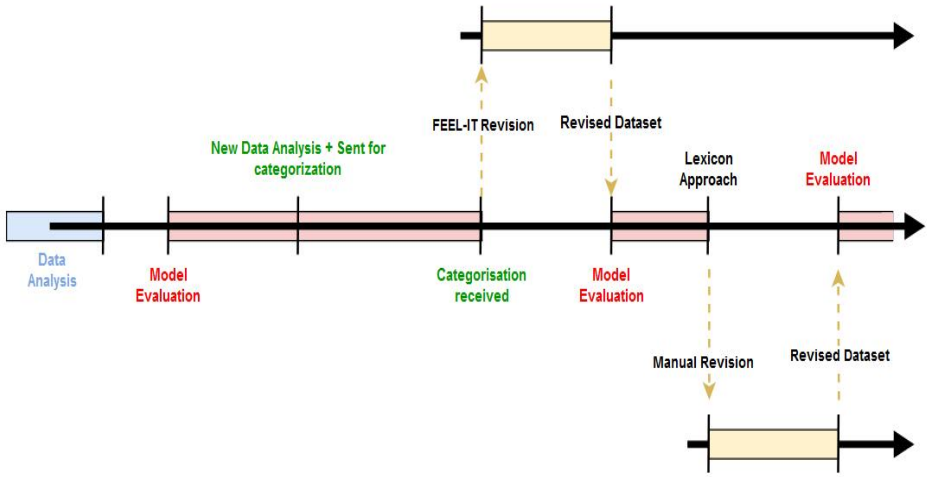


Figure 1.1: Workflow of the project

The development and evaluation model will be discussed in chapter 4 where several models will be evaluated and tested based on some metrics performance.
Lastly, chapter 5 contains the conclusion and some possible future improvements.
A design and implementations part of the solution, with a REST API and the use of Docker will be explained in the appendix A. Moreover, the main libraries and tools used are described in the appendix B.

Chapter 2

Natural Language Processing

Over the past few years, the burgeoning popularity of applying artificial intelligence to textual data can be attributed to the effectiveness demonstrated by AI-based methods in the field of NLP. This success derives from the ability of AI to understand, interpret, generate texts, analyse a large number of texts and be able to identify key relationships and concepts.
The challenge that has to be faced is to make human language understandable to machines, which is the aim of Natural Language Processing (NLP), the branch of Artificial Intelligence specialized in text.
Thanks to NLP, it is possible to create systems for document summarization, machine translation, spam detection, sentiment classification, named entity recognition, question answering and many other tasks [79].

2.1 Why Natural Language Processing?

Nowadays, companies are overwhelmed by data and without the proper technology, it would be impossible to analyse this amount of information effectively. We can just think for example about the quantity of data which are available through the social web.
Before the advent of Web 2.0, users acted in a passive way solely focused on obtaining the needed information, rather than creating content. Today, things have changed, everyone can decide to use any kind of device and be an active user of the web. This brings many possibilities, from posts on social media to reviews in E-Commerce, consequently resulting in the generation of increasingly large volumes of data.
Another simple example regards the process of document digitalization in a company, which for the majority is composed of texts in natural language. With the help of NLP, machines may enhance their understanding of human language, which can bring to solve in an automatic manner, different repetitive tasks, like ticket classification, machine translation, spell checks and summarization.

2.2 Natural Language Processing Tasks

As mentioned before, there are numerous NLP tasks that can be performed and in this section, the main learning problems will be highlighted with a brief description for

each one.

Text Classification
One of the core tasks in the field of Natural Language Processing is Text Classification. It involves assigning a sentence to a predefined class or category. This task can be easily applied in a wide range of domains, from social media to e-commerce, marketing and law. [80]
Some instances of text classification tasks include:

- Spam Detection;
- Sentiment Classification;
- Language Identification;
- News Classification.

Machine Translation
Another prevalent(?) task in NLP is Machine Translation. The goal is to automate the process of translating a sentence from a source language to a target language.
For decades people have been working on solving this task and in the first approach, called **Rule-Based**, the knowledge is encoded directly into rules which are used to perform translation.
More recent approaches are **Statistical** and **Neural-based** Machine Translation, the first one made statistics by using the data, and the second uses an Artificial Neural Network to predict the word based on the previous words in the sequence.
Nowadays, architectures based on Transformers like BERT have reached major improvements in machine translation.

Text Summarization
Another useful and important task is Text Summarization which is an example of Text-to-Text generation like Machine Translation. However in this case the goal is to create a shorter and more concise version of a given paragraph or document that preserves the main ideas and the overall meaning.
Automated Text Summarization approaches are typically categorized into two main types: **Extractive** and **Abstractive** methods. [67]
Extractive methods recognize and select sentences from the original text that contain the main point of the document and compose the summary based on these sentences, resulting in a summary that replicates parts of the original text verbatim.[78]
The second approach, similar to the first, identifies key segments of the document and interprets their contextual meaning, subsequently rephrasing them in a novel manner. The main information is conveyed through the shortest text possible.

Part Of Speech(POS) Tagging
Part Of Speech (POS) tagging is the process of assigning for each word in a sentence, the proper part of speech which is a category of words with similar grammatical properties. For example, common parts of speech in English are nouns, verbs, adjectives, adverbs and so on.
This kind of task can be useful in NLP for several reasons: understanding the grammatical structure of a sentence and disambiguating words with multiple meanings can help to improve also the accuracy of other Natural Language Processing tasks.
The **Rule-based** approach uses pre-defined rules in order to perform the job based on their spelling and context. In **Statistical POS Tagging** alternatively, a machine

learning model is trained on a large annotated corpus of text with the aim of learning patterns and characteristics of different parts of speech.
One more approach is called **Transformation based Tagging**, the main idea is to use rules to convert the tags of words into text based on the context. For instance, a rule might change a word tag from a verb to a noun if it appears after a determiner such as "the.". Rules are applied to the text in a specific order, and the tags are updated after each transformation. [41]

Named Entity Recognition (NER)
Named Entity Recognition (NER) is one of the most popular tasks in semantic analysis and a subproblem of information extraction. The aim of this task is to identify and categorize entities in a text.
Entities can be names, places, organizations and, in general, any word or series of words that refers to the same thing. Extracting entities can be useful for many other problems like machine translation, question answering or summarization.
The approaches to Named Entity Recognition can be categorized into three classes, **Hand-made Rule-based** NER, **Machine Learning based** NER and **Hybrid** NER. As written before, the rule-based method tries to extract entities and names by using numerous human-made rules. The machine learning approach instead, convert the problem into a classification task and try to solve it. A hybrid NER system combines both approaches by using the strongest point from each method.

Question Answering
The Question Answering (QA) task involves developing a model which can answer questions in natural language. This kind of systems are used to automatize the answer to frequently asked question by using a knowledge base as a context.
Other kinds of applications for this task are smart virtual assistants, information retrieval, customer service, and education. [19]
There are several types of Question Answering systems:

- **Extractive Question Answering**: the context, which could be a document or a set of documents, is provided to the model with the question. It extracts the answer from the context and provides it;
- **Generative Question Answering**: In this kind of system, the answer is generated by the model directly based on the context, the answer is the coherent and appropriate generated text from the model, based on its background knowledge.

2.3 Approaches to Text Classification

As already mentioned before, Text Classification is one of the most essential tasks in the field of Natural Language Processing. The goal is to assign a set of predefined labels (classes or categories) to a sentence or a document.
Speaking more generally about classification and not being focused on text, there are three main types of problem instances: **binary**, **multi-class** and **multi-label**.
In the first one, we classify the data into two mutually exclusive groups of classes, which can be labelled with 0 and 1. One example could be spam detection, where an e-mail can be considered spam (1) or not spam (0).
In multi-class classification, each data sample is assigned to one and only one label from three or more classes. For instance, we want to classify an image of an animal into several categories such as dogs, cats, horses, birds, etc.
In the multi-label classification instead, multiple classes may be assigned to a single data sample. For example, in an image, we can find more than one animal, with multi-label we can classify multiple animals. [62]
This thesis will specifically center on addressing binary and multi-class classification problems. The approaches to text classification are different, but generally, there are three main categories, **rule-based**, **machine learning-based**, and **hybrid**.

2.3.1 Rule-based System

In a rule-based approach, we try to encode knowledge directly into rules that are used to classify the text. These rules are utilized to allow systems to identify relevant elements of the text.
They are composed of a left side, which contains specific features of a text, and a right side which contains one of the classes that we want to identify. [93] The occurrence of the features in a test sentence implies the adherence of the sentence to the corresponding rule.
A text can satisfy many rules which right side may have different classes. Usually, the selected class is the most frequent among the rules adhering to the sentence.
Rules can be created based on a variety of characteristics of the text, such as the presence or absence of particular words, specific pattern, parts of speech of words, the length of the text and so on.
One advantage of this kind of approaches is that they are easy to implement and to interpret mainly if we use simple features like the presence of certain keywords. Moreover, systems based on rules can be relatively fast, due to the fact that the process to classify a text involves checking the text against the predefined rules and returning the appropriate label.
However, creating rules is time-consuming and challenging, such systems demand deep knowledge of the domain and complexity in order to handle a variety kind of patterns. Furthermore, they are also difficult to maintain, since rules may need to be updated or modified. [86]
In general, other approaches can perform better and can be more flexible, that's why nowadays they are preferable.

2.3.2 Machine Learning System

Machine Learning systems, which have become very popular not only for text classification, in recent years become good alternatives to the use of manually crafted

rule-based systems.
This kind of approach relies on a huge amount of data, which is used to train machines for classification purposes.
In the supervised paradigm, the data used to train the classifier are pre-labelled, in this way machine learning algorithms can assimilate associations, patterns, context and the specific output (i.e. sentiment) for a specific input (i.e. text).
In this way, machine learning systems are able to understand the data and classify them.
However, in NLP applications, machine learning systems perform a step called **feature extraction** where text in the form of string is converted into a more manageable type of data for machines. This means that we need to provide a representation that can be fitted into the system, in particular, we use a numerical representation in the form of a vector.
One of the most common approaches is called **Bag-Of-Words** where a vector represents the frequency of a word in a predefined dictionary of words [86]. After this transformation, the machine learning algorithm is trained with the features created and the corresponding label that could be for example the sentiment of the sentence.

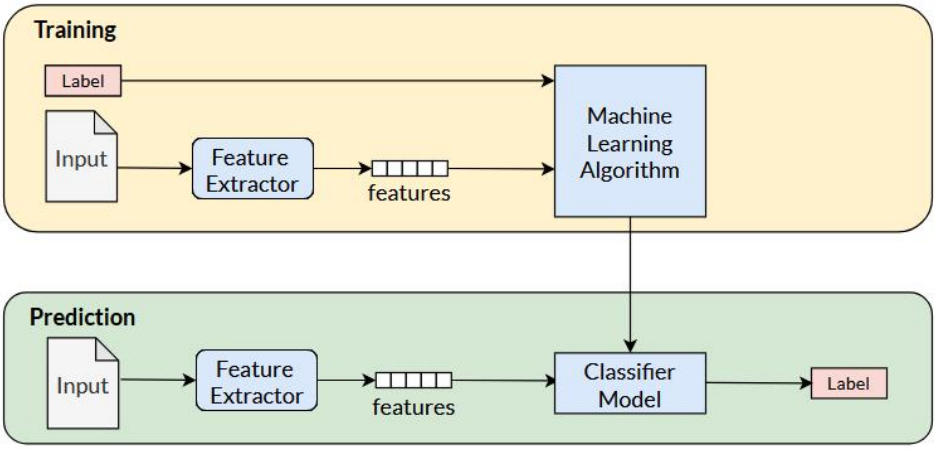


Figure 2.1: Supervised-classification [68]

Following the training part, the machine learning model can perform the predictions. By using the feature extractor, it is possible to derive the features from the input and feed them to the classifier which aims is to predict the label for the task that it was trained for.
In Figure 2.1 is possible to see the general process of training and prediction in a machine learning system for text classification. This kind of technique provides much more accurate results than rule-based systems since machine learning algorithms are able to understand and detect patterns and associations. They are more adaptable and easier to maintain with respect to rule-based systems since they do not require the definition of new rules and it is sufficient to retrain them on new data.

2.3.3 Hybrid System

Hybrid systems try to merge the strength of both the above approaches. Usually, the starting point is the machine learning approach, where we use a model trained on a labelled corpus.
Afterwards getting the predictions, we can improve the performance by exploiting a rule-based expert system which can deal with the false negatives. [55]
The main benefit of this approach is that you can easily perform refinements by adding new specific rules which can avoid conflicting categories in order to improve the capacity of the rule-based system.

2.4 Text Representation

Machine learning algorithms are not able to deal with raw text data directly, indeed text is often unstructured and chaotic. Instead, artificial intelligence systems typically require a numerical representation of the input. The aim of the **feature extractor** (or feature encoding), thus is to transform the documents into a numerical representation typically in a format such as vectors of numbers.
Many techniques have been developed over the years, including bag-of-words, TF-IDF or word embeddings.
Overall, text vectorization is a crucial step in natural language processing that allows machine learning algorithms to work with text data and extract meaningful insights. [13]
However, before actually get a dive into various text representation techniques, **preprocessing** should be performed in order to improve the overall results of the machine learning algorithms.

2.4.1 Preprocessing

Text preprocessing is a vital step, and often the first one, in the pipeline of a Natural Language Processing (NLP) system, with a potential impact on its final performance. This is a crucial stage because it transforms the original corpus into a clean format that can be easily understood and analyzed by machine learning algorithms. [53]
The main idea and the primary goal of this step is to clean the documents and remove the noise in order to highlight the relevant part of them. This can make the text more suitable for tasks such as sentiment analysis, topic modelling or more generally text classification.
There are several techniques that can be used to perform text preprocessing: [60]

- Tokenization
- Capitalization
- Stop-words removal
- Noise removal
- Abbreviation and Spelling Correction

Tokenization

In a typical natural language processing pipeline, one of the most important preprocessing stage is tokenization.
The task performed is to break up a text into smaller components known as tokens

which can be either words, characters or subwords. [21]
The first one and the second one apply tokenization respectively based on words and characters within the sentence. The subword tokenization instead performs the splitting based on subwords, which means that words will be splitted into smaller parts based on character co-occurrence frequency. [72]
As example, word tokenizer would split the sentence `Hello World!` into three different tokens: `"Hello", "World", "!"`

Capitalization

Another significant component of the preprocessing pipeline is a basic operation that converts the uppercase letters into lowercase characters.
This phase can play an important role in determining how effectively machine learning models can process and understand text data. Indeed, lowercasing text guarantees that words with different capitalization are treated in the same way and it also helps by reducing the vocabulary size avoiding the duplication of words. Moreover, tasks such as text classification or POS Tagging might rely on the fact that capitalization of a word does not alter its meaning.
However, there exists an alternative approach that relies on the frequency of unigrams across the entire dataset which is named Truecasing. [63]
It is also important to point out that this technique is not common for every NLP problem, as for some of them capitalization can lead to loss of information, for example in a sentiment analysis classification.

Stop-words removal

Stopwords are words which appear frequently in a language but typically do not add significant meaning to the text. The idea of this part of the preprocessing pipeline is to remove these kinds of words that do not give additional value to the text, in this way is possible to focus on the important part of a sentence.
Indeed, stopwords are available in abundance in any human language, by removing these words we are able to remove, most of the time, the low-level information from the text and give more focus to the important information. [59] [33] In English some examples of stopwords are *a*, *the*, *is*, *are*, for Italian instead we have *a*, *al*, *allo*, *poco*.

Noise Removal

In some cases, even the punctuation can affect the model evaluation process, indeed there are tasks in which it can be removed from the textual data.
This process eliminates irrelevant or unnecessary elements with the objective of enhancing the quality of the data. However, is essential to be careful while removing some commonly used punctuation. For example, the word *don't* will be converted in *don t*; a human could understand easily this word, but a machine can treat *don* and *t* as two separate words. [65]

Abbreviation Removal

Abbreviation removal is a common step in the preprocessing pipeline for natural language processing (NLP) tasks. It involves identifying and expanding shortened

forms of a word or phrase to their full forms to ensure that the text is correctly understood by the NLP models. [60]
For example, "NLP" would be expanded to "natural language processing". This step is important because many NLP models rely on the context and meaning of words to generate accurate predictions, and abbreviations can often introduce ambiguity and confusion.

2.4.2 Bag Of Words

One of the simplest approaches for representing a text is called Bag Of Words (BoW) which represents a document as the frequency of the words in it.
It involves a vocabulary of known words and it turns sentences into fixed-length vectors whose dimensions correspond to the vocabulary size. Each dimension of the vector is associated with a word, the corresponding value is its frequency in the document. This process is often referred to as vectorization [101]. Suppose we need to vectorize the following sentences:

- *the cat sat*
- *the cat sat in the hat*
- *the cat with the hat*

The vocabulary is composed of every unique word present in the set of documents (the sentences), in this case, it will be made of: *the*,*cat*,*sat*,*in*,*hat*,*with*.
The total number of elements in the vocabulary is six, thus, every sentence will be represented as a vector of dimension six.
Every value in the vector will be the frequency of the word in that document.

Sentence	**the**	**cat**	**sat**	**in**	**hat**	**with**
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat sat with the hat	2	1	0	0	1	1

Table 2.1: Frequency of words

In table 2.1 is possible to see the occurrence of each vocabulary word in each text.
This technique is straightforward and easily comprehensible, however, some drawbacks are evident.
The meaning of the sentences are not completely encoded: discarding word order the context is ignored and in turn also the meaning of words in the documents. These two elements can bring greater results in terms of machine-learning model accuracy.
The vocabulary configuration chosen can impact its size and thus the sparsity of the representation. Indeed in some cases, it is preferable to use a vocabulary composed of subwords instead of a complete word, in order to reduce the dimension. This raises another issue, the sparsity of the data representation, indeed, sparse representations are harder to handle for computational reasons in terms of space and time complexity, and for information reasons, where pieces of information are more distributed and fragmented. [14]

To overcome these limitations, more advanced NLP techniques have been developed such as word embeddings.

2.4.3 TF-IDF

Another disadvantage of the bag of words representation regards the importance of a word within the set of documents. The bag of words model does not take into consideration the influence of the frequency of words.
The TF-IDF (Term Frequency - Inverse Document Frequency) try to model the importance of a word in the text which is a fundamental part of information retrieval. This technique is based on the TF-IDF which is a *numerical statistic that is intended to reflect how important a word is to a document in a corpus.* Is based on two distinct components, the **Term Frequency (TF)** and the **Inverse Document Frequency (IDF)**. [50]

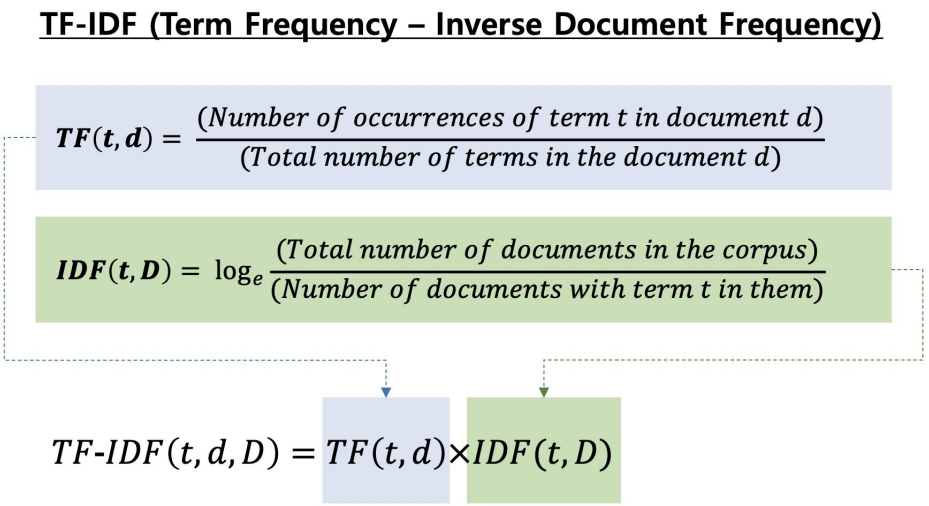


Figure 2.2: Term Frequency - Inverse Document Frequency [1]

In figure 2.2 is possible to see how the score is computed. The TF value is a local measure of the frequency of a term (t) in document (d) and is defined by the ratio of the occurrence in the text to the total number of words in the document.
An example of the term frequency value is available in table 2.2 where the same sentences of the previous example are used.

Sentence	the	cat	sat	in	hat	with
the cat sat	0.3	0.3	0.3	0	0	0
the cat sat in the hat	0.7	0.17	0.17	0.17	0.17	0
the cat sat with the hat	0.7	0.17	0.17	0	0.17	0.17

Table 2.2: Term Frequency values

The IDF allows measuring the importance of words within the entire set of documents, in particular, specifying how common or uncommon a word is in the corpus. It is a global measure computed logarithmically.
The IDF values for each word in the vocabulary can be found in table 2.3.

Word	Value
with	1.09
in	1.09
hat	0.41
cat	0
sat	0
the	0

Table 2.3: Inverse Document Frequency values

Finally, the TF-IDF score is simply the product between TF and IDF, refer to table 2.4 for the values.
By computing this estimation you can get a more precise representation of the text by highlighting which words are more important and which are less important.

Sentence	the	cat	sat	in	hat	with
the cat sat	0	0	0	0	0	0
the cat sat in the hat	0	0	0	0.18	0.07	0
the cat sat with the hat	0	0	0	0	0.07	0.18

Table 2.4: TF-IDF values

As is possible to understand from the tables, Bag Of Words vectors are easier to interpret due to their simplicity, however, TF-IDF performs better in machine learning models because it brings more information about the overall document structure. [90]

2.4.4 Word Embeddings

More advanced techniques than Bag-of-Words and TF-IDF have been developed in recent years. They are neural network-based and they become popular in natural language processing for representing words as numerical vectors in a high-dimensional space.
The peculiarity of these approaches is that they can capture the important semantic and syntactic relationships between words. The final outcome desired is that words with similar content have a similar value in the representation, thus they occupy close spatial positions which can be computed with the *cosine similarity*. A method for obtaining them is by training a neural network on a large corpus of text data, where the network

learns to predict the context words given a target word. [95] Bengio [100] proposed the approach with a neural network composed of an embedding layer, several hidden layers and a softmax activation function in 2003. Due to computational problems, other opportunities based on this method have been defined, such as **Word2Vec**. [98]

In 2013 Tomas Mikolov developed Word2Vec **(Distributed Representations of-Words and Phrases and their Compositionality)** which became one of the most popular implementations for word embedding. It offers two different variants the **Continous Bag-of-Words (CBOW)** and **Skip-gram**.
The first approach, which is visible in figure 2.3, learns the embedding by taking the context of each word as input and trying to predict the word based on that. In this case, the context is a window of neighbour words. [7]

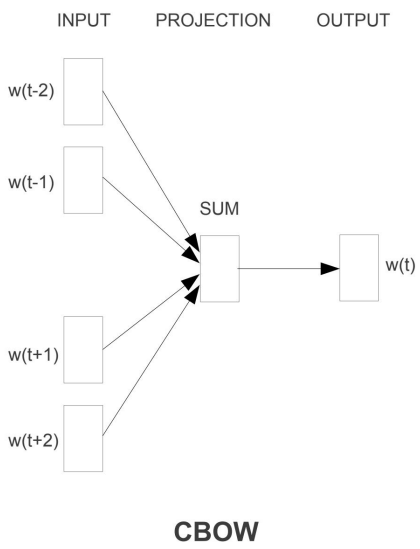


Figure 2.3: Continuous Bag-of-Words approach

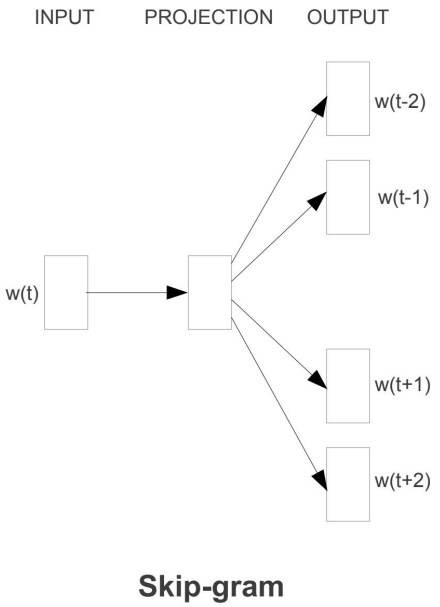


Figure 2.4: Skip-gram approach

The skip-gram procedure is analogous to CBOW, however, from the target word it tries to predict the context.

Both models work by learning words from their local usage context, which, as was written before, is a window of neighbouring words which is a configurable parameter. This variable, as written on [69] "has a strong effect on the resulting vector. Large windows tend to produce more topical similarities, while smaller windows have more functional and syntactic similarity".
High-quality results and an efficient complexity (low space and time complexity) are the key advantages of this approach.

An extension of Word2Vec has been developed with the aim of improving the concept of context. Stanford University designed **GloVe**: Global Vectors for Word Representation. [37]

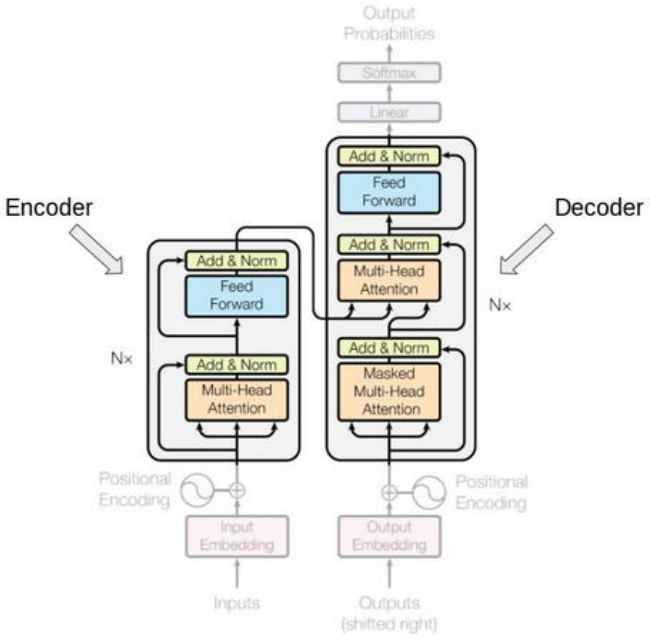
The idea behind this project is to construct an explicit *word-context* or *word co-occurrence* matrix using statistics across the whole text corpus. In this way, you can derive the relationship between the words from statistics in a global manner with the aim of getting word embedding of better quality.

2.5 BERT

In 2018 Google researchers published **BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers), a groundbreaking language representation model that made significant strides in the field of Natural Language Processing.
It is based on **Transformers**, an architecture designed in 2017 in which every output element is connected to every input element and relies on a mechanism called *Attention* to draw global dependencies between them.
Traditional language models such as LSTM and GRU are unidirectional, which means that they can process the input sequence in one direction (from left to right or from right to left). BERT, instead, uses a **bidirectional** approach thanks to the transformer architecture and this allows it to capture the context and relationship in both directions.
One primary innovation of BERT is its training mechanism which is designed to pre-train deep bidirectional representation from unlabelled text by considering the context both from the left to the right side of the sequence in all the layers.
In this way, BERT is able to contextualise words, thereby increasing its capacity for understanding context and ambiguity in language. [64]
Moreover, it can be fine-tuned with one additional output layer for some specific task and in the year of release, it achieved state-of-the-art results for eleven natural language processing tasks such as question answering or semantic textual similarity.
BERT was only the starting point for several variations and improvements, including RoBERTa, AlBERT, and ELECTRA. These models have improved performance by expanding on the original BERT architecture.

2.5.1 How BERT Works

BERT's model architecture is a multi-layer bidirectional Transformer encoder, based on the original implementation which is visible in figure 2.5.
Transformer is a simple network architecture with an encoder-decoder structure, where an input sequence is given to the encoder and mapped to continuous representations, while the decoder generates an output sequence one element at a time. [92]
In the original paper, the encoder is composed of six identical layers, each of them, as we can see from the figure 2.5, is composed of two main components, the *Multi-Head self-attention mechanism* and a simple feed-forward neural network.
The decoder contains the same two components of the encoder, with the difference that it performs the multi-head attention mechanism on the output of the encoder. Moreover, to ensure that the prediction for position i depends only on the output already generated, Masked multi-head attention was added.



