Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in ICT for internet and multimedia

Leveraging AI models for Ticket Classification and Language Detection: A Data Analysis Approach to Improve Service Quality and Customer Experience

Master Candidate

Francesco Fregona

Student ID 1237331

Supervisor

Prof. Michele Rossi

University of Padova

Co-supervisor

Dott. Davide Bastianetto

PAT s.r.l.

Date
03/07/2023

Academic Year
2022/2023

...if I'm going to fall, I don't want to fall back on anything, I want to fall forward. Every graduate has the training and the talent to succeed, but do you have the guts to fail? If you don't fail, you are not even trying, to get something you've never had, you have to do something you never did. Not only take risks, but to be open to life, to accept new views and to be open to new opinions, even though you're scared stiff. It may be frightening, it will also be rewarding. Because the chances you take, the people you meet, the people you love, the faith that you have, that's what's going to define your life. If you fall, remember this, fall forward.

— D. WASHINGTON

Abstract

This thesis focuses on language detection and text classification for efficient language processing tasks. The aim is to develop robust models that can accurately identify the language of a given text and classify it into relevant categories. The research explores different approaches and techniques to achieve optimal performance in both language detection and text classification. For language detection, a comparative analysis of various models, including FastText [36], Spacy[17], and Cybozu [41], is conducted. The results show that the fastest model performs exceptionally well, especially when combined with a hybrid solution. The hybrid approach leverages the model's predictions only when the text contains more than 15 characters, below that threshold a dictionary check is performed. Regarding text classification, the study delves into the development of a reliable model. The thesis includes the implementation and the fine-tune of several models, including baseline models and more complex one like SVM, LSTM [16], StartSpace[53] and BERT [9]. This thesis propose an SVM-based hybrid model. The hybrid model incorporates user feedback in the classification process, particularly when the model's confidence level falls below 50%. This user-driven approach enhances the classification accuracy and provides flexibility in scenarios where the model's confidence may be lower. The experiments and evaluations conducted demonstrate the effectiveness of the proposed solutions in language detection and text classification tasks. The achieved results highlight the practical applicability and performance improvements obtained through the hybrid approaches. The research contributes to the advancement of language processing techniques and provides valuable insights for future developments in the field.

Contents

List of Figures **xi**

List of Tables **xiii**

List of Acronyms **xix**

1 Introduction **1**

1.1 PAT s.r.l. since 1992 1

1.1.1 Overview of the Main Software Platform 2

1.1.2 Zucchetti partner - AI Factory Laboratory 2

1.2 Thesis goals: Enhancing Customer Experience through AI implementations 3

1.2.1 Language detection 3

1.2.2 Ticket classification 4

1.3 Dataset structure 4

1.3.1 Gaming Company Chat Dataset 5

1.3.2 Energy Company Ticket Dataset 5

1.3.3 Papaluca dataset 6

1.3.4 Language Detection dataset 7

1.4 Thesis structure 7

2 Natural language processing **11**

2.1 Why NLP in a buisness scenario? 11

2.1.1 Customer service and process automation 12

2.1.2 Manage big amount of data 12

2.2 Common NLP task 13

2.2.1 Text classification 14

2.2.2 Language modeling 14

2.2.3 Name entity recognition 14
2.2.4 Sentiment analysis 15
2.3 Focus on Text classification: A Comparative Study of Algorithms and approaches 15
2.3.1 Text representation techniques 16
2.3.2 Main approaches for text classification 19
2.4 BERT 21
2.4.1 How BERT works 21
2.4.2 Pre-trained models 26
2.4.3 Fine-Tuning 27
2.4.4 Limitation and challenging in using BERT 27
2.5 Evaluation metrics 28

3 Main tool and libraries **31**
3.1 Programming languages 31
3.2 Main Python libraries 32
3.2.1 Baseline: Pandas, Numpy, Matplotlib 32
3.2.2 Nltk: Natural Language Toolkit and Scikit-learn 34
3.2.3 Spacy, langdetect and FastText 34
3.2.4 Tensorflow and Transformer 35
3.3 Tools 36
3.3.1 JetBrains: PyCharm and IntelliJ 36
3.3.2 Colab/Kaggle 37
3.3.3 Docker and Kubernetes 38

4 Analysis of Ticket Misclassification and its Impact on Business Productivity: A Case Study of PAT srl for Implementing an AI Model for Process Optimization **41**
4.1 Focus on Productive Time: An Empirical Evaluation of Performance and Efficiency 42
4.1.1 Comparison between correct and wrong classification . . . 42
4.1.2 Forwards and waste of time due to forwards 45

5 Data Exploration and Preprocessing for Improving the Performance of the AI Model **49**
5.1 Language detection: Chat bot dataset analysis 49
5.1.1 Preliminary text analysis 50

5.1.2 Text empirical inspection 51
5.1.3 Chat concatenation 52
5.2 Ticket classification: Trouble ticketing dataset 53
5.2.1 Dataset overview 54
5.2.2 One hot encoding 56
5.2.3 Text preprocessing 57
5.2.4 Word analysis 59
5.2.5 Dealing with Class Imbalance 60
5.3 Train, validation and Test split 63

6 Language detection models evaluation: FastText, Spacy, Cybozu 65
6.1 Models overview 65
6.1.1 N-gram approach 65
6.1.2 Neural network based 66
6.2 Model evaluation 67
6.2.1 Test results 67
6.3 Hybrid solution 69
6.3.1 Reduce the number of languages 69
6.3.2 Configuration utils 69
6.3.3 Postprocessing: Hybrid Choice 70
6.3.4 Choice of char threshold 70
6.4 Final results 72

7 Ticket classification model development and performance evaluation 75
7.1 Baseline models and SVM 76
7.1.1 Naive Bayes 76
7.1.2 Logistic regression 77
7.1.3 Support Vector Machine 78
7.1.4 First consideration after the baseline models and SVM implementation 80
7.2 Refining Analysis and Preprocessing for Improved Model Performance 81
7.2.1 Model robustness on different ticket threshold 81
7.2.2 Final threshold choice: 25 tickets 82
7.2.3 Concatenation of the Subject field 83
7.3 Long Short Term Frequency Memory 84

7.3.1 LSTM network architecture . . . 85
7.3.2 LSTM test results . . . 86
7.4 StarSpace . . . 88
7.4.1 StarSpace test results . . . 88
7.5 BERT Fine Tuning . . . 88
7.5.1 BERT based Network Architecture . . . 89
7.5.2 BERT test results . . . 91
7.6 Test Markov chain . . . 92
7.7 Ensemble techniques . . . 96
7.7.1 Bootstrap bagging . . . 97
7.7.2 Bootstrap test results . . . 97
7.8 Memory requirements and resource usage consideration . . . 99
7.9 Hybrid solution:Keep user choice . . . 100
7.9.1 Grid search probability threshold . . . 100
7.9.2 Results . . . 103
7.10 Final results comparison . . . 104

8 From Development to Deployment: Implementing an AI Model into Production Environment 109
8.1 API implementation . . . 109
8.2 Docker's Utility . . . 110
8.3 Unit Test and Integration with CX Studio . . . 112
8.4 Kubernetes . . . 112

9 Conclusions 115
9.1 Practical Implications . . . 116
9.2 Future works . . . 117
9.3 Personal Growth and Accomplishments . . . 118

A Appendix: Detailed Model Performance Metrics 121
A.1 User . . . 121
A.2 SVM . . . 126
A.3 LSTM . . . 130
A.4 StarSpace . . . 134
A.5 BERT: Italian based XXL . . . 138
A.6 GilBERTo . . . 142
A.7 Hybrid System - SVM based . . . 146

References **151**

Ringraziamenti **155**

List of Figures

1.1 Thesis structure. 9

2.1 An histogram that represent the growth of the NLP market. . . . 12
2.2 Text classification process. 14
2.3 NER process. 15
2.4 Sentiment analysis process. 15
2.5 TD-IDF example. 18
2.6 Optimal Hyperplane of SVM. 20
2.7 An RNN architecture. 22
2.8 An simple example of self-attention mechanism. 23
2.9 The Transformer - model architecture (from original paper). . . . 24
2.10 BERT input representation (from original paper). 25
2.11 Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning (from original paper). 26
2.12 A graphical representation of precision, recall and F1-score. . . . 29

3.1 Programming languages logos. 32
3.2 Pandas logo. 33
3.3 Numpy logo. 33
3.4 Matplotlib logo. 34
3.5 Scikit-learn logo. 34
3.6 Principal libraries for language detection. 35
3.7 Main tools for dealing with NLP and machine learning. 36
3.8 Commonly used IDEs for development. 37
3.9 Main cloud-based environment to develop AI models and more. . 37
3.10 Main tools for dealing with containers and images side by side. . 39

4.1 Resolution Time distribution. 43
4.2 Graphical representation of the comparison between resolution time of correct/wrong classification. 45
4.3 Graphical representation of the comparison between number of forward of correct/wrong classification. 47

5.1 Question length distribution. 50
5.2 Question length distribution, with concatenated *Question* grouped by *Chat Session*. 52
5.3 Choice of language to compare with the result of the chat concatenation due to the probability distribution. 53
5.4 First fifty Application_FINAL distribution. 56
5.5 Request length distribution. 57
5.6 Top 20 tokens frequency in the dataset without preprocessing. . . 59
5.7 Top 20 tokens frequency in the dataset preprocessed. 60
5.8 Markov Chains definition. 61

6.1 Hybrid approach diagram. 71
6.2 Grid-search of the best chat threshold. 72

7.1 LSTM model training/validation loss over epochs. 87
7.2 LSTM model training/validation accuracy over epochs. 87
7.3 Illustration of Bootstrap bagging technique. 98
7.4 Grid search for probability threshold, the F1 score is related to the weighted average. 102
7.5 Comparison of F1-score on single classes between Hybrid system and StarSpace. 107
7.6 Comparison of F1-score on single classes between Hybrid system and BERT. 107
7.7 Comparison of F1-score on single classes between Hybrid system and SVM. 108
7.8 Comparison of F1-score on single classes between Hybrid system and User. 108

8.1 Docker containers. 111
8.2 Docker containers. 113

List of Tables

1.1 Dataset details. 7

2.1 Natural Language Processing tasks. 13
2.2 Bag of words example. 17
2.3 TF-IDF example. 18
2.4 Some of pre-trained models that could be used for fine-tuning. . . 27

4.1 The first 15th classes in which is underlined a loss in terms of time between tickets with correct classification and tickets misclassified. 44
4.2 Applications in which is underlined that the mean number of forwards of tickets with correct classification is lower than tickets misclassified. 46

5.1 Some example of user questions. 51
5.2 Some useful details about the Energy Company dataset. 54
5.3 First fifty Application_FINAL details. 55
5.4 One-hot encoding example. 57
5.5 Example of lemmatization process. 58
5.6 Vocabulary size after preprocessing. 60
5.7 Some original ticket from the Energy company dataset. 62
5.8 Some examples of synthetic tickets generated with Markov Chains. 63

6.1 1-gram and 2-gram approach applied to the words sequence. . . . 66
6.2 Details on pretrained models used for the language detector. . . . 67
6.3 Accuracy on the test dataset. 67
6.4 Correspondence on the internal gaming dataset using the the output of the language detected with the entire conversation as input. 68

6.5 Correspondence on the internal gaming dataset using the personalized method in figure 5.3. 68
6.6 Accuracy on the test dataset using the hybrid approach. 72
6.7 Correspondence with hybrid approach on the internal gaming dataset using the the output of the language detected with the entire conversation as input. 73
6.8 Correspondence with hybrid approach on the internal gaming dataset using the personalized method in figure 5.3. 73

7.1 Dataset splitting details for baseline models. 76
7.2 Partial class distribution of classes used to show results of the first part of the model evaluation that regards the baseline models and the SVM. 77
7.3 Naive Bayes classification results. 77
7.4 Naive Bayes single classes results: In the first half the results of the most populated classes, in the second half minority classes (Support field refering to test set). 78
7.5 Logistic regression classification results. 78
7.6 Logistic Regression single classes results: In the first half the results of the most populated classes, in the second half minority classes. 79
7.7 SVM classification results using liner kernel. 80
7.8 SVM classification results using polynomial kernel. 80
7.9 SVM classification results using Radial Basis Function kernel. . . 80
7.10 Support Vector Machine single classes results (linear kernel): In the first half the results of the most populated classes, in the second half minority classes. 81
7.11 Dimension of different dataset depending on ticket threshold. . . 82
7.12 Results of ticket threshold grid-search. 83
7.13 Some examples of uninformative request. 84
7.14 Results of SVM by concatenating the Subject and Request fields. . 84
7.15 Dataset splitting details after refining the analysis and preprocessing. 85
7.16 LSTM model classification results (complete classification report on Appendix A). 86

7.17 StarSpace model classification results (complete classification report on Appendix A). 89
7.18 Pretrained models result using the 4-layer concatenation approach. 92
7.19 Pretrained models result using the pooler output approach. . . . 92
7.20 Dataset details after the addition of synthetic tickets (ticket_threshold = 1000). 93
7.21 Dataset details after the addition of synthetic tickets (ticket_threshold = 400). 94
7.22 Dataset details after the addition of synthetic tickets (ticket_threshold = 200). 94
7.23 Dataset details after the addition of synthetic tickets (ticket_threshold = 50). 95
7.24 Comparison of SVM results using synthetic data. 95
7.25 Comparison of BERT results using synthetic data. 96
7.26 Bootstrap bagging technique classification results compered with a single SVM model trained on the entire dataset. 99
7.27 Comparison between user choice and SVM results. 100
7.28 Support Vector Machine single classes results: In the first half we have the results of the most populated classes, in the second half minority classes, the dataset in exam refers to table 7.15 101
7.29 User manual classification results: In the first half we have the results of the most populated classes, in the second half minority classes (complete classification report on Appendix A). 101
7.30 Weighted average precision, recall, F1 score and accuracy of the hybrid system based on SVM, using different probability threshold.103
7.31 Comparison between user choice, SVM and hybrid approach. . . 104
7.32 Hybrid solution results: In the first half we have the results of the most populated classes, in the second half minority classes (complete classification report on Appendix A). 104
7.33 Final comparison between all the relevant model results. 105
7.34 Comparison between SVM, BERT and StarSpace classification metrics of *HRNext*. 105
7.35 Three tickets of *HRNext*. 106

A.1 The following table provides the classification report based on user choices. It utilizes the same test dataset used for the models. The class labeled as "OTHERS" encompasses categories that are outside the context of the classification. As this classification is performed by a human, there may be instances where misclassifications occur, particularly for classes that were removed during the preprocessing stage. 125
A.2 SVM complete classification report. 129
A.3 LSTM complete classification report. 133
A.4 StarSpace complete classification report. 137
A.5 BERT Italian based XXL pretrained model (pooler output) complete classification report. 141
A.6 GilBERTo (4 layers concatenation) complete classification report. . 145
A.7 Hybrid System SVM based complete classification report. 149

List of Acronyms

ML Machine Learning

DL Deep Learning

AI Artificial Intelligence

HDA Helpdesk Advanced

NLP Natural Language Processing

XNLI Cross-lingual Natural Language Inference

STSb Swiss Transportation Safety Investigation Board

MT Machine Translated

IVR Interactive Voice Response

OCR Optical Character Recognition

BFSI Banking, Financial Services, and Insurance

IT Information Technology

SAP System Application Products

ITES Information Technology Enabled Services

BERT Pre-training of Deep Bidirectional Transformers for Language Understanding

NER Name Entity Recognition

BOW Bag Of Words

TF-IDF Term Frequency-Inverse Document Frequency

SVM Support Vector Machine

LSTM Long Short-Term Memory

NN Neural Network

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

TPR True Positive Rate

FPR False Positive Rate

CSV Comma-Separated Values

NLTK Natural Language Toolkit

IDE Integrated Development Environment

CLI Command Line Interface

GUI Graphical User Interface

CNCF Cloud Native Computing Foundation

RT Resolution Time

RBF Radial Basis Function

GAN Generative Adversarial Networks

1 Introduction

In today's business environment, delivering excellent customer service has become a critical factor for success across all industries. The increasing expectations of customers for personalized and high-quality service necessitate continuous efforts from companies to remain competitive. Within this context, artificial intelligence (AI) and machine learning (ML) play a pivotal role in enhancing customer experience and service quality. ML techniques and advanced engineering tools enable companies to analyze vast amounts of data, gaining valuable insights that can optimize service delivery processes, improve customer interactions, and offer personalized experiences.

The objective of this project is to explore the potential of AI and ML in enhancing customer service and experience within a real-world setting. Specifically, I will develop AI models and assess their utility in automating service tasks, such as language identification in a Chat Bot or automatic ticket classification. Furthermore, I will evaluate the impact of AI-based solutions on service quality, customer satisfaction, and business performance. With the expertise of PAT s.r.l., I will examine the challenges, opportunities, and best practices associated with implementing AI-based customer service solutions.

1.1 PAT s.r.l. since 1992

This master's thesis project is being developed in collaboration with PAT s.r.l., an ICT company that has been operating in the business-to-business sector since 1992. PAT specializes in Service Management, Virtual Assistance, Customer

Relationship Management, Customer Care, and Service Desk. Although the company's headquarters is located in Montebelluna (TV), it has multiple delivery offices in Italy, including Lodi, Milan, Rome, Florence, as well as an international office in Madrid.

PAT's primary objective is to simplify the management of information and relationships across different areas of a company, leading to enhanced business processes, particularly in sectors such as customer service, customer care, service desk, direct interaction, and internal communication.

1.1.1 Overview of the Main Software Platform

1. **Helpdesk Advanced (HDA):** HDA represents a web and mobile-based solution for trouble ticketing that governs services across all organizational areas, recognizing the crucial role of service management in achieving success. This Service Desk solution, known as HDA, has been specifically designed to automate processes and cater to various strategic service governance scenarios through highly configurable IT and business processes. HDA prioritizes an intuitive and user-friendly Service Desk experience, optimizing the User Experience by providing channels and interfaces that are easy to navigate and operate. Additionally, HDA empowers users to generate statistical information in the form of graphs and charts, enabling the monitoring of service efficiency and effectiveness over time.
2. **Engagent & CX Studio:** CX Studio, accessible in both Cloud and local environments, serves as a multi-channel interaction framework that leverages Artificial Intelligence and Machine Learning. This framework incorporates a virtual assistant, enabling real-time interactions 24/7 across various user-preferred channels, including the web, social networks, WhatsApp, Microsoft Teams, and more. The CX Studio framework empowers users to create personalized one-to-one dialogue flows and apply them seamlessly across all channels, guided by principles of proactivity, engagement, and customer involvement. With CX Studio, you gain the ability to efficiently coordinate interactions for different business areas within a single tool.
3. **IC Studio:** ICstudio is a Customer Relationship Management platform dedicated to the needs of different company areas, from the sales force to marketing, from customer care to the call center. ICstudio increases your business by creating winning and lasting relationships with leads, prospects, customers and partners.

1.1.2 Zucchetti partner - AI Factory Laboratory

In June 2013, PAT Group established a partnership with Zucchetti, a prominent software house in Italy known for pioneering payroll processing software.

Over time, Zucchetti has expanded its reach by acquiring several companies that provide diverse solutions in various areas worldwide.

The AI Factory - Innovation Lab is a recent initiative that brings together engineers and developers from PAT and Zucchetti. Its primary objective is to harness the latest advancements in AI and ML technologies and integrate them into PAT solutions, introducing innovative features.

This master's thesis is driven by the AI Factory, receiving continuous updates on the state-of-the-art developments. It includes regular reports on key metrics and achieved results, ensuring that the thesis stays abreast of the latest advancements in the field.

1.2 Thesis goals: Enhancing Customer Experience through AI implementations

The aim of this thesis is to investigate the role of artificial intelligence (AI) in enhancing customer experience within the modern business landscape. Many businesses have turned to AI as a means to improve customer experience, and the results have been promising. Through AI-powered chatbots and virtual assistants, personalized product recommendations, and predictive analytics, businesses are able to provide customers with more efficient and tailored service than ever before.

However, implementing AI technologies can be complex and challenging. To maximize the impact of AI on customer experience, businesses must consider various factors, such as the quality of data, which is a common issue and limitation when implementing AI models.

Furthermore, it is crucial to acknowledge that customers themselves may not fully grasp the potential of AI or be aware of their actual needs. This thesis will analyze the impact of implementing AI models on service quality, highlighting the importance of effectively integrating AI to enhance customer experience.

1.2.1 Language detection

One specific objective of this thesis is to create a language detection model that can be integrated into the Engagent CX Studio solution introduced in the previous paragraph to enhance the customer experience. Chatbots have become

widely used customer service tools, providing customers with a speedy and efficient means of obtaining information. However, a major challenge of chatbots is their ability to understand and respond to queries in various languages.

With this aim, I will develop a language detection model that can accurately identify the language of a customer's message. This will enable the virtual assistant to respond appropriately in the correct language, thereby enhancing the customer experience. Furthermore, this model will eliminate the need for customers to switch languages or for businesses to maintain separate chatbots for each language.

In this thesis, I will test some pre-trained models with the capability to detect different languages, trying to identify the best one suitable for short texts.

1.2.2 Ticket classification

Another goal of this thesis is to design a ticket classification model that can automate the process of categorizing customer support tickets. The data for this model will come from the HDA platform. When customers contact a business through HDA with a question or issue, they have the option to select the topic that is related to the office responsible for handling the ticket and providing support. However, customers often misclassify the tickets, leading to the need for support agents to manually redirect them to the appropriate team or individual for resolution.

This manual categorization process is prone to errors and can be time-consuming, resulting in delays and frustration for customers. To address this challenge, I will develop a ticket classification model that can automatically categorize support tickets based on their content.

To develop this model, I will utilize various natural language processing (NLP) techniques and text classification algorithms. The model will be trained on a large corpus of historical support ticket data, with a particular focus on handling unbalanced datasets.

1.3 Dataset structure

In this section, I will introduce the structure of the datasets that will be used in this project. It is important to note that for the two internal datasets (Gaming and Energy companies), I will not disclose the source due to privacy policy

considerations. Additionally, the names used in the datasets are not related to the actual work environments of the companies.

1.3.1 Gaming Company Chat Dataset

The data that compose this dataset are extrapolated from the actual ChatBot in Engagent Pat solution. Each row represents a single message sent by a customer or the operator's reply. The dataset contains more than 43,321 messages in different languages (refer to Table 1.1). It will be used to test the language detection.

The dataset does not include any information about the language, and therefore, we do not have any labels. In Chapter 5, we will explore how to address this problem.
The most important field are summarize in the following list:

1. **QuestionID:** This is the primary key, a numerical unique identifier associated with a single message request.
2. **Chat Session:**An alphanumeric identifier of the conversation (chat) that identifies the entire chat session between the operator and the customer. A chat session may remain open for more than one day, typically until the problem is resolved.
3. **Who:** This field indicates whether the message is sent by the operator, the user, or the system (in the case of a virtual assistant).
4. **Text:** This field contains the user's request or the system/operator's reply.

1.3.2 Energy Company Ticket Dataset

This dataset consists of 21,570 unique tickets from the HDA solution. Each row represents a single ticket opened by a customer. As shown in Table 1.1, the mean number of characters in this dataset is higher compared to the others due to its nature.
The same ticket may appear multiple times in the dataset (the same TicketID in more than one row) because it tracks the forwarding process. This is the reason why the total number of elements (rows) in the dataset is 38,021. There are several reasons why a ticket can be forwarded, and these reasons will be explained in more detail in the following chapters. The main reason for forwarding a ticket

is when the customer selects the wrong department, and the ticket needs to be redirected to the appropriate one for resolution. The most important fields are summarized in the following list:

1. **TicketID:** This is the primary key, a numerical unique identifier associated with a single ticket.
2. **CreationDate:** This field contains the date and time when a ticket is created.
3. **ClosureDate:** This field contains the date and time when a ticket is closed.
4. **Subject:** This field represents the subject of the ticket.
5. **Request:** This field contains the content text of the ticket, i.e., the actual request.
6. **Application:** This field indicates the application related to the department in which the ticket should be resolved. This field is selected by the customer. Examples of application names include *SAP-DBO*[1], *SIEBEL*, *OIG*, etc.
7. **Application_FINAL:** This field represents the final department in which the ticket is actually resolved. The total number of applications, and hence the hypothetical number of classes that will be identified by the model, is 190.
8. **Action Type:** This field provides descriptive information about the action applied to the ticket. It can have two values: *creation* and *forward*. Tickets that have no forwards will have only one row with the Action Type set to *creation*, indicating that the ticket remained in the same department from creation to closure. Tickets with forwards will have the first row with Action Type set to *creation* and additional rows for each forward, with the Action Type set to *forward*.
9. **DateTime Forward:** This field contains the date and time when the ticket is forwarded.

1.3.3 Papaluca dataset

The Papaluca dataset [34] will be used to evaluate the performance of the language detection model. It was created during The Hugging Face Course Community Event of 2021 and consists of text from different sources, including the multilingual Amazon Reviews Corpus, XNLI, and STSb Multi MT.

[1]SAP stands for System Application and Products, we will see different applications starting with this acronym.

1.3.4 Language Detection dataset

This dataset, available on Kaggle [21] [23], will be used in conjunction with the previous dataset for language detection. The primary source of this dataset is Wikipedia [50]. More details can be found in Table 1.1.

Dataset	**Number of rows**	**Number of languages**	**Mean number of characters**
Gaming Company Chat	43321	10	54
Energy Company	38021	1	500
Papaluca	10000	20	124
Language Detection	10267	17	107

Table 1.1: Dataset details.

1.4 Thesis structure

The structure of this thesis is organized to provide a comprehensive understanding of the research and implementation process. The thesis is divided into several sections, each focusing on specific aspects related to enhancing customer experience through AI implementations in the context of PAT s.r.l.

Chapter 1 provides an introductions to PAT s.r.l., its partnership with Zucchetti and its main software platform. In the introduction the thesis goals are outlined, with a focus on enhancing customer experience through AI implementations. Specifically, two key areas of focus are identified: language detection and ticket classification. In this section is also discussed the dataset structure.

The chapter 2 explores the importance of NLP in a business scenario. Common NLP tasks are introduced, with a focus on text classification presenting a comparative study of algorithms and approaches. It covers text representation techniques and main approaches for text classification. An in-depth exploration of BERT (Bidirectional Encoder Representations from Transformers) is provided, explaining how it works, pre-trained models, fine-tuning, and its limitations and challenges. At the end, evaluation metrics for NLP models are discussed.

The main tools and library used in the entier project are discussed in chapter 3 and than the chapter 4 focuses on a case study of PAT s.r.l., analyzing

ticket misclassification and its impact on business productivity. An empirical evaluation of performance and efficiency, comparing correct and wrong ticket classifications, and thus, the forward and waste of time due to misclassifications.

Chapter 5 concerns data exploration and preprocessing techniques to enhance the performance of both AI models implemented. It includes consideration and some test to deal with class imbalance.

The chapter 6 delves into the implementation of language detection models and present a comprehensive comparison and evaluation of various approaches. Specifically it focuses on three models: FastText, Spacy, and Cybozu. Through this analysis, the aim is to assess the effectiveness and performance of each model in accurately detecting the language of a given text.

Chapter 7 revolves around the development and evaluation of a ticket classification model. It embarks on a systematic and rigorous study to design and implement a model that effectively categorizes tickets based on their content. The chapter encompasses a detailed examination of the model development process and model selection. Various techniques and algorithms are employed to optimize the model's performance and achieve accurate ticket classification. In the final chapters 8 and 9 introduce the deployment part underlining the use of Docker and Kubernetes, the future works and the final conclusion of the thesis.

The Figure 1.1 illustrates the timeline of the development process, highlighting the iterative nature of the analysis, preprocessing, and implementation stages.

The timeline begins with the data sourcing phase, where the relevant data sources are identified and collected. Subsequently, the exploring data analysis (EDA) phase takes place, during which the collected data is thoroughly examined to gain insights and understand its characteristics. This analysis phase helps in making informed decisions about the preprocessing steps and model design.

The preprocessing stage follows the data analysis, where the collected data is cleaned, transformed, and prepared for model training. This step ensures that the data is in a suitable format and quality for effective model development. The iterative nature of the development process allows for revisiting the analysis and preprocessing stages if necessary.

Once the data is preprocessed, the model implementation phase begins. This phase involves selecting an appropriate model architecture, training the model

on the preprocessed data, and fine-tuning it for optimal performance, before going in the last stage, the deployment.

It is important to emphasize that the development process is not strictly linear but rather iterative (dotted lines in the figure). This means that adjustments and refinements in the analysis and preprocessing stages can be made based on the outcomes of subsequent stages.

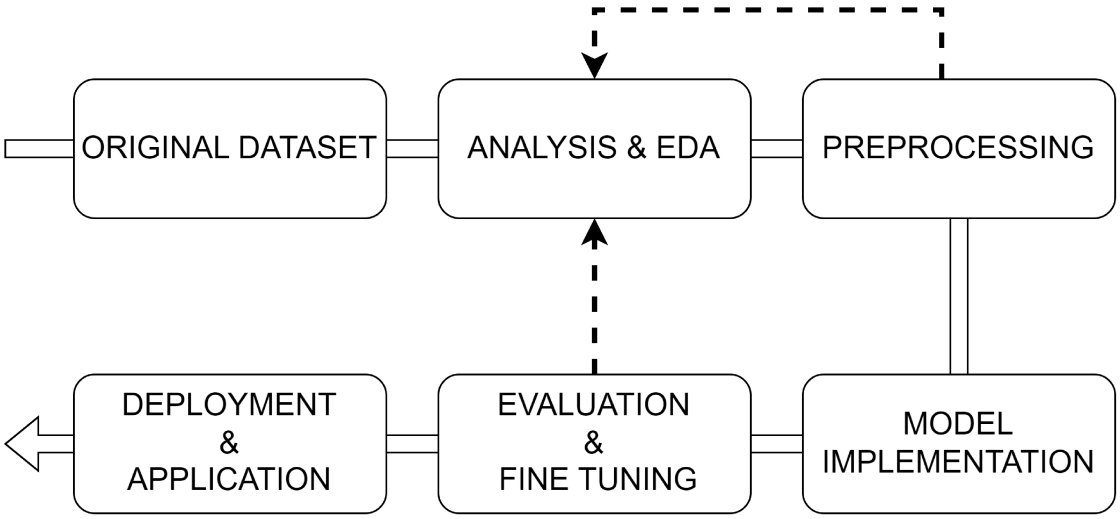


Figure 1.1: Thesis structure.

2

Natural language processing

Natural Language Processing (NLP) is a field of artificial intelligence that aims to enable computers to understand, interpret, and generate human language. It involves the study of how machines can analyze and process human text in order to comprehend and respond in a manner that closely resembles natural language. NLP combines knowledge in different fields, including not only computer science but also linguistics and cognitive psychology.
This field is continuously evolving and rapidly transforming the way we interact with technology, while unlocking new opportunities for innovation across industries.

2.1 Why NLP in a buisness scenario?

There are several reasons why NLP has become one of the most common technological advancements in the global business scenario. Nowadays, more and more companies are implementing this tool as it provides advantages not only regarding customer services but also internal productivity.
Investments in this field are increasing exponentially. According to a 2019 Statista report [44], the NLP market is projected to reach 43.0 billion dollars by 2025, as shown in Figure 2.1.

Actually, observing the trend in the year that this thesis is written, the first part of 2023, we are two years behind. The market size value updated in 2022 is USD 15.7 billion, but latest research confirms the direction that the NLP market will take in the next few years. A MarketsandMarkets study [26] projects that

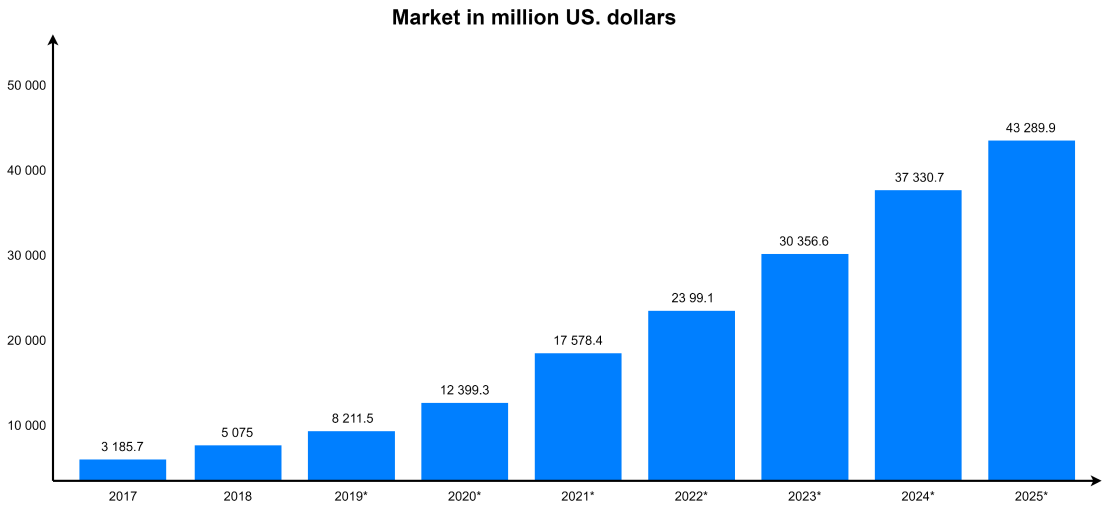


Figure 2.1: An histogram that represent the growth of the NLP market.

the NLP market is poised to reach USD 49.4 billion by 2027.

2.1.1 Customer service and process automation

NLP is used in customer care and provides support for various entities. As previously specified, it is also high-quality internal support because this tool saves considerable time for customer service agents, as I will underline in Chapter 4. Assignments related to understanding, analysis, and responses are all automated, thanks to AI and deep learning, which complement NLP in the best way. Thus, it plays a significant role in process automation. In the business environment, various unstructured data are collected, such as emails, chat conversations, and customer feedback. These data sources are used to trigger automated workflows and business processes. In Chapter 7, I will show an example of a customer service task that will be automated, specifically the ticket classification task, which automates the process of retrieving information by extracting features from text requests and directing tickets to the appropriate department or customer agent.

2.1.2 Manage big amount of data

Companies that operate in the sports betting business can generate more than 10,000 tickets daily, which represents a huge quantity. Managing large amounts of data can be challenging for businesses. However, NLP can help

businesses handle and manage large volumes of data effectively, especially unstructured data, which accounts for 80% of all data generated by businesses. By using NLP techniques, businesses can extract valuable insights and actionable intelligence from data sources, such as customer feedback. NLP can also be used to categorize and tag data, extract relevant keywords, and identify patterns and trends. This enables businesses to make data-driven decisions and improve business processes, even when dealing with large amounts of data, all within a reasonable timeframe.

2.2 Common NLP task

In everyday life, many of us use NLP applications without even realizing it. Spell-checkers, online search engines, translators, and voice assistants like Apple Siri, Amazon Alexa, Samsung Bixby, or Google Assistant all incorporate NLP technology. Table 2.1 illustrates various NLP tasks, and the following sections provide a brief explanation of the most common ones, highlighting their interesting capabilities and potential.

Word Tagging	**Sentence Parsing**	**Text Classification**	**Text Pair Matching**	**Text Generation**
Word segmentation	Constituency analysis	Sentiment analysis	Semantic textual similarity	Language modeling
Shallow syntax chunking	Semantic parsing	Text classification	Natural languange inference	Machine translation
Name entity recognition	Dependency parsing	Temporal processing	Relation prediction	Simplification
Part-of-speech tagging		Coreference resolution		Summarizing
Semantic role labeling				Dialogue
Word sense disambiguation				Question answering

Table 2.1: Natural Language Processing tasks.

2.2.1 Text classification

Text classification is a task that involves assigning tags or labels to text based on its content (see Figure 2.2). This type of classifier has multiple applications, such as structuring or organizing text. I will explore this task in more detail later on.

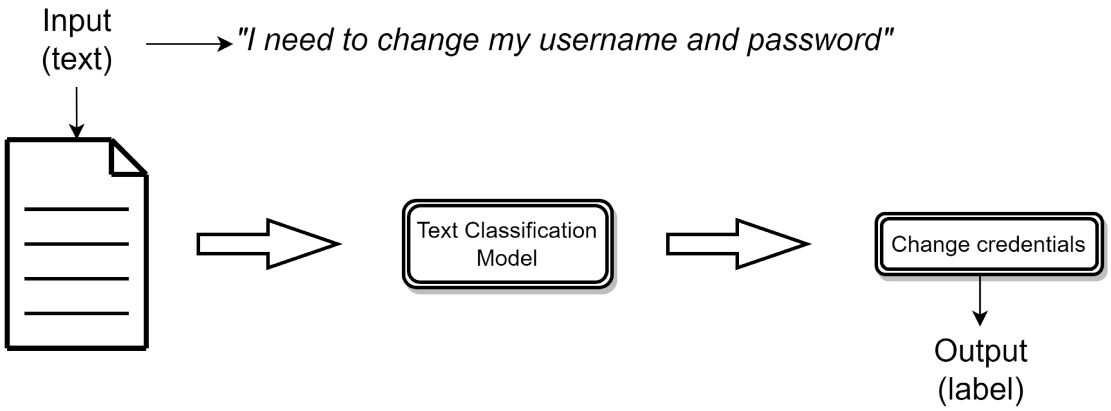


Figure 2.2: Text classification process.

2.2.2 Language modeling

Language modeling is an NLP task that includes next sentence prediction, which we use every day when typing something, such as in the Google search bar. In the next section, I will focus on BERT [9], which has been trained for both next sentence prediction and named entity recognition.

2.2.3 Name entity recognition

Named entity recognition (NER) is a powerful and useful task used to identify entities such as persons, locations, or organizations within the text (see figure 2.3). One of the most important and delicate applications of NER is anonymization. For example, if a bank company desires to obtain statistics or perform analysis on their data, they must ensure the masking of sensitive customer information to prevent disclosure.

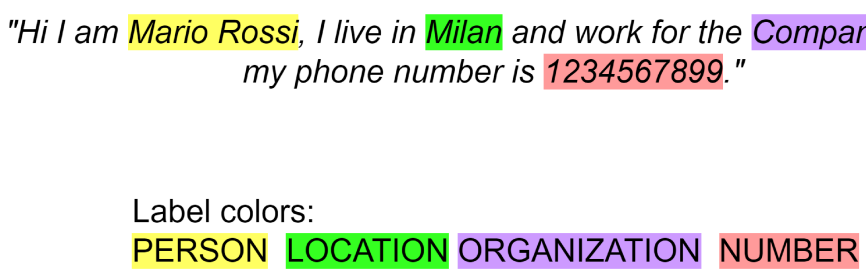


Figure 2.3: NER process.

2.2.4 Sentiment analysis

Sentiment analysis is another common task that involves analyzing a piece of text to determine the underlying sentiment, emotion, or opinion expressed by the author (figure 2.4). The goal of sentiment analysis is to identify whether text is positive, negative. Sentiment analysis is commonly used in business and marketing to evaluate customer satisfaction, track brand reputation, and monitor public opinion on social media.

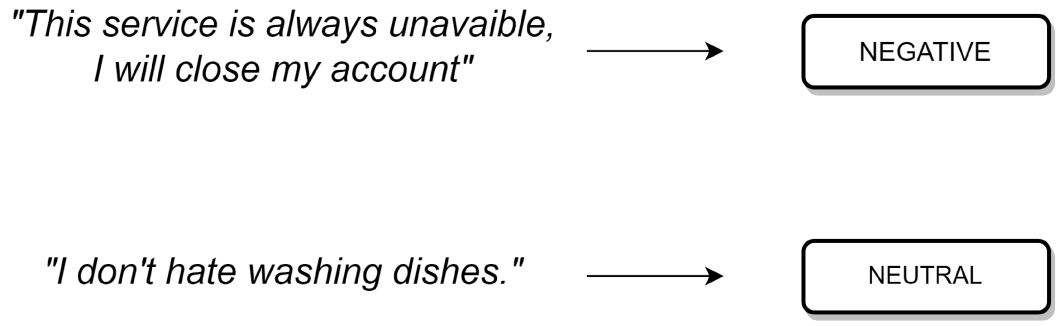


Figure 2.4: Sentiment analysis process.

2.3 Focus on Text classification: A Comparative Study of Algorithms and approaches

Text Classification could be both, supervised and unsupervised learning task, in this specific case it will be treated as the first one since I need to categorize the ticket in the correct department (label), it involves automatically assigning predefined categories or labels to a given text document. This kind of NLP approach is suitable for sentiment analysis, ticket classification, toxic/spam filtering etc. .

Text classification include several challenges, such as handling noisy data, dealing with high-dimensional feature space and identifying relevant features for different categories.
To handle noisy data the role of data analysis and preprocessing is crucial but it is more understandable seen in practise, I will do a deep focus on Chapters 4 and 5, in this section I will focus on the theory of the text representation and I will discover the main machine learning (ML) and deep learning (DL) approaches used to handle with the problem, in a way to have the knowledge base to face the technical details of the implementation in the next chapters.

2.3.1 Text representation techniques

Text representation is a mandatory requirement in NLP, the primary reason why I need it is that the computers cannot understand raw text as is written by humans. Therefore, I need to convert it into a format that the computers can understand and process.
Text representation involves converting data into a numerical form, which can be used as input of ML/DL algorithms. This process is also known as feature extraction or vectorization. The choice of text representation technique is important and can impact significantly the performance of AI models.

Bag of Words (BOW)

The BOW model is a simply commonly used approach to represent a text document as *bag* of its words, the idea behind the procedure is keeping track of the frequency of each word neglecting the grammar.
The first step to follow is creating a dictionary of unique words in the text corpus. For each word in the vocabulary is associated an integer index, then, for each document in the corpus, it's constructed a vector with the same length to the vocabulary size, and the frequency of of each word in the document is placed in the corresponding value of the vector.
The resulting vectors are called *bag-of-words*, an example is shown in table 2.2.

Document	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 with the hat	2	1	0	0	1	1

Table 2.2: Bag of words example.

Term frequency-inverse document frequency (TF-IDF)

TF-IDF is a widely used technique for text representation in NLP. It is based on the statistical distribution of the words in the document, it is a combination of two metrics:

$$\mathrm{TF}(w, d) = \frac{\mathrm{frequency}(w, d)}{max\{\mathrm{frequency}(w, d) : t \in d\}} \tag{2.1}$$

$$\mathrm{IDF}(i) = \log\left(\frac{|D|}{|\{d \in D : w \in d\}|}\right) \tag{2.2}$$

Where the equation 2.1, the term frequency (TF), is the number of times a word appears in the document divided by the total number of words inside the same document, while the equation 2.2, the inverse document (IDF), take track of the rarity, a measure of how important a term is to a document in the corpus, IDF is calculated as the logarithm of the total number of documents in the corpus divided by the number of documents in which the word appears.
The final TF-IDF score is calculated multiplying those two terms together, by this way we are able to identify the most relevant terms in the document. This method is a powerful tool for information retrieval and text classification tasks, as it provides a simple effective way to represent the text putting emphasis on the content.
The figure 2.5 display an example of the TF-IDF score of two words {*the*, *with*}, of the third document in table 2.2, the score referring to the word *the* is equal to 0, this is because the that particular word is present in every document in the corpus, so it gives no relevant information. On the other side, the score of the word *with* is more relevant since it appears in only one document, that's the

reason why its score is 0.22.

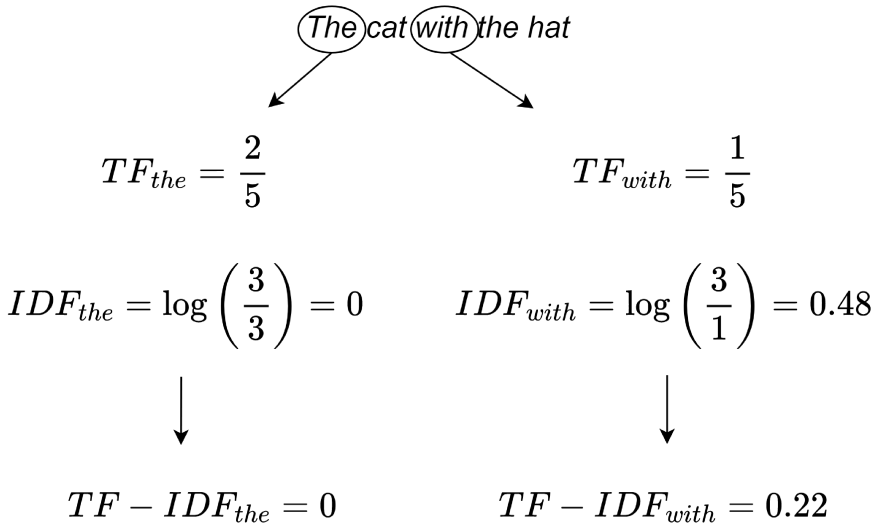


Figure 2.5: TD-IDF example.

In table 2.3 is shown the entire TF-IDF coding.

Document	**the**	**cat**	**sat**	**in**	**hat**	**with**
the cat sat	0	0	0.06	0	0	0
the cat sat in the hat	0	0	0.03	0.08	0.03	0
the cat with the hat	0	0	0	0	0.04	0.2

Table 2.3: TF-IDF example.

Word embedding

Word embeddings are techniques for representing words as vectors in a high-dimensional space. The basic idea is to map words to points in a continuous vector space, where each dimension of the vector represents a different feature or attribute of the word. Word embeddings are typically learned using neural network models, such as Word2Vec [29] or GloVe [35] (unfortunately there are no Italian pre-trained models for them, so they cannot be used in this project), which are trained on large corpora of text data. These models try to predict the context in a given text, by learning a dense, low-dimensional representation of

the words that captures their semantic and syntactic properties. The resulting word embeddings can be used as input to a wide range of NLP tasks, such as text classification, sentiment analysis, and machine translation, and have been shown to outperform traditional bag-of-words models in many cases.
The main characteristic of the words embedding is that they preserve information about the context of the document.

2.3.2 Main approaches for text classification

To solve the task under consideration I will explore several approaches across subfields of artificial intelligence like machine learning and also deep learning that is part of ML itself, DL techniques include the use of neural networks (NNs) and recurrent neural networks (RNNs) [39] which are use to solved more complex tasks, but as I will show they require also more resources, regarding DL, I will focus on it later on a dedicated section for BERT.

Machine learning: Naive Bayes, logistic regression, SVM

Naive Bayes is a common simple probabilistic algorithm used in NLP for text classification. It is based on Bayes Theorem of probability [3] and uses the *naive* assumption of conditionally independence of the features (words) to the class labels. It has been shown to perform well in many text classification tasks [49], but, in the real life the assumption on which this algorithm is based on is almost impossible to adopt.

The **Support Vector Machine** is one of the most used and powerful supervised machine learning algorithm used for text classification tasks. The main idea of SVM is to find the best hyperplane that separates the data into different classes, the goal is to maximize the margin between the hypothetical optimal hyperplane line between the support vectors of different classes, as is shown in figure 2.6.

Logistic regression is a widely used statistical method in the field of NLP for text classification tasks. It is a classification algorithm that is used to model the probability of a certain class or category. It takes a set of input features (such as the presence or absence of certain words in a document) and uses them to predict the probability of a particular class. Logistic regression is a popular choice for text classification due to its simplicity and efficiency.

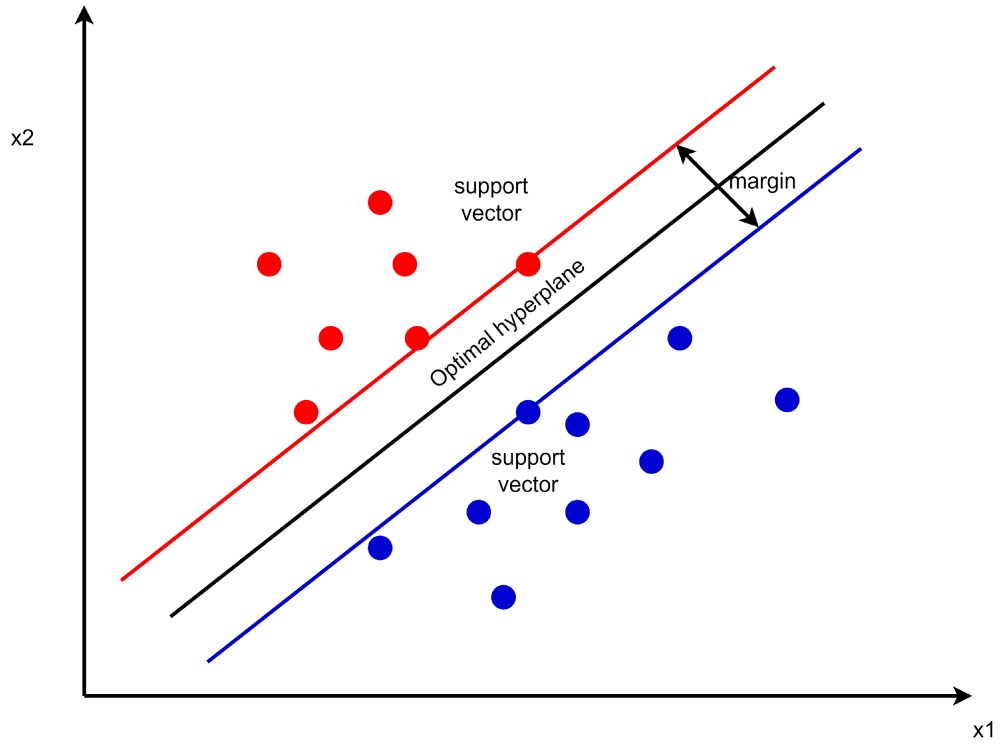


Figure 2.6: Optimal Hyperplane of SVM.

Deep learning: StarSpace, LSTM

StarSpace is a neural network model used for text classification, developed by Facebook researchers [53]. It is based on the idea of learning a low-dimensional embedding of words, phrases, and sentences that preserves semantic similarity. The model uses a multi-task learning objective that combines multiple tasks such as classification, regression, and ranking in a single model. It achieves this by using a shared embedding layer for all tasks and multiple task-specific output layers. The shared embedding layer is trained to minimize the distance between similar items in the embedding space and maximize the distance between dissimilar items. The model can handle large-scale datasets and is computationally efficient, making it suitable for practical applications. It has been shown to outperform existing methods on several benchmark datasets in various NLP tasks. I will show in the next chapters how its performances are efficient without using too many resources.

Long Short-Term Memory [16] (LSTM) is a type of Recurrent Neural Network (RNN) that has shown great promise in modeling sequential data such as speech, audio, and text. LSTMs address the vanishing gradient problem

that can occur in standard RNNs by introducing a gated cell mechanism that allows for long-term memory storage and retrieval. The LSTM cell contains three gates: the input gate, the forget gate, and the output gate. These gates control the information flow into, out of, and within the cell, allowing for selective information retention and removal. The LSTM has been shown to achieve state-of-the-art results in a variety of natural language processing tasks such as language modeling, machine translation, sentiment analysis, and named entity recognition.

2.4 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a powerful language representation model that has played a leading role in the field of NLP in recent years. Developed by Google in 2018 [9], BERT is a deep neural network architecture that is pre-trained on massive amounts of text data and can be fine-tuned for various NLP tasks, such as text classification, question-answering, and text generation.
One of the main advantages of BERT, thanks to the Transformer architecture [48], is its ability to understand the context and meaning of words in a sentence by considering the surrounding words. Unlike traditional language models that process text from left-to-right or right-to-left, BERT is a bidirectional model that reads the entire input sequence of a sentence in both directions, allowing it to capture the context and meaning of words more accurately.
However, as I will proof later on, despite its success, BERT still faces some limitations and challenges, such as its high computational cost and the need for large amounts of training data. Nonetheless, the use of pre-trained BERT models has become prevalent in NLP applications, and many pre-trained models are now available for use by researchers and business developers.

2.4.1 How BERT works

As is mentioned before, the strength of BERT, is the ability to understand the context, this is achieved using Transformer, let's focus on its functioning.

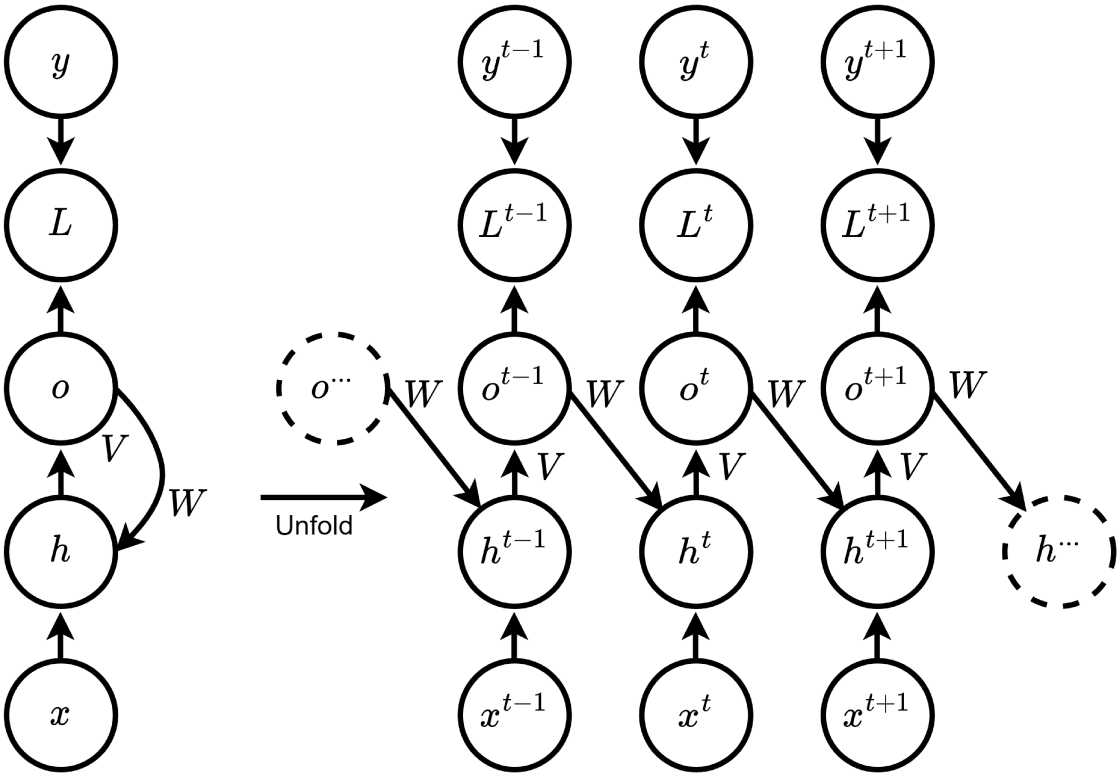


Figure 2.7: An RNN architecture.