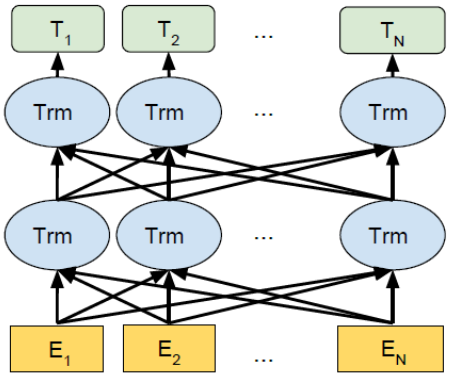


Figure 2.6: Bert architecture [25]

Figure 2.5: Transformers architecture [92]

The attention mechanism is the one that allows learning how to measure the similarity/influence between different elements in the same sequence and this operation is computed by using three different vectors called *query*, *key* and *value* which are generated from the input sequence.
BERT in this way can understand which parts of the sentence are most related to each other, and thus capture dependencies between them with the consequence of a better contextual understanding. [54]
As it is possible to calculate the attention score at once using matrices and speed up the process, a great property of the attention mechanism is its ability to be applied in parallel, enabling us to utilize *Multi-Head attention*.
In figure 2.6 is possible to see a general representation of BERT's architecture where every input (which is an embedding representation of the token) is processed by a series of transformers blocks. Each of them is connected to each other, and the base size of BERT is repeated 12 times. [25]

Input representation

One strong point of BERT is the ability of handle a wide variety of tasks. This is reached also thanks to its input representations which represent without any ambiguity both a single sentence and a pair of sentences.
The sequence input is tokenized with WordPiece embeddings [25], a subword segmentation algorithm used in natural language processing.
Moreover, they add some special tokens: the $[CLS]$ token in the first positions which is used to handle classification tasks, and $[SEP]$ in order to separate the sentences.
Finally, the input representation is computed by summing the token, the segment and positional embeddings. Segment embeddings are used to determine if a token belongs

to sentence A or sentence B, instead, the positional embeddings provide information about the relative position of the tokens in the input sequence. [25]

Pre-training BERT

The first stage BERT has is the *pre-training* using two unsupervised tasks, the *Masked Language Modeling (MLM)* and *Next Sentence Prediction (NSP)*.
The 15% of the input tokens are randomly masked with a specific tag $[MASK]$ and the first BERT task is to predict the original values of these masked token based on their context.
Instead, the Next Sentence Prediction task allows BERT to understand the relationship between two sentences by predicting whether or not two sentences are consecutive in a given text. [25] The combination of MLM and NSP helps BERT to learn contextualized representations of words that capture the meaning of the words in context and the relationships between sentences.

Fine-Tuning BERT

As already mentioned, BERT can be fine-tuned on specific downstream tasks thanks also to the self-attention mechanism.
Once BERT has been pre-trained on a large corpus of text with the modality viewed in the previous paragraph, the fine-tuning procedure includes adapting BERT to a particular task and training it with a task-specific dataset.
In figure 2.7 is visible the pre-train stage and a fine-tune.

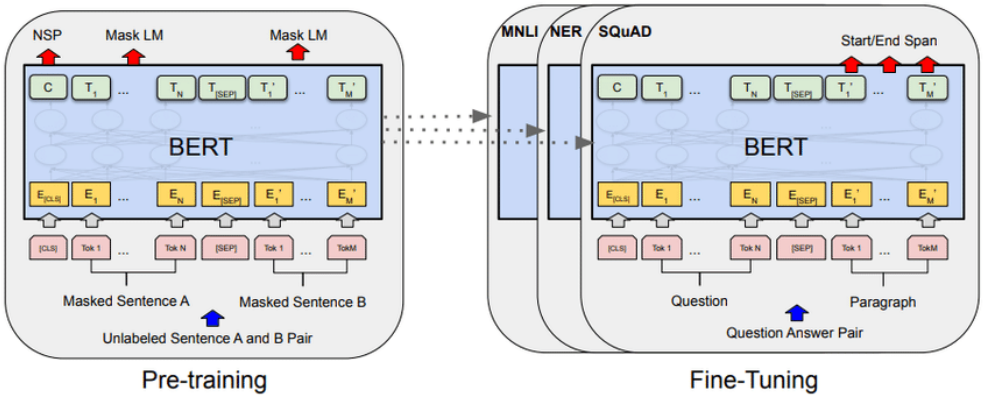


Figure 2.7: Pre-training and Fine-Tuning stage in BERT [25]

For text classification tasks, the $[CLS]$ representation serves as the input to an additional output layer that performs the classification. The same procedure is also used for other tasks such as question answering or sequence tagging.
This allows the model to be fine-tuned for a specific task while still leveraging the rich contextualized word representations learned during pre-training.

2.5.2 Limitations and challenges in using BERT

Despite BERT's state-of-the-art performance on many NLP tasks, this model nevertheless has some drawbacks and difficulties. [43]

Computational cost One of the main disadvantages of using BERT and other big language models is the computational and memory resources needed to be trained. Indeed, in the base version of BERT there are 110 million trainable parameters while the large version contains a total of 340 million parameters, which makes the model hard to train.
However, fine-tuning BERT on a specific task is relatively fast but it can take time when working with large datasets.
As regards memory, the base model is able to manage sequences of length 512 with a hidden state of size 768. This means that it requires memory to store 512 hidden states, each of size 768. For the large version instead, the hidden state size is 1024.

Out-of-Vocabulary words BERT has been pre-trained on a large corpus of text, however, this means that its vocabulary is limited to the training data.
If it encounters a word that is not in its vocabulary, the word will be replaced with a particular token $[UNK]$ which means that is out of the vocabulary.
This limitation could impact the performance when rare domains or specific words are used.

Languages availability While BERT's performance in English language tasks was astonishing, its performance in different languages may be limited. This is because the pre-training phase was carried out with English language texts.
In order to fully exploit the potential offered by BERT, it may be necessary to carry out training for a specific language, and this can be costly in terms of time and resources. However, nowadays, multilingual models or specific-language models are available.

Fine-Tuning BERT, in order to solve specific tasks, needs to be fine-tuned to achieve optimal performance. This training process requires a labelled dataset on that specific job, which may not always be available or may be expensive to obtain.

2.5.3 Pre-trained Italian BERT models

The open-source nature of BERT has contributed to its widespread adoption and popularity in the NLP community. Researchers and developers are contributing to the creation of a wide range of BERT models trained in different languages.
In particular, several pre-trained Italian BERT models are currently available and they can be fine-tuned for some specific tasks. More information about these models are visible in table 2.5.

Name	**Architecture**	**# Parameters**	**Training Data**	**Size Training**
AlBERTo [5]	BERT	110M	200M Italian Tweets	128k terms Vocabulary
Italian BERT [24]	BERT	110M	OPUS Corpora Collection + Wikipedia Italian Dump	13GB
Italian BERT XXL [24]	BERT	110M	OPUS Corpora Collection + Italian OSCAR Corpus	81GB
Italian ELECTRA [24]	ELECTRA	110M	OPUS Corpora Collection + Italian OSCAR Corpus	81GB
GilBERTo [35]	RoBERTa	110M	Italian OSCAR Corpus	71GB
UmBERTo uncased [88]	RoBERTa	110M	Texts extracted from Wikipedia-ITA	7GB
UmBERTo cased [88]	RoBERTa	110M	Italian OSCAR Corpus	69GB
GePpeTto [34]	GPT	117M	Wikipedia Italian Dump + ItWac Corpus	15GB

Table 2.5: Pre-trained Italian BERT models

As is clear from the table 2.5, there are several other architectures in addition to BERT.
RoBERTa (Robustly optimized BERT pretraining approach) is a variation of BERT developed by a Facebook team in order to improve the original implementation. They trained BERT on a larger dataset and they introduced the *dynamic masking* which consists of changing the masked token during the training procedure. [22]
ELECTRA changes the pre-training approach by substituting the Masked language Modeling task with a Replaced Token Detection. The main idea behind this approach is to replace a token with some plausible alternatives generated by a generator network instead of masking completely the token. Then the model trains a discriminative model to predict whether a generator sample replaced tokens in the input. [28]

GPT(Generative Pre-trained Network), is a large transformers-based language model trained on an enormous quantity of data. The objective task is to predict the next word given all of the previous words within some text. [12]

Another fundamental aspect that may vary in the models, is the typology of the training data used for the pre-training part.
The data sources used for training the models are:

- **Twitter**: The tweets available on the social network;
- **OPUS Corpora Collection**: A collection of text translated from the web [71];
- **Wikipedia**: a free, collaborative, multilingual online encyclopedia;
- **OSCAR Corpus**: a project which aims to process data from a wide range of sources on the web, from news sites to forums and more [70];
- **ItWac Corpus**: an Italian corpus made up of texts collected from the Internet [52].

2.6 Evaluating Machine Learning Models

One of the most important aspects of developing a machine learning model is testing its performance in order to determine if the model can generalize well on data that has not been seen before.
A common approach involves splitting the available data into three different sets: a **training** set, a **validation** set and a **test** set. The common percentage size for the sets may vary depending on the specific problem or the size of the total dataset. Typically, the sizes of the training, validation and test sets follow a common practice of approximately 60-80 for training and 10-20 for each of the validation and test sets.

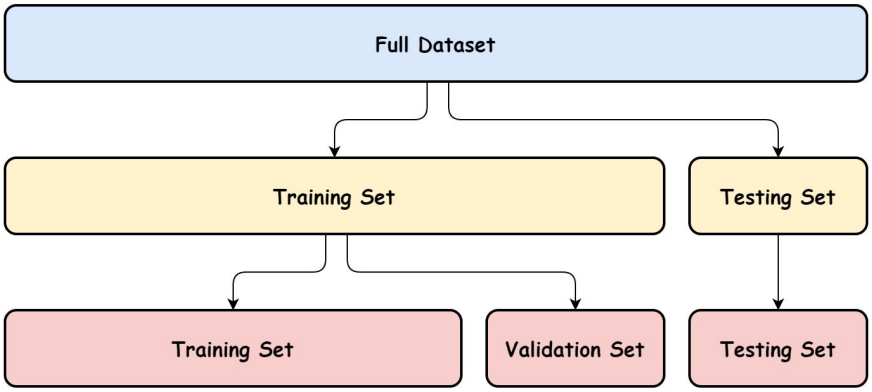


Figure 2.8: Typical Dataset split [23]

The largest set is utilized to familiarize the model with patterns and relationships with the data. This enables the model to make predictions on new data.
Following training, the purpose of a validation set is to evaluate the model's performance thanks to some metrics that will be explained later. Moreover, various hyperparameters (e.g. hyperparameters, epochs, etc.) are tuned in order to optimize the behaviour.
Conversely, the test set serves as a final evaluation, with the key requirement being that it contains unseen data, in order to provide an estimation of how the model performs well on new data. [45]
In evaluating the model's predictive capacity, it is vital to use appropriate metrics, and in the following sections, the most commonly used metrics for classification tasks will be discussed.

2.6.1 Confusion Matrix

A confusion matrix, also known as error matrix, is visible in figure 2.9 and is a table used to evaluate the performance of a trained model for classification tasks. [39]

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Figure 2.9: Confusion Matrix

The columns indicate the model's predicted label, while the rows refers to the ground truth value. In this way there are four possible outcomes:

- **True Positive (TP)**: how many time the model predict correctly the positive class;
- **False Positive (FP)**: how many time the model incorrectly predict the positive class;
- **True Negative (TN)**: how many time the model predict correctly the negative class;
- **False Negative (FN)**:how many time the model incorrectly predict the positive class.

Greater values in the diagonal part of the matrix indicate a better performance from the model.
In the next sections, more metrics for classification tasks computed with the value available in the confusion matrix are explained.

2.6.2 Accuracy

Accuracy is one of the most popular metrics for classification tasks, it indicates how many times a model predicts correctly an element over the total dataset.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

The formula of the Accuracy is visible in the equation 2.1, which is the ratio between the correct predictions and the total possible elements. It is an overall metric of how much the model is correctly predicted on the entire set of data.
One problem with this metric is that does not take into account potential dataset imbalances [39]. For this reason, there are other metrics that may be more useful and concise.

2.6.3 Precision and Recall

Precision, which can be computed with the equation available 2.2 specifies how many positive elements have been classified as actual positive. Thus it explains how often we are correct when we classify a class as positive, thus how much we can trust the model when it predicts a sample as positive. [39]

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

With a high precision value, we have a model that is correctly predicting positive cases and it is making few false positive predictions. Otherwise, a low value indicates that the classifier is predicting incorrectly and generating many false positive elements.

Recall, on the other hand, is a measure that indicates how many of the true positive classes the model can detect. The formula is is visible in 2.3.

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

Intuitively, it measures the ability of the model to find all the Positive units in the dataset, so a high recall indicates that the model is effectively identifying positive cases, otherwise, the classifier is missing them.

2.6.4 F1-Score

The last metric that is presented is the F1-score, which combines both Precision and Recall.

$$Recall = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.4}$$

The F1-Score is the harmonic mean of the two metrics previously described, to compute it is possible to use the equation in 2.4. [39]
This metric is very useful for evaluating the performance of a classifier because it takes into consideration both precision and recall.

Chapter 3

Data Analysis

Data analysis is a crucial component when working with artificial intelligence techniques, particularly in the Natural Language Processing field.
Indeed, this area involves complex and multifaceted data which may not be properly structured.
Languages contain several aspects that highlight the importance of analysis and preprocessing.
For instance, its ambiguity might lead to multiple interpretations which can be reader-dependent. In addition, depending on the geographical, cultural and social context, it can be highly varied and difficult to interpret.
Moreover, the lack of a rigorous structure, such as that of programming languages, further complicates the execution of tasks involving natural language.
This section presents the data analysis performed on an internal dataset of Pat Srl which was already classified with the sentiment value. The dataset contains textual data from a gaming customer.
It was labelled with different kinds of categorizations which will be analyzed.
The goal of this section is to investigate and understand data characteristics, such as length of the sentences and sentiment distribution or any kind of information which may be useful for the task.
Furthermore, an additional non-categorized dataset, which sources regards both chat and email, will be analyzed in order to gather more data for a possible machine learning model.
Preprocessing techniques are being analyzed since their use might significantly influence the final metrics' outcome.
The intricate nature of tasks such as sentiment analysis makes an investigation of these techniques necessary, as some facets of texts such as casing or stopwords can play an important role in underlying emotions of a sentence.
Finally, a description of techniques used to mitigate dataset imbalance will be presented. While common methods like undersampling or oversampling offer straightforwards solutions, a more sophisticated and context-aware approach will be presented, leveraging additional datasets to provide a comprehensive understanding.
Moreover, all the dataset reviews mentioned in chapter 1 will be described.

3.1 Categorized Dataset

In this section will be analyzed the two already categorized datasets, extrapolated from users' question and response within the chatbot.
Indeed, the dataset was already created in PAT Srl but it has never been used before.
Each dataset employed in this project is structured in a tabular format with rows and columns.
Since the main goal is to predict the sentiment values of these sentences, the primary focus during the data analysis will be on the actual users' sentences and the *Sentiment* which indicates with what sentiment the sentence has been categorized.

3.1.1 First Dataset

In the first dataset, the available columns were "*Text*", "*Sentiment*", "*Thread*", "*From*". The latter two serve as providing information regarding chat thread and the corresponding user who sent the message.
In this use case, the focus will be only on single messages available in the "*Text*" column, and on the sentiment value manually assigned available in the column "*Sentiment*".
Within this particular user case, the primary focus will be on single messages with the relative associated sentiment value.
In this first dataset, the sentiment column contains 4 different categorization:

1. Very Negative;
2. Negative;
3. Neutral/Positive;
4. Very Positive.

The dataset was manually labelled by a team member who evaluated the sentences from the customer's perspective, identifying offensive and inappropriate messages as negative or very negative.
The total number of sentences available in this dataset is **5.087** which were categorized with one of the possibility written before.

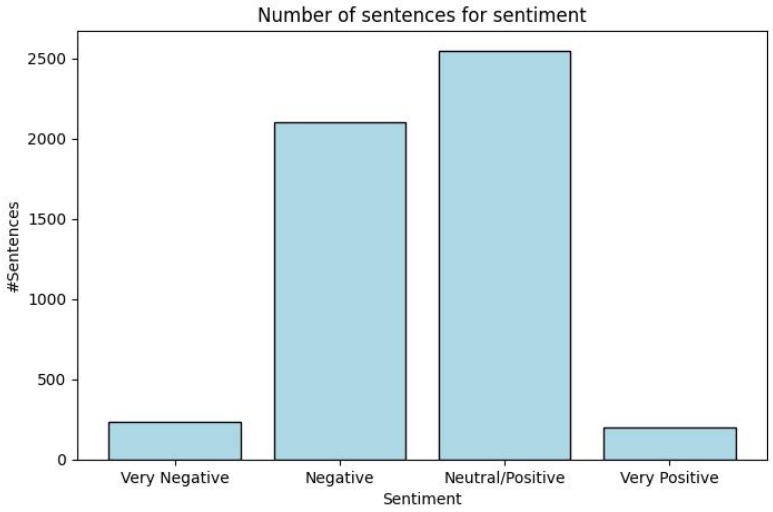


Figure 3.1: First Dataset Label Distribution

Figure 3.1 illustrates the imbalanced distribution of the several sentiments, particularly evident in the *very negative* and *very positive* categories.
The difference between classes with a high and low number of elements is roughly 2.000 items, as it show in the table 3.1 where precisely is possible to see the amount of phrases for each categorization.
Within the field of machine learning, this dataset may be considered as unbalanced, give that there are classes which has more observations than others.

Sentiment	**Number of Sentences**
Very Negative	236
Negative	2.106
Neutral/Positive	2.545
Very Positive	200

Table 3.1: Sentences Distribution by Sentiment in first dataset

In addition to the 5.807 categorized sentences, other 636 elements were available within the dataset.
However, those elements were not categorized, thus, their sentiment contains a *None* value.
By analyzing these sentences, it was noted that they were difficult to categorize with a particular sentiment value.
Indeed, choosing the best categorization is not always that trivial, developing linguistic resources, and more generally data annotation, involves the making of subjective

judgements.
Moreover, deciding which of these judgements are reliable and reproducible has become increasingly important in order to provide resources which are noise-free and they can be used for automatic systems. [36]

Ip 15215-15215
??????
veramente
ma niente
1519d08b0000017f5200
Ore 18.44

Table 3.2: Some example of uncategorized elements

In table 3.2 are shown some instances of the uncategorized sentences from the dataset. This type of phrases can be codes, numbers or texts that cannot be categorized precisely.

3.1.2 Sentiment Field Consideration

A comprehensive analysis regarding the two categories *Neutral/Positive* and *Very positivie* have been conducted.
Indeed, the question attempted to be answered was whether it is truly beneficial to have four different catogrizations and whether this multi-classification give some advantages for customers.
To answer these questions some samples from the categories were were selected and analyzed.

Sarebbe possibile avere un bonus??
Salve volevo sapere quanto tempo ci vuole per chiudere un conto
Salve sono il sig.<NAME> dovrei avere indietro un bonus scommesse
Buongiorno, vorrei chiudere il mio conto gioco' 'Si devo avere oggi
c'è stato un problema di caricamento della pagina ed erroneamente le ultime 2 partite sono uguali ...

Table 3.3: Neutral/Positive sample sentences

grazie
Ok grazie
buona sera
Va bene
BUONGIORNO
grazie della disponibilità

Table 3.4: Very Positive sample sentences

As can be noted, *Very Positive* sentences, which are visible in table 3.4, are mostly gratitude sentences (grazie, grazie della disponibilità) or general greetings (e.g. BUONGIORNO, buona sera).
Concerning instead the *Neutral/Positive* sentences in table 3.3, they exhibit a high degree of generality and they mainly regards users request information.
From a customer's perspective, having an additional categorization only dedicated to greetings may not provide any significant added value.
Moreover, as will be seen in the following sections, the second set of data does not contains this particular categorization.
Taking these factors into consideration, it was decided to aggregate the *Very Positive* category into the *Neutral/Positive* classification, thus, remaining with three different labels.

A similar line of reasoning used for the *Positive* sentiment, has also been applied to the *Negative* values.

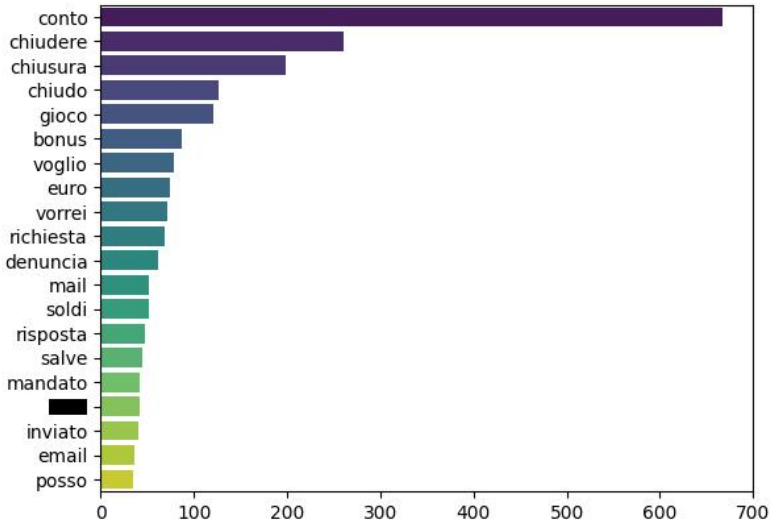


Figure 3.2: 20 most used words in Negative sentiment

Figures 3.2 and 3.3 showcase respectively the top 20 most frequently used words in

Negative and *Very Negative* sentiment groups, excluding stopwords.
Both categories contains words which refers to an account closure, like *conto*, *chiudere* or *chiusura*.
Instead, only in the *Very Negative* classification are visible words that indicate a possible denunciation and therefore a threat.
Since the goal is to detect which users are are not satisfied, the current classification can provide to the customer a better knowledge about the feeling of the user.
However, one potential improvement could limit the frequency of sentences mentioning an account closure within the *Very Negative* category, so as to be able for a possible model to better classify the sentiment without ambiguity.

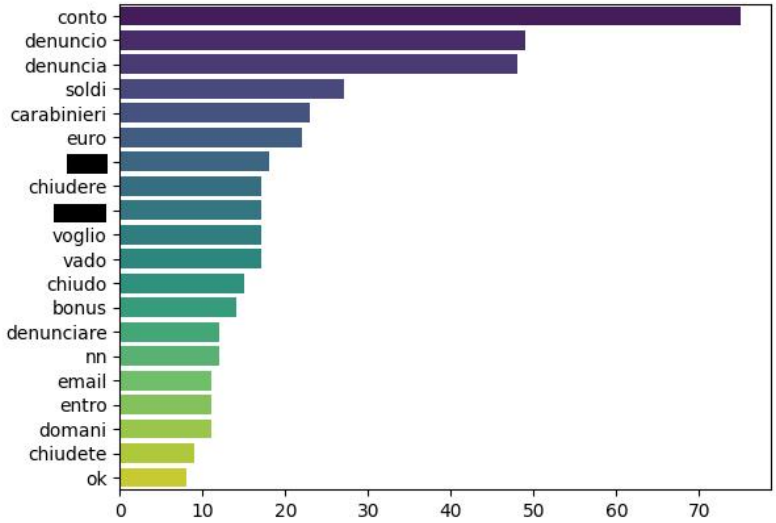


Figure 3.3: 20 most used words in Very Negative sentiment

Alternatively, both the negative groups can be merged in one single category having only two sentiment value (Neutral/Positive and Negative).
In the view of training a possible artificial intelligence model, this choice might simplifies the task by reducing it into a binary classification, which potentially could be easier compared to a multi-classification problem.

3.1.3 Second Dataset

This dataset, as the previous one, was extrapolated from user' question and response within the chatbot, however, additional information are available, indeed, it consists of nine columns *ID*, *Question ID*, *Service*, *Thread*, *Data and Time*, *From*, *To*, *Text*, *Sentiment*.
As previously done, the focus will be on the *Text* and *Sentiment* columns, as the remaining columns provide information about the overall chat rather than the individual messages.

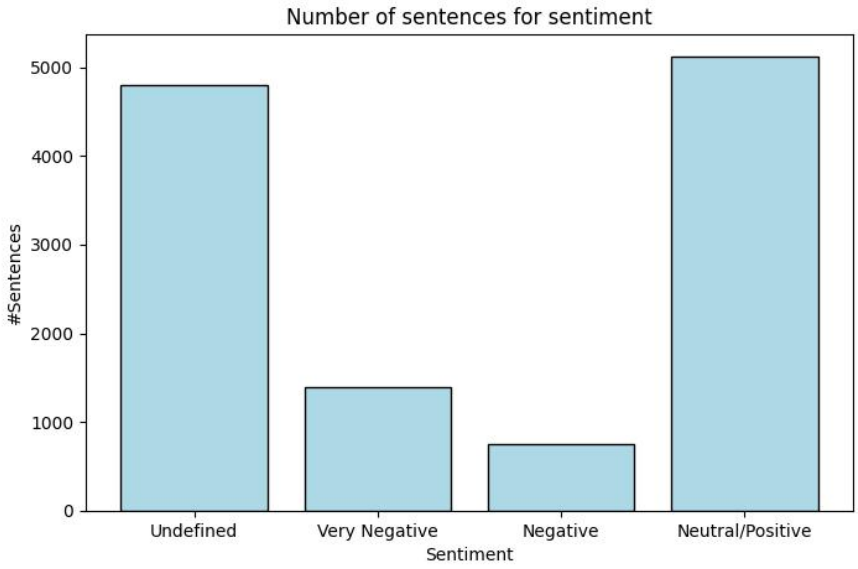


Figure 3.4: Label distribution second dataset

This dataset, as can been seen in figure 3.4 and in table 3.5, have an additional categorization value: *Undefined*.
There are 4796 entries classified as uncertain out of a total of approximately 12.000 entries.

Sentiment	Number of Sentences
Undefined	4796
Very Negative	1402
Negative	757
Neutral/Positive	5116

Table 3.5: Number of sentences for each sentiment in the second dataset

This undefined value is included to specify where the text lacks of clarity, the user's feeling have not been comprehended from the annotator or the language is not clear; examples of such messages are visible in table 3.6.

magione
Gg
,
Non vogliamo mai
..?
gtrgrgr

Table 3.6: Undefined sample sentences

Thus, some of the sentences can be characterized as being incomprehensible or dictated by angry outbursts.
However, by analysing carefully more sentences of this class, it is possible to find expression which have a concrete meaning and thus a possible categorisation.
One simple heuristic that we can use in order to get rid of incomprehensible text is to remove all the sentences which contains only a single words and leave the sentences that potentially could be categorized.

3.1.4 Dataset Concatenation

From the two given datasets, a unique dataframe were created by concatenation, however, without considering undefined sentences from the second dataset.

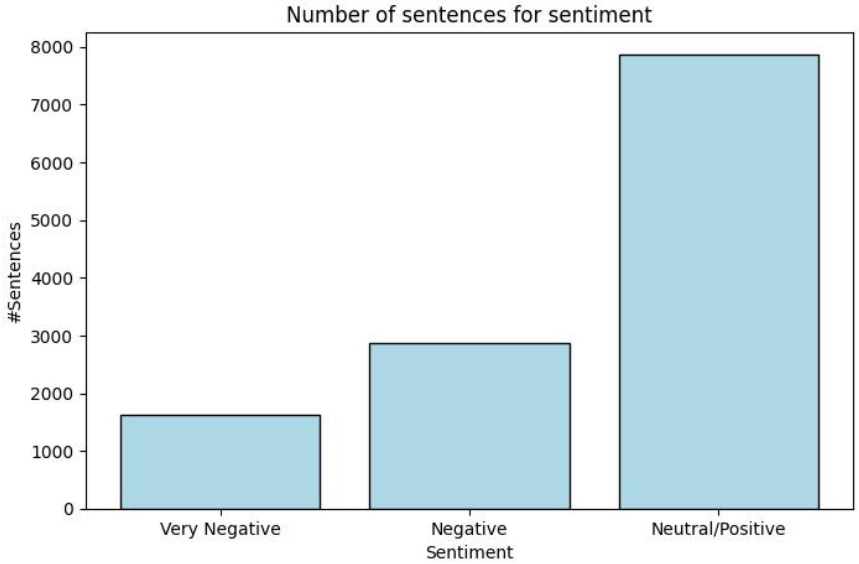


Figure 3.5: Label distribution concatenated dataset (*CustomerData_ 1*)

In Figure 3.5 and in table 3.7 is possible to see the distribution of the final dataset and the specific number of entries for each sentiment.