Attention Is All You Need

Transformer is a deep learning model (architecture on figure 2.9) introduced in the paper *Attention Is All You Need* [48] in 2017, with the goals to go over the limitations of RNNs in sequence modeling tasks. One of the main issue of a RNN is that the vectors generated intermediately for every word (h^* in the figure 2.7) they are not truly represent the context of the word itself, after all the context depends on the word that comes before and the one that comes after, but as shown in figure 2.7 is clear that the signal comes from only the word that is just before it.
Even bidirectional RNNs (BRNNs) suffer of this phenomenon because they just look at the context left-to-right and right-to-left separately and then concatenate the results, this could cause some loss of the meaning.

The key innovation is the *self-attention mechanism*, which allows the model to capture global dependencies between input and output sequences without the need for recurrent connections, and keeping track on which part each word needs to focus on, a simple example is shown in figure 2.8, for each word is create a vector with weights (in the figure represented with darker color the

most related) that represent the relation with the others word.

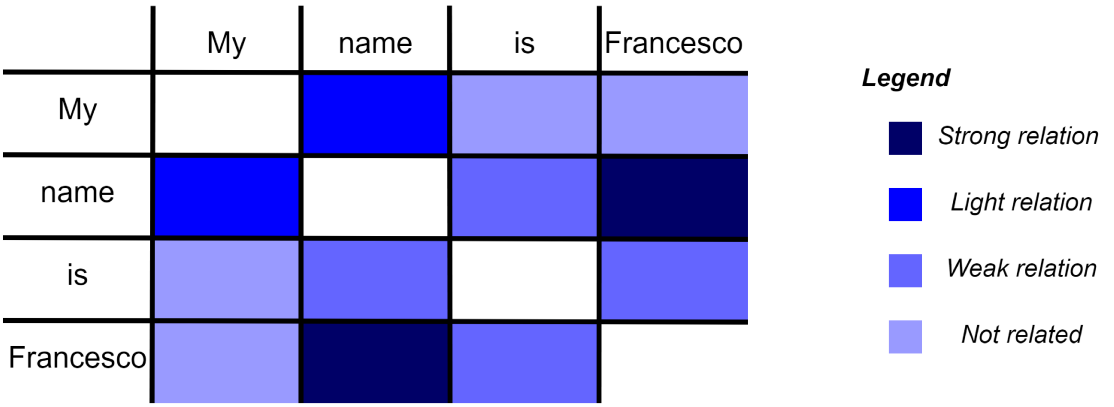


Figure 2.8: An simple example of self-attention mechanism.

How Self Attention score is computed

Let's go more deeper in the details of how the Attention score is computed in the Transformer architecture, shown in figure 2.9, which represent the architecture of a Transformer NN, there are an encoder and a decoder, the first one is used to encode the input sequence into a fixed-length vector representation, while the decoder is used to decode the output sequence from the encoded representation.

The Attention score is calculated in the *Multi-Head Attention* section, each input token is going to have three representations: **Q**uery, **K**ey and **V**alue vector. The query vector is used to determine which other tokens are most relevant to the current token. The key vector is used to calculate the similarity between the query vector and the key vectors of all the other tokens. The value vector is used to compute a weighted sum of the values of all other tokens, where the weights are determined by the attention scores.

The combination of those vector will populate the self-attention matrix:

$$\text{self} - \text{attention} = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

In the formula 2.3 the QK^T, known as *dot product attention*, performs the dot product between query and key vectors, so for every words we will have how much attention we want to focus on all the others. The denominator $\sqrt{d_k}$, that

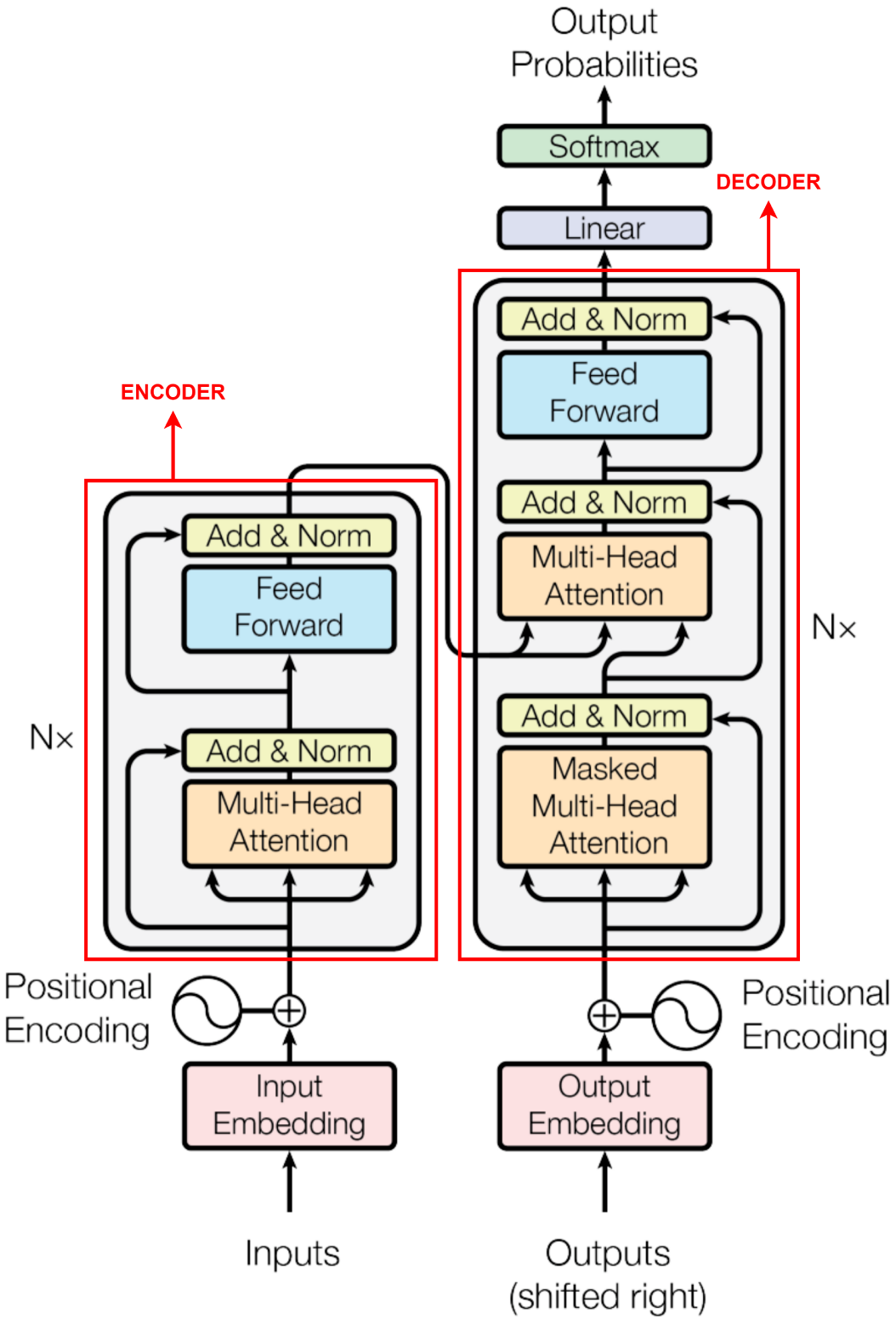


Figure 2.9: The Transformer - model architecture (from original paper).

is the square root of the dimension of the key vector, is used to minimize the variance, the *softmax* is applied to have the distribution probability and than its output multiplied by V, the value vectors, will give us the *attention-score*.

BERT: Encoder-only Transformers

BERT uses the Transformer architecture as the basis for its model. Specifically, it uses a variant of the Transformer known as the encoder-only Transformer. In the encoder-only Transformer, the input is passed through a series of encoding layers, each of which is composed of multi-head self-attention and feed-forward neural network sub-layers. The output of the final encoding layer is used to represent the input text. The input text to BERT is first processed by a special tokenization method called *WordPiece tokenization*. This method breaks down words into smaller subwords or pieces, allowing the model to handle out-of-vocabulary words and improve generalization.
For example, in figure 2.10, the word *playing* it's been splitted into *play*, *##ing*. The double hash symbol (##) is used to indicate that a subword is part of a larger word.
Additionally, BERT also uses special tokens to mark the beginning and end of a sentence, as well as to indicate the presence of a sentence pair in the input. The [CLS] token is inserted at the beginning of the input sequence, and the [SEP] token is inserted at the end of each sentence or sentence pair.
The [CLS] token is also used to represent the entire input sequence in BERT's classification tasks, where the output of the [CLS] token is fed into a final classification layer.

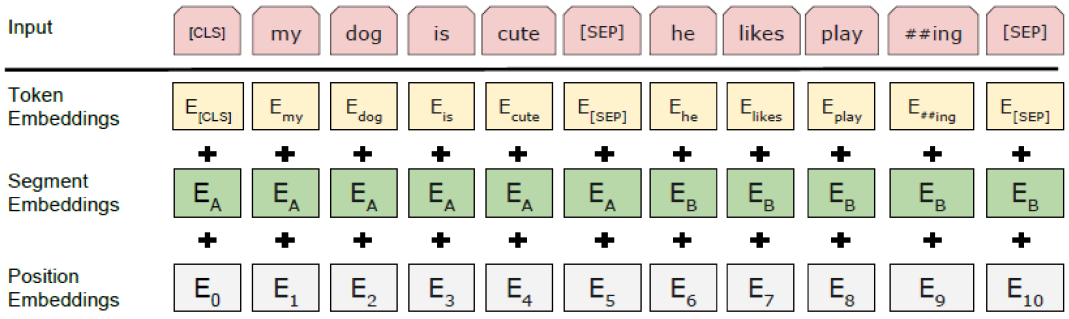


Figure 2.10: BERT input representation (from original paper).

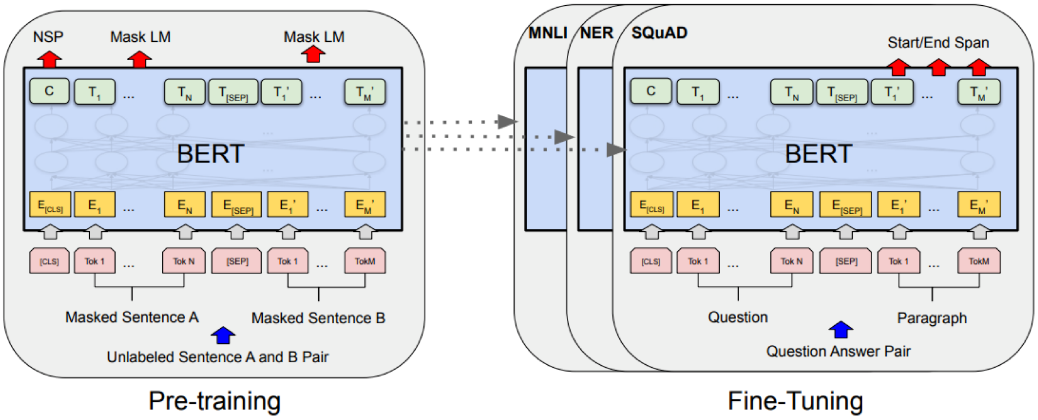


Figure 2.11: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning (from original paper).

2.4.2 Pre-trained models

The capability of BERT to learn general understanding of language is achieved by pre-training the model on large corpus of text using an unsupervised learning approach, without relying on labeled data. This step is presented in the left part of figure 2.11. Two tasks, namely masked language modeling (MLM) and next sentence prediction (NSP), were used for pre-training the model.

In MLM, BERT randomly masks some words in the input sentence and attempts to predict the original masked words based on the context provided by the remaining sentence.
For NSP, BERT tries to predict whether two input sentences are consecutive or not. This task helps BERT to understand the relationships between different sentences and to learn the context between them.

There are several pre-trained models available for text classification tasks. In this particular study, models were selected that are pre-trained even in the Italian language, which is the primary language of the datasets being examined. Table 2.4 presents the principal characteristics of the models that will be tested in the subsequent chapters.

[1]In this cell and later on the thesis in *Main sources* will be mentioned training data corpus, which are online platforms with all large-scale collections of texts in multiple languages, the data are available for download and can be used for research and other purposes such as building natural language processing models.

Model	Main sources[1]	Corpus size	Parameters	Tokens
Italian XXL [24]	Wikipedia OPUS [32] OSCAR [33]	$\approx 81GB$	$\approx 110M$	$\approx 13B$
XLM-RoBERTa [8]	Wikipedia Crawl [7]	$\approx 2.5TB$	$\approx 355M$	$\approx B/T$
AlBERTo [37]	Wikipedia Twitter	$\approx 270GB$	$\approx 370M$	$\approx B$
UmBERTo [47] (RoBERTa based)	Wikipedia OSCAR [33]	$\approx 7GB$	$\approx 110M$	$\approx B$
GilBERTo [13] (RoBERTa based)	Wikipedia OSCAR [33]	$\approx 71GB$	$\approx M$	$\approx 11B$

Table 2.4: Some of pre-trained models that could be used for fine-tuning.

2.4.3 Fine-Tuning

Fine-tuning in BERT involves taking the pre-trained models and training them on a specific downstream task, in the right part of the figure 2.11 the process for example applied to NER task is shown. In this case I will fine-tune it for text classification. Usually the pre-trained model is used in conjunction with a task-specific architecture, for example a linear dense layer, dropout layer etc.. The parameters of both architectures, BERT model and the task-specific one are then fine-tuned using supervised learning.

The fine-tuning include several steps, including the selection of an appropriate BERT pre-trained model, since the characteristics of the model will play a fundamental role in the prediction phase, for example a model that has been trained on a large dataset which include Italian text inside, will work better in this project since the text that I will use for fine-tune it will be mainly in Italian. Another important factor is the choice of the architecture that follows the pre-trained model, which will have some particular characteristics of the specific-task (number of output classes etc.).

2.4.4 Limitation and challenging in using BERT

Despite its impressive performance, BERT has some limitation and challenges that I need to consider, let's summarize some of them.

Computational Resources

BERT models are computationally expensive to train and require a large amount of memory to store, this can make it difficult for small/medium organizations to utilize BERT effectively, to fine-tune BERT in this project it's been used Google Colab [14] and Kaggle [21] online platform because without them the training phase without using a GPU would been to much expensive to face.

Domain-specific Vocabulary

BERT's pre-training is based on a general vocabulary, texts are taken from the web, Wikipedia documents, tweet etc., that may not be suitable for certain domains or languages. Text could include some particular pattern like product code, or some specific word that has a particular meaning for the companies in exam but in general have a completely different explanation. This can result in poor performance when using BERT for tasks that require domain-specific vocabulary.

Multilingual Support

As already mentioned before since I am dealing with human natural language I need to take in consideration models that has been pre-trained on text that are suitable for this specific language domain and there are still many languages that are not well represented in BERT's pre-training corpus, this is a limitation for task that require support for less used languages.

2.5 Evaluation metrics

Choosing the right evaluation metrics is an important step when developing AI models in NLP, especially when dealing with unbalanced datasets. This is because accuracy alone is not always a reliable indicator of model performance when datasets are unbalanced, where one class of data may dominate over the other. In such cases, accuracy can be misleading, as the model can achieve high accuracy even by classifying almost all instances as the dominant class. Let's suppose that the test-set of data contains 95 elements of class A, and only 5 elements of class B, a model that always predicts class A will achieve 95% of accuracy, it is way off base.

Therefore, in addition to accuracy, I need to use evaluation metrics that account for class imbalance, such as *precision*, *recall* and *F1-score*.

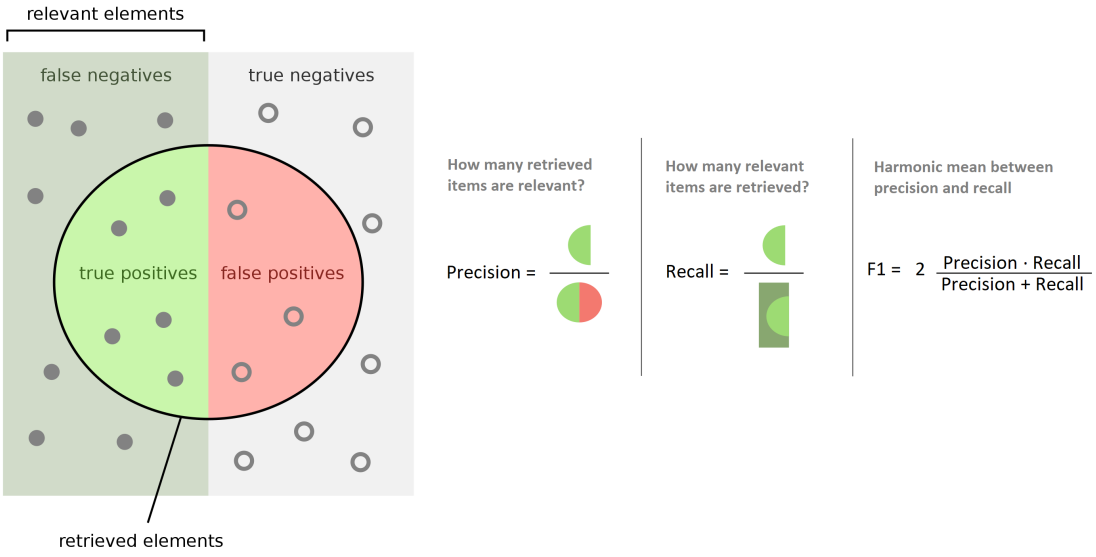


Figure 2.12: A graphical representation of precision, recall and F1-score.

The figure 2.12 shows how those metrics are computed. **Precision** measures how many of the positive predictions made by the model are actually correct, and it is calculated by dividing true positives by the sum of true positives and false positives.

Recall measures how many of the actual positive instances are identified correctly by the model, and it is calculated by dividing true positives by the sum of true positives and false negatives.

F1-score is a harmonic mean of precision and recall, and works also for cases where the datasets are imbalanced as it requires both precision and recall to have a reasonable value. Even if you have a small number of positive cases vs negative cases, the formula will weight the metric value down if the precision or recall of the positive class is low.

Regarding the evaluation I will refer to two different ways to calculate the average metrics, (macro avg./weighted avg.), they are often used for multi-class classification problems:

- **Macro Average (macro avg):** The macro average calculates the metric independently for each class and then takes the unweighted average across all classes. It gives equal importance to each class and treats them as equally important in the calculation of the average metric. It is useful when you want to evaluate the overall performance of the classifier without considering class imbalance.

- **Weighted Average (weighted avg):** The weighted average calculates the metric for each class, but takes into account the class imbalance by weighting the average by the number of samples in each class. It gives more importance to the metrics of classes with a larger number of samples, as they have a greater impact on the overall average. It is useful when you have class imbalance and you want to have a more representative average that considers the contribution of each class based on their sample size.

3

Main tool and libraries

Before going throw the details of implementation in this section I will brief introduce the main tools that I have used to do the analysis and development.

3.1 Programming languages

The are several programming languages used in the development of AI models, for this project I will use *Python* [11], version 3.7 and 3.10, that is one of the most widely used programming languages in the world of AI, particularly in machine learning and natural language processing. Its extensive library of open-source modules and tools makes it an ideal choice for processing large datasets and building complex models.

Regarding the deployment part I will use *Groovy* [12], it is a programming language based on Java designed to enhance its capabilities by providing additional features for scripting and meta-programming. Groovy is a dynamic language, which means that it is able to adapt to a wide range of programming styles and can be used for various purposes, including web development, scripting, and automation. In this project I will use Groovy to build the *adapter* that is used to integrate, through API, the Language Detection model in the Engagent Pat solution.

(a) Python logo.

(b) Groovy logo.

Figure 3.1: Programming languages logos.

3.2 Main Python libraries

As just mentioned in the previous paragraph, one of the main advantages using Python is the large amount of libraries available without any subscriptions or payment. Here I will discover the main Python libraries used for this project.

3.2.1 Baseline: Pandas, Numpy, Matplotlib

Pandas

Pandas [28] is one of the most important and commonly used open-source library in Python. It offers a range of tools for working with large quantity of data, including *Dataframe*, a particular data structure for efficient data processing, powerful data analysis tools, and data visualization capabilities.

The key advantages of Pandas library include:

1. **Flexibility:** It is highly flexible, it can handle different data format including spreadsheet data, comma-separated values (CSV) files, and relational database.
2. **Data processing capabilities:** It provides a range of data processing capabilities, including handling missing data, merging and joining data sets, and filtering and aggregating data.
3. **Powerful data analysis tools:** It provides a range of powerful data analysis capabilities, including statistical analysis, time series analysis, and exploratory data analysis, for example it allows the calculation of standard deviation on thousands values in a very effective way and with a simple command.
4. **Data visualization:** It provides powerful data visualization capabilities, through integration with other libraries such as Matplotlib [18], allowing users to create visually appealing and informative charts and graphs.

Figure 3.2: Pandas logo.

Numpy

Another popular open-source library is **Numpy** [15], it is used for numerical computing and data analysis in Python. It provides powerful tools for working with matrices, vectors and other data structures. Numpy is an essential tools for data scientist and engineers who work with huge amount of data.

The main advantages of Numpy are *broadcasting*, which is a powerful feature that allows operations between elements in arrays with different sizes and shapes, and interoperability with other libraries and tools, including Pandas and Matplotlib [18] making it easy to use.

Figure 3.3: Numpy logo.

Matplotlib

Matplotlib [18] is the most common used library for creating static, animated, and interactive visualizations in Python. It allows users to create many types of plots, including line plots, scatter plots, bar plots, histograms, and more. Almost all the graph that I will show in the next chapters are generated using this library.

Figure 3.4: Matplotlib logo.

3.2.2 Nltk: Natural Language Toolkit and Scikit-learn

There are several libraries that provide tools for working with natural language processing and machine learning tasks in Python, two of them are *nltk* and *scikit-learn*. NLTK [5] is a library designed for working with human language data, including text pre-processing and analysis. It provides different tools for NLP tasks such as tokenization, stemming, tagging, and parsing.

Scikit-learn library is used for machine learning tasks, It provides simple and efficient tools for data mining and data analysis, for example it allows handling with splitting the dataset in train-set, validation-set and test-set in a easy and efficient way and a lot of other useful functions.

Figure 3.5: Scikit-learn logo.

3.2.3 Spacy, langdetect and FastText

Python documentation offers different libraries to deal with NLP pipelines, in this section I will discover *Spacy* [17] by Explosion AI department, *langdetect* [41] by Cybozu Lab. (original project in Java) and *FastText* [36] powered by Facebook Inc., in particular I will use an implementation of their tools to detect the languages.

Spacy library is an open-source library that provides different functionalities for text processing, tokenization, name entity recognition and more, one of the most interesting feature is that it offers pre-trained models that can detect the language of a given text. It is able to detect more than 72 languages.

Langdetect library is based on the features extraction of the *n-grams* of the word. It works by analyzing the character distribution of the input text and comparing it to the character distributions of various languages. The library currently supports over 55 languages.

FastText is an open-source, lightweight library that provides efficient and accurate text classification and representation learning. One of its features is the ability to perform language detection quickly.
FastText's language detection is based on the classification of n-grams, which are contiguous sequences of n items from the input text. In the case of language detection, the n-grams are characters or character sequences. FastText first trains a language classification model on large amounts of text data in different languages. The model learns to associate each language with its characteristic n-grams. When given a text input, FastText breaks it down into n-grams and uses the model to predict the language of the input based on the n-grams present. The model outputs the most likely language of the input, along with a confidence score. More details on this approach will be presented in Chapter 6.

(a) SpaCy logo. (b) Cybozu Lab Inc.. (c) FastText logo.

Figure 3.6: Principal libraries for language detection.

3.2.4 Tensorflow and Transformer

Tensorflow [1] is another open-source machine learning framework developed by Google commonly used. It provides a comprehensive set of tools for building and training machine learning models such as neural networks, decision tree, regression models etc.. The library is written in Python but it offers support for different programming languages (Java, C++ etc.). One of the most important API is *Keras* which allows users to build and train models quickly and easily.

Transformer library by Hugging Face [51] is used for working with a variety of transformer-based models, including BERT and others. It provides APIs for fine-tuning and adapting pre-trained models for specific tasks. The Hugging Face Transformers library also provides access to a large number of pre-trained

transformer-based models, including multilingual models which can be used for a variety of natural language processing tasks, including text classification, named entity recognition, and sentiment analysis.
Since the Hugging Face Transformers library is a powerful tool for working with transformer-based models in Python, it will play a main role in this project.

(a) TensorFlow logo. (b) Hugging Face logo.

Figure 3.7: Main tools for dealing with NLP and machine learning.

3.3 Tools

To develop and deploy a project, it is essential to have access to a set of tools and software services that can facilitate the process. There are several cloud-based platforms that provide the necessary infrastructure and services to build and deploy machine learning models. In this section, I will discuss some of the main software services and tools used to develop and deploy the entire project.

3.3.1 JetBrains: PyCharm and IntelliJ

PyCharm and *IntelliJ IDEA* are integrated development environments (IDEs) developed by JetBrains [19] for Python and Java development, respectively. PyCharm provides developers with an integration with various tools and frameworks commonly used in Python development. IntelliJ IDEA is an IDE for Java that includes support for several other programming languages, such as Groovy, making it a versatile choice for developers. Both of them offer to developers advanced features such as code completion, debugging, and testing. PyCharm and IntelliJ IDEA are widely used by developers for their ease of use, versatility, and extensive support for various programming languages and frameworks.

(a) JetBrains logo. (b) PyCharm logo. (c) IntelliJ logo.

Figure 3.8: Commonly used IDEs for development.

3.3.2 Colab/Kaggle

Google Colab [14] and *Kaggle* [21] are cloud-based platforms that provide a free environment for running machine learning and data analysis tasks. Google Colab (short for "Collaboratory") is a service that allows users to write and run Python code in a Jupyter Notebook, with access to free GPUs and TPUs. It is a part of Google Drive and can be accessed via a web browser. Google Colab also allows users to store, share, and collaborate on notebooks with others.

Kaggle provides both a Jupyter notebook environment and a cloud-based machine learning workbench. These platforms are useful for exploring, analyzing, and visualizing data, as well as for training and deploying machine learning models. Users can compete with other data scientists to solve real-world problems and also collaborate with others on open-source projects. Kaggle also offers various learning resources, including courses and tutorials, to help users improve their data science and machine learning skills.

Both of them could be a good solution for users to scale up their computational resources as needed. In this project, they will be used with a free license, thus with some limitations on the use of GPUs (around 20 hours per week). They also offer a subscription to have unlimited access.

(a) Google Colab logo. (b) Kaggle logo.

Figure 3.9: Main cloud-based environment to develop AI models and more.

3.3.3 Docker and Kubernetes

Docker

Docker [10] is a platform for developing, deploying, and running applications using containerization technology. This technology allows developers to package an application into a lightweight and portable *container* that can run on any infrastructure, whether it is a laptop, a testing environment, or a production server. The reason for this is due to the fact that a container contains not only the application itself but everything that the application needs to run, including the code, dependencies, and system libraries. Thus, Docker is like a layer of abstraction between the application and the underlying operating system and hardware.
Containers are built from *images*, which are essentially pre-configured templates that specify the application and its dependencies.

Docker provides a command line interface (CLI) and a graphic user interface (GUI) for managing containers and images. Users can create, run, stop, and delete containers, as well as build and push images to Docker registries for others to use. Using Docker both models, language detection and ticket classification will be transferred in an internal server (test environment) in a way that they could be tested without worrying about the infrastructure.

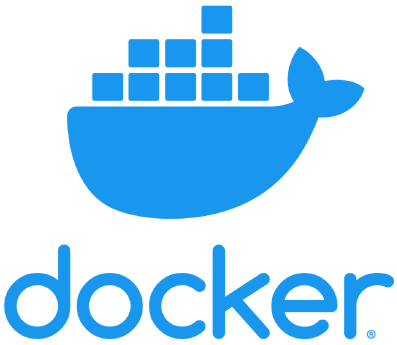
Kubernetes

Kubernetes [22] is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). It provides a way to automate the deployment and scaling of containers, as well as manage containerized applications in a distributed environment.

At its core, Kubernetes is built around the concept of a cluster, which is a set of worker nodes that run containerized applications. Each worker node runs a container runtime, such as Docker, and is managed by the Kubernetes control plane. The Kubernetes control plane is responsible for managing the overall state of the cluster, including deploying and scaling applications, monitoring and logging, and handling failover and recovery.

Nowadays, many companies choose Kubernetes for their production work-

loads to ensure scalability, reliability, and high availability.



(a) Docker logo.

(b) Kubernetes logo.

Figure 3.10: Main tools for dealing with containers and images side by side.

4

Analysis of Ticket Misclassification and its Impact on Business Productivity: A Case Study of PAT srl for Implementing an AI Model for Process Optimization

Most of the time, customers do not recognize the problem, or better, they don't know what they need to improve their business productivity because they are not able or they don't have the attitude to correctly identify the points on which to intervene.

In this chapter, I will delve into the problem of ticket misclassification in the system of trouble ticketing. As I have already said, the misclassification of tickets can cause significant delays in resolving customer issues, leading to customer dissatisfaction and negative impacts on the overall customer experience. Therefore, it is essential to explore the use of AI models and why we need them to help customers correctly classify their tickets and reduce the likelihood of misclassification.

The goal of this analysis is to underline the problem end expose it up to the customer.

4.1 Focus on Productive Time: An Empirical Evaluation of Performance and Efficiency

The dataset used for this analysis is the *Energy Company* (more details in the Chapter 1, on table 1.1). First, I will compare the resolution time (RT) of those tickets that are correctly classified to the tickets that are wrongly classified. After that, I will focus on the number of forwards to check if the number of forwards of a misclassified ticket is higher compared to a ticket in which its correct category has been detected in one shot.

4.1.1 Comparison between correct and wrong classification

To perform this analysis, I need to focus on 4 particular components of the dataset: the *CreationDate* and the *ClosureDate*. Their differences will give the resolution time of a ticket, computed as follows:

$$\text{RT} = \textit{ClosureDate} - \textit{CreationDate} \tag{4.1}$$

The other fields needed for this analysis are *Application*, in which the customer's choice is stored, and *Application_FINAL*, which represents the final correct department. With these ticket details, I can detect the misclassifications. Thus, if:

$$\textit{Application} \; ! = \; \textit{Application\_FINAL} \rightarrow \textbf{\textit{Wrong classification}} \tag{4.2}$$

Resolution time distribution

In the figure 4.1 is displayed the distribution of the RT of the entire tickets in the dataset, most of the tickets are solved in less then 10 days, anyway there could be some cases, the *outliers tickets*, in which the time needed to close the ticket is very high. There are several reasons why the RT of a ticket could take even more that 40 days, the solution process could be complex depending of the nature of the ticket, or maybe is needed some additional information and so the ticket stays in the status *suspended* until the customer gives the information

necessary to complete the procedure, another reason concern the holiday period in which there are fewer operators that works on the ticket resolution. Anyway those cases represent only the 5% of the dataset, so I can neglect them.

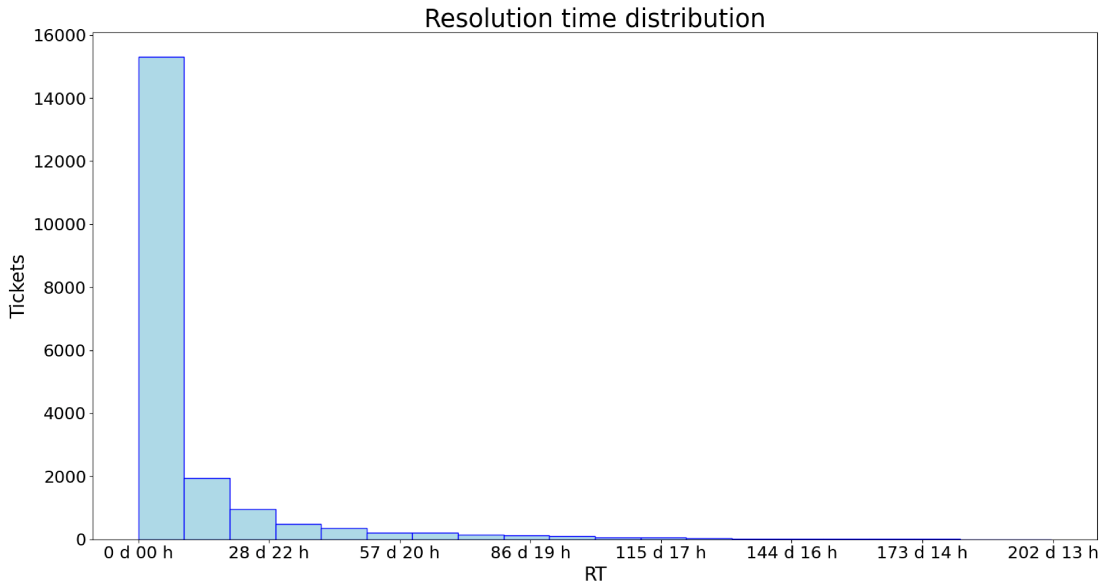


Figure 4.1: Resolution Time distribution.

Applications analysis

The RT of a ticket is strictly related to the complexity of the issue described inside. Let's assume that I have two tickets, A and B. The first is wrongly classified, and the second (B) has the right destination.
The content of ticket A refers to a log-in issue, the user forgot it's credentials (assume that the user doesn't know how to recover them), a problem that takes 5 minutes to be solved, but the ticket arrives to the wrong department and it takes 1 day to be submitted to the right one, so RT(A) is equal to 1 day and 5 minutes.
Ticket B, instead, underlines a connection problem, maybe a server is down and to understand the problem and to solve it's required more time, so RT(B) is equal to 2 days.
In this case the time needed to solve a wrong classified ticket is lower but, actually there is 1 day of wasting time, so comparing all the tickets together putting on one side the correct one and on the other side the misclassifications doesn't make any sense.

For this reason, I will compare the RT between tickets that belong to the same *application*, so the content and the possible issue are comparable.
For this kind of analysis, since I want to focus on the loss in terms of time perceived from the customer, I have considered 1 day composed of 24 hour, considering also the weekend and the holiday.

In table 4.1 and in the figure 4.2 is possible to see the results of the analysis, the *Loss* field is simply the difference between correct and wrong RT, I can notice that in some applications there are a substantial quantity of time wasted.
For example, the application *STAMPE_ISU* which represent the 3.05% of the total amount of tickets, the error rate is quite high, so the impact of that loss (3 days and 11 hours) could be important.

Application_FINAL	**Ticket of total**	**Error rate**	**Mean RT Correct classification**	**Mean RT Wrong classification**	**Loss**
SAP-DBO	6,70%	12,34%	2 d 12 h	6 d 08 h	3 d 20 h
SIEBEL	6,65%	17,24%	3 d 18 h	6 d 07 h	2 d 13 h
SAP-BI	5,75%	32,39%	3 d 23 h	6 d 14 h	2 d 15 h
SAP-BW/4 HANA	4,81%	7,26%	3 d 23 h	7 d 07 h	3 d 8 h
SAP-FI/CA	3,57%	19,25%	4 d 12 h	8 d 02 h	3 d 10 h
OIG	3,54%	18,39%	8 d 14 h	19 d 22 h	11 d 8 h
STAMPE_ISU	3,05%	40,65%	5 d 01 h	8 d 12 h	3 d 11 h
SAP-SSA	2,61%	14,73%	3 d 23 h	9 d 23 h	6 d
SAP-WM	1,80%	9,39%	5 d 02 h	9 d 16 h	4 d 14 h
SAP-MM	1,62%	16,85%	11 d 23 h	12 d 23 h	1 d
SAP-EBDM	1,58%	28,31%	5 d 08 h	10 d 06 h	4 d 22 h
SAP-BW	1,51%	9,23%	3 d 21 h	14 d 19 h	10 d 22 h
SAP-MDG	0,96%	63,64%	13 d 00 h	16 d 04 h	3 d 4 h
SAP-HR	0,69%	40,68%	3 d 15 h	6 d 17 h	3 d 2 h
SAP-SD	0,65%	19,82%	14 d 01 h	20 d 14 h	6 d 13 h
SAP-WASTE	0,63%	14,68%	10 d 06 h	15 d 16 h	5 d 10 h

Table 4.1: The first 15th classes in which is underlined a loss in terms of time between tickets with correct classification and tickets misclassified.

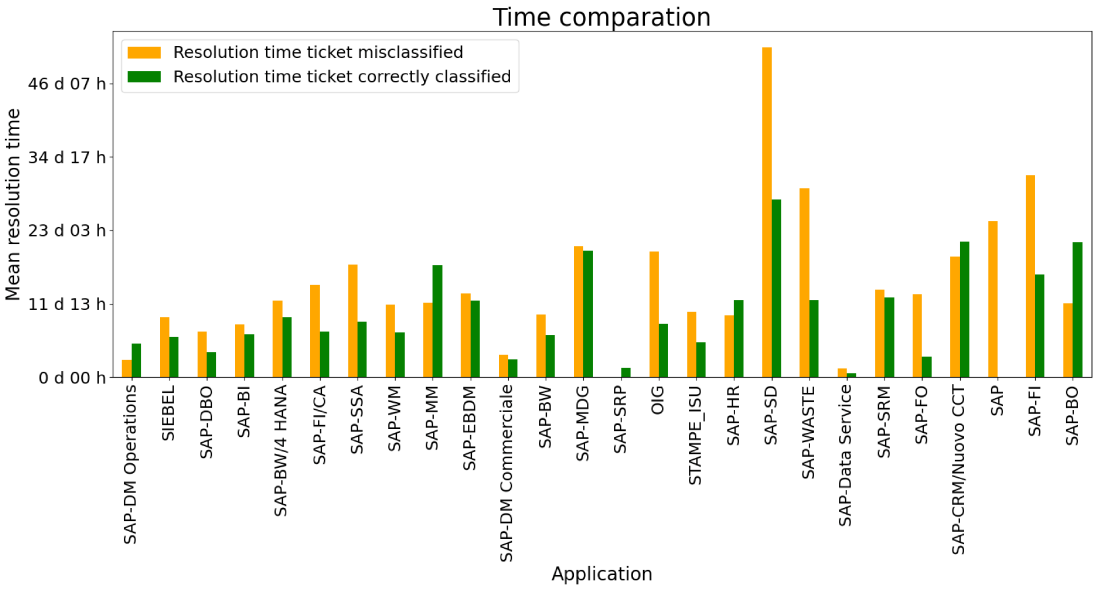


Figure 4.2: Graphical representation of the comparison between resolution time of correct/wrong classification.

4.1.2 Forwards and waste of time due to forwards

In this section, I will analyze the comparison of the number of forwards between the correctly classified tickets and the wrongly classified ones. The forwarding of tickets could happen also when the agent who receives the ticket cannot solve the problem and forwards it to a higher level of expertise. But most of the time, the reason is simply a misclassification.

The comparison of the number of forwards between correctly classified tickets and the wrongly classified ones, shown in table 4.2 and in figure 4.3, will help us understand the impact of misclassification on the efficiency and productivity of the trouble-ticketing system.

Based on the comparison of resolution time and mean number of forwards between correctly classified tickets and wrongly classified ones, it can be concluded that misclassified tickets lead to higher resolution time and more forwarding actions. This finding highlights the importance of accurately classifying tickets in the trouble ticketing system. The longer resolution time and increased forwarding actions not only negatively impact the customer experience but also increase the workload of the support team. Thus, an AI model that can help in the accurate classification of tickets can lead to a more efficient resolution of customer issues and a better overall experience. In the following sections, I will

Application_FINAL	Ticket of total	Error rate	Mean Forwards Correct classification	Mean Forwards Wrong classification
Salesforce Heravendi	7.60%	4.39%	0.1	0.9
SAP-DM Operations	6.52%	59.67%	0.47	1.16
SAP-DBO	6.17%	13.97%	0.33	1.59
Siebel	6.13%	16.72%	0.42	1.35
SAP-BI	5.54%	33.39%	0.5	1.37
SAP-BW/4 HANA	4.80%	7.63%	0.55	1.16
OIG	3.52%	19.47%	1.31	0.66
SAP-FI/CA	3.48%	19.57%	0.49	1.56
STAMPE_ISU	2.82%	39.57%	0.71	1.46
BEAM	2.53%	63.19%	0.4	1.23
SAP-SSA	2.46%	15.09%	0.44	1.99
LIMS	1.81%	2.56%	0.26	2.8
SAP-WM	1.74%	12.27%	0.55	1.67
SAP-MM	1.59%	16.57%	0.56	1.11
SAP-PM	1.51%	58.46%	0.7	1.36
SAP-EBDM	1.44%	27.97%	0.46	2.06
SAP-BW	1.44%	12.86%	0.17	0.6
SAP-DM Commerciale	1.43%	59.22%	0.94	1.11
SAP-MD	1.37%	74.92%	1.22	1

Table 4.2: Applications in which is underlined that the mean number of forwards of tickets with correct classification is lower than tickets misclassified.

discuss the development and evaluation of an AI model to help address this issue.

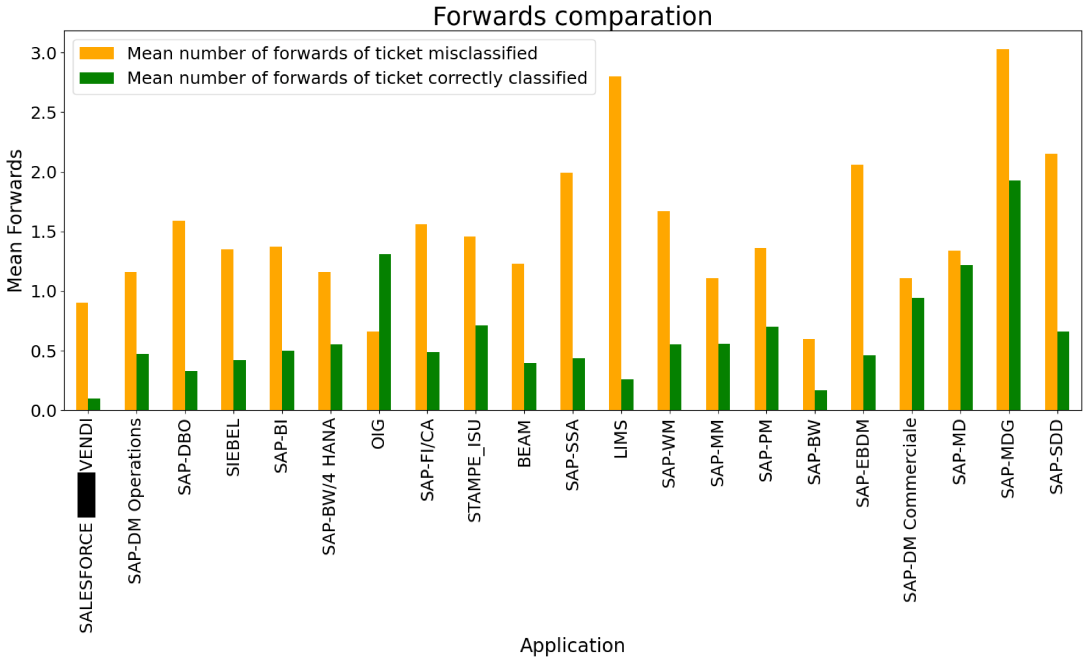


Figure 4.3: Graphical representation of the comparison between number of forward of correct/wrong classification.

5

Data Exploration and Preprocessing for Improving the Performance of the AI Model

The quality of data is one of the most important factors that affect the performance of the AI model. Preprocessing is an essential step in developing accurate and effective AI models in NLP. It involves cleaning and transforming raw data into a format that can be easily understood by the model. Preprocessing can significantly impact the quality of the model's output. Poorly preprocessed data can lead to incorrect or inconsistent results, especially when dealing with noisy text data in real-world scenarios. In this chapter, I will explore what we mean by noisy data and discuss the steps taken to overcome this problem in both the datasets used for the language detection and ticket classification models. By the end of this chapter, I will have the input data ready for the development/training phase.

5.1 Language detection: Chat bot dataset analysis

To perform correct preprocessing, I need to understand data as thoroughly as possible. Additionally, I need to keep the final objective in mind. In this case, my aim is to detect the language. Therefore, even so-called *stopwords* (such as grammar, conjunctions, adverbs, etc.) and *punctuation*, which usually do not

add useful information, could be significant factors in determining the correct language.

Let's begin with data exploration. I will examine various useful details, such as the distribution of lengths, and inspect the data for any particular patterns or text structures that may affect the detection.

5.1.1 Preliminary text analysis

The figure 5.1 displays the length distribution of the *Question* field in the dataset used for the language detection model. In this case, there is a small portion of the dataset, less than 1%, with a length greater than 500. All of these questions are ignored or excluded from further analysis. As expected,

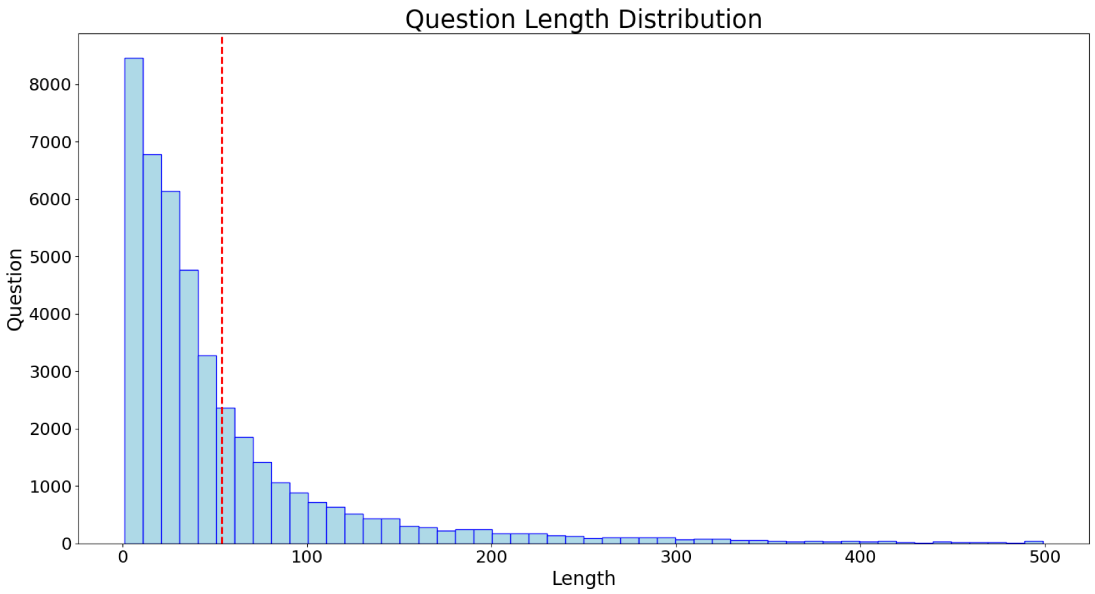


Figure 5.1: Question length distribution.

the average length is around 54 characters (indicated by the red dashed line). This observation indicates that I should be careful when cleaning the text. It's important to note that performing good preprocessing doesn't always mean removing as much data as possible. In this case, I want to avoid losing too much information.

5.1.2 Text empirical inspection

When attempting to detect the language in a specific context, especially in a business scenario, it is possible to encounter an unknown vocabulary universe. Table 5.1 displays a few examples of user questions, where certain patterns may refer to specific product names or IDs. Since the structure of these patterns could be numeric or alphanumeric, they have the potential to confuse the language detector.

Usually, the presence of specific patterns in long texts should not pose a

Question	**Language**
Salve io ho un **Product name*[1]in aa/bb/123	it
No, I need the new codes for the following **Product name** - abc123	en
Necesito asistencia tecnica para la **IDProduct** de 80 cm	es

Table 5.1: Some example of user questions.

problem. However, when texts are short, the noise introduced by these patterns must be taken into consideration.

To address this issue, I can employ *regular expressions*. In the context of text preprocessing for NLP, regular expressions are used to eliminate noisy patterns such as URLs, special characters, HTML tags, or other irrelevant information that might interfere with the analysis.

Regular expressions are powerful tools used to match and manipulate text patterns. They consist of a sequence of characters that define a search pattern and are employed to find, replace, and extract text data. By defining specific patterns to match and replace, regular expressions assist in cleaning the text data and improving the accuracy of downstream tasks.

Thanks to regular expressions, all the noisy data such as *Product name*, *ID-Product*, and others were removed. This implies a highly accurate analysis and manual research to identify and remove them from the text.

[1]In the entire thesis when the symbol (*) appears it means that the word, or part of it, contains some information strictly related to the actual company, for this reason it will be censured.

5.1.3 Chat concatenation

Introducing the dataset, I mentioned that there is no information about the language of the messages. This is why I used the Papaluca and Language detection labeled datasets from the web [34, 23] to evaluate the accuracy of the models.
However, this also means that I need to find a method to evaluate the results on the internal dataset. The best solution would be to manually label all the messages, but this would require a significant amount of time and resources. Additionally, relying on a single person to do the labeling could introduce errors.

As I will show later on, all the models perform quite well on medium-to-long text. Therefore, one possibility is to compare the prediction results of each individual message with the prediction of the concatenated messages belonging to the same chat.

By concatenating all the messages grouped by *Chat Session* (see Figure 5.2) the distribution changes. In this case, the mean number of characters is around 253, resulting in a higher probability of success in language detection. Let's in-

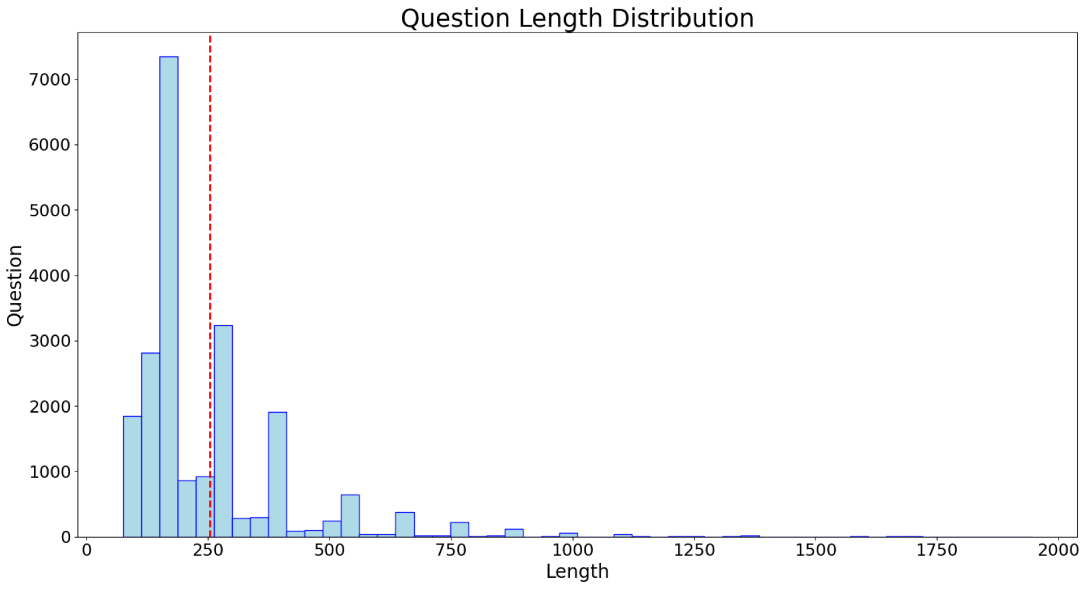


Figure 5.2: Question length distribution, with concatenated *Question* grouped by *Chat Session*.

troduce a new measure to evaluate the performance of the models in this dataset. Since I do not have the certainty of correct predictions due to the missing labels, I cannot rely on accuracy as a metric. Instead, I will refer to the *correspondence*

between the predictions of the concatenated messages and the predictions of the individual messages.
The problem arises when predicting the language in a concatenated chat. I obtain one result for the entire chat, but if the chat is composed of multiple messages, I will have multiple results. So, how do I compare them together?
To address this issue, two approaches will be examined. The first, simpler approach is to assign the language detected by concatenating the entire chat session as the label for each message and then determine the correspondence. Alternatively, in Figure 5.3, a probability distribution has been created for the results. The result to be compared with the chat concatenation result is chosen by selecting the highest probability. This approach takes into account the possibility that sometimes a single message may consist of just one word, such as *menu*, which could be a multilingual word or may not provide enough information to determine the language even for a human being. Therefore, the first method may not be the most appropriate in such cases.
By combining the results of both evaluations, I can gain insight into the correctness and robustness of the predictions for individual messages.

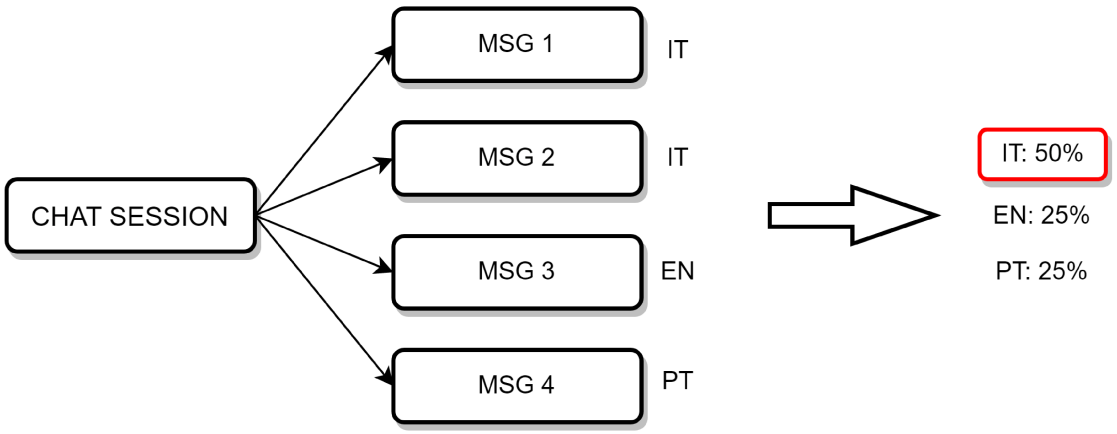


Figure 5.3: Choice of language to compare with the result of the chat concatenation due to the probability distribution.