Sentiment	**Number of Sentences**
Very Negative	1638
Negative	2863
Neutral/Positive	7861
Total	**12.362**

Table 3.7: Number of sentences for each sentiment in the concatenated dataset (*Customer-Data_1*)

In view of what can be seen in figure 3.5 and table 3.7, the dataset has a large number of elements which are *Neutral/Positive*.
Instead, *Negative* class has least than the half of the major class, while *Very Negative* sentences reach more or less 1600 elements.
As is evident, this set of data is unbalance, even if we try to aggregate the negative categories into one unique class.
By virtue of this, some techniques to counteract the imbalance will be necessary.

General Analysis

A more general analysis was conducted on the concatenated dataset, indeed, this paragraph will provide valuable insights of the data.
As might have expected from the previous plot in figure 3.2 and 3.3, in the first position of the top 20 words used in the entire dataset there is the word "*conto*".
Other popular words, which are visible in figure 3.6 are "*bonus*", which is present about 1200 times, or "*operatore*".

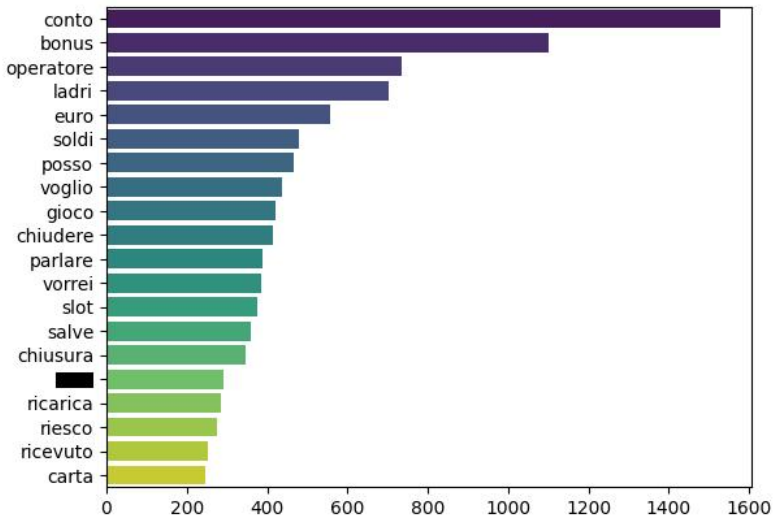


Figure 3.6: 20 most used words in the complete dataset

Furthermore, can be also observe certain content which can be directly connected to a potential negative sentiment expressed by the user: the word "*ladri*" (thieves).
Depending on the context, this term can have both a neutral and negative connotation; the same reasoning can be applied also to all the other words, except for the obscured word, which happens to be a profanity frequently employed in sentences with negative connotations.
Context is a crucial variable that must be taken into account when performing tasks such as sentiment analysis. The potential of BERT, as viewed in the previous chapter, lies in its ability to perform context inclusion thanks to the pre-training phase.
In this particular use case, as the data comes from final user chats, when the word "*ladri*" is used, the sentence can be perceived as negative, since it is a direct message from the user to the client.
Trying to be more precise and make some assumptions, a gaming company which receives many messages containing the word "thieves" could consider this message as offensive and therefore negative.
This type of company operates in an ever-growing and very peculiar sector that can lead to dissatisfied users for many reasons. Therefore, these comments could be dictated by outbursts of anger by the user who perhaps was not completely satisfied or simply because they lost in some way.
Trying to be more precise and making some assumptions, assuming a gaming company receives numerous messages containing the word "thieves", such messages may be

considered as negative.
This kind of business operates in an ever-growing and particular industry where customer dissatisfaction can arise from various factors.
Consequently, the comments could be dictated by outbursts of anger by the user who perhaps was not completely satisfied or simply because he lost in some way.

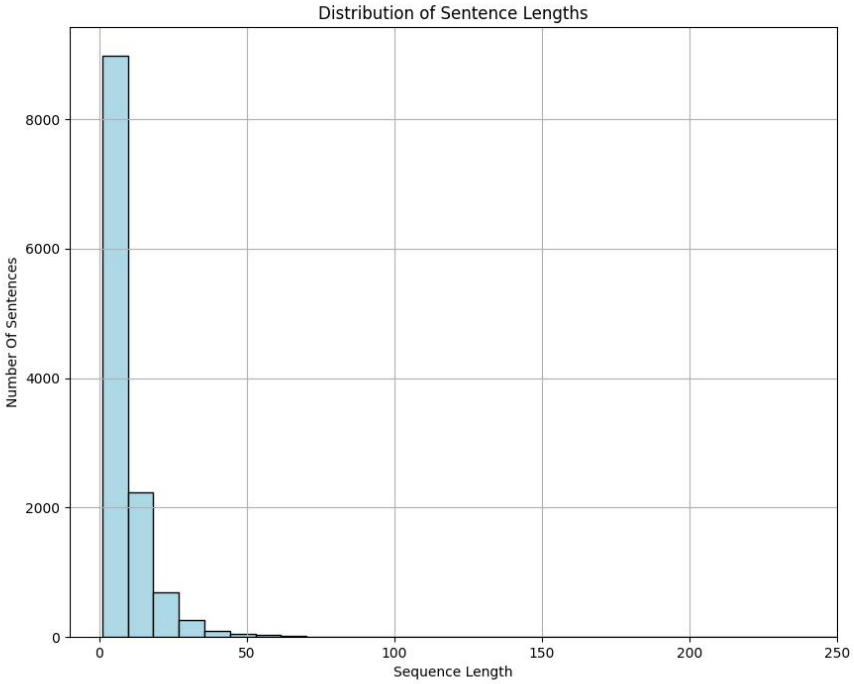


Figure 3.7: Sentence length distribution

Concerning the length of sentences, in figure 3.7 can be seen the sentence length distribution of the entire dataset.
As one might have expected, most of the phrases have a size between 1 and 50 words, due to the fact that the data comes from chat sources, which usually consist of short messages as opposed to other sources such as email which can potentially have longer text.
The sentences' split was performed by dividing the text in words, thus the spaces are not considered in this plot but they are the character used to determine the word boundaries.

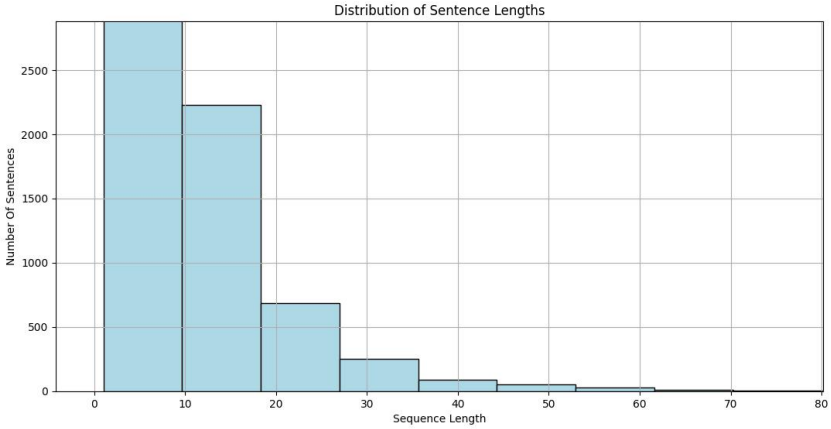


Figure 3.8: Sentence length distribution (2)

In figure 3.8 is possible to see a more detailed picture of the distribution.
More specifically, there are more than 8000 sentences, which represent the 76% of the entire dataset, with lengths between 1 and 10 words.
The number of sentences then drastically decreases; there are roughly 2300 sentences that are between 11 and 18 words in length and nearly 650 sentences between 10 and 27 words in length.
With the help of this research, some outliers were even found; in particular, a sentence with a length of 434.
Investigating this element, was noted that it was not a phrase written by a human but just a text containing some special offers that were available.
Presumably, a user needed some information regarding the promotion, thus copying the text into the chat in order to automatically have details about it from the chatbot.
The sentiment given for this expression was *Neutral/Positive*.
Given this and the requirements that the sentiment to be predicted must pertain to human-written text, it was decided to remove the sentence.

3.2 Non-Categorized Dataset

Data play a key role in the field of artificial intelligence as they are an important component for developing solutions based on automatic reasoning.
Typically, gathering more data leads to better results, thereby enhancing the performance of an application.
For this reason, one activity was undertaken, which involved the analysis and extraction of sentences from an additional dataset that could be classified with a sentiment.
In this set of data, which contains 173.391 elements, the available columns are: *ID Ticket*, *Data Creazione*, *ID Richiedente (Sito)*, *Oggetto*, *Richiesta*, *Soluzione*, *ID Sorgente*, *ID Gruppo Owner*, *ID Servizio*, *Servizio*.
Directing the attention to the "*Sorgente*" column, it was observed that it comprises several sources of text with values such as "*Email*","*Chat*","*Operatore*","*IVR*","*Operatore RST*".
Phrases extrapolated from email and chat, which account for the majority of dataset,

namely 172.961 rows, will be the primary focus.
Other columns such as "*Servizio*", do not bring useful information, as it only takes on a unique value, namely "*S3C*", thus they will not be considered.
The feature that includes the users' messages is "*Richiesta*", and, as will be seen in due course, sentences need to be cleaned in order to obtain the actual request of the users.

3.2.1 Chat Source

By concentrating solely on chat messages, the resultant dataset will comprise 80.132 tickets, some of which only consist of the sentence "*Ticket creato automaticamente della canalità chat.*"
Without any doubt, all these elements can be dropped, since they do not contains human-written messages and thus they are not categorizable.
The remaining 10.331 tickets can be analyzed.
By analyzing the column containing the user's requests, it was possible to find some generic and usual patterns which are visible in table 3.8.

username: *message*
Gestione chat_Il cliente chiede un bonus username: *message*
Gestione chat. Clt chiede un bonus generico. Grazie username: *message*

Table 3.8: Pattern

In particular, the username of the customer followed by dots is the point in which is possible to obtain the real message of the user.
Instead, in other cases, a more complex structure of the sentence is visible, which however remains the same and consequently a possible pattern to be removed.
In table 3.9 are shown some examples of the sentences containing the pattern.

Gestione chat_Il cliente chiede bonus USERNAME: *Ciao, domando: ieri sera puntavo alla roulette anche più di 100 euro alla volta ma non ho ancora avuto nessun bonus per questo*
Gestione chat_Il cliente chiede un bonus USERNAME: *mi puoi accreditare un bonus di omaggio 100€*
Gestione chat. Clt chiede bonus generico .Grazie USERNAME : *Ciao posso avere un bonus per favore*
USERNAME: *volevo sapere se era possibile richiedere un bonus visto la quantità di giocate da me fatte ultimamente*
USERNAME: *Volevo chiederle se fosse possibile ricevere un bonus*

Table 3.9: Sentences examples with pattern

Every user has their own username, thus, it makes it difficult to identify correctly the sentences and identify the pattern.
Nonetheless, sentences which follow this pattern consistently start with the customer's username, followed by a colon.
Additionally, a username analysis reveals that they are written only with letters or numbers, avoiding any special characters.
Consequently, this observation provides an effective means of identifying the username.
A Regular Expression was used in order to extract the messages from the pattern:

```
^(Gestione chat(\_|\.).*bonus\s?(generico\.\sGrazie\s)?)?[A-Za-z0-9]+:(.*)
```

These 8032 elements were found and some instances are visible in table 3.10.
However, even in this case, some outliers have been detected: 1025 elements which do not contains human-written elements.

è possibile avere un bonus omaggio?
*voglio e pretendo un bonus di almeno €50 altrimenti prendo provvedimenti e non verso più un ******
ciao buongiorno scusa se è possibile si può qualche bonus
salve è possibile avere una ricarica bonus oggi impossibilitata a ricarivare non mi accontetate mai nonostante le mie numerose e corpose ricariche

Table 3.10: Extrapolated sentences from patterns

3.2.2 Email Source

One important column in the email source is "*Oggetto*" (subject), indeed, it contains more information with respect to the previous chat source.
According to the subject column, the dataset was divided. Email with *"Website Feedback"* as subject, contains a different pattern with respect to the others, as in the message is specified that the user sent the email using the form available on the customer website.
Furthermore, all the emails that had in the object the word "*Re*" were separated, as they correspond to two or more emails exchanged between users and operators.
Regarding email with subject *Website Feedback*, some examples are shown below, separated by the character -.

```
DA:  (EMAIL)
A: NOME (EMAIL)
CC:
OGGETTO: [Website feedback] Ricariche e Prelievi

11785495 (URL [1]) ha inviato un
messaggio usando il form contatti su URL [2].

si puo avere 1 bonus omaggio reparto da quando mi avete fatto il regalo ho
caricato altri 200 a passare grazie
```

```
[1] URL
[2] URL

---------------------------------------------------------------------

DA:  (EMAIL)
A: NOME (EMAIL)
CC:
OGGETTO: [Website feedback] Promozioni

6031980020025988 NOME (MAIL [1]) ha
inviato un messaggio usando il form contatti su URL [2].

Ciao io sono NOME per me da arrivare ancora Bones di tutti
ok per favore mantattai Bones ok no bloccati mio conto ok io ieri pura fatto
400 ok già perzzo voi ok  io vinci mai però io voleva giocare ok


[1] mailto:MAIL
[2] URL
```

As is visible, even in this case there is a specific pattern that can be found in most of these emails:

```
DA: (EMAIL)
A: ... (EMAIL)
CC:
OGGETTO: [Website feedback] ...
... ha inviato un messaggio usando il form contatti su URL [2].

message

[1] ...
[2] ...
```

However, this is not a properly common pattern for every sentence since emails and names are different for every consumer.
With a regular expression that defines a generic email is possible to get rid of this pattern and obtain a specific message from the users.
The regex that permits obtaining a generic email is

```
[A-z0-9._%+-]+@[A-z0-9.-]+\.[A-z]{2,}
```

As regards the URL, it never changes since is the contact form of the company.
By removing this pattern in these kind of emails, is possible to get the specific message from the user and the total entries obtained with *Website Feedback* emails is 12.307.

A first cleaning procedure, with a similar approach as seen before for the email with the subject "*Website Feedback*", was performed for the email with the subject "*Re*" in order to get the specific message.
Some examples can be found in the table 3.11.

However, it is not trivial to extract the actual user's message in such emails from the entire conversation.
In fact, the number of emails exchanged between a user and an operator could be high. This then leads to a sentence that contains considerable noise, including various responses received and email signatures.
In this case, it was decided to take the first message that can be found, leaving out the operator's replies or other user requests from the user.

Messages Obtained
Salve viste le ricariche effettuate e i movimenti fatti dal mia conto gioco sarebbe possibile ricevere un bonus? Numero conto gioco:11835709.
MALEDETTI LADRI MA E POSSIBILE CHE SAPETE SOLO TRUFFARE...SEGALERO TUTTO ALL AAMS....COSI VEDIAMO SE AVETE RAGIONE VOI.
Buongiorno, io ho aperto da poco questo conto gioco ma non ho mai ricevuto nemmeno tutt'oggi il bonus di benvenuto senza deposito vorrei avere spiegazioni.
Buonasera,È possibile ricevere un bonus giocabile?Cordiali saluti.
Buongiorno. Ho effettuato due ricariche da 20 euro ciascuna, ma non ho i soldi dentro a disposizione per giocare anche se mi ha dato in entrambe le operazioni "versamento effettuato con successo". Come mai?.
11 euro...? Metteteveli...nel.....Il mer 30 mar 2022, 12:38 CUSTOMER_NAME <CUSTOMER_EMAIL.
Salveda oggi non sarò piu una vostra clientenon dico di vincere ma i+almeno un pò di divertimento.
Non mi interessa il messaggio che mandate in automatico esigo RISPETTO!!MANDATEMI UN ACCREDITOIl mer 16 feb 2022, 20:45 CUSTOMER_NAME <CUSTOMER_EMAIL
Salve non riesco a recuperare la password del mio CUSTOMER_NAME e quando metto il codice fiscale mi dice che è impossibile recuperarlo.
siete un sito vergognoso, software di poker piu truccato di quello di pokerstars,fate vomitare,da me non vedrete mai piu un centesimo ladri.

Table 3.11: Sentences from *"Re"* emails

Analyzing the sentences, a substantial number of them ask the firm for a bonus. Some of them potentially could be considered as Neutral/Positive since inappropriate or offensive terms are not used.
Conversely, others are explicitly negative in tone.
Some sentences, however, still need to be processed in the best possible way as they retain some elements that are unnecessary, such as date, time and email.
Below is available an example of a request that contains more than one email.

```
DA: USER_NAME (USER_EMAIL)
```

```
A: CUSTOMER_NAME (CUSTOMER_EMAIL)
CC:
OGGETTO: Re: CUSTOMER_NAME risponde [T7242260C]

Buongiorno..in quanto a professionalità direi scadente pur essendo un
grande sito di scommesse,dite e scrivete di approvare i voucher entro 72
ore,ne stanno passando 150 di ore e ancora nulla,voglio dire sono 40 euro
mica 400.000..chissa quando avrò il privilegio di poter ritirare la mia
vincita, intanto io continuo a non giocare su questo conto,e mi sa che lo
abbandonerò dopo queste disavventure..dai diamoci una mossa.....!!!!!!!!!

Il ven 4 mar 2022, 12:34 CUSTOMER_NAME <CUSTOMER_EMAIL> ha scritto:

> Ciao USER_NAME,
>
> Dopo aver inviato la richiesta, il voucher sarà disponibile entro un
> termine massimo di 72 ore necessario per svolgere verifiche sulla sicurezza
> del conto gioco. Potrai in ogni momento verificare lo stato della richiesta
> accedendo a Il Tuo conto -> Prelievi da elaborare/Prelievi voucher
> approvati.
>
> Saluti
> CUSTOMER_NAME
>
> ------------------------------------------------
> La tua richiesta era:
>
> DA:  (USER_EMAIL)
> A: CUSTOMER_NAME (CUSTOMER_EMAIL);CUSTOMER_NAME (CUSTOMER_EMAIL)
> CC:
> OGGETTO: Ricariche e Prelievi
>
> volevo sapere quando mi approvate il voucher di prelievo grazie
>
```

An optimal idea would be to take all sentences/requests from the user.
In this case, as visible in the example above, the first phrase could be categorised as negative, while the second (*volevo sapere quando mi approvate il voucher di prelievo grazie*) could be instead categorised as positive.
The implemented idea instead took only the first sentence, as since each message may vary the number of e-mails exchanged and it may be difficult to retrieve correctly all the user's messages.

3.3 Text Preprocessing

Cleaning and transforming the raw data text into a more manageable format is an essential step in a natural language processing pipeline.
The preprocessing application to the data, such as tokenization, stopwords removal, and noise reduction, can significantly enhance analysis efficiency and, especially, improve the performance metrics of the chosen model.
The following section will discuss the preprocessing techniques analyzed and used for the dataset.
The results that these methods provide to the model's metrics will be explored in detail later on the next chapter where a deep study regarding the model development will be explained.

3.3.1 Emoji Removal

In contemporary communication channels like chats or social networks, many people use emojis to express themselves and the usage of such expressions has grown significantly. Their inclusion in tasks such as sentiment analysis might affect the overall sentiment of a sentence, by taking them into account can lead to better accuracy of the results. [16]

However, upon careful examination of the dataset, it was revealed that it was devoid of any kind of emoji.
The inspection was conducted with a combination of specific Python libraries for emoji analysis [30] in conjunction with regular expressions and their corresponding unicode definitions.

The absence of emojis could be attributed to the context or the unsuitability of their usage within the business-oriented chat environment.
While social media platforms provide the most conducive setting for the effective use and interpretation of emojis, in this case, chats are derived from business activity.
Thinking about a future model that will perform the sentiment prediction, it will be trained on an emoji-free corpus, thus for this reason, in the preprocessing pipeline that will be used for cleaning the new data, the emoji will be removed.

3.3.2 Sentence Capitalization

Most of the time, the capitalization of words often plays an important role in the field of sentiment analysis.
Typically in contexts such as chat and social networks, the use of uppercase or lowercase might have distinct meanings.
For instance, uppercase letters may manifest someone who is shouting, potentially reflecting anger or strong emotions.

Therefore, the consideration of this preprocessing step must be taken in account this particular scenario.
As already seen in chapter 2, an approach that tries to avoid treating equally every word is called *Truecasing*.
However, before actually using it, the dataset was analyzed in order to understand whether, the tendency to use uppercase, is often present within the negative category

of the dataset.

Sentiment	# Sentences	# UP Sentences	% Over Total
Very Negative	1.638	861	7 %
Negative	2.863	71	0.6 %
Positive	7.861	160	1.3 %
Total	12.362	1.092	8.9 %

Table 3.12: Uppercase sentences statistics

As is visible in table 3.12, the 8,9 % of the total dataset contains sentences completely written in uppercase.
Moreover, most of this percentage is composed by phrases which are labelled as very negative, thus we can see slightly the tendency mentioned before.
To apply the truecasing methodology in the dataset is necessary to train a model to understand the frequency of the uppercase words.
To achieve these results, the truecaser developed by the University of Tartu was used. It processes sentence-first and all uppercase words using a unigram probabilistic approach. [84].
The outcome obtained after the model application, are available in the table 3.13, with a substantial decrease of the upper case sentences with sentiment negative and positive.

Sentiment	# Sentences	# UP Sentences	% over total
Very Negative	1.638	697	5,6 %
Negative	2.863	3	0.02%
Positive	7.861	5	0.04%
Total	12.362	705	5,7 %

Table 3.13: Uppercase sentences statistics after truecasing

One point that has to be noted is that the truecasing is not necessarily always better. Indeed, in the following tables, two sentences were taken as examples. The first one categorized as neutral/positive, and the second one as negative.

Original Sentence
HO RITROVATO UNA MIA VECCHIA SCHEDA SULLA QUALE VI è UN ATTIVO. nON HO PIù LE CREDENZIALI, COME FARE PER RIATTIVARLA
IO PENSAVO CHE ALMENO SULLE SCOMMESSE ERAVATE UN MINIMO SUFFICIENTI. IO NON HO MAI VISTO UNA COSA SIMILE IN 20 ANNI

Table 3.14: Example of original sentences

Truecased Sentence
ho ritrovato una mia vecchia SCHEDA sulla quale vi è un attivo. nON ho PIù le credenziali, come fare per RIATTIVARLA
io pensavo che almeno sulle scommesse eravate un minimo SUFFICIENTI. io non ho mai visto una cosa simile

Table 3.15: Example of truecased sentences

As is evident from sentences available in table 3.14 and 3.15, this methodology is keeping some specific words in capital letters.
The capitalization of the words does not readily convey whether the sentiment is positive or negative in both the original and truecased versions.
For instance, in table 3.14, the first sentence is categorized as positive, despite being mostly written in capitals.
However, the second sentence which is categorized as very negative, can be subjected to the same reasoning, but this time, the casing can be considered as part of an assumption that leads to conclusion that sentence is negative.
Moreover, performing truecasing on these sentences, will not give the possibility to a better exploitation of the available information.
That's why, a common approach with lowercase might be more helpful with respect the truecasing.
Models will treat each word in a equal manner, in this way they will not be confused from any kind of strange capitalization.

3.3.3 Stopwords removal

A general step of a preprocessing pipeline is the stopwords removal.
Indeed, these kind of words may introduce noise into sentences, thus, removing them, might help to captures relevant features or keywords and get the main essence of a text.
However, is not always a suggested approach specifically for some task where each word is meaningful such as machine translation, text summarization or question-answering problems.
In contrast, removing stopwords can enhance the accuracy of several classification tasks, like spam filtering and genre classification, by highlighting the essential components of the sentences.

As regards sentiment analysis, it has been showed that in the case study of Twitter, the using of pre-compiled stopwords has a negative impact on classification performance. [42]
Moreover, after conducting a general analysis of the dataset, it was discovered that the 35% of the dataset's total word count, consisted of stopwords.
Consequently, it is important to evaluate their suitability for the task at hand. This finding raises the question of whether including stopwords would positively or negatively impact the learning process and the resultant performance of the model.
Therefore, a thorough investigation is warranted to determine the most appropriate approach.
For this reasons it was decided to test the impact of the stopwords with the given dataset and the results are available in the Development and Validation Model chapter 4.

3.3.4 Text Normalization

Despite the dataset is not directly collected from social networks like Twitter, the available features are similarly noisy to the features which can be found on the web. Indeed, is possible to notice that there are particular instances of entities that do not contribute to the sentiment identification within the sentence.

Features	**# Sentences**
Money	742
Date	111
FreeSpin	96
Time	84
Percentage	32
Phone	26
Email	22
Ticket	21
Punctuation Only	9
Fiscal Code	6
Total	1149

Table 3.16: Features Statistics

Abbreviation	**# Replacement**
x	per
grz	grazie
cmq	comunque
xke	perchè
xk	perchè
ke	chec
qlc	qualche
nn	non
xò	però
dv	dove
tt	tutto
gg	giorni
k	che
nnt	niente

Table 3.17: Abbreviation List

Table 3.16 illustrates the frequency of sentences containing distinct features that are typically irrelevant to sentiment analysis.

Inspired by a paper which introduce a BERT-Based Pipeline for Italian Twitter Sentiment Analysis [66], these features were replaced with specific tags during pre-processing to minimize the number of unique tokens in the text, thereby eliminating redundancy and variability.

Abbreviations underwent a similar process, whereby a list of prevalent Italian abbreviations was compiled (table 3.17) and subsequently utilized to replace them within the sentences.

3.4 How to deal with unbalanced Dataset

As is clearly visible in figure 3.5, the number of samples in the Neutral/Positive instance is much higher with respect to the other two classes.
Datasets with such characteristics are commonly referred to as **unbalanced datasets**.
Training machine learning models with this kind of set can present various challenges and issues that need to be addressed.
Indeed, a model may not be able to generalize well to samples of the minority class due to the disparity in the number of training examples it has received compared to the majority category.
Thus, the model's prediction performance may not be adequate, especially in a case where the minority class is the one of interest.
Moreover, when dealing with an unbalanced dataset, the evaluation metrics have to be selected carefully.
For instance, relying only on accuracy can be misleading as it may yield high scores even if the model performs poorly on the minority class.
In this section, methods for handling unbalanced datasets will be discussed. Some of them are well-known, while others are alternative approaches that have been tested.

3.4.1 Over-sampling and Under-sampling Techniques

Over-sampling and under-sampling are two simple and fast resampling techniques used to reduce dataset unbalance.
The first one adds more copies to the minority class and it is used when data are not sufficient.
When enough data are available, the opposite approach known as under-sampling can be effective. This strategy involves reducing the size of the majority class.
Specifically, the *random* over-sampling or under-sampling, duplicate or delete some random samples from respective class. [15]
Both approaches can be completed using the simple Python library *imblearn*. [51]

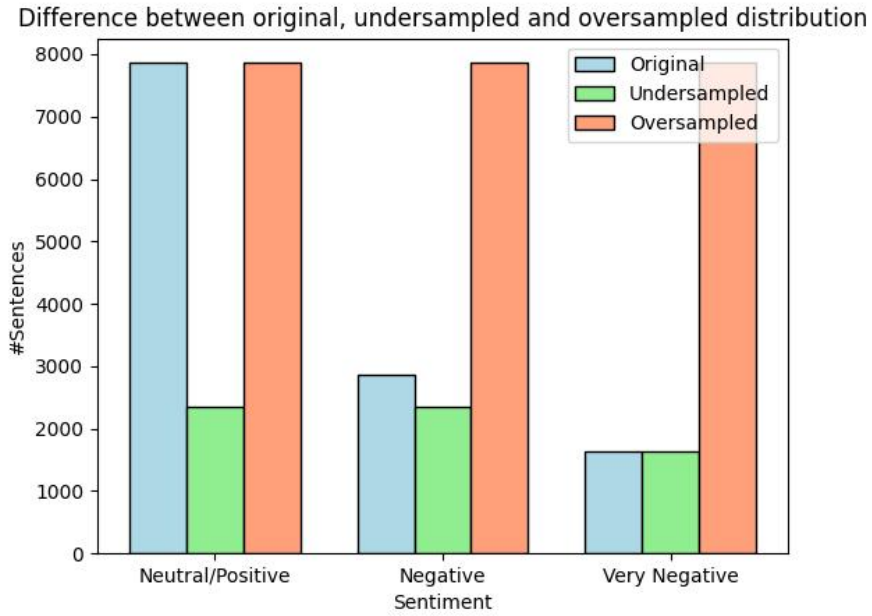


Figure 3.9: Difference between distributions

In figure 3.9 is possible to see the difference between the original dataset distribution (blue), the under-sampled distribution (green) and the over-sampled distribution(orange).
Actually, in the under-sampling case, the minority class is *very negative* with a total number of 1.638 elements.
Although the classic approach selects the very negative classification as a minority class, doing so would result in the useless loss of data for the negative category.
Indeed, in comparison to the very negative class, the negative classification does not contain so many more sentences to unbalance the dataset.
For this reason, it was decided to use the negative categorization of the minority class.
This reduces the neutral/positive instances to the same value as the negative instances, i.e. 2.863 elements.
In the opposite case, the over-sampling technique results in an equal number of instances for each categorization, matching that of the majority class, i.e. the positive class, with 7.861 elements.

3.4.2 Dataset Augmentation

The idea behind this approach stems from the request to make the dataset more generic, rather than focusing solely on a particular customer type.
While there is a wealth of data available online, they are often raw and lacking in categorization.
Therefore, an approach that aims to generalize the dataset could lead to better results, both in terms of the overall model and dataset balance.
Motivated by these goals, a research was conducted for a dataset that could provide phrases similar to those found in a chat conversation, particularly those that express

strong negative sentiments.
By incorporating a different dataset, this approach has the potential to yield benefits in terms of the generalization of the model and balancing of the dataset.
The most interesting dataset found was not directly used for sentiment classification, but rather for a different type of task which however is still related.
Specifically, the **Italian YouTube Hate Speech Corpus** [20] is a dataset consisting of manually annotated comments extracted from YouTube videos between January 2020 and May 2020.
This dataset has the potential to be valuable for the sentiment classification task due to the presence of strongly negative comments, despite being originally intended for a different purpose.
The dataset was annotated by type of hate speech and by the target audience to which it refers. In fact, two fields of the dataset are called *type* and *target*.
Type can assume 4 distinct values:

- **Appropriate**;
- **Inappropriate** (5.426 elements), the comment contains terms that are obscene, vulgar; but the text is not directed at any person specifically;
- **Offensive** (3.059 elements), the comment includes offensive generalisation, contempt, dehumanisation, and indirect offensive remarks;
- **Violent** (626 elements), the comment's author threatens, indulges, desires or calls for (physical) violence against a target,it also includes calling for, denying or glorifying war crimes and crimes against humanity.

On the other hand, the field *target* can take on up to 14 values such as "*racism*", "*individual*", "*politics*" or a specific category called "*other*".
To start, sentences classified as *appropriate* in the youtube corpus were excluded, as the sentiment of such sentences could not be determined with absolute certainty.
A naive approach would incorporate all the entries of the inappropriate, offensive, and violent types into the very negative categorization.
While doing so would lead to a better-balanced dataset, however, it would also lose the integrity of the starting dataset.
Nevertheless, in order to optimize the utility of the YouTube HateSpeech Corpus while preserving the integrity of the source data, it was decided to restrict the number of elements to be incorporated.

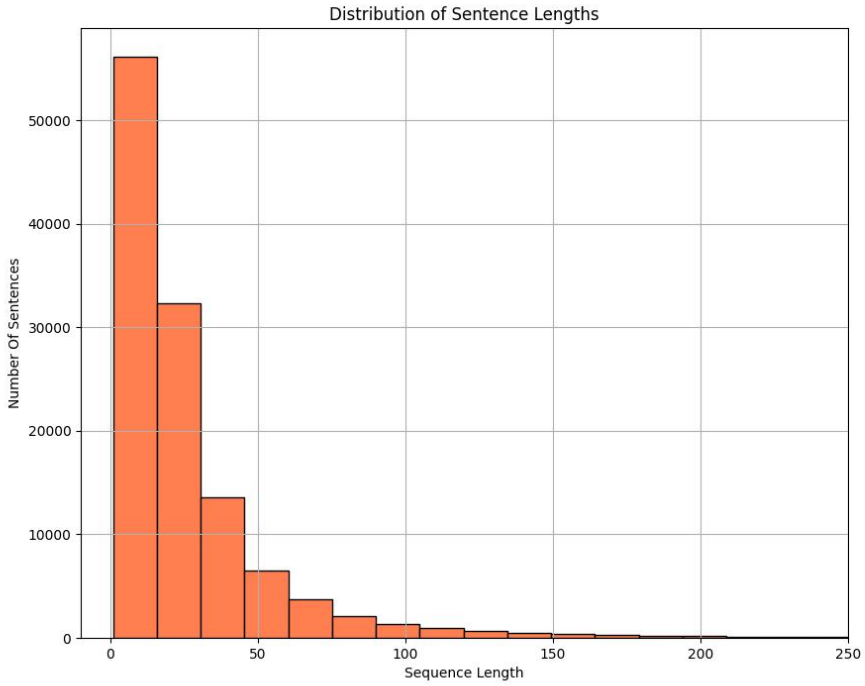


Figure 3.10: Sentence Length distribution Youtube HateSpeech Dataset

As is possible to see in figure 3.10, the distribution of the sentence length is higher with respect to the original dataset distribution visible in figure 3.7.
For this reason, a decision was made to impose a limit on the sentence length to be included, specifically 64 words.
In addition, the YouTube corpus was also filtered for the *target* column, indeed, in order to prevent contamination of the original dataset with expressions containing targeted insults, the decision was made to exclusively exploit the "*other*" category.
Considering the aim of creating a more generic dataset, a more meticulous analysis of the language was conducted, leading to the adoption of a secondary filtering approach for the sentences.

The second approach used for filtering the Youtube Corpus is visible in 3.11.
From the original dataset, the most used words from the very negative classification were extracted. Afterwards, these words were used to filter the hate speech dataset.
To increase the generality of the model, the youtube comments that did not contain words already present within the original dataset, were extrapolated.
The main principle behind this method is that when the model is required to discern highly negative expressions, a greater diversity of typology will enhance its ability to do so effectively.
However, due to the issue of category balancing, some additional sentences that incorporate the most frequently utilized words were selected.

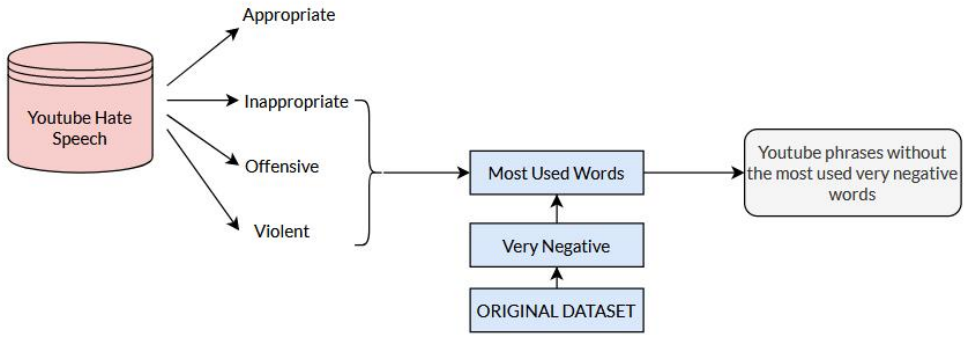


Figure 3.11: Filtering approach by most used words

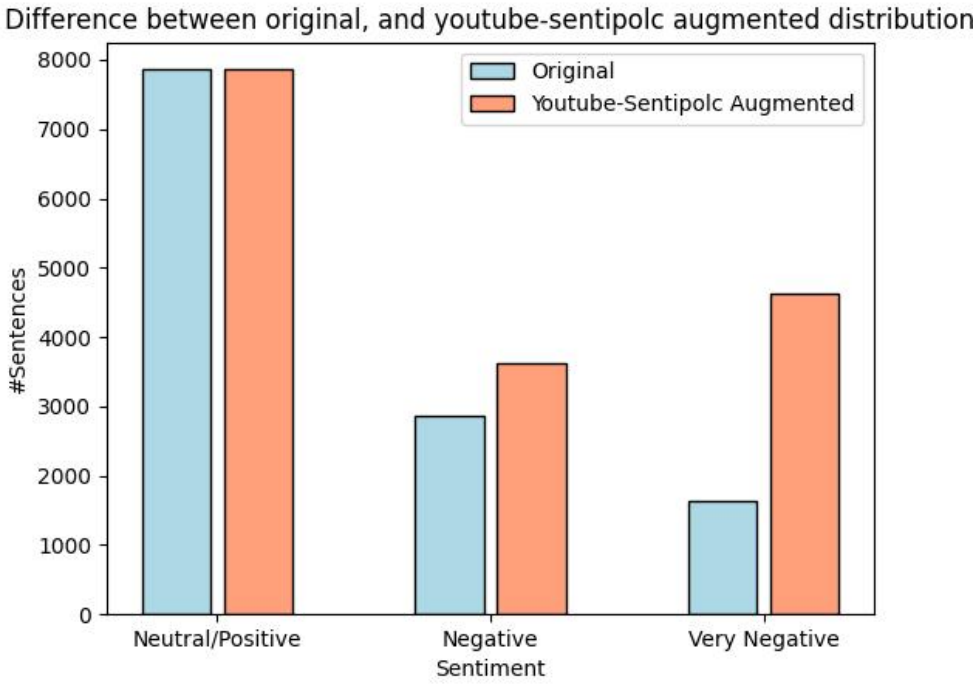


Figure 3.12: Difference distribution Original (*CustomerData_1*) and Youtube-SentiPolc Augmented (*CustomerData_1-YT-SP*) Datasets

In figure 3.12 is visible the distributions difference between the original and augmented dataset.
As also shown in table 3.18, the number of the very negative elements has increased to reach 4.638.
A second dataset used with the same approach as YoutubeHateSpeech was **SENTIPOLC** which is a SENTIment POLarity Classification organised within Evalita 2016, the fourth evaluation campaign of Natural Language Processing and Speech tools

for Italian. [77]
The set consists of Italian tweets which actually is a typology of data that can be similar to the one available in a chat.
The available texts may belong to three different topics: "*political*", "*socio-politicals*" and "*others*" and the sentiment can be either positive or negative.
As in the case of youtube, it was selectively filtered the negative elements pertaining to "*other*" topics, purposefully omitting any content related to political or socio-political issues.
By doing this, a total number of 765 elements were added to the dataset in the negative categorization and the results is visible in figure 3.12 with a more precise definition in table 3.18.
One peculiarity that has to be noted, is the Hashtag elements which are available in Twitter posts. They were preprocessed and removed from the sentence.

Sentiment	**Number of Sentences**
Very Negative	4.638
Negative	3.628
Neutral/Positive	7.861
Total	**12.362**

Table 3.18: Number of sentences for each sentiment in the youtube augmented dataset (*CustomerData_1-YT-SP*)

3.5 Dataset Review

As already mentioned in chapter 1, some reviews of the dataset were performed due to the label's inconsistency.
After the conclusion of the data analysis, the initial phase model evaluation commenced. However, new data was sent to a colleague for sentiment categorization based on the three defined labels, aiming to increase the number of negative and very negative values.
After receiving the new data, they were used to increase the training set and in table 3.19, is visible the new final dataset (*CustomerData_2*) distribution given by the sum of the original data (*CustomerData_1*) with the new dataset.

Sentiment	**Number of Sentences**
Very Negative	13.137
Negative	3253
Neutral/Positive	1951
Total	**18.341**

Table 3.19: *CustomerData_2* distribution

Following that, during the model evaluation phase that will be described better in the next chapter, it was realized that certain sentences deemed positive were categorized as "negative or "very negative", and vice versa.
Moreover, sentences with identical meanings were categorized differently.
In virtue of this, a first "automatic" review was performed to mitigate this problem.
After the first revision, in an advanced stage of the model evaluation, a lexicon approach was used to try to improve the model's performance.
However, even in this case, it was noted again the inconsistency of the sentiment labels in the dataset.
To address completely the problem, a manual review of the entire dataset was performed.

3.5.1 Automatic Review with Feel-IT

Both the original dataset and the new data were analyzed. A word-targeted investigation was performed before diving into a semi-automatic data review using a model.
Indeed, by seeking out particular words or phrases that conveyed a potentially negative sentiment, it was noted that some sentences with the same meaning were categorized in different ways.
This leads to a possible model, not fully understand the categorisation with the consequence of performing a wrong prediction.

As is possible to see in tables 3.20, 3.21, 3.22, 3.23 for sentences which contain specific terms, the categorization is inconsistent.

Messages	**Sentiment**
sempre quello dite *vergognatevi*	Negative
questa cosa è *vergognosa* che voi non accontentate più i clienti	Negative
vergonatevi se vi guardate allo specchio che cazzo vedete vergona *vergogna*	Very Negative
vergogna	Very Negative
lo.sapete che siete passibili di denuncia e tramite querela posso denunciarli risalendo al turno e a.chi risponde dietro la tastiera? é *vergognoso*	Very Negative

Table 3.20: Sentences with inconsistent label (1)

Messages	**Sentiment**
c è qualcuno?	Negative
c'è qualcuno?	Neutral/Positive
vorrei parlare con qualcuno	Neutral/Positive
vorrei parlare con qualcuno	Negative
c'è qualcuno che risponde?	Negative

Table 3.21: Sentences with inconsistent label (2)

Messages	**Sentiment**
se volete mi fate denuncia non ho paura di finanzza di nesuno	Neutral/Positive
mi fate denuncia non mi interesse pju	Negative
siete da denuncia	Very Negative
vado a denunciarvi faccio prima e poi chiudo il conto come faranno la maggior parte dei giocatori	Negative
domartina inviero fax di denuncia a il mattino e a striscia la notizia	Very Negative

Table 3.22: Sentences with inconsistent label (3)

Messages	Sentiment
chiusura conto	Neutral/Positive
chiusura conto	Neutral/Positive
chiusura conto gioco	Negative
chiusura conto	Negative

Table 3.23: Sentences with inconsistent label (4)

Thus, sentences with the same meaning were categorized differently. For this reason, the labels of some sentences have been modified so as not to create discrepancies between phrases with the same possible meaning.
Similarly, to ensure uniformity with the original dataset, certain elements in the new dataset underwent modifications to adjust their associated sentiment values.
Moreover, to reduce also the unbalance of the dataset, the sentences which contains only one single word were analyzed.
In figure 3.13 are shown the 15 sentences composed only by one single word in the Neutra/Positive classification.
The term "*operatore*" appears more than 150 times throughout the entire dataset. Since single words potentially do not lead to substantial contributions in terms of performance, it was decided to reduce the number of each element to around 10 to 20.

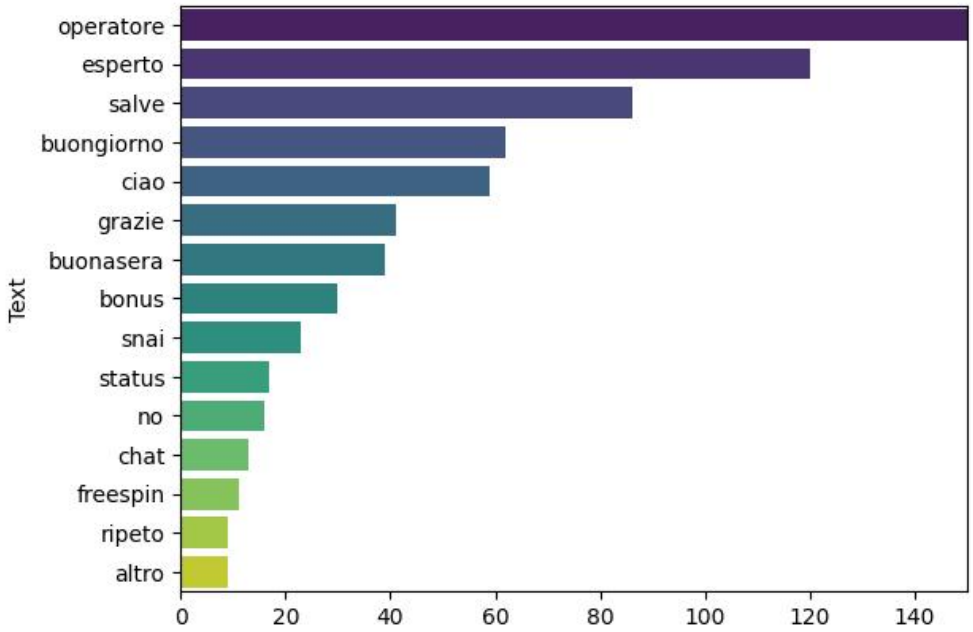


Figure 3.13: Most used single-word positive sentences

The automatic review was performed thanks to another sentiment analysis model called Feel-IT, which is a fine-tuned UmBERTo model on an Italian Twitter-based dataset

for a sentiment analysis task. It was able to obtain state-of-the-art performances on different benchmark corpora. [31]

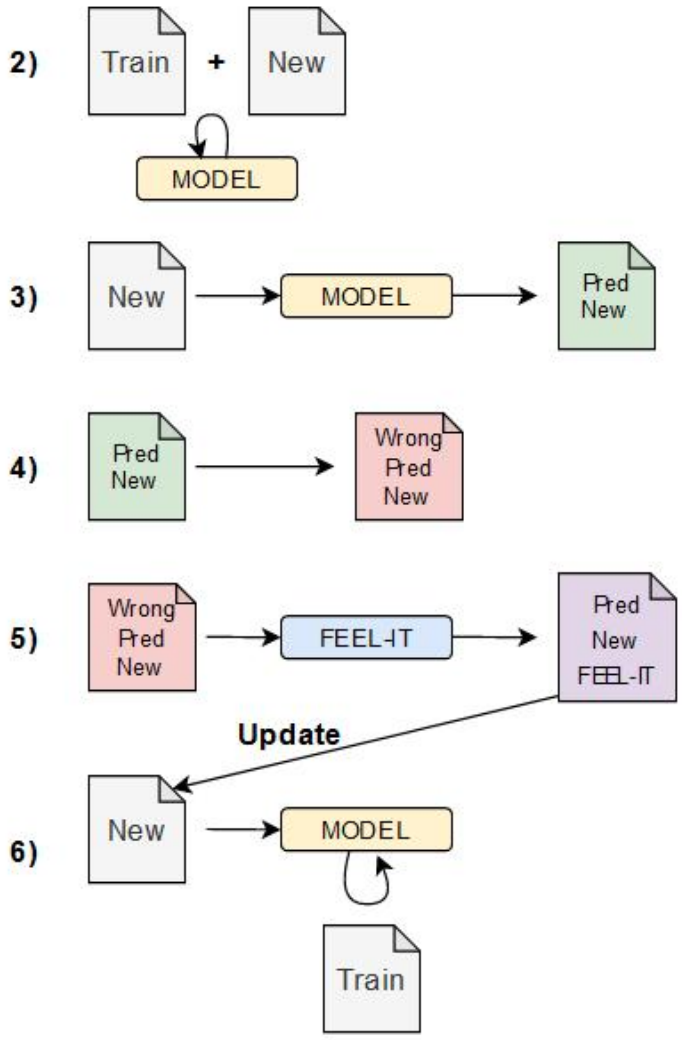


Figure 3.14: Semi-automatic review of the new data

In figure 3.14 is visible the workflow used for this approach which is also based on another model, in particular, a BERT architecture that was tested during the model evaluation stage.
The BERT model was trained with both the original data and the new data (2) and subsequently was used for predicting again the new data (3).
One particularity that must be mentioned and which was discussed also in chapter 2 is that usually, in a machine learning pipeline, the test set and training set are completely separated.
In this case, a deliberate decision was made to do the opposite in order to not com-

pletely waste the colleague's work.
In point number (4), sentences that exhibited dissimilar sentiment values when comparing the model predictions and the manually assigned categories were gathered and considered as "wrong".
The Feel-IT model is what comes into play at this juncture. The "wrong" predictions, were categorized using the Feel-IT model which could categorise a sentence as positive or negative.
Finally, these sentences were updated with the obtained prediction from Feel-IT (5-6).

This approach was designed in order to give a certain amount of decision to the artificial intelligence model which was trained also with the new data and thus context was amplified.
At the same time, the work done by the operator has not been completely replaced since that was a comparison between the Feel-IT predictions and the operator categorization.

3.5.2 Manual Review

Following a lexicon-based approach used for improving the model performances that will be explained in the next chapter, it was noted that some data were still inaccurately categorized.
Therefore, a manual review of all data was performed, aiming to balance the dataset according to the predefined categories.
This revision process was essential to ensure the reliability of the dataset and thus the validity of the results. In this way, errors were corrected and inconsistencies mitigated.

During the examination of the dataset, some rules were defined.
Indeed, there were sentences with the same semantic meaning but with different sentiment categorizations.

Semantic Meaning	**Sentiment**
Sentence with inappropriate words	Very Negative
Wants an account closure	Negative
Ask why there is no one to answer	Negative
Asking for a bonus request	Neutral/Positive
Asking if someone is present in the chat	Neutral/Positive

Table 3.24: Rules during manual review

Rules are available in table 3.24 where the semantic meaning of the sentence and the chosen sentiment are shown.

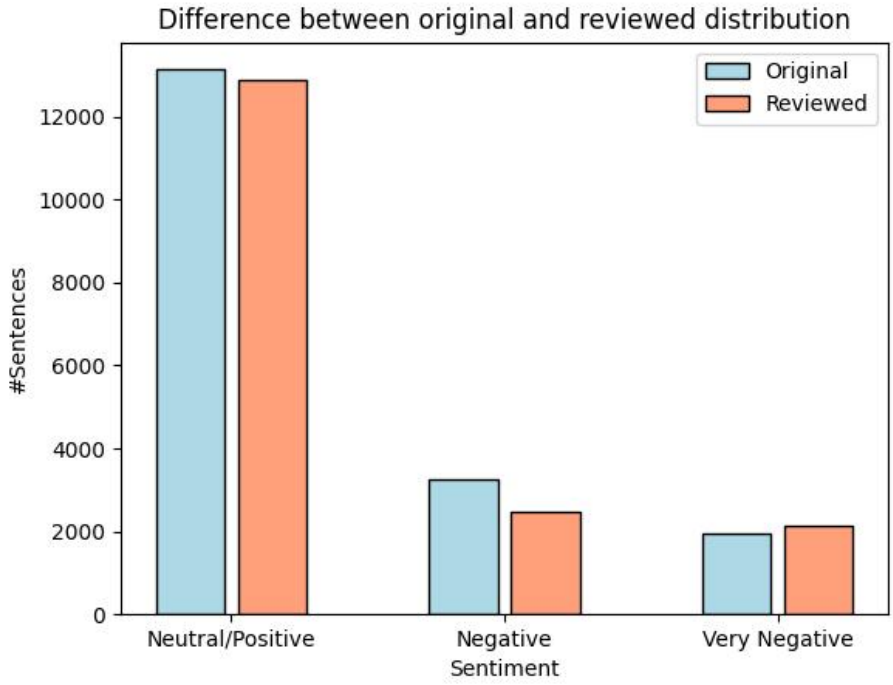


Figure 3.15: Distribution comparison between original and reviewed dataset

Sentiment	**#Sentence *CustomerData_2***	***#Sentence CustomerData_3***
Neutral/Positive	13137	12895
Negative	3253	2480
Very Negative	1951	2348
Total	**18.341**	**17.723**

Table 3.25: Number of sentences for each sentiment value in *CustomerData_2* (Original) and *CustomerData_3* (Reviewed)

In figure 3.15 and table 3.25 is visible the difference between the original distribution and the reviewed version of the dataset.
The new data were concatenated and no technique for unbalanced data were used. Quite substantial is the difference of negative values. However, by doing so the dataset discrepancy is contained.

Moreover, other outliers have been identified and removed, some of them are visible in the table 3.26

Outliers
reference 97.267d7a5c.1651251158.28bad5eb
15 15 40
Df07e5081d14cda6fe03
an error occurred while processing your request.reference 97.67d7a5c.1651252422.38b1dbab

Table 3.26: Example of outliers removed

Overall, the use of this approach in conjunction with the Feel-IT model proved to be effective in improving the accuracy and validity of the dataset, and therefore the overall quality of the analysis as will be visible in the next chapter.
Furthermore, this approach of manual revision and balancing of the dataset was chosen over the use of automated tools, as it allowed for a more nuanced and context-dependent categorization of the data, taking into account the subtleties of language use and meaning.

Chapter 4

Development and validation of the Model

Following the data extraction and analysis stage, a fundamental step in the machine learning pipeline is the training and evaluation of models.
Within the field of artificial intelligence, numerous architectures and algorithms can be used for many purposes.
Therefore, is vital to understand which kind of technology can impact in a better manner the final evaluation.
In this phase, models are trained and evaluated on a labelled dataset and metrics are used to determine which technique performs better.
The purpose of this chapter is to explore models, techniques, and architectures, and evaluate them in terms of metrics suitable for a sentiment analysis task.
Moreover, some important aspects of the task had to be analyzed in order to decide how to continue the model definition.
A common issue in the development and validation of a machine learning model is called overfitting, which occurs when the trained model fails to generalize the new data, leading to poor performance.
Part of this stage is also to apply techniques in order to reduce this problem and increase performance on an unseen dataset.
Thus, this chapter will provide an overview of the tested approaches, from baseline models to fine-tuned BERT architectures. All the choices that have been made will be discussed, in addition to the results obtained in terms of metrics.
In particular, BERT will be discussed in a thorough way since is the model which provides better results. Some different architectural structures for BERT will be analyzed and tested in order to understand which one could give the best performance.
As already mentioned in previous chapters, the dataset has undergone several modifications. The results of the reviews will be discussed in a specific subchapter.
In summary, the development and validation phase is crucial in a machine learning pipeline. It involves meticulous algorithm selection, evaluation metrics and techniques to achieve good results of the model on unseen data.
This phase serves also to determine the best possible configuration that yields optimal results.

4.1 Initial Experiments

In the first phase of the model evaluation, the dataset used was the original one, without any kind of modification.
The total number of elements in the dataset is 12.362 which are subdivided into three categorizations as visible in the table 3.7.
Nevertheless, before actually choosing a possible model and evaluating it, certain considerations must be taken into account.
Even if the main idea is to perform a multi-class categorization, an interesting experiment which could be undertaken is a **binary** classification trial.
Indeed, in this scenario, a possible idea is to consider *very negative* and *negative* classification into a unified category.
This could prove beneficial for two reasons, the first pertaining to dataset balancing, and the second to model performance. Indeed, merging the two categories is feasible to alleviate the problem of class imbalance.
Moreover, from a model's perspective, a binary classification could be easier to generalize with respect to a multi-class problem.
Another alternative to be explored might concern the task itself. Indeed, a standard approach for a classification problem works by using a **softmax** layer with the aim of providing the category's probability.
A possible option is to treat the problem as a regression task. The categorical variable *sentiment* may be translated into a numerical variable and used for training a regression model. In this way, the model returns as output a real number and is possible to classify the value with the closer integer.
As already mentioned in chapter 2, different paths can be followed for the text representation. A Bag-Of-Words approach based on the TF-IDF representation is a commonly used methodology.
However, in tasks such as sentiment analysis, words order and compound words might be crucial to determine the feeling of the sentence.
Words Embeddings instead, allow capturing the relationship between words and understand better the semantic and syntactic words connection. Is possible to develop specific embeddings or used a pre-trained model in order to get them.