5.2 Ticket classification: Trouble ticketing dataset

The Energy Company dataset under examination in this case includes labels, indicating that I am dealing with a multi-classification problem. In this scenario, preliminary analysis and text preprocessing are crucial to comprehend the data

and make appropriate adjustments or assumptions to enable optimal learning by the models. It is important to emphasize that preprocessing can be modified even after implementing the model, as discussed in Chapter 1 in the thesis structure section (see Figure 1.1). By analyzing the prediction results, I can revisit and adjust or add preprocessing steps based on those results.

5.2.1 Dataset overview

Let's delve deeper into the analysis of the dataset's features. The first aspect to consider is the number of unique tickets. Since the goal is to classify the correct class, I can neglect information related to the creation, closure, and forwarding time in this phase. Instead, I will focus on the columns *Subject*, *Request*, *Application*, and *Application_FINAL*.

Table 5.2 provides a summary of the most important features of the dataset. It's worth noting that sometimes the user does not make any choice regarding the application. However, this does not impact the training process since the column that will be used as the label is *Application_FINAL*.

Features	**Description**
Number of rows	38021
Number of unique tickets	21570
Number of classes	190
Null value in *Application* column	1510
Language	Italian

Table 5.2: Some useful details about the Energy Company dataset.

Label distribution

In Figure 5.4 and Table **??** below it, it's possible to observe the ticket distribution for the first fifty applications. It is evident that I am facing one of the most common challenges in the NLP environment: class imbalance. The table displays only 30% of the total applications, yet I can already see a significant difference in ticket counts between the first application, *SALESFORCE ***VENDI*, with 1640 tickets, and *SAP-FI*, with only 91 tickets. This indicates that during the implementation phase, I must set a threshold for the number of tickets. All

classes that do not meet this threshold will be excluded to ensure consistency in the training process, more on this in Chapter 7.

Dealing with an unbalanced dataset can be a complex task, and I will try to address this problem in the last section of this chapter.

Application_FINAL	Ticket	Ticket of total	Application_FINAL	Ticket	Ticket of total
SALESFORCE ***VENDI	1640	7.6%	HDA	232	1.1%
SAP-DM Operations	1406	6.5%	SALESFORCE GWM	221	1%
SAP-DBO	1331	6.2%	DOC1	218	1%
SIEBEL	1322	6.2%	GEOCALL	214	0.9%
SAP-BI	1195	5.5%	ORACLE HCM	193	0.9%
SAP-BW/4 HANA	1035	4.8%	SAP-SDD	188	0.8%
OIG	760	3.5%	Sito ***Comm	180	0.8%
SAP-FI/CA	751	3.5%	SAP-SRP	165	0.7%
STAMPE_ISU	609	2.8%	Portale Web Terzisti	165	0.7%
BEAM	546	2.5%	SAP-SD	150	0.7%
SAP-SSA	530	2.5%	SAP-HR	147	0.7%
Sito Gruppo***	513	2.4%	SAP-EHS DPI	141	0.7%
HRNext-Payroll	404	1.9%	SAP-WASTE	133	0.6%
LIMS	390	1.8%	SAP-Data Service	127	0.6%
ARCHIFLOW Protocollo	385	1.8%	YUBIK LEGALE	122	0.6%
SAP-WM	375	1.7%	HEROKU	121	0.6%
SAP-MM	344	1.6%	SAP	111	0.5%
SAP-PM	325	1.5%	Sito In***	105	0.5%
SAP-EBDM	311	1.4%	xxx Altro	104	0.5%
SAP-BW	311	1.4%	ESRI GIS	104	0.5%
SAP-DM Commerciale	309	1.4%	HRNext-MasterData	100	0.5%
SAP-MD	295	1.4%	SAP-SRM	98	0.5%
SAP-MDG	264	1.2%	SAP-FO	92	0.4%
YuBSC	245	1.1%	SAP-CRM/Nuovo CCT	91	0.4%
HRNext-Time	235	1.1%	SAP-FI	91	0.4%

Table 5.3: First fifty Application_FINAL details.

Length distribution

Tickets in this dataset describe several problems or information request, as we can notice in the figure 5.5 the mean length is equal to 500 characters, as expected

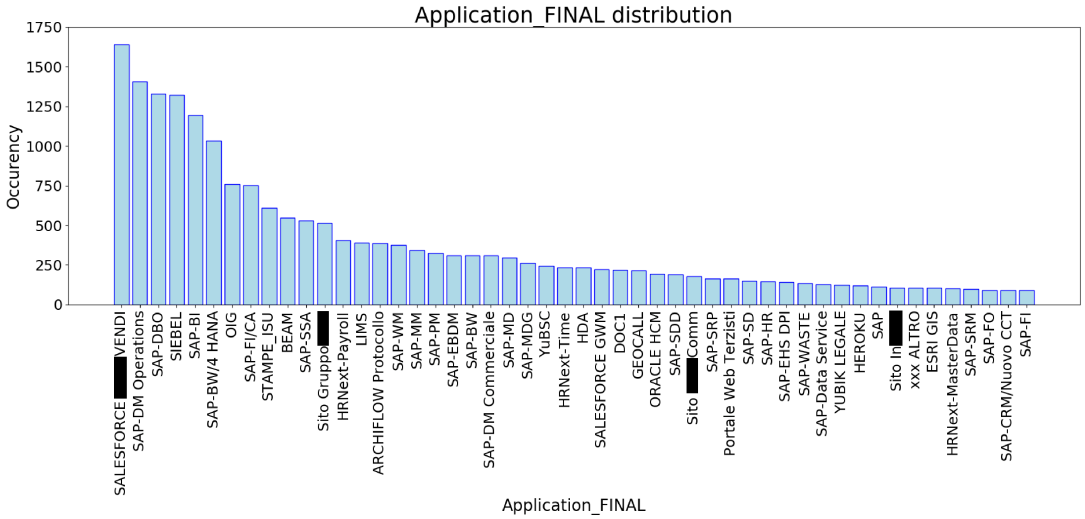


Figure 5.4: First fifty Application_FINAL distribution.

is higher than the previews dataset in which there was Chat Bot messages.

This information will assume more importance when we will see in details the BERT embeddings.

5.2.2 One hot encoding

I have already mentioned that machines process numbers, which applies to both input text and labels. One-hot encoding is a technique used in machine learning and data analysis to represent categorical variables as binary vectors. It transforms categorical variables into a binary representation where each category is represented by a binary value (0 or 1) in a separate column.

In the example provided in Table 5.4, I have created a simplified illustration. We have three sentences (in this thesis, they correspond to tickets) and their corresponding sentiment labels. Each label (Positive, Neutral, Negative) has its own binary column. For each data point, the respective category is represented by a 1 in its column, while the other columns contain 0s.

This table showcases the one-hot encoded representation of categorical variables. It enables machine learning algorithms to work with categorical data by converting them into a numerical representation. In this thesis, I will utilize one-hot encoding with LSTM and all pretrained BERT models.

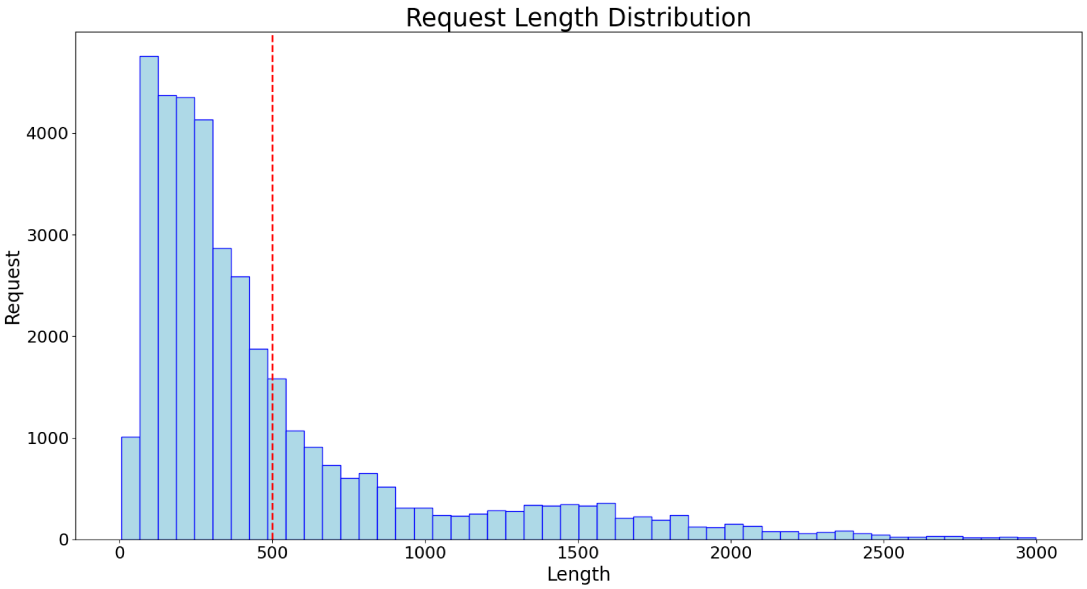


Figure 5.5: Request length distribution.

Sentence	Positive	Neutral	Negative
I had a fantastic time at the beach with my friends.	1	0	0
The weather today is neither too hot nor too cold.	0	1	0
I was disappointed with the poor service at the restaurant.	0	0	1

Table 5.4: One-hot encoding example.

5.2.3 Text preprocessing

Cleaning text

HDA can handle request in different way, it offers the possibility to change the format of the text editor in html and it can also receive/send emails, when a ticket is opened by sending an email, it's been copied all the information about sender, receiver, carbon copy (CC) etc. . This means that inside the *Request* I can find some html tags, special characters like *newline* or *tabular*, and all the noise that could have an email text format. Here an example:

```
   DA: (EMAIL)
A: NOME (EMAIL)
CC: (EMAIL), (EMAIL), (EMAIL)
OGGETTO: [Website feedback] Segnalazione blocco
```

```
ID UTENTE: 12345678 (URL [1]) ha inviato un
messaggio usando il form contatti su URL [2].
Buongiorno, segnalo blocco in [ID SERVER]
[FIRMA UTENTE]

[1] URL
[2] URL
```

Inside the tickets there are a lot of words or symbols that do not add any useful information, even the so called *stopwords* or link to some website, all of them are removed with the same method used in the previews section, *regular expression*.

Lemmatization

Lemmatization is the process of grouping inflected words together to analyze them as a single unit, identified by the word's lemma or dictionary form. Simplemma is a library [42] that offers a straightforward and multilingual approach to find base forms without requiring morphosyntactic information. It is designed to be fast, user-friendly, and suitable for a wide range of cases, although it may not be as powerful as more complex lemmatization solutions. Simplemma is particularly useful in low-resource contexts, educational settings, or as a baseline system for lemmatization and morphological analysis. Currently, it supports 49 languages.

This process is beneficial for reducing the size of the vocabulary. By reducing words to their lemmas, the vocabulary size decrease by 9% in this case. Table 5.5 presents some examples of lemmatization applied to a verb and a plural word.

Words	**Lemma**
giocare, giocato, giochiamo, gioco	giocare
informazioni, informazione	informazione

Table 5.5: Example of lemmatization process.

5.2.4 Word analysis

Performing a word analysis can be useful in various natural language processing tasks. It allows us to identify the most frequent words, patterns, and structures in the text, which can provide insights into the language and content of the text. By analyzing the words, we can identify the key concepts and topics that are being discussed, and this can help in determining the appropriate preprocessing steps, such as verify stop-word removal and lemmatization. By analyzing the frequency of specific words or phrases, we can identify potential biases, anomalies, or errors in the text, which can inform further analysis or the refinement of the NLP model. In the figure 5.6 and 5.7 we can notice the difference between the analysis before and after preprocessing, the first position in the original text are occupied by *punctuation* and *stopwords* and than instead, after the preprocessing, we can see that the most frequent tokens are more informative words. This analysis assists in determining the effectiveness of

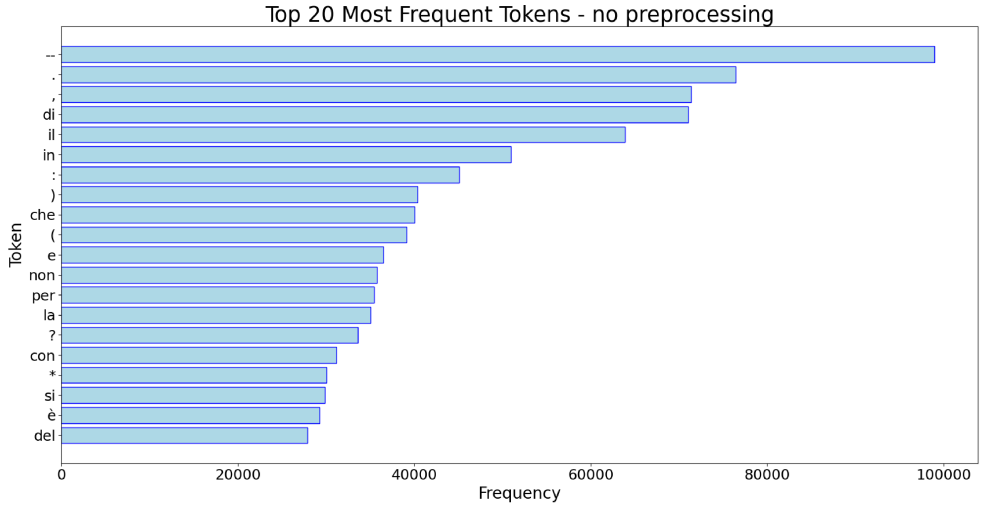


Figure 5.6: Top 20 tokens frequency in the dataset without preprocessing.

stop-word removal and lemmatization processes, as well as identifying tokens or words that can be removed. For instance, words like *anche* or *oppure* are typically deleted. However, as already mentioned, caution must be exercised not to remove too many words, as they might be useful for BERT in understanding the ticket context.

Table 5.6 presents a summary of the vocabulary size at different preprocessing stages.

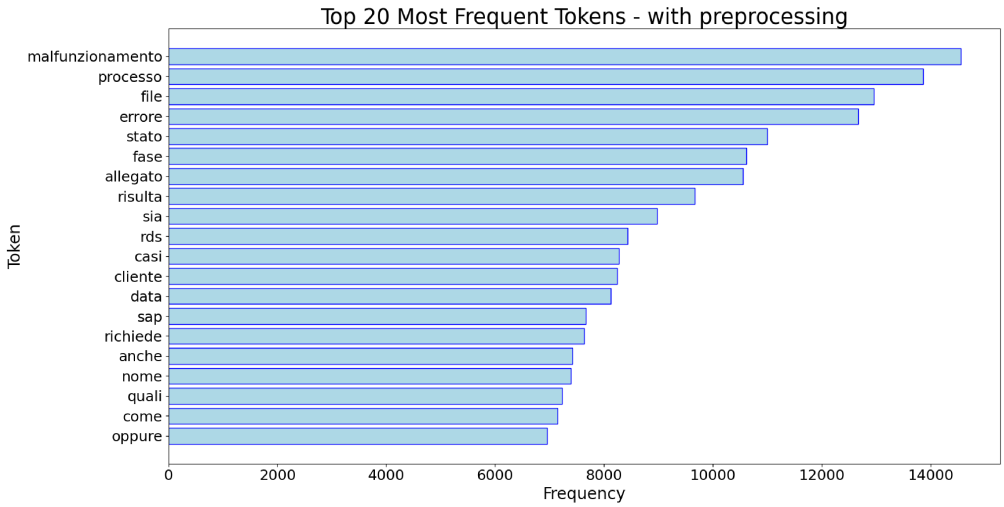


Figure 5.7: Top 20 tokens frequency in the dataset preprocessed.

Vocabulary size	**Size**	**Size reduction**
Original	94727 tokens	0%
After text cleaning	68255 tokens	28%
After Lemmatization	61841 tokens	9% (34% of original)

Table 5.6: Vocabulary size after preprocessing.

5.2.5 Dealing with Class Imbalance

In any classification problem, having a balanced dataset is crucial for achieving optimal model performance. However, in real-world scenarios like this, achieving a balanced dataset is often challenging.

As mentioned in the *Label distribution* section of this chapter, the dataset exhibits a high degree of imbalance, with certain classes having significantly more examples than others. This class imbalance can negatively impact the classification performance, as the model may become biased towards the majority class.

Addressing this issue requires careful consideration. One approach is to handle the class imbalance directly in the network architecture implementation by assigning appropriate class weights. However, with 190 classes, applying grid search methods becomes resource-intensive in terms of computation power and

time.

Another way to tackle class imbalance is by generating synthetic data for minority classes. This involves creating new data points that are similar to the existing data but have slight variations, thereby improving the model's ability to distinguish between classes. In this section, I will introduce the Markov Chains method [2], which is widely used and highly regarded in the scientific community for its simplicity and low resource requirements.

Markov Chains

As explained in [2] this method is based on the definition of Markov Chains [4], it is used to generate synthetic text based on transition probabilities between terms. By looking at the figure 5.8 it could be explained as follows: given some stages $S = s_1, s_2, ..., s_r$, transition probability $p_{i,j}$ is the probability that the stage s_i transit to the stage s_j. The probability of the current transition is independent of previous transition. The start stage may be chose particularly or randomly selected by the probability p_i.

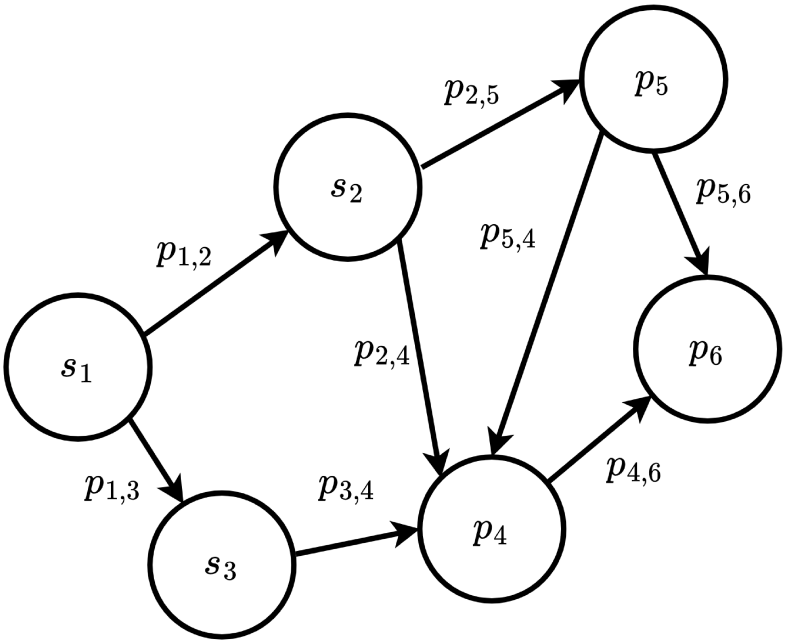


Figure 5.8: Markov Chains definition.

In text generation, Markov Chains generates text with random walking. Each stage represents the token or the word, the transition probabilities of terms are learned by exploring all text in the corpus during the training step. When

the training step already finished, the synthetic text can be generated as the following steps:

1. **Seed word:** The start term can be particularly chosen or used with the weighted random select. The weight of each term is determined by the probability of the occurrence of term in the corpus.

2. **Next term:** The next terms can be generated by weighted random select. The weight of each term is determined by the probability of the occurrence of term when previous term is given. Which can be calculated by the following equation:

$$W_{t_i} = P(t_i|t_{i-1}) \tag{5.1}$$

When t_i denotes the considering term, t_{i-1} denotes the previous term. W_{t_i} denotes the weight of t_i that is the probability of the occurrence of term when previous term is given.

In practise, based on its original paper in Python there is the library Markovify [27] which implement the methods required to generate data, in this project I have generate text for some minority classes. In tables 5.7 5.8 some examples:

Original ticket	**Application_FINAL**
Il processo fase che presenta il malfunzionamento: Tipo Sottotipo TipoOperazione Servizio Dispositiva Contratti Indica il tuo Nome Cognome e Telefono per eventuali contatti relativi a questo TK *OPERATOR NAME* Qual è il comportamento anomalo riscontrato nel processo fase L order *ORDER ID* risulta in attesa conferma.	SALESFORCE ****VENDI
Si segnala l interruzione del *SERVER ID* in data *DATE* Errore: *ERROR CODE* lanciato *DATE*, variante INR_NUMERI, nome utente *USERNAME* La sessione int. è stata interrotta con l errore di run time sopra citato	SAP-DM Operations
Si richiede di sbloccare il seguente *PROCESS ID* Grazie, saluti	SAP-SSA

Table 5.7: Some original ticket from the Energy company dataset.

The results achieved so far are satisfactory. However, as I will discuss in the subsequent chapters when evaluating the model with the dataset trained using synthetic data, there is no substantial improvement in performance. The reason for this is quite evident. The synthetic data generation method works best when classes have a substantial corpus and a large vocabulary size to train on. In this

Synthetic ticket	Application_FINAL
Il processo fase Che Salesforce quando si è attivato solo il Cambio offerta sia *** che *** deve scendere il tipo voltura corretto Dopo esserti confrontato con i colleghi che lavorano su partenr community si richiede bonifica dello stato della pratica Rif. *ID PRATICA*	SALESFORCE ****VENDI
Si segnalano le seguenti *PROCESS ID* in errore per: Inserire i dati nel tracciato XML non ci consente di iniziare le lavorazioni per la corretta prosecuzione della campagna di telelettura.	SAP-DM Operations
Si richiede gentilmente di sbloccare i seguenti *PROCESS ID* in stato *STATE ID* e fase *ID* consolidato. Grazie	SAP-SSA

Table 5.8: Some examples of synthetic tickets generated with Markov Chains.

case, some minority classes have less than 5 tickets, resulting in an insufficient vocabulary size to generate high-quality synthetic data. In Chapter **??**, I will explore a more powerful method that can be used for this purpose, albeit at a higher resource cost.

5.3 Train, validation and Test split

During model implementation and evaluation, splitting the dataset into train, validation, and test sets is a critical step. The following are the key characteristics of each set:

- **Train set:** The training set is used to train the model. It consists of labeled examples that the model uses to learn the underlying patterns and relationships between input features and target outputs. The model adjusts its parameters based on the training data to minimize errors and enhance its predictive capabilities.
- **Validation set:**The validation set is used to fine-tune the model during the training process. It helps in assessing the model's performance on unseen data and optimizing its hyperparameters. By evaluating the model on the validation set, I can make adjustments and improvements to enhance its performance.
- **Test set:**The test set is used to evaluate the final performance of the trained model. It serves as an unbiased measure of how well the model can generalize to unseen data. The test set should be representative of the real-world data the model will encounter. By evaluating the model on the test set, I can obtain an unbiased estimate of its performance and assess its ability to make accurate predictions in practical applications.

6

Language detection models evaluation: FastText, Spacy, Cybozu

6.1 Models overview

n this section, I will walk through all the implementation steps of the Language Detection model. I will first examine the details of the approach used by the three tested tools and then compare their performances to make a choice. Additionally, I will perform some postprocessing to optimize the model for the intended purpose.

6.1.1 N-gram approach

As mentioned in Chapter 3, both Cybozu's solution and Fasttext are based on the n-gram approach.

To illustrate this approach, consider Table 6.1. If we focus on the 2-gram approach, we would break down the sentence into all possible pairs of consecutive words. Each pair is considered a feature and is assigned a weight based on its frequency in the training data. These weights are then used to classify new text based on the presence or absence of these features. Both Cybozu Lab and Fasttext employ variations of the n-gram approach for language detection. Cybozu Lab uses 5-gram character sequences, treating each word as a separate entity by splitting the characters. On the other hand, Fasttext combines n-gram character and word sequences.

Sentence	The quick brown fox jumps over the lazy dog
1-gram	[The], [quick], [brown], [fox], [jumps], [over], [the], [lazy], [dog]
2-gram	[The quick], [quick brown], [brown fox], [fox jumps], [jumps over], [over the], [the lazy], [lazy dog]

Table 6.1: 1-gram and 2-gram approach applied to the words sequence.

This approach is effective because it captures not only individual words but also the context in which they appear. Some words alone may not provide sufficient information to determine the language they belong to, but considering the surrounding context enhances the accuracy of language detection.

6.1.2 Neural network based

Fasttext utilizes a neural network in addition to the n-gram approach to improve accuracy. It employs supervised learning, training on a large corpus of text with known language labels to create language-specific models for language detection.

In contrast, spaCy's language detection algorithm does not rely on the n-gram approach. It employs a convolutional neural network (CNN) model trained on extensive text data in various languages. The model uses character-level embeddings and convolutional layers to extract features from input text. These features are then passed through a fully connected neural network to predict the language.

Pretrained models

Each method utilizes pretrained models. In the case of the `langdetect()` method in Cybozu's detector, there is no need to manually load any language model. It employs an internal set of probability models that are built into the library for language detection.

For Fasttext and spaCy, pretrained models need to be loaded. SpaCy offers various tools, including language detection, and users can download the most suitable model for their specific task from the official spaCy model hub [43].[20], more details in the table 6.2.

	Model's name	Main sources	Size
SpaCy	en_core_web_sm	OntoNotes 5 [31] ClearNLP Constituent-to-Dependency Conversion [6] WordNet 3.0 [52]	12MB
Fasttext	lid.176.bin	Wikipedia Tabotea [46] SETimes [40]	126MB

Table 6.2: Details on pretrained models used for the language detector.

6.2 Model evaluation

In this section, I will present the results of the tests conducted on the dataset. First, I will analyze the two labeled datasets, namely the Papaluca dataset and the Language Detection Dataset, to assess the real accuracy performance. Then, I will evaluate the performance on the internal dataset using the methods explained in the previous chapters.

6.2.1 Test results

In table 6.3 is shown that every methods perform well on the test dataset, SpaCy and Cybozu achieve the same results and it seems that Fasttext outperformed them in terms of accuracy.

Method	Accuracy on Language Detection dataset	Accuracy on Papaluca dataset
Cybozu	95%	92%
SpaCy	95%	92%
Fasttext	99%	96%

Table 6.3: Accuracy on the test dataset.

Regarding the internal *Gaming Company Chat Dataset*, I compared the language detected in a single chat message with the language detected using the entire conversation as input. I used the two methods described in Chapter 5. Additionally, an important aspect to evaluate is the time required to detect the languages. The final model will be used to detect language in a large amount of

data, not just in a ChatBot. The results are presented in Tables 6.4 and 6.5 below.