For this evaluation stage, the sentiment values were converted using the One Hot Encoding offered by the sklearn library.
This type of encoding represents each sentiment with a vector whose length is the number of possible sentiments.
In every vector, only one element is 1 and the rest are 0s. The position of the 1 represents the corresponding category.
In table 4.1 is shown the one-hot encoding for each sentiment value.

Sentiment	**Number of Sentences**
Very Negative	[1,0,0]
Negative	[0,1,0]
Neutral/Positive	[0,0,1]

Table 4.1: One Hot Encoding

This target representation leverages the use of the *softmax* output function.
Specifically, the multi-classification created model, employs this activation function as the final layer of the neural network, enabling it to generate a probability distribution as output that determines the predicted sentiment.

Model evaluation involves splitting the dataset into distinct sets. Thus, to test these conditions, the dataset was partitioned into two distinct subsets, namely *training set*, and *test set*.
The separation was specifically carried out to allocate 80% of the data for the training set and 20% for the test set.
Moreover, data is split in a stratified fashion, in order to ensure the test set has all the possible categories to test.

Dataset	**Train Set**	**Test Set**	**Total**
CustomerData_1	9.889	2.473	12.362

Table 4.2: Train and Test elements number

In table 4.2 is shown the number of elements for training and test set for each dataset evaluated.
This evaluation process involved the development and training of simple models using Tensorflow and Keras libraries.
These experimental models were specifically designed to assess the various scenarios discussed earlier.

4.1.1 Embeddings Evalutation

The evaluation process started with the examination of different embedding types, including TF-IDF, Word Embeddings created with the help of the Keras library and contextualized Embeddings with BERT.
Performance metrics then will be compared across these different variations.

TF-IDF

The TF-IDF embedding methodology was implemented using the scikit-learn library. Specifically, the *TfidfVectorizer* object permits the creation of vector representation of a sentence based on TF-IDF values and is subject to an initial training phase.
Subsequently, the training procedure was applied to the training data and the trained TfidfVectorizer object was used to generate sentence embeddings.
A simple model with Keras was developed and trained with the obtained embeddings.

Word Embeddings With Keras

In the case of the word embeddings generated using Keras, a tokenizer was trained on the training set to develop the words' vocabulary.
The tokenizer maps each word of the training set into a unique integer value, thus, is possible to generate integer sequences from each sentence, with an additional padding necessary to standardize the length of the sequences.
Once the tokenization and padding were complete, embeddings were generated through a dedicated layer in the neural network known as the *Embedding layer* [29].
This layer works by manipulating the input numerical sequence and converting it into a dense vector representation of a fixed length facilitating the downstream analysis of sentence meaning.

BERT Embeddings

By leveraging the capabilities of the HuggingFace library, tests using BERT were conducted. HugginFace provides access to a wide range of pre-trained BERT models, which can be utilized for text embedding or fine-tuning purposes.
In this case, AlBERTo [5] model, which was trained on a large quantity of Twitter posts, was used to generate the sentence embeddings.
It was chosen because, with respect to the other trained models, the type of data it was trained on, is more similar to the possible data available in a chat environment.
Indeed, the language used on Twitter is comparable to the language that is possible to find on chat platforms.
To prepare the data to be processed by the model, special tokens $[CLS]$ and $[SEP]$ were added to each sentence.
HuggingFace provides two particular classes called *AutoTokenizer* and *AutoModel* which are specifically designed to fetch tokenizers and models from pre-trained configurations.
The tokenizer performs the sentence tokenization, after having automatically added the special tokens. In addition, if phrases are excessively long they are truncated.
As already seen in 2, BERT was originally trained on two tasks, one of which was the Next Sentence Prediction. Thus, it expects input in the form of pairs of sentences, marked with a binary value of either 1 or 0.
However, in this instance, as only a single sentence was utilized, an array of 1 was

generated for each sentence. [10]
Afterwards, tokens and vectors were passed through the loaded model with the "*output_hidden_states*" flag set to True in order to obtain each intermediate state of the model.
Finally, a unified representation for each sentence was produced by computing the mean of the penultimate hidden layers associated with each token.

Embeddings Results

In table 4.3 are shown the metric values with the different embedding techniques.

Embeddings	Accuracy	Precision	Recall	F1-Score
TF-DF	0.83	0.82	0.80	0.81
Keras Embeddings	0.81	0.80	0.78	0.79
BERT	**0.85**	**0.83**	**0.82**	**0.82**

Table 4.3: Metrics value with different embeddings

As expected, BERT embeddings are able to contextualize better the sentence and consequently obtain better performance with respect Keras and TF-IDF embeddings. Nevertheless, it is important to note that the alternative approaches to BERT also yield highly promising results, achieving accuracy levels of around 80%.
The embeddings generated by Keras do not reach TF-IDF values, which may be attributed to the requirement for high-quality and high-quantity data to get optimal embeddings.

4.1.2 Regression Evaluation

The evaluation process also involved exploring alternative approaches to the task typology. Through specific operations, is possible to treat a classification problem as a regression problem.
To perform that, labels were mapped to integer values: 0 for *very negative*, 1 for *negative*, and 2 for *neutral/positive*.
Each model utilized approximately the same number of trainable parameters. Moreover, hyperparameter values, such as learning rate and epochs, remained consistent.
Furthermore, to derive the outcomes, standard loss and metric functions employed in regression tasks were utilized, specifically the *Mean Squared Error (MSE)* function, which is commonly used in regression scenarios.
Finally, the various embeddings representations were tested to determine their respective performance under these alternative approaches.

Regression Results

In table 4.4, metric values for the regression task are shown, divided by the embedding typology.

Embeddings	Accuracy	Precision	Recall	F1-Score
TF-DF	0.82	**0.83**	0.80	**0.80**
Keras	0.79	0.78	0.75	0.76
BERT	**0.83**	0.79	**0.82**	**0.80**

Table 4.4: Metrics value with different embeddings - Regression

Even in this case, the embeddings define with BERT lead to better performance in the majority of the metrics.
However, the precision value for the TF-IDF approach achieved a remarkable 0.83, compared to 0.78 and 0.79 respectively of Keras and BERT.
Also in this case the embeddings defined with BERT lead to better performances. However, the precision in the TF-IDF case, reaches 0.83 against the 0.78 and 0.79 respectively of Keras and BERT.

4.1.3 Binary Evaluation

An additional experiment was performed based on the number of possible classifications. Indeed, in this specific instance of the problem, from multi-classification is possible to derive a binary classification task.
In this case, *very negative* and *negative* categories were merged into a unique category, thus leading to a new dataset version with only 2 categories.
Each model utilized approximately the same number of trainable parameters. Moreover, hyperparameter values, such as learning rate and epochs, remained consistent.
Furthermore, as in the case of the regression task, in order to derive outcomes, alternative loss and metric functions were used, namely respectively *Binary_ CrossEntropy* and *sigmoid* which are typically chosen for binary problems.
Finally, the various embeddings representations were tested to determine their respective performance under these alternative approaches.

Binary Results

In table 4.5, metric values for the binary problem are shown, divided by the embedding typology.

Embeddings	Accuracy	Precision	Recall	F1-Score
TF-DF	0.85	0.84	0.84	0.84
Keras	0.84	0.83	0.82	0.83
BERT	**0.88**	**0.87**	**0.86**	**0.87**

Table 4.5: Metrics value with different embeddings - Binary

While BERT remains the best-performing embedding approach, reaching exceptional

accuracy levels of up to 88%, an overall increase in performance metrics was observed across each of the tested embeddings.
Indeed, all metric scores surpassed the threshold of 80 percent, indicating consistently high performance across the board.
This can be attributed to different factors, such as an improvement in the balance of the dataset, and fewer mistakes made by the model as it was trained using just two categories.

4.1.4 Initial Experiments Conclusion

As observed in the various scenarios, BERT often tends to yield the best results, thus, further investigation must be performed.
Indeed, the models developed in previous studies were relatively simple and they were employing diverse embedding techniques and approaches to the problem.
BERT also offers the possibility of fine-tuning a pre-trained model for a specific task, making it crucial to assess its applicability in this domain.
The values obtained by evaluating the dataset in the binary attempt exhibit promising results.
However, the main project's objective is to better characterize the user's emotions. The limitation of having only two sentiments in the dataset may be limiting the impact of the result from this point of view.
Moreover, the use of regression, on the other hand, did not provide excellent results like those of multi-classification approaches.

4.2 Baseline Models

In machine learning pipelines, baseline models are simple models that serve as a reference for evaluating the performance of more powerful techniques.
The main idea is to start from a simple, well-known and easy-to-implement algorithm in order to set a minimum level of performance which can be undoubtedly achieved on a given task.
Baseline models will serve as benchmarks for future more advanced models because the evaluation is completely dependent on the use case that is analyzed.
Moreover, one more key benefit is that allows us to understand better the data. It is possible to check if the data are adequate for the task. [8]
This section will cover the baseline models analyzed for this task of sentiment classification.
Precisely, the spotlight will be on Naive Bayes, Support Vector Machine, and Decision Trees which are three algorithms that have established themselves as fundamental baselines for text and sentiment classification. [9]
TF-IDF embeddings were utilized as input for baseline models due to their simplicity and comparable performance to BERT, while the creation of BERT embeddings proved to be more challenging than other methods.
Models were trained for a multi-classification task, thus no binary or regression evaluations were performed.
The dataset used to train the baseline models consists of the concatenation of the original data and the new data categorized, thus the dataset *CustomerData_2*
In table 4.6 is possible to see the distribution of this dataset.

Sentiment	**Count**
Neutral/Positive	13.137
Negative	3.253
Very Negative	1.951
Total	**18.341**

Table 4.6: Distribution of *CustomerData_2* for baseline models

The dataset was partitioned into a training set and a test set, constituting 80% and 20% of the total dataset, respectively, for both the baseline models and starspace.
Moreover, in the Fine-Tuning BERT approach, an additional validation set was used, derived from the training set and comprising 10% of its size.

4.2.1 Naive Bayes

A simple probabilistic approach often used in Natural Language Processing for text classification is Naive Bayes.
It is a probabilistic model based on *Bayes Theorem* which is used to define a probability of a hypothesis by exploiting new evidence using conditional probabilities.
Is called "Naive" because it assumes that features (in this case words) are conditionally independent of the category labels.
This assumption makes this approach efficient and fast and can perform well even with a small dataset. [44]

Accuracy	Precision	Recall	F1-Score
0.81	0.83	0.60	0.65

Table 4.7: Metrics value for Naive Bayes

In table 4.7 is possible to see the metrics obtained with the Naive Bayes approach.

4.2.2 Support Vector Machine

Another popular supervised algorithm is called Support Vector Machine (SVM) which can be usef both for classification and regression tasks.
The idea behind the SVM is to find the best hyperplane in an N-dimensional space which classifies data into separate categories.
Specifically, finding the hyperplane with the maximum margin, which is the distance between data points and the hyperplane, brings better results.
An illustrative figure is available in 4.1
The support vector machine's peculiarity is that can be trained using different kernels, such as linear, polynomial or radial basis function (RBF) kernels which can capture the relationship between input and output. [32]

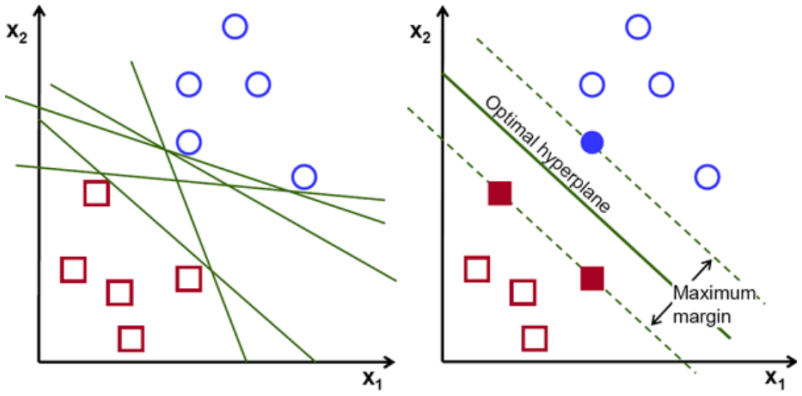


Figure 4.1: Support Vector Machine [32]

In table 4.8 is possible to see the metrics obtained with the Support Vector Machine

Accuracy	Precision	Recall	F1-Score
0.80	0.84	0.57	0.62

Table 4.8: Metrics value for Support Vector Machine

4.2.3 Decision Tree

Decision Tree is a popular machine learning algorithm easily to understand, interpret and implement.
It works by defining a tree structure from a starting root node, it constructs internal nodes which define decisions based on a feature value, and leaf nodes which represent a class label.
Building a decision tree works by splitting internal nodes by selecting the best attributes. In order to determine the best splitting, a measure of information gain or some other criteria must be computed. Some commonly used examples are *Entropy* or *Gini impuriti*.
Decision Trees are easy to interpret and are widely used in machine learning for tasks where interpretability and visualizability are fundamental.
This is because it is straightforward to understand the reasoning behind a specific classification.
However, when data are noisy and tree is not that wide, they can be prone to overfitting which can be avoided by applying pruning techniques. [75]

Accuracy	Precision	Recall	F1-Score
0.80	0.74	0.72	0.73

Table 4.9: Metrics value for Decision Tree

In table 4.9 is possible to see the metrics obtained with this approach.

4.2.4 Final Baseline considerations

Figure 4.2 is shown the comparison between metrics of the baseline models.
As regards the training time, it took few seconds to train Naive Bayes and Support Vector Machine and few minutes for Decision Tree.
As is clearly visible, models perform on average in the same way with a generic drop in the recall values.
On the whole, the precision and the accuracy reached by models is *0,80*, followed by F1-Score with *0,67*, and recall with values equal to *0,63*.
When evaluating the models as a whole, the decision tree model shows a relatively constant trend, with no significant changes observed in recall and f1-score, in contrast to Naive Bayes and SVM, which exhibit a significant drop in performance.

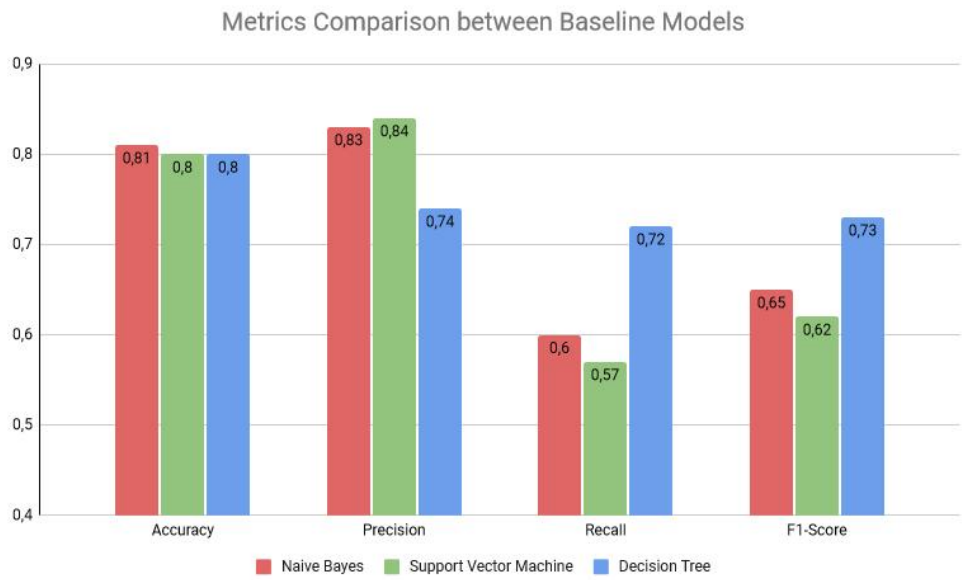


Figure 4.2: Metrics Comparison between Baseline models

In this specific use case, the final goal is to best identify complaints and negative comments. Thus, is important to have a higher recall as these metrics allow to measure the ability of the classifier to identify correctly every instance of a particular category. Thereby, improving recall performance is essential in order to achieve better results. Upon examining the prior assessments, with a particular focus on the embeddings test 4.3, it becomes apparent that using a basic neural network and despite an imbalanced dataset, BERT embeddings were able to attain, and at times surpass, the values achieved by the baseline metrics.
To sum up, the baseline metrics establish a threshold that can be surpassed by leveraging the contextual understanding offered by BERT, and the computational power of neural networks, ultimately producing better results.

4.3 Starspace

Pat Srl has been utilizing robust and efficient models, already established in the field, enabling them to accomplish diverse NLP-specific tasks in an accurate and faster way. Starspace [94], indeed, is a general-purpose neural embedding model developed by Facebook AI Research (nowadays called Meta AI) which can be exploited to solve a wide variety of problems such as text classification, ranking entities or recommendations.
The main idea behind this model is to learn embeddings of different entity types into the same N-dimensional space. By doing so, entities will be comparable with one another.
In order to learn embeddings, is crucial to understand how to compare entities.
The Starspace approach uses an objective function based on entity pairs which assign higher scores to correct input-output pairs than to incorrect ones.
This model has proven effective for several tasks as it allows learning robust and general

embedding by leveraging share information across tasks.
Thus, Starspace embeddings provide a straightforward, efficient and strong baseline for any of the mentioned tasks.
Thanks to a simple interface provided by a pre-written library available, a Starspace model was trained and used for sentiment classification.

Accuracy	**Precision**	**Recall**	**F1-Score**
0.86	0.82	0.77	0.79

Table 4.10: Metrics value for Starspace

In table 4.10 is possible to see the metrics obtained by the Starspace model.
In comparison to the previously discussed baselines, values are higher.
This is another evidence that an embedding model can lead to better performance for this use case. However, some drawbacks surfaced during the utilization of this model.
Initially, it is worth noting that the used library is compatible only with a specific version of Python.
Is possible to overcome this problem simply through the utilization of docker contains, which facilitate the definition of a suitable environment for application.
Nevertheless, compatibility issues that may arise when integrating different components used in diverse libraries could be difficult, thereby impacting the overall performance and reliability of the system.

Another disadvantage that must be taken in account regards the prediction elements. In fact, considering the complete dataset, in 3669 test elements, 49 sentences were not categorized from Starspace and they had a *None* value.

È verfognoso
asspetto
bhu
Pederas
LADFRI
novita?
operatrore

Table 4.11: Sentence not categorized

In table 4.11 are shown some examples of these not categorized sentences.
This behaviour is probably attributed to the fact that Starspace is not fully capable to understand misspelled words.
Indeed, most of the words are grammatically incorrect and lead to a general confusion from the model.

To overcome this situation, a Naive Bayes model was trained in parallel with Starspace with aim of predicting the phrases which Starspace is not able to classify.
To sum up, Starspace is an effective and fast model which can lead to great results. The training runtime takes only few minutes even with a higher number of epochs. The limitation, even if they are easily manageable, must be taken into account. However, it will be shown that BERT is able to exceed these limitations and provide better results.

4.4 Fine-Tuning BERT

Embedding definition is a possible approach to the use of BERT, however, a possible alternative use the pre-trained model to perform a fine-tuning.
In this section, two possible fine-tuning procedures will be discussed and analyzed. Both implementations exploit a pre-trained BERT model. The first one uses the HuggingFace library which provides an interface for training and testing a fine-tuned model for sequence classification.
The second approach instead, renamed as *BERT as a layer*, exploiting both HugginFace and keras libraries.
Indeed, a pre-trained BERT model can be loaded with HuggingFace, while it can be used in a Keras neural network and perform the fine-tuning.

For each methodology, the pre-trained BERT model utilized for the analysis was *GilBERTo* [35], because it has been shown in the literature [83] that outperform other pre-trained models in similar tasks.
As written in [46], in order to avoid the *catastrophic forgetting*, the learning rate was set to *2e-5*.
Other hyperparameters such as batch size and number of epochs were set to respectively to 32 and 5.
Moreover, deep-learning neural networks such as BERT, may suffer of the *overfitting* phenomenon, where the model is not able to generalize well on unseen data.
For this reason, a validation set, created with 10% of the training set, was introduced in order to analyze the problem.

Total	**Train Set**	**Test Set**	**Validation Set**
18.341	13.204	3.669	1.468

Table 4.12: Train, Test and Validation elements number

In table 4.12 is visible the total number of sentences within each set.

4.4.1 HuggingFace Interface

The HuggingFace library provides classes which are optimized for training transformer models. Thus, it makes easier the training procedure by avoiding writing a own training loop.
Before actually dive into the evaluation part, let's understand better how the classifier

is defined.
HuggingFace provide a class called *AutoModelForSequenceClassification* which instantiate a pre-trained BERT model, based on a configuration (usually the model name). Moreover, as is visible in figure 4.3, an additional layer called *head* is attached over the [CLS] output token.

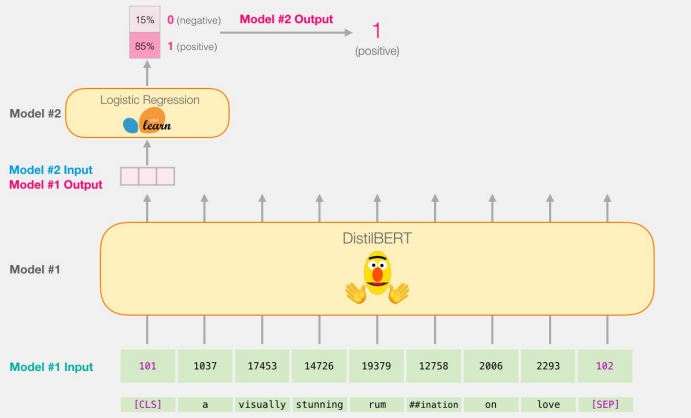


Figure 4.3: Example of additional layer to BERT [4]

Note that in figure 4.3 a Logistic Regression from scikit-learn library is used as the head for the BERT model, while *AutoModelForSequenceClassification* add a simple *linear classifier* that predict one of the possible categorizations.
The BERT tokenizer is used to convert input sentences into an indexed list, serving as input for the model.
The training process performed by the HuggingFace *Trainer* class, enables training of both the model and classifier.
Finally, the classifier outputs a single category, obtained by taking the [CLS] BERT token which represents the entire input sequence.

Accuracy	Precision	Recall	F1-Score
0.870	0.816	0.824	0.819

Table 4.13: Metrics value for Fine-Tuned BERT with HugginFace

In table 4.13 metrics value for the fine-tuned BERT are shown.
The accuracy value of 0.870 is the best found so far. Moreover, precision, recall and F1-score obtain results generally higher with respect to the other baseline analyzed.
This is achieved thanks to the BERT property of contextualization and relationship between words.
Another salient point to note regards the time required for training the model.
The baseline models and Starspace trained respectively in a few seconds and minutes, whereas fine-tuning BERT with the complete dataset, took more than 30 minutes using the available GPU in Google Colaboratory [38] platform.

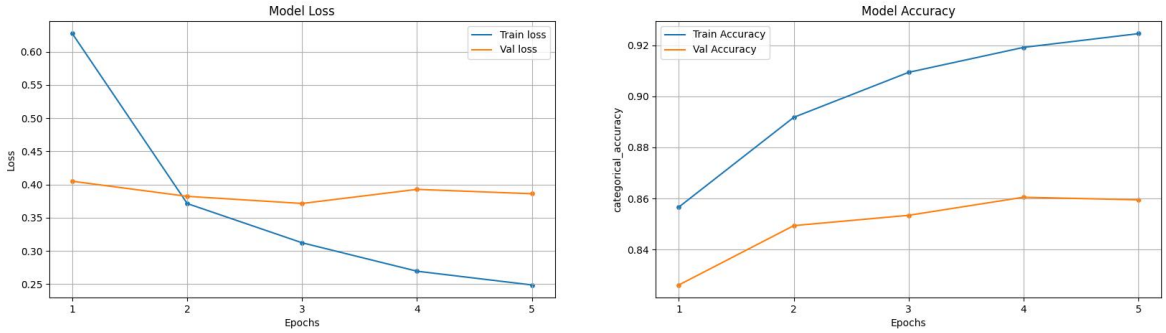


Figure 4.4: HuggingFace Interface Loss and Accuracy during training

The training and validation loss during the training phase can be observed in the left plot of Figure 4.4, while the right plot showcases the training and validation accuracy of the model.
The graphs suggest a slight inclination of the model towards overfitting, with better performances observed in the training set compared to the validation set, indicating a possible manifestation of this phenomenon.
Thus, some regularization techniques must be adopted in order to mitigate this problem.

4.4.2 BERT as a Layer

An alternative approach to fine-tuning BERT, may be carried out by exploiting the HuggingFace portability to the Tensorflow framework.
Indeed, with a class called *TFAutoModel*, a pre-trained BERT model can be loaded and utilized.
The general approach can be seen in figure 4.5

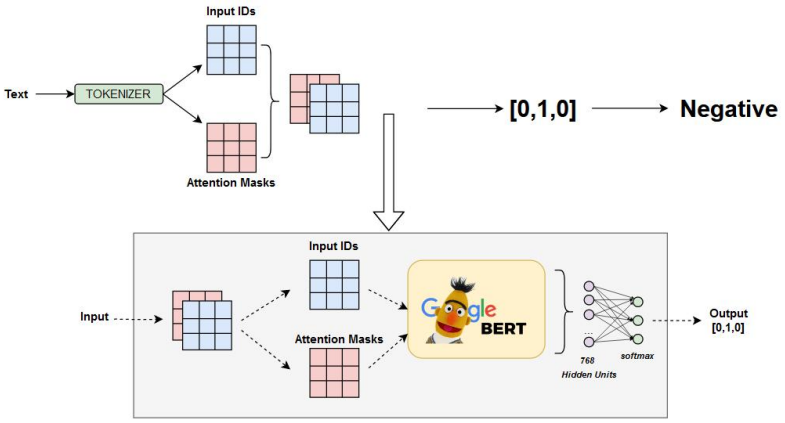


Figure 4.5: BERT as a layer approach

To begin with, as previously seen, the tokenizer is employed to extract input ids and attention masks from a list of sentences.
The grey block denotes the complete neural network, which, for simplicity, receives

only one input: the concatenation of the input ids and attention masks.
Subsequently, the merged object is partitioned to derive the input ids and attention masks as two distinct elements, which serve as inputs for the BERT model, that essentially become a neural network layer.
Finally, the last layer of the network employs a softmax activation function, which yields the predicted final outcome.
For this reason, the sentiment values are encoded with the one-hot encoding.

Accuracy	Precision	Recall	F1-Score
0.865	0.805	0.818	0.811

Table 4.14: Metrics value for BERT as layer

In table 4.14 are shown the value of the metrics obtained by using this approach. The results are comparable to those obtained with the previous approach even if they are slightly lower.
As regards the training time, like the previous approach, also BERT as a layer took more than 30 minutes to finish.

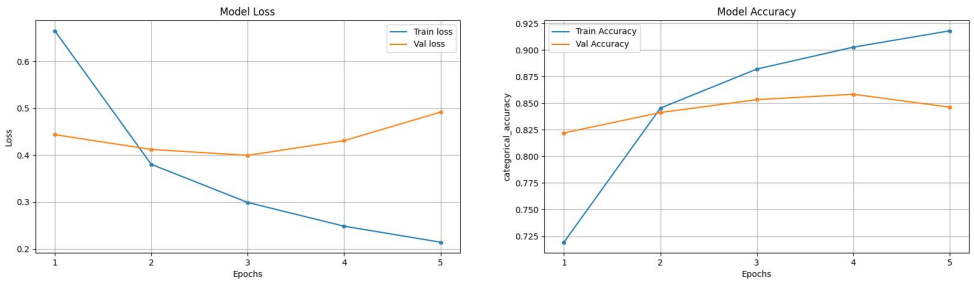


Figure 4.6: BERT as a layer Loss and Accuracy during training

The training and validation loss during the training phase can be observed in the left plot of Figure 4.6, while the right plot showcases the training and validation accuracy of the model.
Also, in this case, graphs tend to show a tendency for the model to overfit.
Therefore, it will also be necessary to apply regularization techniques to mitigate the problem.