The correspondence metric suggests that Fasttext is slightly more consistent

Method	**Correspondence**	**Mean time per single string**	**Languages detected** [1]
Cybozu	79%	17.3 ms	39/37
SpaCy	79%	5.19 ms	39/37
Fasttext	88%	19.2 μs	80/58

Table 6.4: Correspondence on the internal gaming dataset using the the output of the language detected with the entire conversation as input.

Method	**Correspondence**	**Mean time per single string**	**Languages detected**[1]
Cybozu	81%	17.3 ms	39/37
SpaCy	81%	5.19 ms	39/37
Fasttext	89%	19.2 μs	80/58

Table 6.5: Correspondence on the internal gaming dataset using the personalized method in figure 5.3.

in its results, as there is less difference between message-to-message detection and chat-to-chat detection. It is important to note that at this stage, I am not evaluating the accuracy of the detection, but rather the robustness on short text data.

Regarding the time differences between the methods, Fasttext performs the best on average, while Cybozu's solution is the slowest.

As expected, the correspondence values in Table 6.4 are slightly lower compared to those in Table 6.5, which confirms the information presented in Chapter 5.

Another aspect to highlight is the number of languages detected. When using the entire chat session as input, fewer languages are detected. This indicates

[1]The first number refers to the number of languages detected with each single chat messages as input, the second instead refers to the number of languages detected using the entire chat conversation as input.

that the methods are more precise when applied to long text data.

6.3 Hybrid solution

Based on the analysis of the results, the critical point is the length of the sentences. When the message is too short, the models tend to detect the wrong language. To address this issue, I propose a hybrid solution.

The hybrid approach involves using the language detection model when I am reasonably confident that the detected language is correct. However, under certain conditions, I will perform a search in a special dictionary created based on the context in which I am working.

Before discussing the solution in detail, I need to define some preliminary steps, which are explained in the following sections.

6.3.1 Reduce the number of languages

During the tests on the internal dataset, it became evident that the number of detected languages is overestimated. Since I am working within a specific context and are aware of the languages used by the final users, I can reduce the number of detectable languages by defining a vector of *admitted languages*.

This vector is used to filter the model's output. It is one of the conditions applied during the postprocessing step. If the detected language is not present in this vector, the *default language* will be used. The default language can be pre-defined or chosen based on the browser language. In this project, the vector of admitted languages consists of the following eight principal languages:

```
admitted languages = [Italian, English, German, French, Spanish,
                      Portuguese, Polish, Russian]
```

6.3.2 Configuration utils

I have already discussed the issue of short text. To address this problem, a special dictionary has been created for each admitted language. These dictionaries contain words that I want to associate with a specific language. The goal is to populate these dictionaries in a personalized way. For example, I can decide that the word *menu* should be associated with the Italian language because I know that the majority of final users are in Italy. This prevents situations where

words are misidentified as belonging to a different language, as would be the case with Fasttext's identification of *menu* as *id: Indonesian*.

The words that populate the dictionary can be personalized for each customer based on the context and business environment. In this case, a pre-detection was performed, and the most frequent words for each language were selected. After manual inspection, these words were inserted into the respective dictionaries.

6.3.3 Postprocessing: Hybrid Choice

I have all the necessary components to delve into the details of how the final decision is made. Figure 6.1 represents the structure of the postprocessing algorithm implementation.

The first step is to make a prediction using the original pretrained model. If the output probability and the text length exceed their corresponding thresholds (the threshold for probability, Th_prob, is set to 70%), and the detected language is in the admitted languages vector, the model's result is chosen.

If both thresholds are not met, the algorithm performs a check in the dictionary. The dictionary check works as follows: each word in the message is checked for its presence in the dictionary, and the dominant language associated with the words is chosen. It is possible that no words are found in the dictionary. In this situation, if the language predicted by the model is allowed (i.e., present in the admitted languages vector), I trust the model's prediction, even if the probability is lower than the threshold. Otherwise, the default language is chosen.

The dictionary check can be performed even if the conditions for choosing the model's prediction after the first step are met. This occurs when the probability is high, but the language is not allowed. In this case, if the text length is below the threshold (to save resources and time), the check is performed.

6.3.4 Choice of char threshold

As explained in the previous section, I activate the hybrid solution when the text length falls below a certain character threshold. To determine the optimal threshold, a grid search was performed on the truncated text of the Language Detection Dataset, using only the first N characters in each sentence (`N = [5, 10, 15, 20, 30, 50]`). The grid search was conducted using Fasttext, given its fast performance.

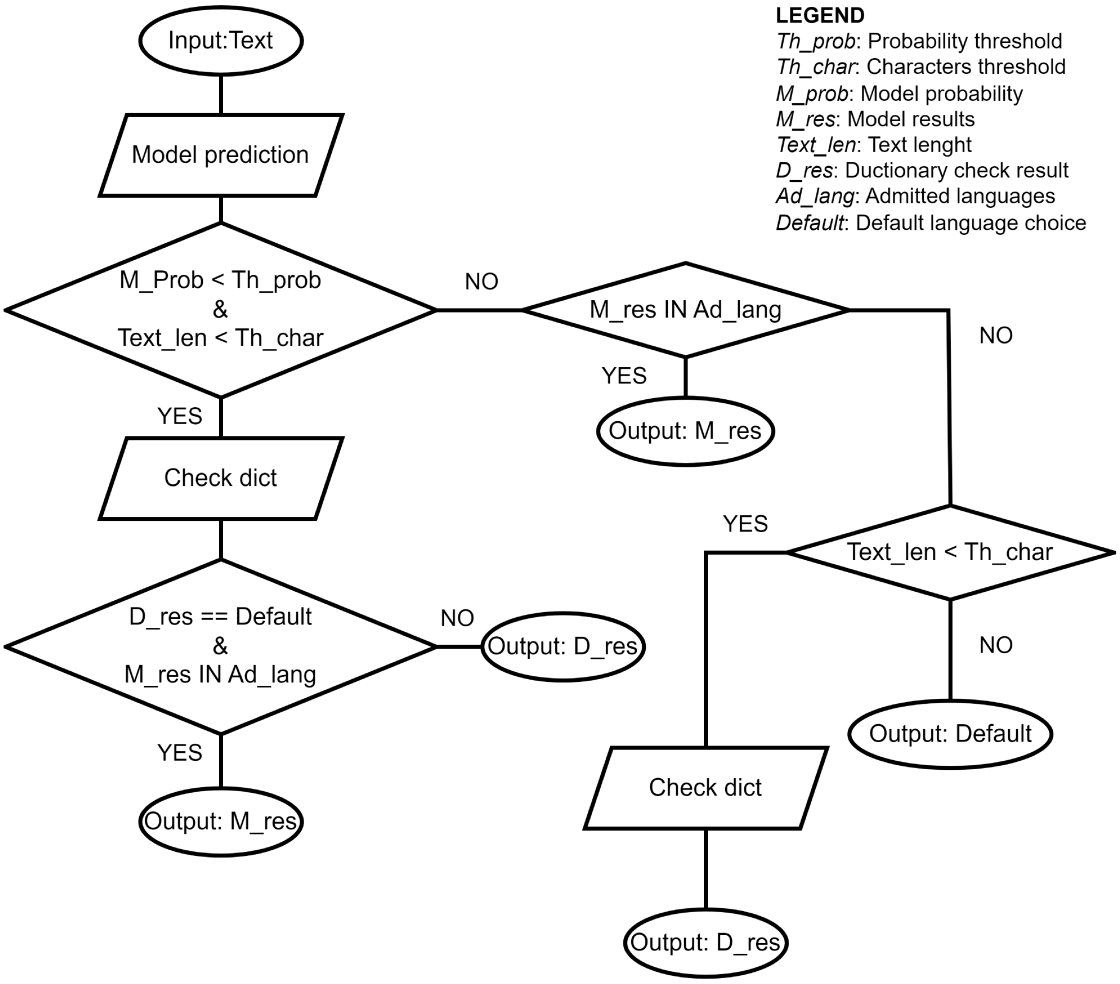


Figure 6.1: Hybrid approach diagram.

By analyzing the results shown in Figure 6.2, the accuracy starts to reach reasonable levels after the character threshold of 15. Therefore, I have set the character threshold for the hybrid approach to 15.

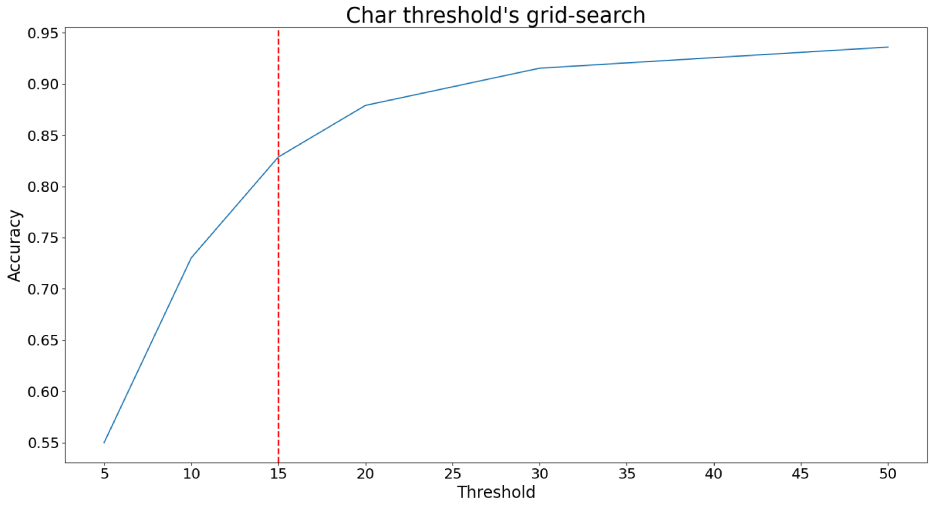


Figure 6.2: Grid-search of the best chat threshold.

6.4 Final results

The same starting test were performed using the hybrid approach, results in tables 6.6, 6.7, 6.8:

Method	**Accuracy on Language Detection dataset**	**Accuracy on Papaluca dataset**
Cybozu	99.2%	99.5%
SpaCy	99.2%	99.5%
Fasttext	99.2%	99.5%

Table 6.6: Accuracy on the test dataset using the hybrid approach.

[2]In brackets the improvement in percentuage respect the results without the hybrid approach.

Method	Correspondence[2]	Mean time per single string	Languages detected
Cybozu	84% (+5%)	32.6 ms	8
SpaCy	90% (+2%)	9.8 ms	8
Fasttext	84% (+5%)	6.6 μs	8

Table 6.7: Correspondence with hybrid approach on the internal gaming dataset using the the output of the language detected with the entire conversation as input.

Method	Correspondence[2]	Mean time per single string	Languages detected
Cybozu	89% (+8%)	32.6 ms	8
SpaCy	89% (+8%)	9.8 ms	8
Fasttext	94% (+5%)	6.6 ms	8

Table 6.8: Correspondence with hybrid approach on the internal gaming dataset using the personalized method in figure 5.3.

Fasttext

Upon examining the results achieved with the hybrid approach, each model demonstrates a higher correspondence, indicating increased robustness to short text. The time required for prediction is slightly longer due to the postprocessing operation, but it remains reasonable.

The final model selected for deployment is Fasttext by Facebook Inc. It has shown to be the most robust model for short messages and it is faster compared to the other models. As mentioned earlier, this model will be utilized not only in a Chat Bot system but also in the creation of a Python library, as described in Chapter 8. The library will enable users to perform language detection and generate statistics on the number of languages detected in a dataset. This emphasizes the importance of fast predictions, as even a small difference in the prediction time of a single string can become crucial when applied to a large amount of data.

7

Ticket classification model development and performance evaluation

In this chapter, I will discuss the implementation of a ticket classification model using various machine learning algorithms and natural language processing (NLP) techniques. I will focus specifically on the internal *Energy Company Ticket Dataset*. My aim is to develop an effective model for classifying tickets in the context of the HDA Pat software.

To begin, I will introduce the baseline models that serve as reference points for evaluating approaches. These baseline models provide a benchmark against which I can compare the performance of the models.

Next, I will delve into the implementation details of the models I experimented with. Specifically, I will explore the Support Vector Machine (SVM), StarSpace, Long Short-Term Memory network (LSTM), and Bidirectional Encoder Representations from Transformers (BERT) models. For each model, I will provide an overview, explain the architecture and training approach, and present the results of the experiments.

Finally, I will analyze the strengths and weaknesses of each model and propose a solution that best aligns with the requirements of the HDA Pat software.

7.1 Baseline models and SVM

When developing a ticket classification model, it is crucial to establish a baseline performance that can serve as a reference point for evaluating the effectiveness of more complex models. Baseline models are relatively simple algorithms that provide a basic level of classification metrics.

In this section, I will discuss the Support Vector Machine and two widely used baseline models: Naive Bayes and Logistic Regression.

7.1.1 Naive Bayes

Let's start with the implementation of a simple model, Naive Bayes. As already mentioned in Chapter 2, this algorithm is based on the Bayes Theorem of probability. Naive Bayes is known for its simplicity and efficiency, making it a popular choice for text classification tasks.

As an initial step, I will consider the entire dataset. The details of the division between the training and test sets can be found in Table 7.1. Additionally, Table 7.2 provides a partial distribution of the classes, allowing us to examine how the models perform on both majority and minority classes.

Subset	**Number of ticket**	**Number of classes**
Train set	19393	190
Test set	2177	190

Table 7.1: Dataset splitting details for baseline models.

Thanks to the scikit-learn Python library, developing a Naive Bayes classifier is an easy task. The Multinomial Naive Bayes classifier is already predefined in the package. By calling `sklearn.naive_bayes.MultinomialNB()` and fitting it to the data, I can obtain the first results, which are shown in Table 7.3.

Is possible to observe that the performance of the Naive Bayes classifier is poor. Even with the applied preprocessing, there is only a slight improvement, but the results are still unsatisfactory. By examining Table 7.4, which shows the results of training with the preprocessed dataset, it is understandable that this is primarily due to several classes having a lack of tickets. This problem highlights how class imbalance negatively affects the performance. This is evident from the

Class	Ticket	% of total
SALESFORCE ****VENDI	1640	7.6%
SAP-DBO	1331	6.2%
SIEBEL	1322	6.2%
SAP-BI	1195	5.5%
OIG	760	7.7%
HRNext-MasterData	100	0.5%
AZURE DataBricks	66	0.3%
App Prevent. Lavori	40	0.2%
Sito Estenergy	10	0.01%
RPA-DOE	8	0.001%

Table 7.2: Partial class distribution of classes used to show results of the first part of the model evaluation that regards the baseline models and the SVM.

	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Without preprocessing	12%/44%	6%/39%	6%/32%	39%	0.3s	0.03s
With preprocessing	14%/47%	8%/44%	9%/38%	44%	0.3s	0.04s

Table 7.3: Naive Bayes classification results.

macro average, which is less than 10%. However, it does confirm the simplicity, usability, and efficiency of the Naive Bayes classifier, as it does not require many resources and provides quick results.

7.1.2 Logistic regression

The Logistic Regression algorithm was implemented using the same data split as shown in Table 7.1. The logistic regression model is readily available in the scikit-learn library as `sklearn.linear_model.LogisticRegression()`. Let's evaluate the results obtained from this implementation:

Looking at the results in Table 7.5 and 7.6, the logistic regression algorithm outperforms the Naive Bayes algorithm. The overall metrics show significant improvement, but the macro average still highlights the problem of class im-

Class	Precision	Recall	F1-score	Support
SALESFORCE ****VENDI	81%	95%	87%	164
SAP-DBO	72%	83%	77%	133
SIEBEL	79%	89%	84%	132
SAP-BI	73%	80%	76%	120
OIG	80%	87%	83%	76
HRNext-MasterData	80%	40%	53%	10
AZURE DataBricks	40%	57%	47%	7
App Prevent. Lavori	33%	25%	29%	4
Sito Estenergy	0%	0%	0%	1
RPA-DOE	0%	0%	0%	1

Table 7.4: Naive Bayes single classes results: In the first half the results of the most populated classes, in the second half minority classes (Support field refering to test set).

	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Without preprocessing	34%/65%	25%/68%	27%/64%	68%	161.4s	0.05s
With preprocessing	35%/66%	25%/68%	27%/64%	68%	150.5s	0.05s

Table 7.5: Logistic regression classification results.

balance in the distribution. Another noteworthy factor from Table 7.5 is the increased training time compared to the Naive Bayes algorithm. Logistic regression may struggle with high-dimensional data, as is often the case in text classification tasks, where the feature space can be large, especially when using techniques like bag-of-words or TF-IDF. This can result in increased computational requirements and potential sparsity issues.

7.1.3 Support Vector Machine

Support Vector Machines (SVM) are widely used for text classification due to their ability to handle high-dimensional data and capture complex decision

Class	Precision	Recall	F1-score	Support
SALESFORCE ****VENDI	74%	96%	84%	164
SAP-DBO	52%	80%	63%	133
SIEBEL	70%	89%	78%	132
SAP-BI	52%	80%	63%	120%
OIG	63%	86%	73%	76
HRNext-MasterData	100%	20%	33%	10
AZURE DataBricks	67%	29%	40%	7
App Prevent. Lavori	0%	0%	0%	4
Sito Estenergy	0%	0%	0%	1
RPA-DOE	0%	0%	0%	1

Table 7.6: Logistic Regression single classes results: In the first half the results of the most populated classes, in the second half minority classes.

boundaries. In this section, I will discuss the results obtained from training SVM models for text classification and explore different possibilities for training SVMs, focusing specifically on approaches well-suited for text classification tasks.

One crucial aspect to consider is the choice of the kernel function, which plays a significant role in SVMs by mapping the data to a higher-dimensional feature space. In text classification, commonly used kernel functions include linear, polynomial, and radial basis function (RBF) kernels. In the following tables, namely Tables 7.7, 7.8, and 7.9, you can compare the results of different kernel implementations. The kernel functions used are as follows:

- *sklearn.svm.LinearSVC():* This function is optimized for liner kernel implementation.
- *sklearn.svm.SVC(kernel='poly | rbf', degree=2):* This is used to test the polynomial and the RBF kernels.

It is evident that the best solution is to use the linear kernel with its optimized function. By examining the performance metrics, it is the superior choice in terms of both metrics and training time. It outperforms the polynomial and RBF kernels, particularly in terms of robustness on the minority classes. However, it is worth noting that even though the linear kernel performs better, there is still space for improvement in terms of the minority classes' performance.

Linear kernel	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Without preprocessing	54%/75%	49%/76%	50%/75%	76%	5.4s	0.01s
With preprocessing	55%/75%	51%/77%	52%/75%	77%	3.5s	0.01s

Table 7.7: SVM classification results using liner kernel.

Polynomial kernel (d=2)	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Without preprocessing	40%/70%	26%/66%	30%/64%	66%	445.4s	18.7s
With preprocessing	40%/71%	26%/66%	29%/64%	66%	351.4s	15.4s

Table 7.8: SVM classification results using polynomial kernel.

RBF kernel	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Without preprocessing	41%/70%	29%/70%	32%/67%	70%	351s	19.2s
With preprocessing	42%/71%	30%/70%	33%/67%	70%	272.3s	16s

Table 7.9: SVM classification results using Radial Basis Function kernel.

7.1.4 First consideration after the baseline models and SVM implementation

It appears that SVM has outperformed both Naive Bayes and Logistic Regression in terms of performance. Naive Bayes algorithm is the fastest (0.3s), while SVM with a polynomial kernel (degree = 2) is the slowest (6m) when preprocessing is applied. Preprocessing has shown a general improvement in metrics, but it is clear that using a restricted vocabulary leads to a less resource-intensive training phase. However, SVM also faces the same problem as the baseline models. The analysis of the single class results in tables 7.10, 7.6, and 7.4 reveals that the precision, recall, and f1-score for the minority classes are very low, sometimes even 0%. This indicates the need to revisit the analysis and preprocessing steps in order to find a solution to this issue, if one exists.

Class	Precision	Recall	F1-score	Support
SALESFORCE ****VENDI	81%	95%	87%	164
SAP-DBO	72%	83%	77%	133
SIEBEL	79%	89%	84%	132
SAP-BI	73%	80%	76%	120
OIG	80%	87%	83%	76
HRNext-MasterData	80%	40%	53%	10
AZURE DataBricks	40%	57%	47%	7
App Prevent. Lavori	33%	25%	29%	4
Sito Estenergy	0%	0%	0%	1
RPA-DOE	0%	0%	0%	1

Table 7.10: Support Vector Machine single classes results (linear kernel): In the first half the results of the most populated classes, in the second half minority classes.

7.2 Refining Analysis and Preprocessing for Improved Model Performance

Given the initial results, it is necessary to review the preprocessing phase to address the class distribution issue. In some cases, there are not enough tickets belonging to certain classes to ensure proper training. Therefore, classes that fall below a certain ticket threshold will be removed.

In this section, I will conduct several tests to determine the optimal threshold for improving the macro average, which is a significant metric for evaluating performance on minority classes. For instance, if I set the threshold at 1000 tickets, it means that I will train the models only with those classes that have at least 1000 tickets. I will perform tests using the following threshold values:

```
Ticket_threshold = [1000, 600, 400, 200, 50, 25]
```

7.2.1 Model robustness on different ticket threshold

In table 7.11, you can find the corresponding details of the datasets used for the grid search of the correct threshold. It is important to note that the threshold

was applied to the entire dataset, so it is not directly related to the minimum number of tickets used for training.

Ticket threshold	**Number of ticket**	**% of total**	**Number of classes**
1000	7929	36.8%	6
600	10049	46.6%	8
400	12042	55.8%	13
200	16716	77.5%	29
50	20193	93.6%	62
25	20804	96.4%	80

Table 7.11: Dimension of different dataset depending on ticket threshold.

The results in table 7.12 suggest, as expected, that with a consistent number of tickets and a balanced dataset, all the models perform well. The SVM achieves around 90% accuracy. The Naive Bayes algorithm starts to consistently decrease in performance from a threshold of 400, while the logistic regression algorithm collapses starting from a threshold of 50. The SVM, once again, is the only model that maintains good performance across all tested thresholds. Even in the last stage, it still achieves an F1-score of 70%, which is a good result considering the dataset's unbalance and the presence of 80 classes.

7.2.2 Final threshold choice: 25 tickets

After careful analysis and consideration, I have determined that setting a threshold of at least 25 tickets per class for the training phase is the optimal choice. This decision is based on several factors, including the performance of the SVM model and the usability of the resulting classification system.
As observed in the previous paragraph, the SVM performs well even when the threshold is set to 25 tickets. Another important consideration is the practical usability of the model. While high performance and accuracy are important, it is equally crucial to cover a diverse range of classes in the ticket classification system. By choosing a threshold of 25 tickets per class, I can include a broader spectrum of classes in the training set, making the model more versatile and applicable to real-world scenarios. On the other hand, setting a higher threshold, such as 1000 tickets per class, might yield higher performance scores, but it

Ticket threshold	Model	Precision	Recall	F1-score	Accuracy
1000	**SVM**	**89%**	**88%**	**88%**	**89%**
1000	Log. Reg.	86%	86%	86%	86%
1000	N. Bayes	81%	78%	79%	79%
600	**SVM**	**88%**	**87%**	**88%**	**88%**
600	Log. Reg.	85%	83%	84%	84%
600	N. Bayes	82%	71%	74%	74%
400	**SVM**	**90%**	**89%**	**90%**	**89%**
400	Log. Reg.	88%	86%	87%	85%
400	N. Bayes	86%	70%	75%	73%
200	**SVM**	**85%**	**81%**	**83%**	**84%**
200	Log. Reg.	83%	75%	77%	79%
200	N. Bayes	75%	41%	46%	57%
50	**SVM**	**82%**	**72%**	**75%**	**79%**
50	Log. Reg.	78%	57%	62%	73%
50	N. Bayes	35%	20%	22%	48%
25	**SVM**	**77%**	**67%**	**70%**	**78%**
25	Log. Reg.	69%	48%	53%	71%
25	N. Bayes	27%	16%	17%	45%

Table 7.12: Results of ticket threshold grid-search.

would significantly limit the number of classes covered.
Therefore, I believe that the chosen threshold strikes a balance between performance and usability. It ensures that the model performs well while covering a sufficient number of classes, enabling it to handle a diverse range of ticket classification tasks effectively.

From this point forward, I will continue to use the SVM due to its good performances. However, I will set aside the baseline models since their performances on minority classes make their usability not worthwhile.

7.2.3 Concatenation of the Subject field

During the model implementation, it was discovered that some tickets had an empty or uninformative request field, while the relevant information was

mentioned in the subject field. This is another case in which I came back to analysis and preprocessing (Figure 1.1). In table 7.13, there are a few examples of this. To make the training more effective, a decision was made to concatenate the subject field with the request field, treating them as a single text input. This approach allowed the inclusion of additional information from the subject field, enhancing the training data and potentially improving the model's performance in understanding the tickets. By leveraging both fields together, the model can benefit from a more comprehensive representation of the ticket content, leading to more accurate classification and better decision-making.

In table 7.14, I have retrained the SVM model with this modification on the data. The overall performances have improved as a result.

Subject	**Request**
RNO in errore	In allegato tematica
problema nell'aggiornamento di due voci del listino prezzi	Buongiorno, in allegato tk per ****
BONIFICA PUNTI LATO SAP	Vedi allegato.
problemi lentezza sap e immagini fatture non disponibili	Si apre ticket come da oggetto

Table 7.13: Some examples of uninformative request.

	Precision	**Recall**	**F1-score**	**Accuracy**
SVM	78%/81%	74%/81%	75%/80%	81%

Table 7.14: Results of SVM by concatenating the Subject and Request fields.

After considering all the factors discussed in this section, the table 7.15 presents detailed information about the dataset utilized for the model implementations in the subsequent sections.

7.3 Long Short Term Frequency Memory

Once I have refined the dataset, I am ready to test more complex models. Let's start with Long Short-Term Memory (LSTM). In this section, I will try a

Subset	Number of ticket	Number of classes
Train/Validation set	18723	80
Test set	2081	80

Table 7.15: Dataset splitting details after refining the analysis and preprocessing.

simple architecture, and I will use the same test dataset that has been used up until this point. The validation set will consist of 20% of the training set.

7.3.1 LSTM network architecture

The implementation of the network is done using sequential node by *keras*, here's breakdown of the different components:

1. **Embedding Layer:**
   - *Input:* The input to the model is a sequence of integers representing words in the text.
   - *Output:* The embedding layer maps each word index to a dense vector representation of fixed size (100). It learns to represent words in a continuous space, capturing semantic relationships between words.

2. **SpatialDroupout1D:**
   - *Input:* The output of the embedding layer.
   - *Output:* SpatialDropout1D applies dropout, a regularization technique, to the embeddings by randomly setting a fraction of features to 0 at each timestep during training. It helps prevent overfitting and encourages the model to learn more robust representations.

3. **LSTM Layer:**
   - *Input:* The output of the SpatialDropout1D layer.
   - *Output:* LSTM layer, it processes the input sequence, updates its internal state, and generates an output at each timestep. The LSTM layer in this architecture has 100 units (output dimensions) and applies dropout and recurrent dropout with a rate of 0.3.