4.5 Experiments with Pre-Processing

So far, tests were performed without applying any kind of text preprocessing.
In this section, the preprocessing methodology discussed in chapter 3 will be analyzed and tested, aiming to illustrate how variations in preprocessing can impact the metric values.
In particular, for each sentence, the emoji have been removed and the lowercase has been applied. In addition, the text normalization process has been carried out and stopwords were not removed.

Specifically, two models were employed: Starspace and BERT as a layer.
The selection of the former was based on its proven effectiveness, as it has been previously utilized by the company and acknowledged for its potential.
The latter one instead was employed as it was the best model in terms of performance so far.

Model	Accuracy	Precision	Recall	F1-Score
BERT ***With*** *preprocessing*	**0.889**	**0.847**	**0.848**	**0.847**
BERT ***Without*** *preprocessing*	0.865	0.805	0.818	0.811
Starspace ***With*** *preprocessing*	0.865	**0.828**	**0.782**	**0.802**
Starspace ***Without*** *preprocessing*	**0.867**	0.826	0.776	0.797

Table 4.15: Metrics value with and without preprocessing for Starspace and BERT

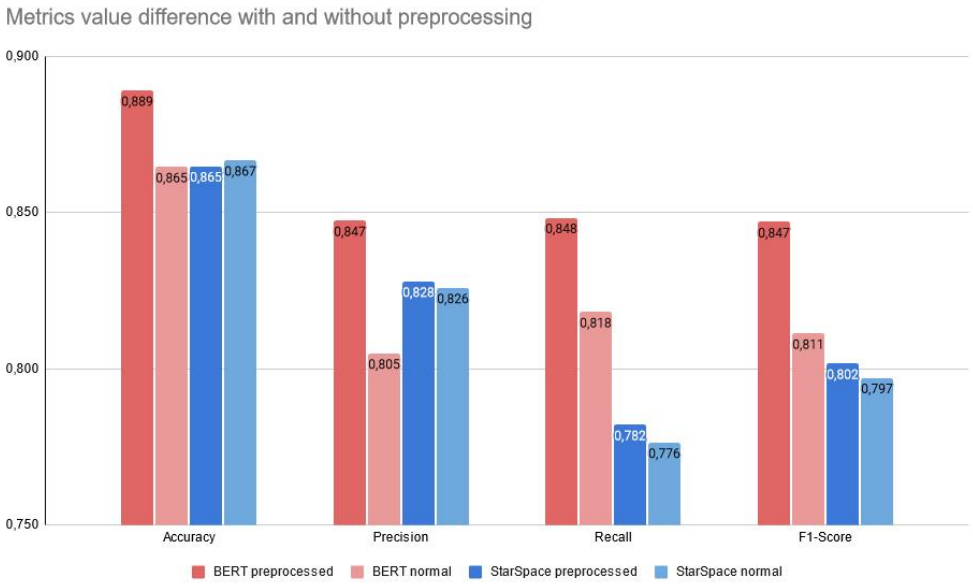


Figure 4.7: Metrics value with and without preprocessing for Starspace and BERT

In table 4.15 is visible the difference between the value of the metrics obtained by both models with and without preprocessing, with a graphical view in figure 4.7.
It is evident that BERT displays significant improvement across all metrics, even with a marked increase.
One possible intuition about this behavior regards the reduction in the number of $[UNK]$ tokens, which indicate tokens not present in the global dictionary.
Starspace on the other hand, performs in overall roughly similar to using the unprocessed dataset, with a slight improvement in precision, recall, and F1-score.

However, It is worth noting that the applied preprocessing technique did not perform the removal of the stopwords from the text.

Model	Accuracy	Precision	Recall	F1-Score
BERT without stopwords	0.878	0.828	0.841	0.834
Starspace without stopwords	0.858	0.807	0.781	0.793

Table 4.16: Metrics value without stopwords for Starspace and BERT

Table 4.16 displays instead the metrics obtained through the preprocessing procedure to which was added the stopwords removal.
Both tables show an improvement in metric scores when using a preprocessing technique that retains stopwords, in contrast to the same model trained on an unprocessed dataset.
However, a slight decrease in values is observed in comparison to the preprocessing technique that also eliminates stopwords.

This test has demonstrated that stopwords are an important component of this task. In fact, stopwords add inherent value to a sentence which can influence the value of the sentiment.
For instance, a common Italian stopword is "*non*", whose usage can introduce significant changes as it defines a negation.
Consider the sentence "*Io ti voglio bene*"(*I love you*) which can simply be categorized as positive.
However, the sentence "*Io non ti voglio bene*"(*I don't love you*) maintains the same semantic meaning while changing the possible sentiment.

4.6 Final Considerations on the Baseline Models

Following the tests conducted and detailed in the previous sections, some decisions have been made.
The advantages that BERT offers are clearly visible, both the use of the contextual embeddings and the fine-tune mode leads to a performance improvement, surpassing baseline models and Starspace.
Thus, a more detailed investigation on using BERT must be performed to understand the potentiality it can offer and to evaluate how much the performance may vary.

One more important aspect that must be analyzed carefully regards which fine-tuning technique to use.
There is little disparity between the outcomes of the two, however, some more consideration must be taken into account.
Indeed, Hugging Face offers a simple interface for training and also performs inference, however, is not trivial to modify the classification layer if needed.
The class *pipeline* from the library, provides an object which can be easily used for performing inference.

On the other hand, the BERT as a layer modality allows a more customizable approach to design the neural network, however, the inference methodology is not as trivial as with pipeline, which is not available for Keras.
This consideration must be discussed as this is a real-world scenario and not an academic evaluation of models. A more detailed discussion is available in appendix A.

As regards the preprocessing stage, particularly in the use of BERT, its inclusion resulted in an improvement in the overall performance.
Therefore, it was deemed essential to preserve and integrate it seamlessly into the pipeline.
Lastly, despite the quite good performance achieved thought the regression method, a multi-classification approach was chosen as it appears to be more intuitive from a logical point of view.

4.7 In-depth BERT Analysis

So far, BERT approaches have exhibited superior results when compared to the other tested technologies. The potentiality that it offer are evidently substantial, yet a more detailed study is necessary in order to obtain further refinement in the results achieved. Going more in detail, as visible in the documentation, HuggingFace's models returns three different elements [47]:

- **Last Hidden States**: final output of the transformer model, produced at the end of the encoding process. They are 768-dimensional embeddings obtained from the last hidden layers for each token.
- **Pooler Output**: a representation of the entire input sentence;
- **Hidden States**: Hidden states of the model at the output of each layer plus the optional initial embedding outputs.

The pooler output, according to the HuggingFace documentation, is the last hidden state of the first token of the sequence $[CLS]$, further preprocessed by a linear layer and a tanh activation function.
Although it is used for classification purposes, HuggingFace documentation warns that the output may not be the best representation of the semantic content of the input [48].
A suggested alternative, which may be able to get better results, is to pool the hidden layer sequence for the entire input.
For this reason, it was decided to further investigate the possible architectural solutions and the results that could be obtained.
However, while the interface provided by HuggingFace is useful, it may not offer the same level of architectural customization as Keras.
For this reason, the architectural tests were performed only with the BERT as a layer approach.
Hyperparameters did not change in the following tests, thus, GilBERTo was the pre-trained model used.

4.7.1 Pooler Output

As previously mentioned, the BERT as a layer approach exploits *TFAutoModel* to load a pre-trained BERT model and leverage its output.
This class allows the use of the previously described outputs so that they can be manipulated and exploited before actually feeding them into the output layer of the neural network, which contains the softmax function.

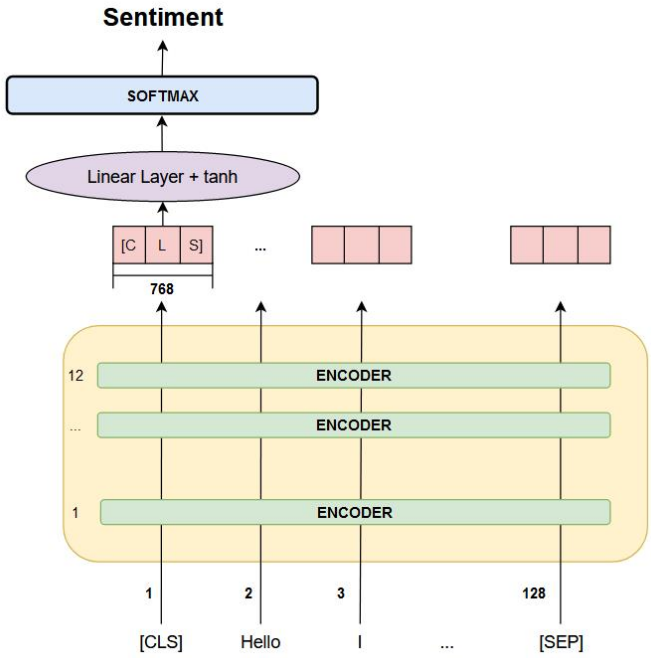


Figure 4.8: Pooler Output Architecture

A general overview of the entire architecture that use the pooler output is visible in figure 4.8.
As already described, the embedding of the $[CLS]$ token is further preprocessed with a linear layer and a hyperbolic tangent function.
The obtained encoding is fed into the softmax layer of the neural network defined which finally provide the sentiment value of the given sentence.
This is the architecture used in the tests seen so far.

Model	Accuracy	Precision	Recall	F1-Score
Pooler Output	0.889	0.847	0.848	0.847

Table 4.17: Metrics value with Pooler Output Architecture

In table 4.17 metrics values are visible with the pooler output architecture.

4.7.2 CLS Token Concatenation

An alternative methodology tested was inspired by the official BERT paper [25] and another interesting paper regards how to fine-tune BERT models. [46]
Indeed, the BERT paper claims that the feature-based approach, where features are extracted from the pre-trained model, gives some advantages.
The tests in the paper were conducted by applying the feature-based methodology without fine-tuning any parameters of BERT.
The best procedure works by concatenating the token representation from the top four

hidden layers of the large BERT model. By doing so, BERT were able to performs competitively with state-of-the-art methods.
The underlying concept is that each layer of BERT is able to captures distinct features of the input text.
Instead, in "*How to Fine-Tune BERT for Text Classification?*" [46], is shown that even with the fine-tuning approach, the last four hidden layers allows to obtain competitive performance.
Thus, the $[CLS]$ token in the last 4 hidden layers were concatenated and sent to the softmax activation function which computes the probability of the sentiment.

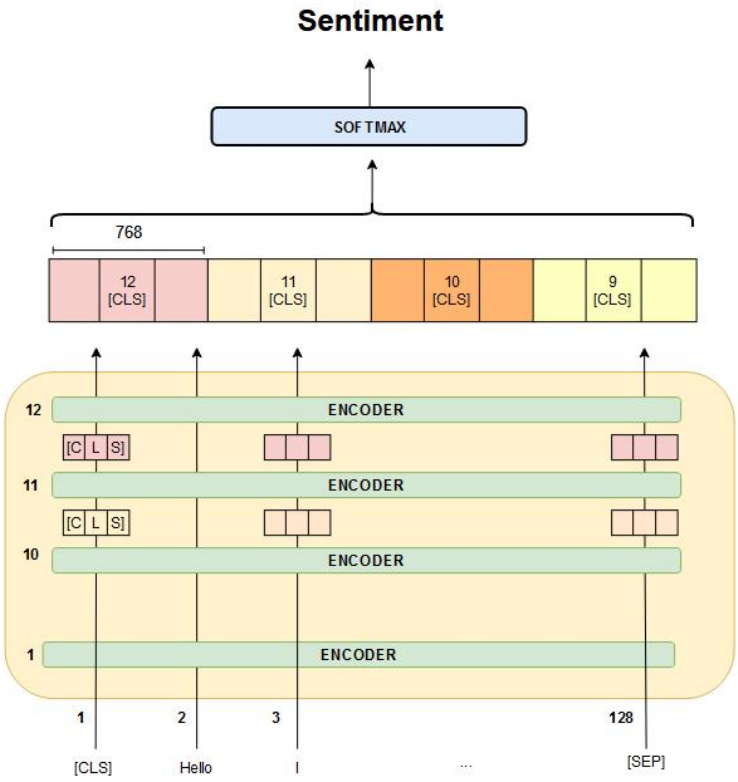


Figure 4.9: CLS Concatenation Architecture

In figure 4.9 the overall architecture is shown and the value of the metric obtained are visible in table 4.18 in comparison to the metrics achieved by exploiting the Pooler Output.

Model	Accuracy	Precision	Recall	F1-Score
Pooler Output	**0.889**	**0.847**	**0.848**	**0.847**
CLS Concatenation	0.887	0.841	0.840	0.846

Table 4.18: Metrics value with CLS Concatenation and Pooler Output architecture

Upon observation, the CLS concatenation architecture exhibits similar metric values compared to the Pooler Output.
Nevertheless, despite the previously cited paper and the warning notes in the HuggingFace documentation, the values still persist at a lower level with respect to the Pooler Output.

4.7.3 Global Average Pooling

As already mentioned before, the HugginFace documentation state that the pooler output is not a good representation

> *This output is usually not a good summary of the semantic content of the input, you're often better with averaging or pooling the sequence of hidden states for the whole input sequence.*

To achieve a more concise sentence representation, it is possible to employ a *Global Average Pooling* on the sequence of all hidden states. [18]
The idea behind the global average pooling is to take the average of each feature map [99] in order to exploit each token of the sentence.
Thus, instead of taking into consideration only the $[CLS]$ token, an average pooling operation is performed in the entire last hidden layers.

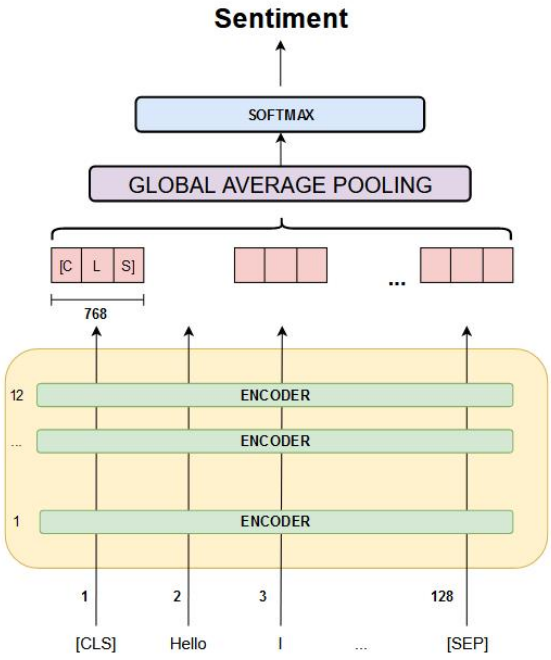


Figure 4.10: Global Average Pooling Architecture

In figure 4.10 is possible to see the overall architecture of this technique.

Model	Accuracy	Precision	Recall	F1-Score
Pooler Output	**0.889**	**0.847**	**0.848**	**0.847**
CLS Concatenation	0.887	0.841	0.840	0.846
Global Average Pooling	0.880	0.834	0.834	0.834

Table 4.19: Metrics value with Global Average Pooling and others architecture

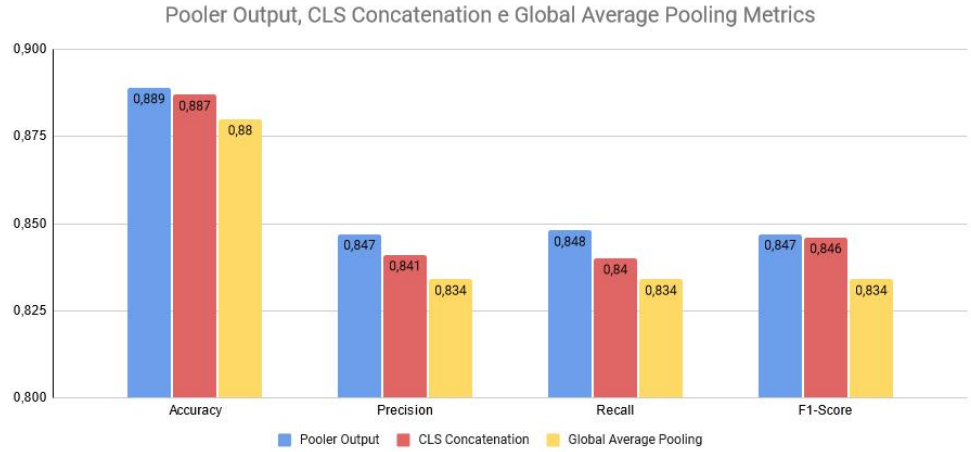


Figure 4.11: Metrics difference between architectures

In table 4.19 are visible all the metrics values obtained by the different architecture with a graphical representation in figure 4.11.

Despite the claims on the HugginFace official website, values obtained from the pooler output tend to be higher than those generated by other two tested architectures.
For all the metrics, the Global Average Pooling is surpassed by all other architectures, while CLS concatenation reaches values comparable to the pooler output's values.
As is visible, metrics values remain approximately consistent across all three architectures, each of them exceeding those obtained from the baselines.

4.7.4 Grid Search

To find the optimal hyperparameter combination which results in the best model performance, a grid search evaluation was performed.
It involves testing the different hyperparameters and evaluating the obtained metrics. Hyperparameters such as learning rate, batch size, number of neurons, and regularization technique can affect in a consistent way the model performance and lead to better classification.
As mentioned in chapter 2, several Italian BERT models are available, and each of them was trained on a diverse large set of data.

Thus, one more hyperparameter that can be evaluated is the chosen pre-trained model. Indeed, models might be trained with different kinds of data and thus, their vocabulary may vary.
For this reason, in order to perform a better estimation, it was decided to test the performance that models can offer.
In this test, for the BERT as a layer modality, the pooler output architecture was used, and the other hyperparameters remained the same as before.

Model	Accuracy	Precision	Recall	F1-Score
GilBERTo	**0.889**	0.847	0.848	0.847
Italian Base XXL	0.8758	**0.8654**	0.8397	0.8502
UmBERTo	0.8581	0.8634	0.8069	0.8299
AlBERTo	0.8706	0.8553	**0.8538**	**0.8545**

Table 4.20: BERT as a layer metrics values

Model	Accuracy	Precision	Recall	F1-Score
GilBERTo	0.880	0.845	0.809	0.824
Italian Base XXL	**0.892**	**0.854**	0.839	0.846
UmBERTo	0.877	0.834	0.813	0.823
AlBERTo	0.891	0.850	**0.841**	**0.849**

Table 4.21: HuggingFace interface metrics values

As visible in table 4.20 and in table 4.21, where are shown the metrics value respectively for BERT as a layer and HuggingFace interface, is evident the generic positive trend that BERT models lead to this task where each metric reach at least 80% of the score.
Furthermore, the difference between the metrics values obtained in both cases is not significantly pronounced.
Thus, model choice might affects the global results, but not in such a way as to guarantee the certainty that the chosen model leads to better performances.
The BERT as a layer approach exhibits the lowest recall, with UmBERTo achieving the smallest value of 0.8069, which is the least among all.
In contrast, when utilizing HuggingFace's interface, GilBERTo yields the worst recall results.
Even if AlBERTo was trained on data similar to chat and the obtained results were satisfying, it was decided to keep using and testing the **Italian Base XXL** model for mainly two reasons.
The data used to train the Italian Base XXL model were more generic and less specific with respect to the AlBERTo training data.
Thus, this implies a better generality of the model, still managing to obtain good performance in the test set.

Furthermore, Italian Base XXL was trained on a larger corpus compared to all other pre-trained models, thereby increasing the probability of encountering fewer unknown tokens in the dataset.

It is important to note that from this point, all the tests performed were based on the Italian Base XXL.
During the internship, also other pre-trained models were tested with different hyper-parameters.
However, for clarity, the discussion will limit to the tests conducted using the Italian Base XXL model, which in the end was chosen also for the reasons previously stated.

Due to the limited computational power offered by Google Colaboratory and Kaggle, a fixed number of epochs was used while checking for the best possible learning rate value.
Indeed, in the following tests, models were trained with 2 epochs.
In table 4.22 are shown the value of the metric obtained by the HuggingFace approach with various learning rates. In the same way, in table 4.23 there are the BERT as a layer results.
Performance does not change significantly using a learning rate between 1e-5 and 5e-5, however, is visible that the models with a learning rate 2e-5 performs slightly better with respect to the other.
It is important to note that 2e-5 is the suggested learning rate to avoid the phenomenon of the *catastrophic forgetting* described in [46].
Indeed, as counter-evidence, two higher learning rates (2e-3 and 2e-4) were tested. However, the results are not satisfactory, with a drastic drop in all metrics values, both for HuggingFace Interface and BERT as a layer approaches.

LearningRate	**Accuracy**	**Precision**	**Recall**	**F1-Score**
2e-3	0.71627	0.23876	0.33333	0.27823
2e-4	0.80703	0.54327	0.63323	0.58434
1e-5	0.89071	0.85554	0.83922	0.84713
2e-5	**0.89425**	**0.85892**	0.84832	**0.85346**
3e-5	0.88907	0.84912	0.84578	0.84741
4e-5	0.88689	0.85008	0.83821	0.84403
5e-5	0.88989	0.84898	**0.84945**	0.84918

Table 4.22: HuggingFace interface metrics values

LearningRate	Accuracy	Precision	Recall	F1-Score
2e-3	0.71627	0.23876	0.33333	0.27823
2e-4	0.71627	0.23876	0.33333	0.27823
1e-5	0.85827	0.82628	**0.87022**	0.83741
2e-5	**0.88744**	**0.84853**	0.83988	0.84414
3e-5	0.88427	0.83513	0.82398	0.82263
4e-5	0.88419	0.84586	0.85449	**0.84955**
5e-5	0.88226	0.83332	0.83362	0.83272

Table 4.23: BERT As A Layer metrics values

The performance achieved with only a few epochs raises concerns about the necessity of a high number of epochs.
For this reason, setting the learning rate at 2e-5, which had yielded the optimal results for most metrics, a number of epochs in the range of 2 to 5 were tested.

Epochs	Accuracy	Precision	Recall	F1-Score
2	0.89425	0.85892	0.84832	0.85346
3	0.88825	0.84904	0.84694	0.84786
4	**0.89452**	**0.86132**	**0.84994**	**0.85541**
5	0.88525	0.84602	0.84203	0.84348

Table 4.24: HuggingFace interface metrics values

Table 4.24 presents the metrics values obtained with varying numbers of epochs for the HugginFace interface method.
As visible, all the values are similar, however, the best performance measure achieved is with 4 epochs.
In the other approach, where the results are visible in table 4.25, with 2 epochs is possible to reach the best accuracy, precision and F1-score, while the recall value with 5 epochs reaches the best score of 0.85.

Epochs	Accuracy	Precision	Recall	F1-Score
2	**0.88744**	**0.84853**	0.83988	**0.84414**
3	0.87708	0.84670	0.84349	0.84163
4	0.88307	0.83511	0.83854	0.83642
5	0.87326	0.81501	**0.84566**	0.82947

Table 4.25: BERT As a Layer metrics values

Going decreasing learning rate, the training time starts increasing, reaching more than 10 minutes for each epoch.
However, this is as expected from theory, indeed, setting too low learning rates usually increase the training time since there will be many updates needed to reach the least possible error.
Indeed, setting a too large learning rate can lead to divergence.

4.7.5 Freezing Layers

One additional way to train the classifier in the head of the BERT layer is to "freeze" the BERT parameters and train only the head classifier which performs the prediction. By freezing the BERT layers, there is a significant reduction in the number of parameters to be trained. Thus, the training time will be lower with respect to the training time without freezing BERT.
However, a possible drop in the value of the metric may occur, as BERT is not able to learn specific-domain words and sentences from the dataset.

Model	Accuracy	Precision	Recall	F1-Score
Without *Freezing Layer*	0.892	0.854	0.839	0.846
With *Freezing Layer*	0.716	0.239	0.333	0.278

Table 4.26: Metrics difference between training and not training BERT layers.

In table 4.26, are visible the value of the metric after training BERT with and without freezing the layers. Not training BERT parameters results in a significant decrease across all metrics.
An interesting observation is the high value obtained by the Accuracy. This is a clear example of how this metric can be misleading.
In fact, since the model predicts the same outcome(neutral/positive) and the neutral/-positive class dominates also the test set, the accuracy metric increase.

4.7.6 Regularization

As previously stated, a primary problem when dealing with large models is overfitting, which can occur when models are not able to generalize well on unseen data.

The validation set can be utilized during the training process to assess the model's performance on new data.
The main overfitting symptom is a substantial difference between training and validation loss.
In order to alleviate this issue, several commons techniques might be applied, in particular, some regularization modalities [57] used were:

- **Dropout**: With a certain probability, during the training procedure, some number of layer outputs are randomly ignored (dropped out).
- **L2 Normalization**: Also known as weight decay, it adds a norm penalty in the loss function.
- **Label Smoothing**: The idea is to add randomness in order to reduce model variance and lower the generalization error. Label Smoothing add noise at the output targets (the labels).
- **Early Stopping**: This technique stops the training when the training loss is no longer decreasing, but the validation error is starting to rise.

With the HuggingFace interface approach, the dropout technique was not used since the customizability of this method is not trivial as BERT as a layer.
Moreover, overfitting is easily visible in training with a higher number of epochs. The early stopping may help reduce error by blocking training actually before the end of all the epochs.
During some tests, through early stopping, the model was often stopped using 2/3 epochs, thus blocking the impact of overfitting in the model. The metrics obtained as fewer epochs have been seen previously and, although not the best, they turn out to be very high.

Moreover, an alternative to the classic Adam optimizer was tested, the **RectifiedAdam** [40].
It is a novel variant of the Adam optimizer which introduces a term to rectify the variance of the adaptive learning rate. By doing this, it tries to solve the Adam bad convergence problem.
Thanks to a TensorFlow addon, which implements this optimizer, the RectifiedAdam was tested on the BERT as a layer approach.

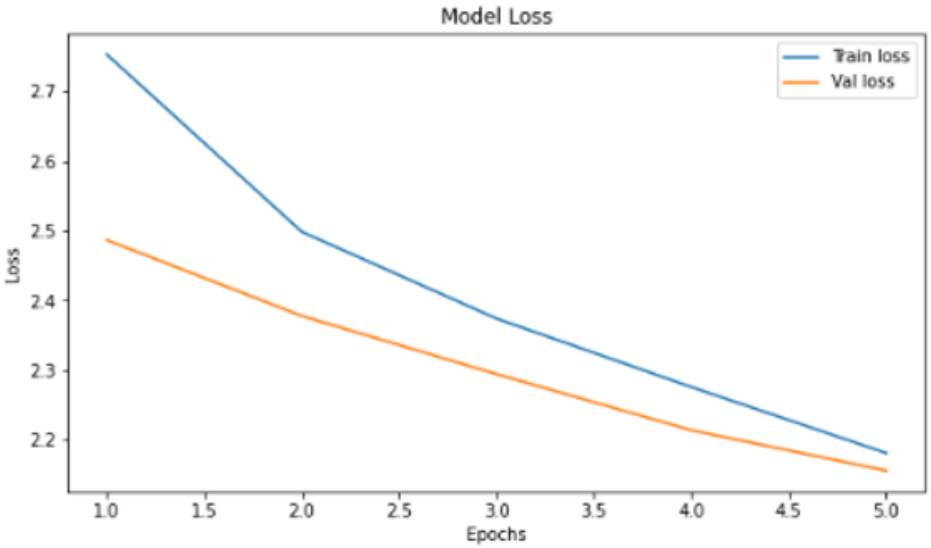


Figure 4.12: Loss with Adam Rectified

In figure 4.12 is visible the loss obtained by the Adam Rectified optimizer with BERT as a layer approach.
As clearly visible, there is no overfitting and the trend of the validation and train curves is very similar.
However, despite the downward trend, the loss remains high, with a value of around 2.2 after 5 epochs.
Instead, employing the HuggingFace approach yields a lower loss value, reaching approximately 0.2 with 2 epochs.

4.8 Further Experiments

The conducted tests have yielded satisfactory results so far, with an overall performance of approximately 82/84% for each metric.
However, an observation is that the confusion matrices obtained have a common characteristic.
Indeed, as can be seen in figure 4.13, BERT demonstrates a tendency to misclassify neutral/positive sentences as negative, highlighting a difficulty in distinguishing between these sentence types.

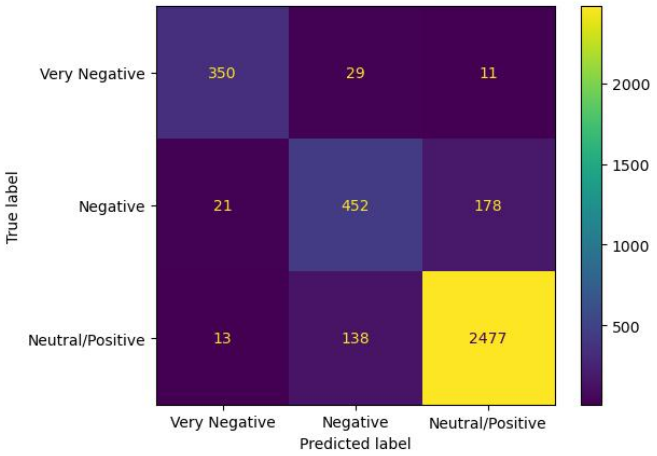


Figure 4.13: Confusion Matrix

Indeed, the confusion matrix displays how many negative sentences are actually classified as neutral/positive and vice versa.
In this case, 178 elements were actually negative but BERT predict a neutral/positive label, while 138 neutral/positive elements were classified as negative.
This was an initial indication of further data investigation, followed by the confirmation using the lexicon-based approach and the actual data inconsistency, noted during the manual review.
In fact, with the goal of improving BERT prediction, a lexicon approach has been performed in order to integrate it into the BERT results.
However, the attempt to use this approach led to a complete dataset review due to the inconsistency of the sentences.
In this section, a description of the lexicon approach and further experiments regarding the dataset reviews will be explored.

All the metrics values from this point will regard only to the final chosen method which is the HuggingFace interface.
The relatively similar metric values observed between the two approaches, along with the non-substantial impact of using different architectures, led to the choice of this approach.
However, an in-depth study of the model implementation within the working context, which will be discussed in the subsequent chapter, influenced the choice.

4.8.1 Lexicon Approach

In sentiment analysis, with the lexicon approach, words in phrases are categorized with a possible sentiment with the help of a so-called *valence* dictionary [11], which defines for instance the word "great" as positive and the word "bad" as negative.
With a complete dictionary, is possible to compute an overall sentiment score simply by counting the numbers of positive and negative words in a sentence and combining them.

One popular formula to compute this score is visible in 4.1.

$$SentimentScore = \frac{Number\ Positive\ words - Number\ Negative\ words}{Total\ Number\ of\ Words} \tag{4.1}$$

Thus, with a negative sentiment score, the text is classified as negative. Otherwise, if it is greater than 0, it means a positive classification.
This methodology has limitations and is not effective as a machine learning model because the classification depends only on the valence dictionary.
Nevertheless, evaluating its efficacy may be useful in order to test a possible enhancement in complex sentences that may be difficult for BERT.

This idea has been taken into consideration as it offers the opportunity to exploit the existing categorized dataset.
By utilizing negative and positive lexicons generated and comparing them to the lexicon generated by the sentence to categorize, it might be feasible to understand the sentiment of the sentence.

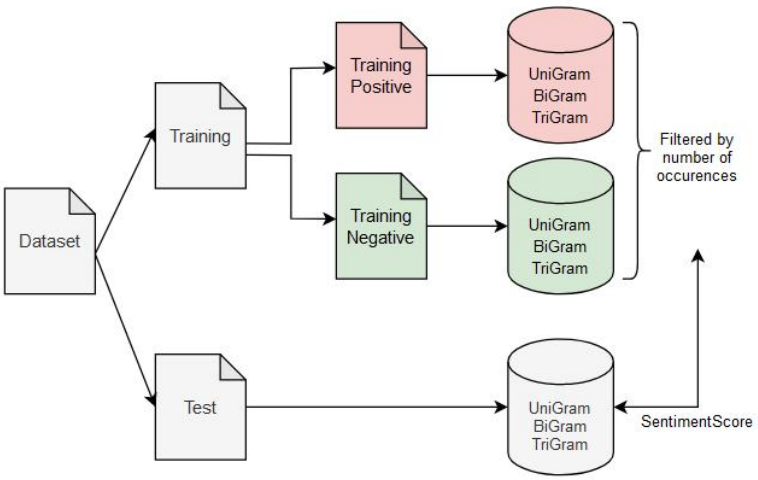


Figure 4.14: Lexicon Approach

Figure 4.14 shows the workflow for the creation of a lexicon with unigrams, bigrams and trigrams.
After a stopwords removal, negative and positive sentences were obtained from the dataset. As regards the negative lexicon, it comprised both "very negative" and "negative" categories.
For each sentence, in both the lexicon, unigrams, bigrams and trigrams were extrapolated.
However, might be possible that occurrences appear in both lexica. As an example, the unigram "*conto*" exists within both dictionaries.
Hence, to address this was opted to filter the lexicon based on the frequency of occurrence. More precisely, the number of times a unigram, bigram or trigram appears in positive and negative training dataset was counted.

Then, the respective n-grams go inside the positive lexicon if it appears more frequently among the positive sentences and vice versa.
In addition, any terms with a frequency of 1 were excluded from consideration.
The unigram "*conto*" appears 623 times in training positive while 472 times in negative, thus in this case it will be present in the positive lexicon.
The idea of using bigrams and trigrams starts from this point. Indeed, while the unigram "*conto*" on its own may not carry a negative connotation, adding another word such as "*chiusura*" could potentially transform the bigram "*chiusura conto*" into a negative expression.

This approach is incorporated into the prediction of BERT as follows:

1. Generate the lexicon from the training set
2. Set a THRESHOLD
3. Extract unigram/bigrams/trigrams from sentence to be predicted
4. Use the lexicon in 1 to get positive elements of sentence
5. Use the lexicon in 1 to get negative elements of sentence
6. Compute SentimentScore
7. If (6) $<$ - THRESHOLD, reduce to one step the BERT predicted sentiment
8. If (6) $>$ THRESHOLD, upgrade to one step the BERT predicted sentiment

By analyzing the results obtained by this approach, was visible a worsening of the global performance, indeed, this methodology lead to a change in prediction which was categorized correctly by BERT.
The only positive outcome was a better understanding of how the sentiment categorization must be reviewed. Indeed some clearly positive terms were put in the negative lexicon which was derived directly from the training set.
Thus, this term appears more frequently in negative sentences instead of positive ones.
For this reason, the complete review of the data, discussed in 3 was performed.

4.8.2 Dataset Reviewed Test

Following the two revisions explained in chapter 3, the final dataset appears to have a distribution that is visible in figure 4.15 with a detailed overview for each sentiment in table 4.27.

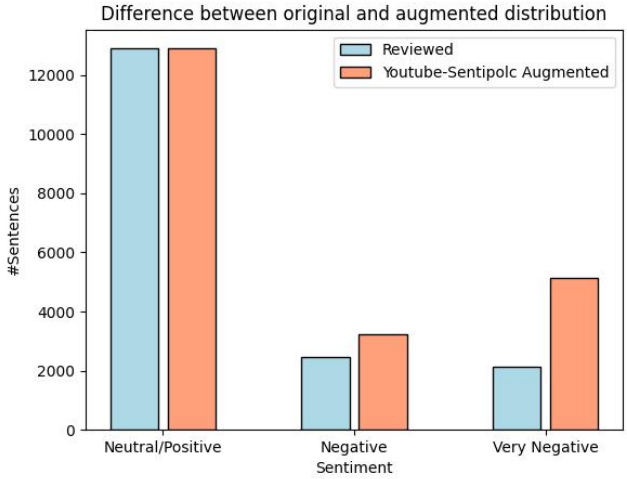


Figure 4.15: Reviewed (*CustomerData_3*) and augmented (*CustomerData_3-YT-SP*) dataset distribution

Sentiment	***CustomerData_3***	***CustomerData_3-YT-SP***
Neutral/Positive	12.895	12.895
Negative	2.480	3.245
Very Negative	2.348	5.148
Total	**17.723**	**21.288**

Table 4.27: Number of elements for *CustomerData_3* and *CustomerData_3-YT-SP* dataset

In table 4.28 instead, is possible to see the marked difference between metrics obtained after the review.
The accuracy value surpassed 90% for the first time, with a significant increase also in the other metrics.

Dataset	**Accuracy**	**Precision**	**Recall**	**F1-Score**
CustomerData_2	0.892	0.854	0.839	0.846
CustomerData_3	**0.916**	**0.876**	**0.862**	**0.870**

Table 4.28: Metrics difference between original (*CustomerData_2*) and reviewed (*CustomerData_3*) dataset

4.8.3 YouTube and Sentipolc Augumentation

Following the dataset review, a test using the Youtube and Sentipolc augmentation approach was performed.
In this way, very negative and negative sentences were added from a YouTube Hate Speech and Sentipolc dataset to enhance the dataset balance.
As visible in figure 4.15, the unbalanced nature of the dataset remain unaffected.
However, the main idea behind this test is to add variety in negative and very negative sentences, and understand if the model is able to distinguish them in a better way against the neutral/positive elements.

Dataset	Accuracy	Precision	Recall	F1-Score
CustomerData_2	0.892	0.854	0.839	0.846
CustomerData_3	0.916	0.876	**0.862**	0.870
CustomerData_3-YT-SP	**0.921**	**0.890**	0.861	**0.874**

Table 4.29: Metrics difference between original (*CustomerData_2*), reviewed (*CustomerData_3*) and augmented (*CustomerData_3-YT-SP*) dataset

In table 4.29 are visible the metrics difference between the tested dataset.
It is important to note that the test in the augmented case, was performed only on the data of the client. Thus, Youtube and Sentipolc texts were used only for training.
Results are slightly better with respect to the reviewed dataset, this could mean that model is able to better distinguish the categories.
Recall value does not change significantly, while Precision reaches a score very close to 90%.

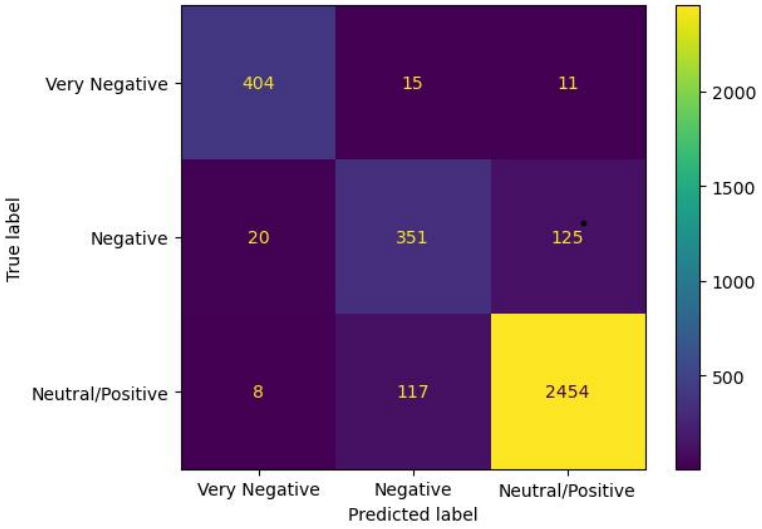


Figure 4.16: Confusion Matrix

In figure 4.16, is possible to see a lower error for the true neutral/positive sentences, with a reduction of 53 elements in comparison to 4.13.
Instead, as regards the true negative sentences, is possible to see that 21 phrases were this time categorized correctly.
As visible, the variety of the data obtained by Youtube and Sentipolc yields an improvement of the metrics even if they are not closely related to them.
However, the nature of the data source (Youtube and Tweets), is a component which is able to connect the two datasets with the original customer dataset.

Chapter 5

Conclusions

Language is part of a complex system defined by signs, symbols and rules that allow communication between human beings.
It is not fixed but is mutable and constantly evolving, changes that may depend on various factors including the geographical basis, social class, education and many others.
As a result, tasks related to text, particularly those pertaining to Natural Language Processing, are not easily solvable, necessitating the utilization of an expansive model with substantial capabilities.
The main objective of the thesis was to investigate the potential of artificial intelligence to enhance the customer experience offered by PAT Srl solutions.
The key component that initiated it all was the necessity of a sentiment analysis model based on users' requests provided in the chatbot.
Moreover, the framework's architecture of PAT solutions had to be studied in order to integrate the intelligent solution into Engagent and CXStudio.
In summary, the whole journey of researching and developing the optimal sentiment analysis model in terms of metrics, along with the integration of it into various deployment platforms was the key goal of the thesis.
An additional and important component was Data Analysis, indeed a large amount of high-quality data is required in artificial intelligence techniques.
Exploring data and applying preprocessing techniques were two fundamental steps to put a solid foundation for an effective machine learning model.
A more in-depth study was performed for the casing of sentences and the utilizing of stopwords, since their use is a common approach in natural language processing tasks, however, in sentiment analysis, it can lead to performance variations.
Some common approaches were used, while others, following a study, were applied based on how much the performance of the various models improved, such as stopwords removal.
Furthermore, the unbalanced nature of the data led to the use of an approach that could reduce this problem, exploiting other types of texts that could be suitable for the analyzed use case.
Several artificial intelligence approaches were tested, from basic baselines to state-of-the-art methods which lead to great results. The conducted tests out have demonstrated the limited capabilities of models such as Support Vector Machine, Decision Tree or Naive Bayes, which however are solid and consolidated baselines for text classification.

Instead, BERT's great ability to understand and assimilate the words' context was a fundamental component to overcoming other models that have already been used within the company such as Starspace.
In-depth studies about BERT architecture have not led to substantial improvements in results, contrary to what has been observed in the HuggingFace documentation.
However, one thing that managed to improve the results was the overall data improvements. It is vital, to have a high-quality set of data in order to improve the final outcome, in fact, as can be seen from the various tests carried out, reviews and augmentation attempts have led to an increase in terms of performance.
Moreover, leveraging large models like BERT in different environments with respect to academia can be challenging. Space and time issues must be taken into account in order to keep high performance, especially when the amount of requests is substantial. Therefore, it is crucial to optimize the equilibrium between efficiency and effectiveness.
During this thesis, I was able to deal with a real-world scenario, which, although in some respects similar to the academic one, is different from others.
For example, in companies, maintenance and support are critical components of the software development life cycle. Moreover, collaboration and effective communication between team members is essential for successful software development.
Having had the opportunity to enter a reality like the one offered by PAT Srl has provided me with a great boost in terms of both technical and non-technical skills.
The achieved outcomes have proven satisfactory. Nonetheless, some future improvements or possible alternative paths can be undertaken.
In the next sub-chapter, some potential improvements that may be interesting will be introduced.

5.1 Future Work

5.1.1 ChatGPT

In the recent period, it has been more and more heard about ChatGPT and its potential.
ChatGPT is a large language model developed by OpenAI based on an architecture called GPT (Generative Pre-trained Transformers).
It is able to understand and generate human-like responses thanks to a particular training procedure on a wide variety of internet text sources.
A language model is trained on a large dataset with different domains in order to learn patterns and probabilities of word sequences.
Once trained, a language model is able to generate a text by predicting the most likely next word based on all the previous words.
ChatGPT in particular was trained with a paradigm called *Reinforcement Learning from Human Feedback* (RLHF) where human experts incorporate additional information in the feedback used by the agent to learn.
In [82], it has been demonstrated that the predecessor GPT-2 which was at the time the largest model with 1.5B parameters, was able to achieve state-of-the-art results in 7 out of 8 tested language modelling datasets in the zero-shot setting.
In particular, zero-shot learning is a setup where a model is able to recognize things that it hasn't explicitly seen before in training.
Moreover, more recently OpenAI, with GPT-3, a 175 billion parameter language model, managed to reach strong performance in many NLP datasets, including tasks where on-the-fly reasoning or domain adaptation is required. [3]
Therefore, a possible future work involves leveraging Generative-Pretrained-Transformers and their ability to understand the most probable word for the prediction, offering possible benefits not only in sentiment analysis but also in other tasks.

5.1.2 Sentiment Analysis based on the conversation history

In this use case, sentiment analysis was applied to predict the emotion of a specific chat-based sentence.
Another approach that might be interesting to study, regards a more global conversational sentiment analysis.
Given that the dataset is obtained from the chatbot provided by PAT, it might present the possibility to extract not only customer messages but also the corresponding operator response.
Naturally, the final goal remains unchanged, predict the sentiment of the customer. However, the main difference between conversational sentiment analysis and single sentence sentiment analysis is the existence of context information which might influence the sentiment of an utterance in dialogue.[17]
Models such *BiERU: Bidirectional Emotional Recurrent Unit for Conversational Sentiment Analysis* recently were able to outperform the state of the art in many cases thanks to its enhancing capability of capturing dialogue information.
This possible future project necessitates an in-depth analysis of potential solutions, accompanied by an examination of the resources required to implement scientific research within work environment constraints such as time or economics.

Appendix A

Design and Implementation

Design and implementation are two fundamental steps in the software development process, especially in a business context.
Designing refers to understanding and defining how the software system might work and how it will be built.
Nowadays many technologies are available, providing a variety of options to achieve our objectives. Therefore, it is necessary to have a clear understanding of the system's requirements, be able to analyze effectively which technologies to choose and define the interaction between components.
The implementation step consists of actually building the software system based on the design carried out. Writing code, integrating components, testing the system, and debugging, are all phases which are required in this step.
During this internship, design decisions were made on two different levels: the artificial intelligence model and Docker's containers. In fact, efficiency and effectiveness are highly sought after in software development, thus, for this reason, a careful evaluation was conducted to determine which models to use, by studying the approach using HuggingFace and TensorFlow.
Moreover, from the point of view of the size, the weight of BERT models is very high. Thus, an approach with Docker Volume was performed in order to save space for the container.
Afterwards these considerations, a REST API with Tornado was developed which call is able to perform inference of a given sentence with the trained model.
In this way, the API can be called by the Adapter written in Groovy which interacts with the chatbot.
Moreover, an alternative deployment method was implemented through Kubernetes. Specifically, a sentiment analysis library was developed, which was integrated into a Kubernetes worker - a component responsible for application execution and managing workloads.
In addition, unit tests were created to evaluate the correctness of the API and the code was logged according to the rules present in Pat Srl.

A.1 API Creation with Tornado

An integral part of the internship project provides the development of an API used to perform inference on the trained model.
Indeed, by defining an API, it is possible to define a communication system and interact with a computer or a server to retrieve some information, which, in this particular case, is the predicted sentiment of the model.
A REST API is an API which conforms to the REST (Representational State Transfer) architectural style constraints. [96]
In a REST API information are exchanged in one of the following formats: JSON, HTML, XLT, Python, PHP, or plain text.
The API, developed with the Tornado Web Framework, define a URL which is used to call the predictive model, thus is used only for inference.
The response, elaborated by the server, is in a JSON format as shown below.

```
{
   "sentence": "Ciao, come va?",
   "language":"it",
   "model":"BERT",
   "sentiment":"Neutro/Positivo"
}
```

The JSON structure contains four distinct elements:

- **Sentence**: The evaluated sentence;
- **Language**: The model's language;
- **Model**: The model used for the prediction;
- **Sentiment**: The prediction obtained by the model.

The two keys "*Language*" and "*Model*" are fixed respectively to "*italian*" and "*BERT*" due to the fact that there are no other models available.
They were a requirement asked from the tutor, which might be useful for future improvements. Indeed, it will be possible to expand the API by simply adding other models if it is necessary.
The API, the code and the model are integrated into a Docker container which was defined starting from a Docker Image.
In the next section, more information about the creation of the container will be provided, with a specific focus on addressing space-related issues caused by the large size of BERT models.

A.2 Docker's Utility

As already mentioned in chapter B, Docker is a powerful open-source platform to facilitate the deployment, distribution and running of applications in lightweight and portable environments called Containers.
Containers include the application code, system libraries and dependencies, in this case, even the BERT model.
They are built starting from *Docker Images* which incorporate all the essentials (installation, application code, dependencies) requirements to configure a fully operational container environment and be able to run a specific application. [2]
Docker images can be created by defining a *Dockerfile* which contains all the instruc-

tions and specifications for building the image.
From the image, is possible to create the container thanks to the command
`docker build -t [IMAGE-NAME] [PATH-TO-DOCKERFILE]`
Once the container is created, its environment is isolated from the host system and other containers, with its own filesystem, network, and process namespace, enhancing security and flexibility performance. Moreover, containers may be started, stopped or restarted in case of need.
In particular, due to the fact that the application which the container has to execute is a REST API, the container must expose a port which will provide the clients an entrypoint to the predictive model with the URL defined thanks to the Tornado framework.

A.2.1 Space problems

A request that has been made was to minimize the size of the Docker image as much as possible. Indeed, the first attempts led to the creation of quite heavy images, around 2GB/3GB, which included also the predictive model.
For this reason, a more in-depth analysis to fix this issue was performed.
An initial approach to reduce the size of the Docker image consider the library used. In fact, as previously discussed in chapter 4, there were two approaches to utilizing BERT: the HuggingFace interface or the use of BERT as a layer.
While the first method uses the HuggingFace library to load the model, and PyTorch under the wood as the neural network engine, the latter approach still utilizes HuggingFace to load the model but employs Keras and Tensorflow as the neural network engine.
In order to use the BERT as a layer approach, Keras and Tensorflow are mandatory, due to the fact that PyTorch is not able to load and use the model developed with the two libraries.
Instead, is possible to use Tensorflow with the HuggingFace interface approach due to its portability capacity, however, as will be seen, is not convenient in terms of size.
Indeed, by analyzing the Keras and Tensorflow libraries, it has been determined that in a Windows environment, their amounts is about **1.14 GB**.
Instead, the PyTorch suite might be installed without the GPU environment, due to the fact that the API is used only for inference.
Thus, the PyTorch library will roughly occupy **800 MB** and, in addition, the HuggingFace library, which size is around **130MB**, must be added in order to use the models.
Thus, thanks to these "trick" with PyTorch, is possible to save extra space. With PyTorch and HuggingFace together the size turns out to be even smaller than the size of Tensorflow alone.
Moreover, one additional component of the image must be the model, otherwise, it will be impossible to perform the prediction.
Due to all these considerations, even installing PyTorch without the GPU environment, image dimensions were still high. Thus, some studies were performed to overcome this issue.

A.2.2 Docker Volume

As already mentioned before, in order to make predictions, it is necessary to access the model to perform inference.
Some tricks with the libraries have been performed, however, most of the weight of the images comes from the BERT models which can add more or less **500MB**.
Rather than installing the model within the image, an alternative approach, which exploits the capabilities of Docker, is known as **Docker Volume**.
As previously noted, containers have their own environment, with their operative system, dependencies and filesystem containing files.
A docker volume is a way to keep files intact even when a docker container is deleted. Indeed, is a type of directory which act as an additional filesystem but exists outside the container itself. [89]
They are designed to give the possibility to exchange information between the host and the containers and they are used for several reasons:

- Send data into a Docker container;
- Save date into a Docker container;
- Exchange data between Docker containers.

Thus, is possible to exploit volumes to store the models and allow the containers to use them as if they were actually inside the image.
By doing so, is possible to reduce the size of the Docker images and improving also performance in terms of generalization.
Indeed, it might be possible to design an application which calls a specific model inside the volume, without creating any new images or containers.
A Docker Volume can be defined in different ways, with the command *docker volume*, as an argument of the *docker run* command which is used to run a container, or inside the *dockerfile* with the instruction `VOLUME \path`.
However, with this latter approach, is not possible to rename the volume or assign a directory to the host, which instead can be done using the method that utilizes the "*docker run*" command.
Sometimes, creating a container may require multiple additional parameters, such as volume instructions, resulting in overly structured command line commands.
To simplify the process of container creation and avoid this complexity, the **Docker Compose** [27] tool can be used.
It works with an additional file which must be defined, the **docker-compose.yaml** and it contains all the necessary information to create the container, including the name, path to the Dockerfile, exposed ports and volumes.

An example is shown below.

```
version: "3.9"
services:
    sentimentanalysis:
        container_name: sentimentcontainer
        build: .
        ports:
            - "9000:9000:"
        volumes:
            - Path\To\Models: /volumePath
```

A.2.3 Design choice with Docker

Docker Volumes are a powerful method to save space for containers and may be helpful in many situations.
However, is necessary to take a design choice to understand how many volumes need to be set up. In this particular case, they might be used to contain specific models (BERT, Starspace, SVM) or all the models for a specific client.
The selected design option was made with a long-term and not specific-client perspective, with possible future models that could be used and it can be seen in figure A.1.

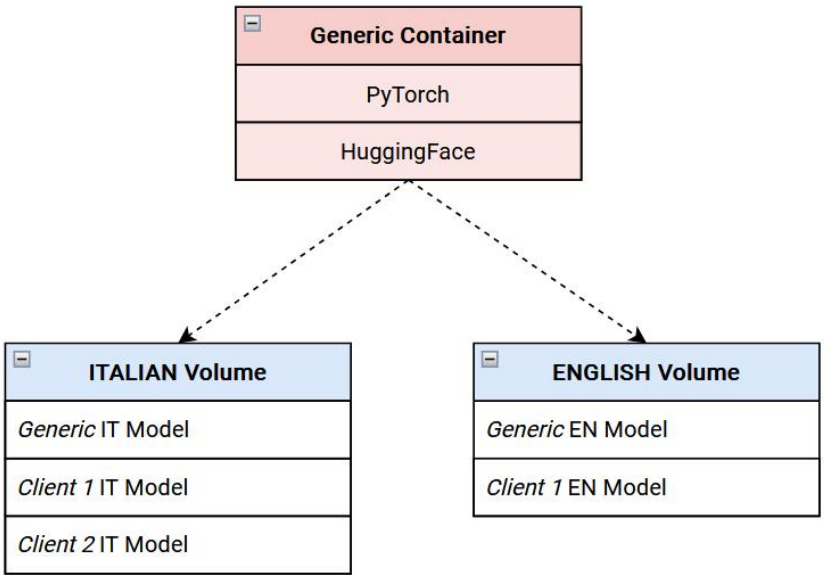


Figure A.1: Design choice with Volume

Two volumes were defined, one for the Italian language and another for English. Ideally, each of them might contain a generic model for sentiment analysis which is trained with general sources of text. Moreover, they might have client-specific models, as the one fine-tuned so far.
This approach allows a possible addition of further models that are specific to particular clients, if available. Furthermore, the logical separation of different languages is much more intuitive and less confusing in comparison to a potential separation based on clients. The latter may result in unnecessary duplication of generic models.
Indeed, it is not always feasible to train a model for a specific client since data are needed to perform the training. Therefore, in each volume of every client, a generic model would be necessary in the absence of a specific model.

A.3 TensorFlow vs PyTorch

As already seen, a specific point for solving the space issues has turned out to be the size of the library used. Indeed, the dimension of Tensorflow is even higher with respect PyTorch and HuggingFace together.

Starting from this point, a further investigation was performed in terms of the libraries chosen.
Indeed, in a real-world scenario, decisions can not be dictated solely by model metrics. Other factors, such as prediction time and code structure, must be also taken into account. After all, the written code will be read by other developers in the future who need to comprehend its functionality.