4. **Dense Layer:**
   - *Input:* The output of the LSTM layer.

- *Output:* The dense layer is a fully connected layer that maps the LSTM layer's output to the number of classes (80) in the classification task. It uses the *softmax()* activation function to produce class probabilities, this function is commonly used for multiclass classification.

Regarding the loss function, the model is compiled with the *categorical cross-entropy* loss, which is suitable for multi-class classification problems. The *RectifiedAdam optimizer* from the TensorFlow library is applied with a learning rate of 0.01.

7.3.2 LSTM TEST RESULTS

I trained the LSTM model for 10 epochs, and the results of the prediction are shown in table 7.16. One notable observation is the training time, which takes minutes rather than seconds. It is important to note that I am using the Kaggle platform with GPUs, and this extended training time is primarily due to the complexity of the LSTM network.

Upon analyzing the results, it becomes evident that LSTM requires a significant amount of memory to store and update the cell. Examining the figures 7.1 and 7.2, it is apparent that the quality and size of the dataset alone are not sufficient to prevent overfitting. Therefore, considering the overall results and the resource requirements, I have concluded that LSTM networks are not suitable for this specific type of text classification.

LSTM	**Precision**	**Recall**	**F1-score**	**Accuracy**	**Time to train**	**Time to predict**
Without preprocessing	58%/68%	52%/68%	53%/67%	68%	30m 43s	9s
With preprocessing	57%/68%	51%/68%	52%/67%	68%	31m 31s	10s
Subject concatenated	**62%/72%**	**57%/72%**	**57%/71%**	**72%**	**28m 7s**	**12s**

Table 7.16: LSTM model classification results (complete classification report on Appendix A).

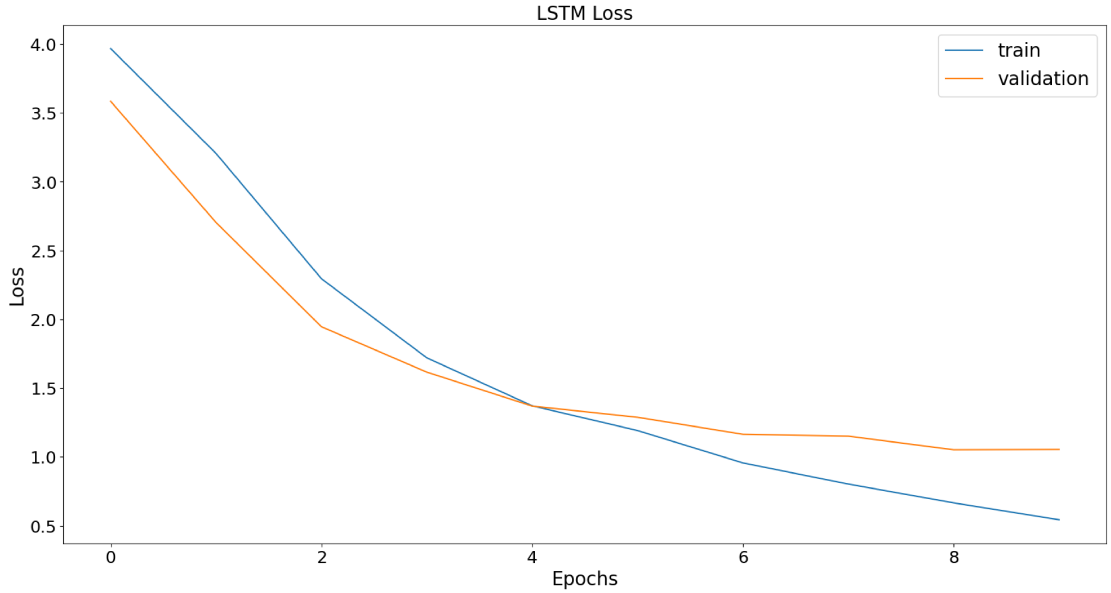


Figure 7.1: LSTM model training/validation loss over epochs.

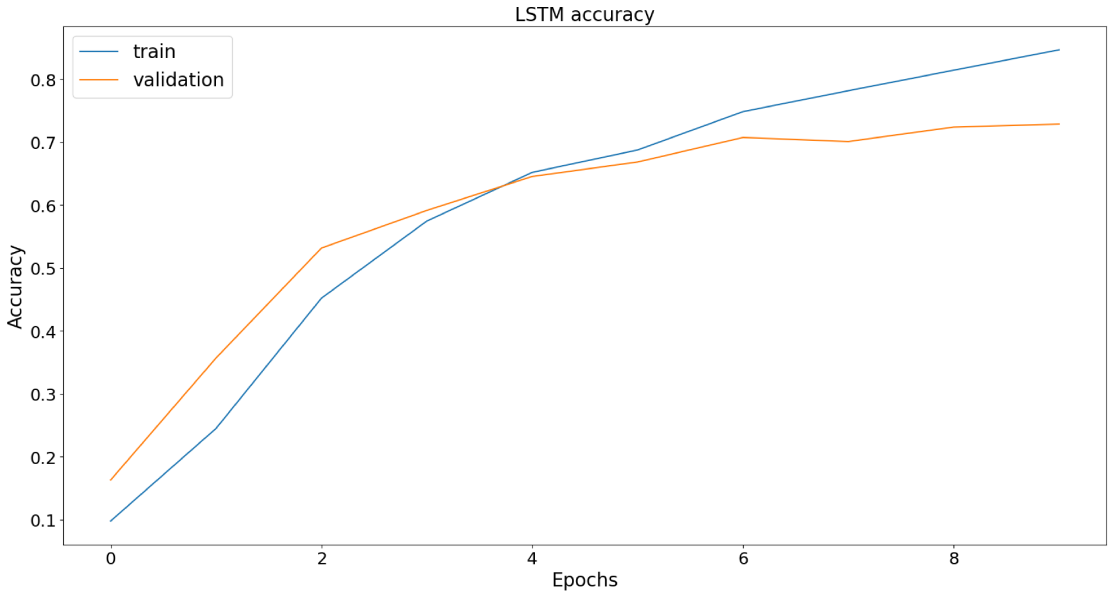


Figure 7.2: LSTM model training/validation accuracy over epochs.

7.4 StarSpace

StarSpace, introduced in Chapter 2, is an embedding-based model that focuses on dense representations of words and documents. It can capture the semantic relationships between them by leveraging co-occurrence patterns in the training data. StarSpace embeddings can be used as input features for downstream classification models. In this project, a Zucchetti private library was utilized, which implements the original network but with a more user-friendly interface. It accepts a list of text along with their corresponding labels as input. Additionally, it allows setting the number of epochs and provides other useful optional parameters. For instance, a list of stopwords can be specified to be excluded during training.

If SVM fails to capture the context of a sentence and LSTM is too resource-intensive, StarSpace could provide a middle ground. One of its most significant advantages is scalability and efficiency, making it an ideal solution for large-scale text classification tasks.

7.4.1 StarSpace test results

In Table 7.17, is displayed the test results of StarSpace. The training time is slightly longer than SVM but significantly shorter than LSTM, confirming its efficiency. The overall results are good, although StarSpace also suffers from data imbalance, as indicated by the approximately 10% difference between macro average and weighted average metrics.

An important aspect to evaluate here is the improvement related to the application of preprocessing, to notice in this case the differences between performances are more evident with respect to the previous models, especially using the concatenation of Subject field , the reason could be related to several factors, the fact that StarSpace is more sensible to the context means also that is less robust to the noise, like links, special characters etc., so in this kind of models the preprocessing part plays a crucial role.

7.5 BERT Fine Tuning

The architecture of BERT was introduced in Chapter 2. In this section, the focus will be on the fine-tuning aspect of BERT.

StarSpace	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Without preprocessing	64%/73%	60%/73%	61%/72%	73%	2m 17s	11s
With preprocessing	67%/75%	61%/75%	62%/74%	75%	2m 22s	10s
Subject concatenated	**73%/78%**	**69%/78%**	**70%/78%**	**78%**	**2m 42s**	**11s**

Table 7.17: StarSpace model classification results (complete classification report on Appendix A).

The output of BERT consists of multiple layers, the number of which depends on the specific variant of BERT being used. The original BERT model developed by Google includes 12 transformer layers for BERT-base and 24 layers for the larger model, BERT-large.

In this master's thesis, two task-specific architectures that handle the output of the BERT layer using different approaches were discovered:

- **4-layer concatenation:** For each input token in the sequence, BERT produces a corresponding contextualized embedding. These embeddings capture the representation of each token based on its context within the input sequence. The token-level output of BERT is a matrix of shape [batch size, sequence length, hidden size], where each element represents the embedding for a specific token in the input sequence. In literature there are several ways in which this layers it's been used [45], we will try to concatenate last four layers to capture a deeper representation of the input text.
- **Pooler output:** BERT also generates a pooler representation that summarizes the entire input sequence. The pooler output is obtained by applying a pooling operation (typically mean or max pooling) over the token-level embeddings. This results in a fixed-size vector of length equal to the hidden size. The pooler output provides a high-level representation of the entire sequence and can be useful for tasks that require a sequence-level understanding.

7.5.1 BERT based Network Architecture

As previously discussed the architecture is a TensorFlow-based model using BERT as the underlying transformer model. The two approaches mentioned in the previous section share the same initial part:

1. **Input Layers:**

   - *input_ids:* This layer represents the input sequence as a tensor of integer IDs. It has a shape of (512,).
   - *attention_masks:* This layer represents the attention mask for the input sequence, indicating which tokens should be attended to and which should be ignored. It also has a shape of (512,).

2. **BERT Model:**

   - *bert_model:* It initializes a BERT model using `TFAutoModel` from *Transformer* library, passing the name of the pre-trained BERT model to be used. This BERT model is then made trainable.
   - *bert_output:* It applies the BERT model to the input data and retrieves the output, which includes the hidden states, pooler output and other relevant information.

From this point the architecture structure differs depending on the approach, in the following paragraph will be explained the two procedure.

First approach: 4-layer concatenation

- **Hidden states:** The above mentioned `bert_output` includes the hidden states, they contains the embeddings of the hidden layers, as initial step they are stacked to form a tensor representing all the hidden states throughout the model layers.
- **Concatenation:** The last 4 hidden states are concatenated to capture a broader context from the input sequence.
- **Dense layer:** The resulting concatenated tensor is passed through a dense layer with 1536 units (the reason of this number is simply related to the high resources consuming, it's been decide to half the complexity) and a *ReLU* activation function to extract higher-level features. To prevent overfitting, a dropout layer is applied, which randomly drops out 30% of the neuron outputs.

Second approach: Pooler layer

- **Pooler output:** `bert_output` includes also the pooler output, it represents the contextualized representation of the [CLS] token, which can be considered as a representation of the entire input sequence. It captures the overall semantic meaning of the input.

- **Dense layer:** The pooler output is passed through a dense layer with 384 units and even here a ReLU activation function and a dropout layer with a randomly drops out of 30%.

Finally the output of those two different approaches is passed to a final dense layer with as many units as the final number of classes (80) using *softmax()* as activation function. The model is then trained with a *RectifiedAdam optimizer* with learning rate of $2e^{-5}$ (the choice of learning rate is based on studies done in [45]) and *categorical cross-entropy* as loss function.

7.5.2 BERT test results

I have tested both approaches for each models introduced in the Chapter 2, the tables 7.18 and 7.19 display the evaluation results.

When considering the last 4 layers concatenation approach, the *Gilberto* and *Italian XXL* models demonstrate the best overall performances. They both perform well, even in terms of macro average. The *RoBERTa*, *AlBERTo* and *UmBERTo* models are comparable, achieving good results but slightly lower than the first two.

Observing the results of the second approach, which uses the pooler layer, it appears that the*Italian XXL* model achieves the best performances in terms of precision, recall, F1-score, and accuracy, especially in the weighted average. It is followed by *Gilberto*, which is still better than *RoBERTa*, *AlBERTo* and *UmBERTo*, which show relatively lower performances.

Regarding training time, there is a consistent increase in resource consumption. On average, all models took more than 2 hours to complete training. However, the most important factor to consider is the prediction time. If the architecture lacks sufficient memory or computational power, this mechanism could potentially cause a blockage in an entire business environment. It is worth noting that some customers generate thousands of tickets per day, so this must be taken into account when choosing the right models

Based on the results, the *GilBERTo* model (with 4 layers concatenation) and the *Italian XXL* model (with the pooler output) consistently delivered the best overall performances (complete classification reports on Appendix A). Therefore, one of them would be the recommended choice.

I have decided to used the *Italian XXL* model for the tests in the next sections. The reason for this choice is related to the corpus size. *GilBERTo* model has been trained on a larger amount of data, which allows it to capture more diverse

patterns and language nuances. This can potentially enhance its performance on real-world data.

Pre-trained model	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Italian XXL	73%/80%	72%/78%	72%/78%	78%	2h 20m 17s	1m 57s
RoBERTa	73%/77%	67%/76%	67%/75%	76%	2h 37m 23s	2m 1s
AlBERTo	75%/77%	68%/77%	69%/77%	77%	2h 28m 37s	2m 0s
UmBERTo	71%/78%	68%/78%	68%/78%	78%	2h 20m 58s	2m 2s
GilBERTo	**76%/80%**	**72%/80%**	**73%/79%**	**80%**	**2h 25m 19s**	**2m 7s**

Table 7.18: Pretrained models result using the 4-layer concatenation approach.

Pre-trained model	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Italian XXL	**76%/79%**	**68%/79%**	**70%/78%**	**79%**	**2h 24m 50s**	**2m 2s**
RoBERTa	69%/75%	62%/76%	63%/75%	76%	2h 35m 57s	1m 56s
AlBERTo	71%/75%	65%/75%	66%/74%	75%	2h 25m 48s	2m 7s
UmBERTo	62%/75%	62%/77%	60%/75%	77%	2h 19m 39s	2m 0s
GilBERTo	72%/78%	67%/79%	67%/77%	79%	2h 19m 27s	2m 2s

Table 7.19: Pretrained models result using the pooler output approach.

7.6 Test Markov chain

In the development of ML models, one common challenge is the quality of the data. While powerful and efficient models and algorithms are available, they perform best when working with balanced and high-quality datasets. However, in real-world environments, it is often difficult to obtain such datasets, and data may be dirty and completely unbalanced.
To address this problem, I will test the performance of a Markov Chain Text Generator, as introduced in Chapter 5. The models I have chosen to test are SVM and BERT, using the Italian base XXL pretrained model. These models will be evaluated with different splits of the dataset. The aim is to test whether adding synthetic data improves the training phase when classes are balanced, as well as whether performance improves even when the classes are unbalanced. To achieve this, I have divided the dataset using the following ticket thresholds:

```
Ticket_threshold = [1000, 400, 200, 50]
```

To avoid introducing redundancy, I have decided not to augment each class to the same number of tickets. The maximum increase is around 20% for each class.

For all the tests that will be introduced, the test set does not contain any synthetic tickets. The data generated by the Markov Chain were added only during the training/validation phase. Regarding the evaluation metrics, I will consider only the weighted average. This choice is made because by adding synthetic data, I may lose unbalanced information. The weighted average provides a more comprehensive measure of overall model performance.

Categories with at least 1000 tickets

In the table 7.20 are the details of the dataset augmented, it's been added the 24% of tickets (2704), in this case since I have a large corpus of each class I have decided to augment each one to 1640 tickets (the same number as the most populated one).

	Original	**With synthetic data**
Number of classes	6	6
Train/Validation set	7136	9840 (+24%)
Test set	793	793

Table 7.20: Dataset details after the addition of synthetic tickets (ticket_threshold = 1000).

Observing the results achieved in tables 7.24 and 7.25, it is evident that BERT has shown improvement in performance, while SVM seems to have achieved worse results. However, in this situation where there are fewer and balanced classes, the overall performances are still good.

Categories with at least 400 tickets

In table 7.21, the details of the augmented dataset are provided. In this case, 14% of tickets (1467) were added. It should be noted that the number of tickets per class is not the same for all classes, as the augmentation was applied only to

classes that had less than 1000 tickets. This decision was made considering that majority classes already had a substantial corpus of sentences.

	Original	**With synthetic data**
Number of classes	13	13
Train/Validation set	10837	12304 (+14%)
Test set	1205	1205

Table 7.21: Dataset details after the addition of synthetic tickets (ticket_threshold = 400).

The results achieved in tables 7.24 and 7.25 indicate a similar situation to the previous test. Even in the case of BERT, training the model with synthetic data does not lead to significant improvement in performance.

Categories with at least 200 tickets

In table 7.22, the details of the augmented dataset using classes with at least 200 tickets are provided. In this case, 36% of the tickets (5460 tickets) were added. As there are more classes to augment, a higher number of synthetic data points were generated. The same increment percentage used in previous tests was maintained for consistency.

	Original	**With synthetic data**
Number of classes	29	29
Train/Validation set	15044	20504 (+36%)
Test set	1672	1672

Table 7.22: Dataset details after the addition of synthetic tickets (ticket_threshold = 200).

Based on the results achieved in tables 7.24 and 7.25, both SVM and BERT demonstrate improvements in their performances. The range of applications that I am considering is becoming more consistent. Let's now wait for the results of the last ticket threshold to make final overall considerations.

Categories with at least 50 tickets

In the case of the threshold considering classes with at least 50 tickets, the dataset details are provided in table 7.23. In this scenario, 15% of the tickets (2550 tickets) were added. Considering the higher imbalance present in this threshold, it was decided not to increase the number of tickets for each class significantly. The rationale behind this decision is that training the generator using only 50 tickets may result in redundancy.

	Original	**With synthetic data**
Number of classes	62	62
Train/Validation set	18173	20723 (+15%)
Test set	2020	2020

Table 7.23: Dataset details after the addition of synthetic tickets (ticket_threshold = 50).

The results achieved in tables 7.24 and 7.25 surely underline the decreasing of the overall performances due to addition of classes and imbalance but also show that the trend of the results of the models using synthetic data is the same.

Ticket threshold	**Dataset**	**Precision**	**Recall**	**F1-score**	**Accuracy**
1000	Original	90%	90%	90%	90%
1000	Synthetic	89%	89%	89%	89%
400	Original	89%	89%	89%	89%
400	Synthetic	88%	88%	88%	88%
200	Original	84%	84%	83%	84%
200	Synthetic	85%	85%	85%	85%
50	Original	79%	79%	79%	79%
50	Synthetic	79%	79%	78%	79%

Table 7.24: Comparison of SVM results using synthetic data.

Final considerations on Markov Chain text generator

The overall findings suggest that BERT demonstrates more pronounced advantages when utilizing synthetic data, although the improvements are sporadic

Ticket threshold	Dataset	Precision	Recall	F1-score	Accuracy
1000	Original	89%	89%	89%	89%
1000	Synthetic	90%	90%	90%	90%
400	Original	89%	89%	89%	89%
400	Synthetic	87%	87%	87%	87%
200	Original	84%	83%	83%	83%
200	Synthetic	84%	83%	83%	83%
50	Original	77%	77%	76%	77%
50	Synthetic	78%	77%	76%	77%

Table 7.25: Comparison of BERT results using synthetic data.

yet significant. This can be attributed to the different embeddings used by the models. It is important to note that SVM relies on TF-IDF, while the tickets generated from the Markov Chain generator originate from the same vocabulary universe, resulting in a consistent vocabulary size. Consequently, the introduction of synthetic data in this scenario may provide limited benefits and potentially introduce interference in predictions. Conversely, BERT, which effectively captures semantic relations and contextual information in sentences, is better equipped to handle varying perspectives and relationships, leading to slight performance enhancements.

It is worth mentioning that I did not test the generator using the entire dataset, as I limited the increment of generated text due to the aforementioned reasons. As already introduced in Chapter 5 one limitation of this method is that it faces the same issue it aims to resolve, which is the lack of data. Markov Chain works best when trained on a large corpus of data, and in this specific situation, the improvements are minimal, if any. Therefore, I have made the decision not to employ any synthetic data. However, this algorithm could be further explored with alternative approaches, as discussed in more detail in Chapter **??**.

7.7 Ensemble techniques

Ensemble techniques in the context of implementing models for tasks such as text classification involve combining multiple individual models to achieve improved predictive performance. These techniques leverage the principle that

combining diverse models can lead to better overall predictions compared to relying on a single model alone.

Ensemble methods can be a compelling solution for text classification tasks due to their ability to capture different aspects of the data and address various challenges. By combining the strengths of multiple models, ensembles have the potential to enhance robustness, generalization, and accuracy in classification.

One possible solution is to implement a voting system that combines the predictions of four distinct models: SVM, StarSpace, BERT, and LSTM. Each model brings its unique approach and strengths to the ensemble, enabling us to leverage the diversity of techniques for improved classification accuracy. The voting system aggregates the predictions of these models and makes a final decision based on the collective opinion, effectively harnessing the power of multiple perspectives. However, using all four models together can be complicated due to their dependencies and memory requirements.

In this section, I will explore the bootstrap bagging solution with SVM, which is another example of an ensemble technique that is less resource-consuming.

7.7.1 Bootstrap bagging

This technique involves creating multiple bootstrap samples or bags from the original dataset and training individual SVM models on each sample. As shown in Figure 7.3, the predictions of these models are combined together in a voting system. The aim is to obtain a more robust and accurate classification outcome. Bootstrap bagging with SVM helps to mitigate overfitting and increases stability by incorporating variations introduced through resampling..

For the tests conducted in this thesis, I have considered three different bag sizes corresponding to 40%, 60%, and 80% of the original training dataset. Each training bag contains 7489, 11233, and 14978 tickets, respectively. The tickets belonging to each class are randomly chosen.

7.7.2 Bootstrap test results

Analyzing the results in Table 7.26, which compares the performance of a single classifier with the ensemble technique, I notice that the time needed for training and prediction is obviously larger when using a bag size equal to 80% of the entire dataset. However, in this case, there is an improvement in precision, while there are no significant gains in the other metrics.

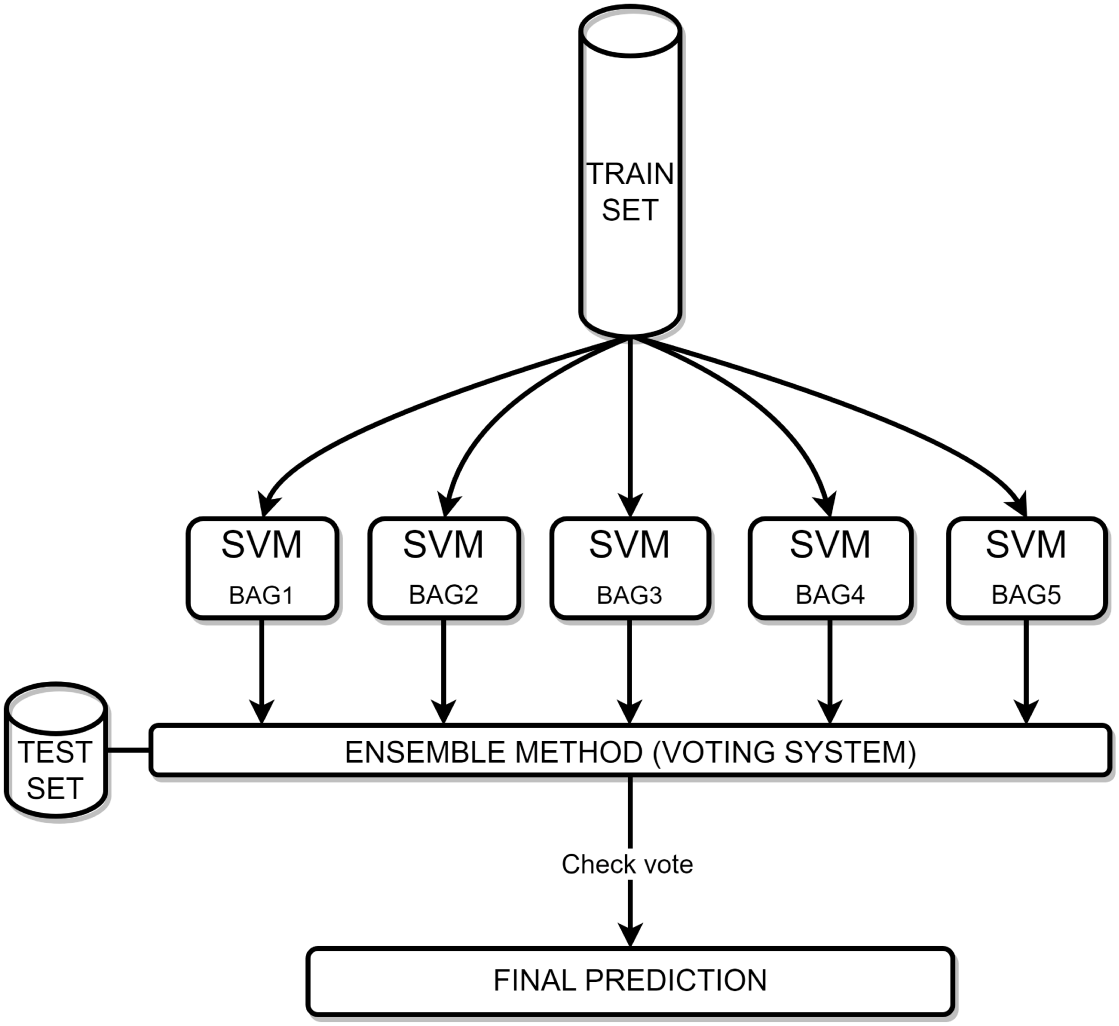


Figure 7.3: Illustration of Bootstrap bagging technique.

	Precision	Recall	F1-score	Accuracy	Time to train	Time to predict
Bagging 80%	79%/81%	74%/81%	75%/80%	81%	49s	6s
Bagging 60%	77%/79%	72%/80%	73%/79%	80%	37s	6s
Bagging 40%	77%/79%	71%/80%	72%/79%	80%	26s	5s
SVM	78%/81%	74%/81%	75%/80%	81%	3.5s	0.01s

Table 7.26: Bootstrap bagging technique classification results compered with a single SVM model trained on the entire dataset.

7.8 Memory requirements and resource usage consideration

In a business scenario, it is important to consume resources efficiently and minimize resource usage for several reasons:

- *Cost Savings:* Efficient resource consumption directly translates into cost savings. By using fewer computational resources, such as CPU, memory, and storage, businesses can reduce their infrastructure costs, especially when dealing with large-scale deployments or processing massive amounts of data.
- *Scalability and Optimization:* Efficient resource usage allows businesses to scale their operations. When resources are utilized optimally, it becomes easier to handle increased workloads and accommodate growing demands, ensuring smooth scalability of the business infrastructure. By minimizing resource consumption, businesses can allocate available resources more effectively.
- *Scalable Deployment:* With limited resources, it becomes crucial to optimize resource usage for deploying models or applications on resource-constrained devices or edge devices. By minimizing resource consumption, businesses can ensure that their solutions can be deployed in various environments without requiring significant hardware upgrades.

After careful consideration, the final model choice for this business scenario is SVM. It balances all these aspects better. However, if I don't consider resource consumption, I would prefer either StarSpace or BERT. Although their overall performances are slightly worse than SVM, they have a better understanding of contextualization and are more robust to changes in context. In contrast, SVM is trained with a specific vocabulary corpus, making it less adaptable to contextual variations.