Description	**PyTorch**	**Tensorflow**
Library Size	**797 MB**	1.14 GB
Model Size	**423MB+952KB(Tokenizer)**	444MB+952KB(Tokenizer)
Prediction Modality	**Pipeline**	Tokenizer+Stack+Model
Inference Test Time	45sec / 3505 elements	46sec / 3505 elements
Inference Single Sentence Time	0.01 sec	0.01 sec

Table A.1: Comparison between PyTorch and Tensorflow

In table A.1 is visible the results of the analysis. Starting from the library size, which was already discussed, also the effective model size was checked.
The model via PyTorch and the HuggingFace interface is slightly lighter with respect to the BERT as a layer approach of TensorFlow, however, the difference is not that substantial. In both cases, the tokenizer is still used which is therefore necessary and weighs 952KB.
The inference time, and therefore the time required to obtain the predicted sentiment for 3505 sentences, was roughly 45 seconds for both solutions. This means that for each sentence models took 10 milliseconds to perform inference.
The last key point that needs to be analyzed regards the prediction modality. With Tensorflow, and thus with the BERT as a layer approach, the prediction method is quite complex and structured.
Indeed, the tokenizer is used to tokenize each sentence and obtain input_ids and attention masks. Subsequently, they are stacked together into a single object which will be fed into the neural network. Finally, the result is a probability distribution, thus, the most probable value is taken as the predicted sentiment.
Thus, is necessary to define a function that from a sentence returns the manipulated text which can be sent to the neural network. The **pipeline** abstraction instead, is an object offered by the HuggingFace library and allows to easily perform inference from a pre-trained model.

```
from transformers import pipeline

mod = AutoModelForSequenceClassifica-
↪  tion.from_pretrained("path_to_saved_model")

tok = AutoTokenizer.from_pretrained("path_to_saved_tokenizer")

pip = pipeline("text-classification",model=mod,tokenizer=tok)
```

As seen in the code above, utilizing the pipeline object only necessitates loading the tokenizer and the model and providing them as arguments to the constructor.
To actually perform inference, a sentence or a list of sentences can be provided to the pipeline object as visible below.

```
pip("Ciao, come stai?")
```

The results obtained by the instruction is the following:

```
[{'label': 'NEUTRAL/POSITIVE', 'score': 0.99645}]
```

In addition to the actual predicted label, the pipeline object returns a score which indicates the confidence of the prediction.
This approach, compared to the alternative, is simpler in several factors. The code's functionality is easily comprehensible without comments, and it is also easy to maintain, not too intricate, and capable to perform the task with few lines of code.
Thus, after all these considerations such as the library/model dimension and the predicting method, the HuggingFace Interface approach was preferred to the BERT as a layer methodology.

A.4 Adapter Development and Unit Test

Once the API is created, the last developed component was the Adapter. This element is what connects the API, and therefore the sentiment analysis model, to the ChatBot. More generally, adapters are the communication interface between the core component of the PAT framework and external services.
They are used to collect data and information to allow a correct evaluation of the rules. Thanks to them, it is also possible to integrate further artificial intelligence services, whether they relate to intent recognition, sentiment detection, or others.
Adapters need to define special methods that populate the necessary variables.
For example, the main function that performs the task is the *run* method which has to obtain the sentence, call the API REST developed for the prediction of the sentiment, understand the predicted value from the JSON obtained as the answer and set some context property which will be read from the CXStudio framework.

In order to verify the correctness of the program and the individual components, an important aspect of software development is unit tests.
They are useful since they help to find and fix potential errors earlier. Moreover, other advantages obtained by unit tests are that allow developers to quickly understand how to make changes in the code, allowing easier refactoring and they are a good component of project documentation.
In these kinds of tests, which usually are automatic, every individual unit or component of the software is tested in an isolated way to verify its correctness. The idea is to

define automatic tests which may be tested many times and in a fast way.
The definition of a unit depends on the developer who performs the test and how the software is structured, which usually might be an individual function, method, procedure, module or object. [91]
Overall, they are an important part of software development, useful to improve code quality, detect errors and as documentation.

A.5 Kubernetes Deployment

Kubernetes is a popular open-source platform used to manage and orchestrate various containers. It is essential to have an understating of management practises, especially when the number of applications begins to increase significantly.

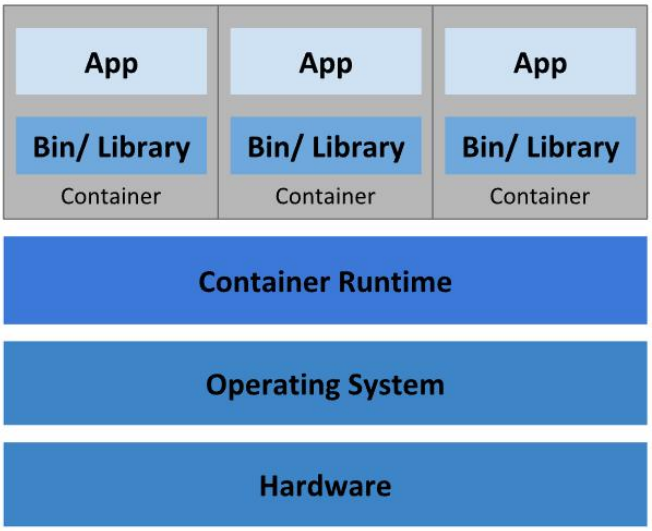


Figure A.2: Container Deployment [97]

Nowadays, the container deployment approach is increasingly gaining popularity due to the multiple advantages that they offer.
Containers are lighter with respect to virtual machines due to the fact that they share the operating system and at the same time, they contain their own filesystem, CPU, memory, PID (Process ID). [97]
In figure A.2 is possible to see a general overview of this kind of approach.
The benefits provided by containers regard simplicity and efficiency in container images in contrast to utilizing virtual machines, the high portability across various clouds and operating systems and well as enhanced resource efficiency.
In a production environment, effectively managing containers that execute applications is vital to prevent service interruptions. For instance, if containers shut down unexpectedly, is necessary to restart it. Kubernetes is the solution for handling such scenarios.

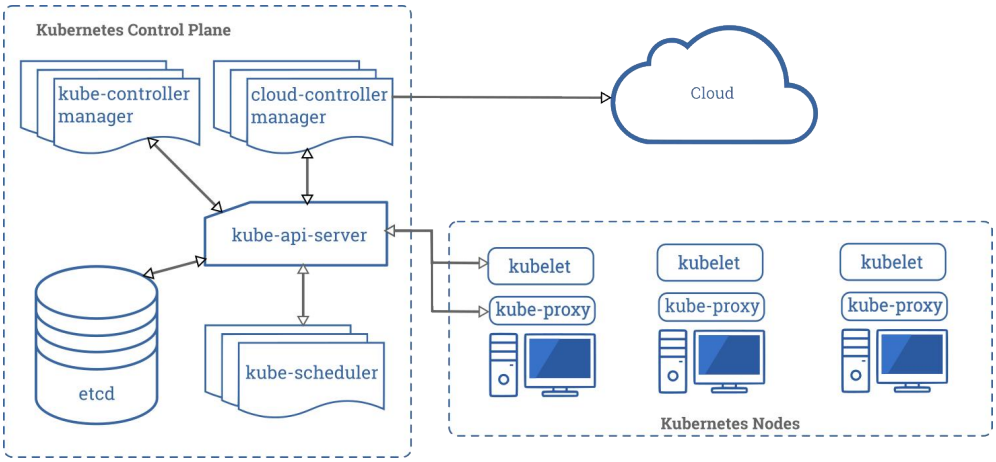


Figure A.3: Kubernetes' components [97]

In figure A.3, is visible the main components of a Kubernetes cluster which is obtained by the deploying of Kubernetes.
It is composed of two main components: a set of worker nodes which run containerized applications and services, managed by the Kubernetes control plane.
Workers run containers that host the application deployed on the cluster, while the control pane manages the overall state of the cluster, including scheduling, monitoring their health, and scaling up or down the number of nodes as needed.
In this internship, one additional requirement was the Kubernetes deploy modality, thus, after training with PAT developers explaining how Kubernetes works and how to implement nodes, a worker with the aim of predicting the sentiment of a sentence was defined.
In order to do that, a library was defined by following the direction of PAT Srl thanks to *Setuptools* which is a library that allows creating Python packages by simplifying the process of building and distributing.

Appendix B

Main Libraries and tools used

Libraries and tools are essential components of software development, as they provide pre-written code and functionality that can be reused in multiple projects, they help to streamline the development process, reduce errors and speed up the time to market. Tools, on the other hand, are software programs that are used to perform specific tasks or automate certain aspects of the development process.
Being more precisely on programming languages, **Python** and **Groovy** have been used for developing the solution.
The first one is a simple and versatile language and in addition, provides numerous libraries for machine learning and data analysis.
Groovy instead is an object-oriented programming language that was designed to be more powerful and flexible than Java, while still maintaining compatibility with the Java platform.
It has some features that make it seem more similar to Python such as dynamic typing, built-in support for regular expressions, and concise syntax.
In particular, Python will be used for developing the sentiment analysis machine learning, while Groovy is the main programming language in order to create the adapter that will be used in the chatbot.
In this section, main libraries and tools used to develop and deploy the entire project will be analyzed.

B.1 Libraries

Python boasts a vast variety of libraries used in diverse fields of study.
Throughout this part, primary libraries utilized within the project and their significance will be discussed.

B.1.1 Pandas

Pandas [73] is an open-source and free-to-use Python library which is widely for data manipulation and analysis.
It offers data structures for working with tables, matrices and time series data, with a set of tools in order to read or write data from a variety of sources, such as CSV, Excel, SQL database, and web APIs.
Pandas provides a fast and efficient DataFrame object for data manipulation with

integrated indexing. It represents a two-dimensional table-like structure, where each column can hold different data types.
Operation suchs as filtering, grouping, aggregation, merging and reshaping with DataFrame are performed in an exceedingly efficient manner.
Its popularity has grown more and more in recent years, the open-source nature of software, makes it an excellent tool for data scientists, researchers and analysts.
Figure B.1 shows the Pandas library logo.

Figure B.1: Pandas logo

B.1.2 Scikit-learn

Scikit-learn [76] is a comprehensive Python library for machine learning which provides a range of powerful tools and algorithms for various tasks in the field of data science.
It is built on other popular libraries in Python such as NumPy, SciPy and Matplotlib, this reduces the difficulty of integrating with existing workflow.
Task as classification, regression, clustering and dimensionality reduction can be performed with the algorithms provided by this library.
Moreover, there are available also pre-preprocessing and feature selection tools which are crucial steps in any machine learning pipeline.
Some of the popular algorithms available in Scikit-learn include linear regression, decision trees, random forests, k-nearest neighbours and support vector machines.
Scikit-learn is open-source and free to use, which has contributed to its popularity among data scientists, researchers, and developers.
Its ease of use, extensive documentation, and active community make it an excellent tool for beginners and experienced practitioners alike.
Thanks to the extended documentation, numerous examples, and an active community, this library is also an excellent tool for beginners.
Figure B.2 shows the Scikit-learn library logo.

Figure B.2: Scikit-Learn logo

B.1.3 Tensorflow and Keras

Tensorflow [85] and *Keras* [58] are two popular Python libraries used for developing and training (deep) neural networks.
Tensorflow, which was developed by Google Brain, is an open-source framework that provides a low-level interface for building and training custom neural networks.
Keras, acts as an interface for the Tensorflow library and provides a high-level neural network API that runs on top of Tensorflow.
Thanks to a simplified interface, it makes it easier to get started with deep learning. Keras allows defining each layer of the neural network, the activation functions, optimizers, hyperparameters and so on.
Both libraries can be used for performing a wide range of tasks, from computer vision to speech recognition. Moreover, they support highly scalability to train models on large datasets.
They are open-source and free to use and become one of the best alternatives for deep learning model development.
Tensorflow logo is visible in figure B.3, Keras logo in figure B.4

Figure B.3: Tensorflow Logo

Figure B.4: Keras Logo

B.1.4 Hugging Face

In the field of Natural Language Processing, *Hugging Face* [49] is a popular open-source library that offers a wide range of tools and pre-trained models for numerous NLP tasks such as text classification, language generation and so on.
The library makes available implementations of transformers models which are compatible with the most popular libraries for deep learning including Tensorflow and PyTorch.
Furthermore, it contains pre-trained models which are state-of-the-art in various NLP task. These models can be fine-tuned in a simple way thanks also to the simplicity of the API provided.
In addition to that, Hugging Face Inc, which developed the library, provides other extensions for other useful tasks such as dataset preprocessing and model evaluation.
Figure B.5 shows the HuggingFace logo.

Figure B.5: HuggingFace logo

B.1.5 Tornado

A popular open-source web framework and asynchronous network library is Tornado [87]. It is used for building high-perfomance, non-blocking web applications.
Tornado give the possibility to develop the REST API since is designed to handle a large number of simultaneous connection, thus it is suitable for real-time web services, chat applications, and other network-intensive applications.
Figure B.6 is visible the Tornado logo.

Figure B.6: Tornado logo

Bibliography

Reference list

[3] Open AI. *Language Models are Few-Shot Learners*. 2020. URL: https://arxiv.org/abs/2005.14165 (cit. on p. 99).

[6] Matteo Marcuzzo Andrea Gasparetto Alessandro Zangari and Andrea Albarelli. *A survey on text classification: Practical perspectives on the Italian language*. 2022. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0270904.

[9] Márcio Guia - Rodrigo Rocha Silva - Jorge Bernardino. *Comparison of Naïve Bayes, Support Vector Machine, Decision Trees and Random Forest on Sentiment Analysis*. 2019. URL: https://www.scitepress.org/Link.aspx?doi=10.5220/0008364105250531 (cit. on p. 68).

[16] Julia Rayz Byungkyu Yoo. *Understanding Emojis for Sentiment Analysis*. 2018. URL: https://journals.flvc.org/FLAIRS/article/view/128562 (cit. on p. 44).

[17] Wei Li - Wei Shao - Shaoxiong Ji - Erik Cambria. *BiERU: Bidirectional Emotional Recurrent Unit for Conversational Sentiment Analysis*. 2021. URL: https://arxiv.org/abs/2006.00492 (cit. on p. 99).

[20] Matteo Cinelli et al. *Online Hate: Behavioural Dynamics and Relationship with Misinformation*. 2021 (cit. on p. 50).

[22] Diogo Cortiz. *Exploring Transformers in Emotion Recognition: a comparison of BERT,DistillBERT, RoBERTa, XLNet and ELECTRA*. Vol. abs/2104.02041. 2021. URL: https://arxiv.org/abs/2104.02041 (cit. on p. 21).

[25] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. URL: https://arxiv.org/abs/1810.04805 (cit. on pp. 18, 19, 81).

[36] Gagliardi Gloria. *Inter-annotator agreement in linguistica: una rassegna critica*. 2018. URL: https://hdl.handle.net/11585/661797 (cit. on p. 30).

[39] Margherita Grandini, Enrico Bagli, and Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. 2020. DOI: 10.48550/ARXIV.2008.05756. URL: https://arxiv.org/abs/2008.05756 (cit. on pp. 23–25).

[40] Liyuan Liu - Haoming Jiang - Pengcheng He - Weizhu Chen -0 Xiaodong Liu - Jianfeng Gao - Jiawei Han. *On the Variance of the Adaptive Learning Rate and Beyond*. 2019. URL: https://arxiv.org/abs/1908.03265v1 (cit. on p. 89).

[42] Harith Alani Hassan Saif Miriam Fernandez. *On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter*. 2014. URL: http://oro.open.ac.uk/40666/ (cit. on p. 47).

[46] Chi Sun - Xipeng Qiu - Yige Xu - Xuanjing Huang. *How to Fine-Tune BERT for Text Classification?* 2019. URL: https://arxiv.org/abs/1905.05583 (cit. on pp. 73, 81, 82, 86).

[53] Mohammad Taher Pilehvar Jose Camacho-Collados. *On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis*. 2018. URL: https://arxiv.org/abs/1707.01780 (cit. on p. 10).

[55] Sara Lana-Serrano Julio Villena-Román Sonia Collada-Pérez. *Hybrid Approach Combining Machine Learning and a Rule-Based Expert System for Text Categorization*. 2011. URL: https://www.researchgate.net/publication/221438964_Hybrid_Approach_Combining_Machine_Learning_and_a_Rule-Based_Expert_System_for_Text_Categorization (cit. on p. 10).

[60] Kowsari et al. *Text Classification Algorithms: A Survey*. 2020. URL: https://arxiv.org/abs/1904.08067 (cit. on pp. 10, 12).

[63] Lucian Vlad Lita, Abraham P Ittycheriah, Salim Roukos, Nanda Kambhatla. *tRuEcasIng*. 2003. URL: https://www.semanticscholar.org/paper/tRuEcasIng-Lita-Ittycheriah/2ce1ac4094b3cbedf34d9b8f3cfe355d1a5ae497 (cit. on p. 11).

[65] Mansooreh Karami, Ahmadreza Mosallanezhad, Michelle V Mancenido, Huan Liu. *"Let's Eat Grandma": Does Punctuation Matter in Sentence Representation?* 2020. URL: https://arxiv.org/abs/2101.03029 (cit. on p. 11).

[68] Kishor Datta Gupta Md Mamimur Islam Subash Poudyal. *MAPREDUCE IMPLEMENTATION FOR MALICIOUS WEBSITES CLASSIFICATION*. 2019. URL: https://www.researchgate.net/publication/336146056_Map_Reduce_Implementation_for_Malicious_Websites_Classification (cit. on p. 9).

[69] Mohamed Morchid. *Neural Network Methods in Natural Language Processing*. 2017, p. 128 (cit. on p. 15).

[79] Diksha Khurana - Aditya Koli - Kiran Khatter - Sukhdev Singh. *Natural language processing: state of the art, current trends and challenges*. 2022. URL: https://doi.org/10.1007/s11042-022-13428-4 (cit. on p. 5).

[82] Alec Radford - Jeffrey Wu - Rewon Child - David Luan - Dario Amodei - Ilya Sutskever. *Language Models are Unsupervised Multitask Learners*. 2019. URL: https://paperswithcode.com/paper/language-models-are-unsupervised-multitask (cit. on p. 99).

[83] Fabio Tamburini. *How "BERTology" Changed the State-of-the-Art also for Italian NLP*. 2020 (cit. on p. 73).

[92] Ashish Vaswani et al. *Attention is All you Need*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. 17, 18).

[93] Latha Parameswaran Vikas K Vijayan Bindu K.R. *A Comprehensive Study of Text Classification Algorithms*. 2010. URL: https://ieeexplore.ieee.org/document/8125990 (cit. on p. 8).

[94] Ledell Wu - Adam Fisch - Sumit Chopra - Keith Adams - Antoine Bordes - Jason Weston. *StarSpace: Embed All The Things!* 2017 (cit. on p. 71).

[99] Min Lin - Qiang Chen - Shuicheng Yan. *Network In Network*. 2013. URL: https://arxiv.org/abs/1312.4400v3 (cit. on p. 83).

[100] Rèjan Ducharme Yoshua Bengio, Pascal Ducharme, and Christian Jauvin. *A Neural Probabilistic Language Model*. 2003. URL: https://www.researchgate.

net/publication/2413241_A_Neural_Probabilistic_Language_Model (cit. on p. 15).

Websites

[1] *[NLP] TF-IDF (Term Frequency - Inverse Document Frequency)*. URL: https://rfriend.tistory.com/745 (cit. on p. 13).

[2] *A Beginner's Guide to Understanding and Building Docker Images*. URL: https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/ (cit. on p. 102).

[4] Jay Alammar. *A Visual Guide to Using BERT for the First Time*. URL: http://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/ (cit. on p. 74).

[5] *AlBERTo*. URL: https://github.com/marcopoli/AlBERTo-it (cit. on pp. 21, 64).

[7] Nilesh Barla. *The Ultimate Guide to Word Embeddings*. URL: https://neptune.ai/blog/word-embeddings-guide (cit. on p. 15).

[8] *Baseline Models*. URL: https://towardsdatascience.com/baseline-models-your-guide-for-model-building-1ec3aa244b8d (cit. on p. 68).

[10] *BERT Word Embeddings Tutorial*. URL: https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/#23-segment-id (cit. on p. 65).

[11] Aline Bessa. *Lexicon-Based Sentiment Analysis: A Tutorial*. URL: https://www.knime.com/blog/lexicon-based-sentiment-analysis (cit. on p. 91).

[12] *Better language models and their implications*. URL: https://openai.com/research/better-language-models (cit. on p. 22).

[13] Jason Brownlee. *A Gentle Introduction to the Bag-of-Words Model*. URL: https://machinelearningmastery.com/gentle-introduction-bag-words-model/ (cit. on p. 10).

[14] Jason Brownlee. *A Gentle Introduction to the Bag-of-Words Model*. URL: https://machinelearningmastery.com/gentle-introduction-bag-words-model/ (cit. on p. 12).

[15] Jason Brownlee. *Random Oversampling and Undersampling for Imbalanced Classification*. URL: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/ (cit. on p. 48).

[18] Riccardo Cantini. *Play with BERT! Text classification using Huggingface and Tensorflow*. URL: https://riccardo-cantini.netlify.app/post/bert_text_classification/ (cit. on p. 83).

[19] Fabio Chiusano. *Two minutes NLP — Quick intro to Question Answering*. URL: https://medium.com/nlplanet/two-minutes-nlp-quick-intro-to-question-answering-124a0930577c (cit. on p. 7).

[21] Codecademy. *Text Preprocessing*. URL: https://www.codecademy.com/learn/dsnlp-text-preprocessing/modules/nlp-text-preprocessing/cheatsheet (cit. on p. 11).

[23] *Datasets and Complete Guide to the Keras Dataset*. URL: https://wikidocs.net/178811 (cit. on p. 23).

[24] *DBMZ BERT models*. URL: https://github.com/dbmdz/berts (cit. on p. 21).

[26] *Docker*. URL: https://www.docker.com/.

[27] *Docker Compose Overview*. URL: https://docs.docker.com/compose/ (cit. on p. 104).

[28] *ELECTRA*. URL: https://huggingface.co/docs/transformers/model_doc/electra (cit. on p. 21).
[29] *Embedding Layer Keras*. URL: https://keras.io/api/layers/core_layers/embedding/ (cit. on p. 64).
[30] *Emoji*. URL: https://github.com/carpedm20/emoji (cit. on p. 44).
[31] *FEEL-IT*. URL: https://github.com/MilaNLProc/feel-it (cit. on p. 57).
[32] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (cit. on p. 69).
[33] Kavita Ganesan. *What are Stop Words?* URL: https://kavita-ganesan.com/what-are-stop-words/ (cit. on p. 11).
[34] *GePpeTto*. URL: https://github.com/LoreDema/GePpeTto (cit. on p. 21).
[35] *GilBERTo*. URL: https://github.com/idb-ita/GilBERTo (cit. on pp. 21, 73).
[37] *GloVe: Global Vectors for Word Representation*. URL: https://nlp.stanford.edu/projects/glove/ (cit. on p. 15).
[38] *Google Colaboratory*. URL: https://colab.research.google.com/ (cit. on p. 74).
[41] Atul Harsha. *Understanding Part-of-Speech Tagging in NLP: Techniques and Applications*. URL: https://www.naukri.com/learning/articles/pos-tagging-in-nlp/#6 (cit. on p. 7).
[43] *How Does BERT NLP Optimization Model Work?* URL: https://www.turing.com/kb/how-bert-nlp-optimization-model-works#disadvantages-of-the-bert-language-model (cit. on p. 20).
[44] *How to Create Naive Bayes Document Classification in Python?* URL: https://www.turing.com/kb/document-classification-using-naive-bayes (cit. on p. 69).
[45] *How to Train and Test Data Like a Pro*. URL: https://sdsclub.com/how-to-train-and-test-data-like-a-pro/ (cit. on p. 23).
[47] *Hugging Face Documentation - Model Outputs*. URL: https://huggingface.co/docs/transformers/main_classes/output (cit. on p. 80).
[48] *Hugging Face Documentation - Model Outputs Warning*. URL: https://huggingface.co/docs/transformers/main_classes/output#transformers.modeling_tf_outputs.TFBaseModelOutputWithPoolingAndCrossAttentions (cit. on p. 80).
[49] *Hugging Face Library*. URL: https://huggingface.co/ (cit. on p. 113).
[50] Purva Huilgol. *Quick Introduction to Bag-of-Words (BoW) and TF-IDF for Creating Features from Text*. URL: https://www.overleaf.com/project/63c017e7b6a34d0eff7f83d8 (cit. on p. 13).
[51] *Imblearn library*. URL: https://imbalanced-learn.org/stable/ (cit. on p. 48).
[52] *itWaC: Italian corpus from the .it domain*. URL: https://www.sketchengine.eu/itwac-italian-corpus/ (cit. on p. 22).
[54] Prateek Joshi. *How do Transformers Work in NLP? A Guide to the Latest State-of-the-Art Models*. URL: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/ (cit. on p. 18).
[56] *Kaggle*. URL: https://www.kaggle.com/.

[57] Sergios Karagiannakos. *Regularization techniques for training deep neural networks*. 2021. URL: https://theaisummer.com/regularization/ (cit. on p. 89).
[58] *Keras Library*. URL: https://keras.io/ (cit. on p. 113).
[59] Chetna Khanna. *Text pre-processing: Stop words removal using different libraries*. URL: https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a (cit. on p. 11).
[61] *Kubernetes*. URL: https://kubernetes.io/.
[62] Ajitesh Kumar. *Difference: Binary, Multiclass and Multi-label Classification*. URL: https://vitalflux.com/difference-binary-multi-class-multi-label-classification/ (cit. on p. 8).
[64] Ben Lutkevich. *BERT language model*. URL: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model (cit. on p. 17).
[67] Matthew Mayo. *Approaches to Text Summarization: An Overview*. URL: https://www.kdnuggets.com/2019/01/approaches-text-summarization-overview.html (cit. on p. 6).
[70] *OPUS (Open Super-large Crawled Aggregated coRpus)*. URL: https://oscar-project.org/ (cit. on p. 22).
[71] *OPUS Corpura Collection*. URL: https://opus.nlpl.eu/ (cit. on p. 22).
[72] Aravindpai Pai. *What is Tokenization in NLP?* URL: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/ (cit. on p. 11).
[73] *Pandas Library*. URL: https://pandas.pydata.org/ (cit. on p. 111).
[74] *PyCharm*. URL: https://www.jetbrains.com/pycharm/.
[75] Anshul Saini. *Decision Tree Algorithm – A Complete Guide*. URL: https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/ (cit. on p. 70).
[76] *Scikit-Learn Library*. URL: https://scikit-learn.org/stable/ (cit. on p. 112).
[77] *SENTIPOLC*. URL: http://www.di.unito.it/~tutreeb/sentipolc-evalita16/ (cit. on p. 53).
[78] Shrivarsheni. *Text Summarization Approaches for NLP – Practical Guide with Generative Examples*. URL: https://www.machinelearningplus.com/nlp/text-summarization-approaches-nlp-example/ (cit. on p. 6).
[80] Anuj Gupta Sowmya Vajjala Bodhisattwa Majumder. *Practical Natural Language Processing*. URL: https://www.oreilly.com/library/view/practical-natural-language/9781492054047/ch04.html (cit. on p. 6).
[81] Beata Stefanowicz. *16 Essential Benefits of Chatbots [+ Challenges in 2023]*. URL: https://www.tidio.com/blog/benefits-of-chatbots/ (cit. on p. 1).
[84] *TartuNLP Truecaser*. URL: https://github.com/TartuNLP/truecaser (cit. on p. 45).
[85] *Tensorflow Library*. URL: https://www.tensorflow.org/?hl=it (cit. on p. 113).
[86] *Text Classification: What it is And Why it Matters*. URL: https://monkeylearn.com/text-classification/ (cit. on pp. 8, 9).
[87] *Tornado*. URL: https://www.tornadoweb.org/en/stable/ (cit. on p. 114).
[88] *UmBERTo*. URL: https://github.com/musixmatchresearch/umberto (cit. on p. 21).
[89] *Understand and manage Docker container volumes (IT)*. URL: https://www.ionos.it/digitalguide/server/know-how/volumi-dei-container-docker/ (cit. on p. 104).

[90] *Understanding TF-IDF for Machine Learning*. URL: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/ (cit. on p. 14).

[91] *Unit Testing Tutorial – What is, Types Test Example*. URL: https://www.guru99.com/unit-testing-guide.html (cit. on p. 108).

[95] *What Are Word Embeddings for Text?* URL: https://machinelearningmastery.com/what-are-word-embeddings/ (cit. on p. 15).

[96] *What is a REST API*. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (cit. on p. 102).

[97] *What is Kubernetes?* URL: https://kubernetes.io/it/docs/concepts/overview/what-is-kubernetes/ (cit. on pp. 108, 109).

[98] *Word Embeddings - Papers With Code*. URL: https://paperswithcode.com/task/word-embeddings/latest (cit. on p. 15).

[101] Victor Zhou. *A Simple Explanation of the Bag-of-Words Model*. URL: https://towardsdatascience.com/a-simple-explanation-of-the-bag-of-words-model-b88fc4f4971 (cit. on p. 12).