7.9 Hybrid solution:Keep user choice

In this section, I will analyze the results of the trained model, particularly SVM, in the context of user classification. Let's refer to table 7.27, which provides information about the actual scenario where the end customer is responsible for classifying the tickets. While the current state-of-the-art models have already outperformed user choices in terms of overall performance, it is important to examine table 7.28, which focuses on the SVM results for specific classes, and table 7.29, which presents the corresponding classification report based solely on user choices (without AI model application).

Upon analysis, I have encountered the recurring issue of the misclassification probability of minority classes. Although the model has demonstrated good overall performance, certain minority classes still exhibit a relatively high misclassification rate. To address this challenge, I propose a hybrid solution that incorporates user choices and leverages the model's predictions only when the probability of accurate classification exceeds a certain threshold. This approach aims to enhance the reliability of the model by prioritizing cases where there is a higher likelihood of accurate predictions. As a result, it optimizes the user experience and ensures more accurate outcomes.

Classifier	**Precision**	**Recall**	**F1-score**	**Accuracy**
User	71%/80%	70%/77%	69%/77%	77%
SVM	**78%/81%**	**74%/81%**	**75%/80%**	**81%**

Table 7.27: Comparison between user choice and SVM results.

7.9.1 Grid search probability threshold

Before delving into the details of the results of the hybrid solution, it is necessary to determine the appropriate probability threshold. As SVM naturally does not provide well-calibrated probabilities, I will introduce the `CalibratedClassifierCV` library.

In scikit-learn, the `CalibratedClassifierCV` class is utilized to calibrate the probabilities predicted by a classifier. When training an SVM classifier, the decision values generated by the classifier are typically used for ranking instances, rather than direct probability estimates. However, in this case, accurate

Class	Precision	Recall	F1-score	Support
SALESFORCE ****VENDI	82%	98%	89%	164
SAP-DBO	74%	81%	78%	133
SIEBEL	89%	94%	91%	132
SAP-BI	70%	72%	71%	120
OIG	93%	83%	88%	76
SAP-BW	79%	74%	77%	31
HRNext-MasterData	64%	70%	67%	10
xxx ALTRO	75%	30%	43%	10
SAP-CO	50%	40%	44%	5
DOCFLOW	100%	33%	50%	3

Table 7.28: Support Vector Machine single classes results: In the first half we have the results of the most populated classes, in the second half minority classes, the dataset in exam refers to table 7.15

Class	Precision	Recall	F1-score	Support
SALESFORCE ****VENDI	78%	98%	87%	164
SAP-DBO	56%	85%	68%	133
SIEBEL	90%	79%	84%	132
SAP-BI	57%	74%	65%	120
OIG	1%	82%	90%	76
SAP-BW	79%	87%	83%	31
HRNext-MasterData	91%	100%	95%	10
xxx ALTRO	0%	0%	0%	10
SAP-CO	100%	100%	100%	5
DOCFLOW	100%	100%	100%	3

Table 7.29: User manual classification results: In the first half we have the results of the most populated classes, in the second half minority classes (complete classification report on Appendix A).

probability estimates are required, which reflect the true likelihood of a sample belonging to a specific class. The `CalibratedClassifierCV` class enables prob-

ability calibration by fitting additional models on top of the SVM classifier. It internally performs a form of cross-validation to estimate the probabilities.

To summarize, by employing `CalibratedClassifierCV`, I can obtain improved probability estimates from the SVM classifier. This is particularly useful in the present scenario, where reliable probability information is needed for decision-making.

To decide which is the correct probability threshold I have tested the system with different values of probabilities:

```
Probability_threshold = [0.1, 0.2, 0.3, 0.4, 0.45, 0.5, 0.55, 0.6,
                         0.7, 0.8, 0.9]
```

To notice that with `Probability_threshold = 0` means considering only the model, and `Probability_threshold = 1` only the user choice.

Grid search results

The table 7.30 and the figure 7.4 display the result of the evaluation, based on that, the final choice of the probability threshold falls to:

```
Probability_threshold = 50%
```

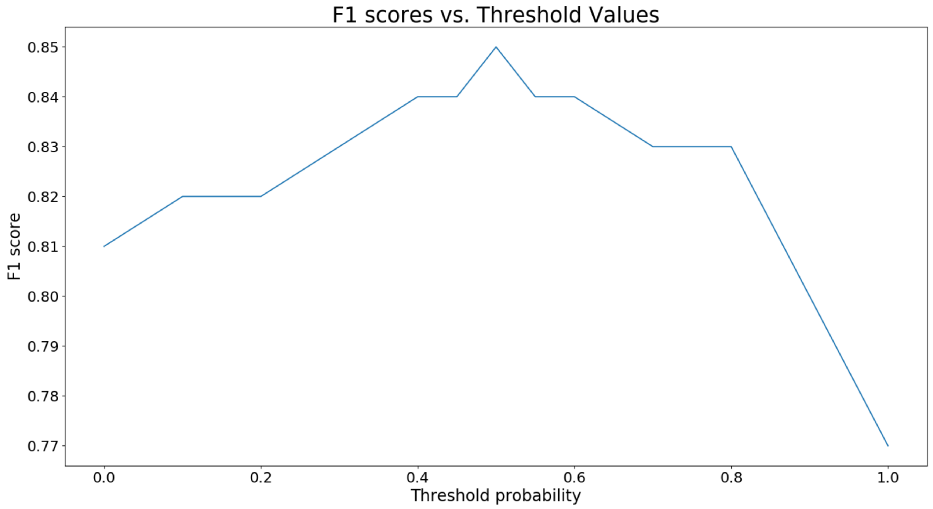


Figure 7.4: Grid search for probability threshold, the F1 score is related to the weighted average.

Probability threshold	Precision	Recall	F1-score	Accuracy
0.1	82%	83%	82%	83%
0.2	83%	83%	82%	83%
0.3	83%	83%	83%	83%
0.4	84%	84%	84%	84%
0.5	85%	85%	85%	85%
0.6	84%	85%	84%	85%
0.7	84%	84%	83%	84%
0.8	84%	83%	83%	83%
0.9	82%	81%	80%	81%

Table 7.30: Weighted average precision, recall, F1 score and accuracy of the hybrid system based on SVM, using different probability threshold.

7.9.2 Results

The results presented in tables 7.31 and 7.32 show that using a hybrid approach can significantly improve performance. In particular, there are noticeable improvements in the classification of minority classes such as *App Prevent*. *Lavori* and *SAP-VIM*, and even the majority classes have experienced enhancements. For instance, the overall metrics for *SALESFORCE* ****VENDI* and *SIEBEL* are higher.

However, it is worth noting that in certain cases, specifically for the class *xxx ALTRO*, the model may make correct predictions but assign a low probability to the outcome (less than 50%). On the other hand, the user consistently fails to classify instances correctly, as indicated by the recall metrics of *xxx ALTRO* in table 7.29 being set to 0%. This situation results in a decline in yield for these classes. It is important to carefully consider this trade-off when utilizing the hybrid approach. While it improves overall performance, there may be instances where the model's low-confidence predictions lead to reduced performance for certain classes.

Classifier	Precision	Recall	F1-score	Accuracy
Hybrid System	**81%/85%**	**81%/80%**	**80%/85%**	**85%**
User	71%/80%	70%/77%	69%/77%	77%
SVM	78%/81%	74%/81%	75%/80%	81%

Table 7.31: Comparison between user choice, SVM and hybrid approach.

Class	Precision	Recall	F1-score	Support
SALESFORCE ****VENDI	83%	95%	89%	164
SAP-DBO	79%	86%	82%	133
SIEBEL	92%	91%	91%	132
SAP-BI	74%	81%	77%	120
OIG	91%	93%	92%	76
SAP-BW	88%	90%	89%	31
HRNext-MasterData	90%	90%	90%	10
xxx ALTRO	33%	10%	15%	10
SAP-CO	75%	60%	67%	5
DOCFLOW	100%	100%	100%	3

Table 7.32: Hybrid solution results: In the first half we have the results of the most populated classes, in the second half minority classes (complete classification report on Appendix A).

7.10 Final results comparison

In this final section, I will summarize the best results achieved from the models and provide a final overall comparison. I will not consider the baseline models.

Table 7.33 shows the performance of the principal models. All of these results were achieved using the dataset modified after removing all the classes with fewer than 25 tickets. The input for these models is the concatenation of the Subject and Request fields.

The results suggest that SVM outperformed other solutions. This is an inter-

Classifier	Precision	Recall	F1-score	Accuracy
User	71%/80%	70%/77%	69%/77%	77%
Hybrid System	**81%/85%**	**81%/85%**	**80%/85%**	**85%**
SVM	78%/81%	74%/81%	75%/80%	81%
BERT	76%/79%	68%/79%	70%/78%	79%
StarSpace	73%/78%	69%/78%	70%/78%	78%
LSTM	62%/72%	57%/72%	57%/71%	72%

Table 7.33: Final comparison between all the relevant model results.

esting evolution of the implementation, as SVM, in this specific case, performs better than powerful and complex neural network-based models. The difference is made by the nature of the dataset and the embeddings used. When dealing with quite long texts/sentences, the probability of finding words that are strictly related to the class itself inside the text of the tickets grows. This means that for TF-IDF, those words will play a crucial role, making this type of embedding more powerful, especially if that word is present only in that kind of tickets and if the number of available tickets for training is low. BERT and StarSpace, on the other hand, will give less importance to the single word, so they lose this kind of information. If they do not have enough tickets to understand the context as well as possible, they will fail in prediction with a higher probability. To understand it better, let's focus on an example. In Table 7.34, the classification metrics of SVM, BERT, and StarSpace related to one specific application/class (*HRNext*) are shown. The F1-score of SVM is clearly the best. In Table 7.35, there are three examples of tickets that belong to that class, and as expected, the name of the class itself is present more times inside the tickets.

Classifier	Precision	Recall	F1-score
SVM	75%	100%	86%
BERT	100%	33%	50%
StarSpace	33%	33%	33%

Table 7.34: Comparison between SVM, BERT and StarSpace classification metrics of *HRNext*.

Application	Request
HRNext	non riesco ad approvare giustificativi in **hrnext** se entrando in **hrnext** clicco l opzione vista collaboratori non appare niente
HRNext	problema di accesso portale rilevazione presenze time di **hrnext** entro 22 08 buongiorno stamattina il tentativo di accedere da postazione di lavoro al portale **hrnext** risulta impossibile diversamente da altre volte
HRNext	incident 20221003 malfunzionamento **hrnext** wifi user rpa elaborazioni pianificate risultano indisponibili i seguenti sistemi **hrnext** rpa accesso all area interserver transiti wifi user

Table 7.35: Three tickets of *HRNext*.

One-vs-One Model Results

In Figures 7.5, 7.6, 7.7, and 7.8, we present the one-versus-one comparison of the most relevant classes (some classes are not displayed to ensure figure visibility). Despite the inherent challenge of achieving a 100% F1-score in a model, like the user do in some classes, the results obtained in these cases are highly satisfactory, with performance metrics consistently surpassing 80%.

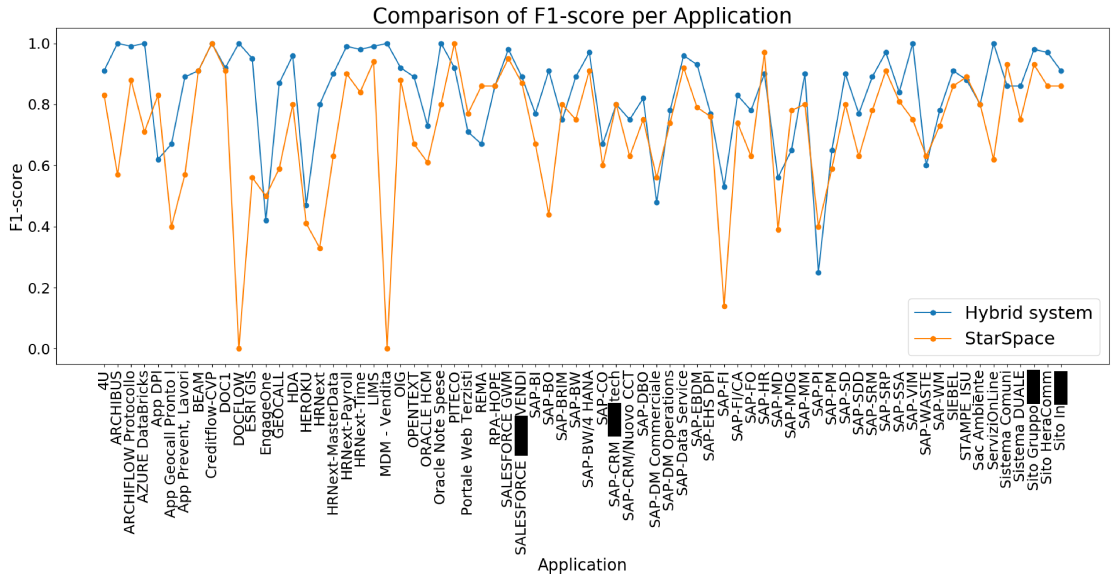


Figure 7.5: Comparison of F1-score on single classes between Hybrid system and StarSpace.

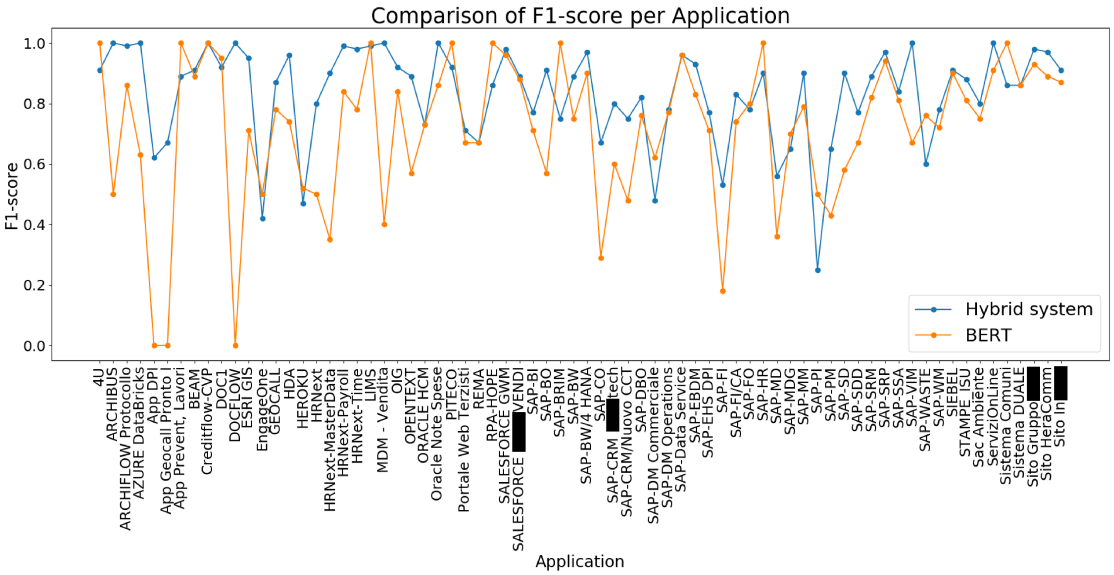


Figure 7.6: Comparison of F1-score on single classes between Hybrid system and BERT.

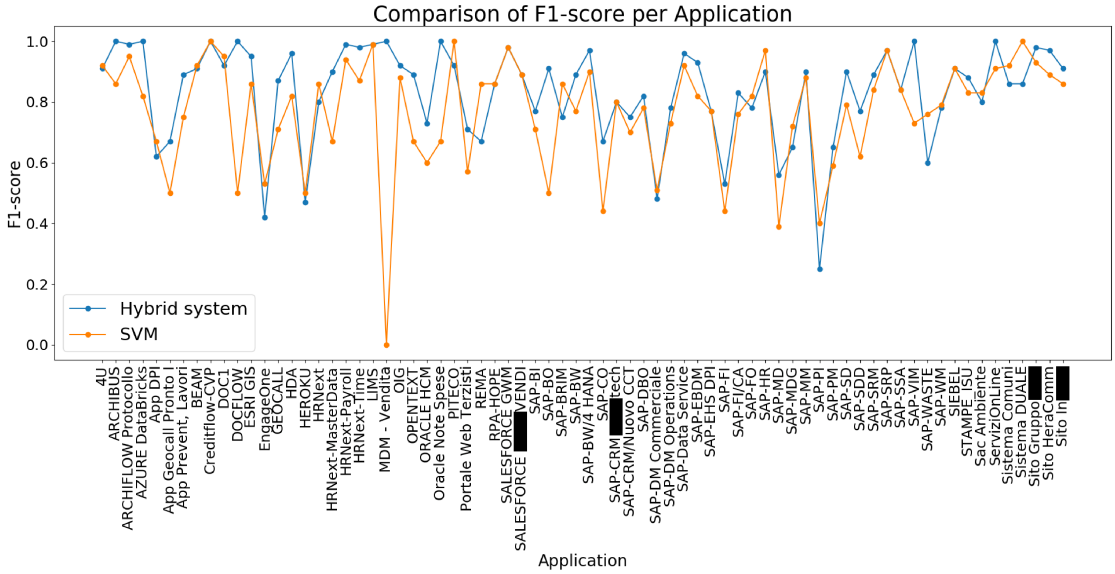


Figure 7.7: Comparison of F1-score on single classes between Hybrid system and SVM.

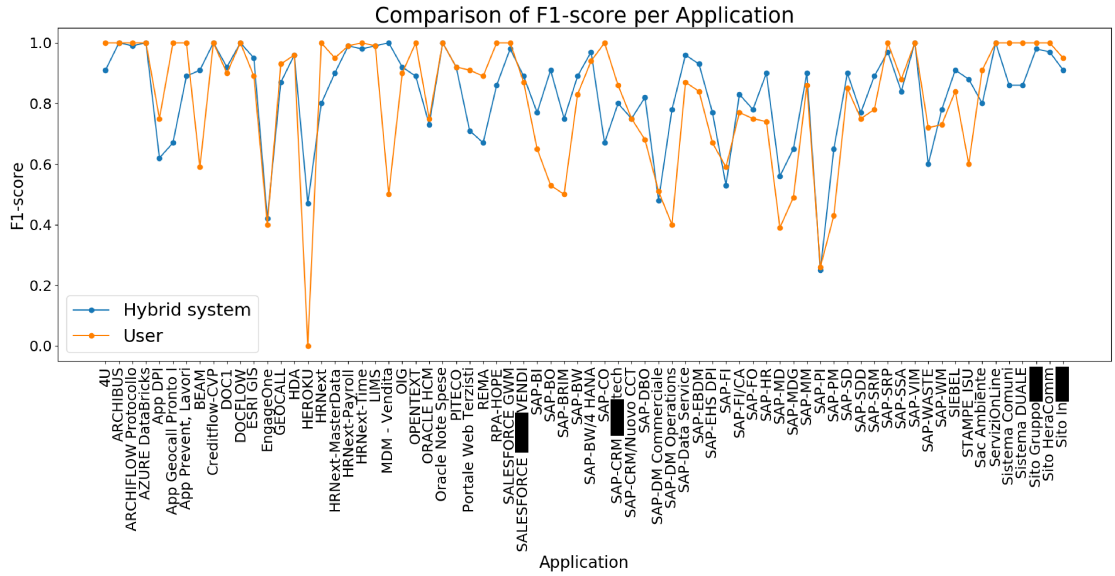


Figure 7.8: Comparison of F1-score on single classes between Hybrid system and User.

8

From Development to Deployment: Implementing an AI Model into Production Environment

Deployment is a crucial step in transforming a machine learning model from a prototype to a scalable and accessible solution. In this section, I will delve into the steps of deploying implemented models and enabling its seamless integration into production systems. The deployment strategy involves the creation of an Application Programming Interface (API) to facilitate easy communication with the model. Additionally, we leverage the power of containerization through Docker, which allows to encapsulate the model and its dependencies into a portable unit. Furthermore, I will explore the advantages of utilizing Kubernetes, a powerful orchestration platform, to manage the deployment.

8.1 API implementation

The both models, language detector and ticket classification will be used by a web application (client), CX Studio & Engagent and HelpDesk Advanced respectively, to do that I have implemented the REST API using Tornado library in python.

The REST (Representational State Transfer) API is a widely adopted architectural style for designing networked applications. It provides a standardized

approach for building web services that allow different systems to communicate and interact with each other over the internet. In the context of deploying machine learning models, a REST API acts as an intermediary layer between the model and the clients who want to make predictions or utilize the model's capabilities.

The API follows a client-server model, where the client initiates requests to the server (which hosts the deployed model) using predefined HTTP methods, in this case, since I want to retrieve information I will use the GET method. When a client sends a request to the REST API, it processes the request, invokes the appropriate model or model endpoint, and returns a response to the client.

The response from the REST API will includes the prediction generated by the model. The data is transmitted in a JSON (JavaScript Object Notation) structured format (see code snippets below), which is a lightweight data interchange format commonly used in web APIs.

```
{
  "text": "Hi, my name is Francesco",
  "language": "en"
}
```

Listing 8.1: JSON response code for language detection.

```
{
  "ticket": "non riesco ad approvare giustificativi in hrnext se
    entrando in hrnext clicco l opzione vista collaboratori non appare
    niente",
  "application": "HRNext"
}
```

Listing 8.2: JSON response code for ticket classification.

8.2 Docker's Utility

As already mentioned in the Chapter 3 Docker is a very useful platform which allows to use applications in a easy way without warring about the infrastructure, in figure 8.1 an example of the Docker containers structure.

In practise, the first step to go throw is the creation of the Dockerfile. A

Dockerfile is a text file that contains instructions on how to build a Docker image. Start by creating a new file named *Dockerfile* (with no file extension) in the project directory. Within the Dockerfile has to be specified the base image, the file that will be copied inside the container, using command `COPY`. To install the dependencies needed, the `RUN` instruction in the Dockerfile allows to execute commands that install the required packages using a package manager like `pip`. Once the Dockerfile is created is possible to build the image and run the container using the windows command line or the Docker GUI.

With the container running, I can test Python models by interacting with the exposed endpoints or functionality. By sending HTTP requests to the container's exposed endpoints. So both models were been packaged into a Docker images and containers, making them portable and easy to deploy on different environments.

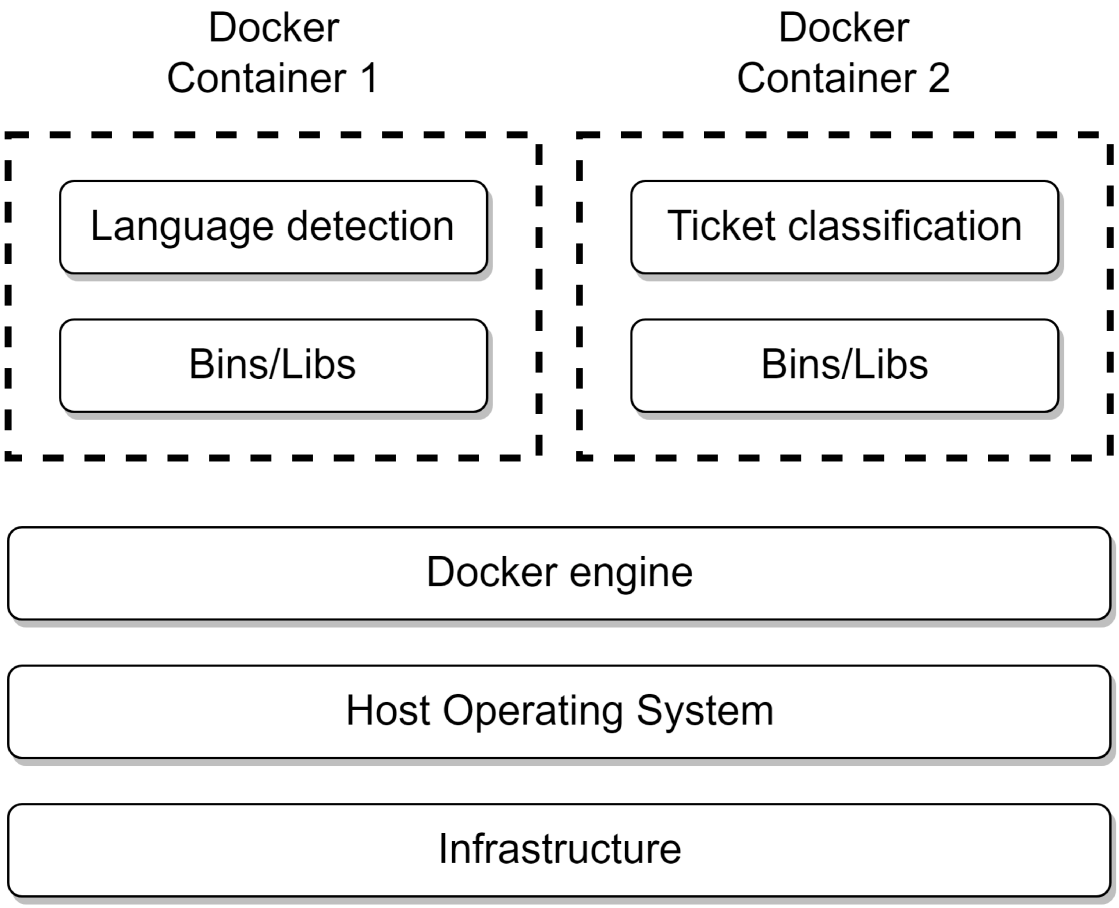


Figure 8.1: Docker containers.

8.3 Unit Test and Integration with CX Studio

An important step to do before deploy models in a production environment are the unit test. Unit tests are a type of software testing where individual components, or units, of a software system are tested in isolation. The purpose of unit testing is to verify that each unit of code, such as a function or a method, works correctly as expected. They ensure that each unit of code performs as intended, detects and prevents bugs early on, and promotes code maintainability and stability. In this project those test were performed on both models implemented.

The last development step regarding the language detection model is the creation of the *Adapter*. This element is used to connect the model to the ChatBot. It is a groovy script that integrate the artificial intelligence services with CX Studio by calling the corresponding API REST and managing the response, in this case the JSON file.

8.4 Kubernetes

Figure 8.2 shows the architecture of Kubernetes, in the right part are displayed the worker nodes, a *worker* is a container host, it runs the *kubelet* process which is responsible for communicating with the kubernetes clusters (left part of figure). Kubernetes system allows to configured the so called *Desired State Management* by configuring the *.YAML* file, it contains all the configuration needed to the control plane to manage the state.

In the *.YAML* file there are different configurations, let's focus on two fundamental pieces:

- **POD:** It is the smallest unit of the kuberenetes configuration, it contains the container of the applications, in this case the one that I have created in Docker for language detector and ticket classification. Is possible to configure different container images.
- **Replica:** For each pod is possible to configure how many replicas to have in different worker. Replicas are used to provide scalability and high availability for applications running in the cluster. By defining the number of replicas for a pod, you can control the desired level of redundancy and ensure that a specified number of identical pod instances are running.

The left part of figure 8.2 represent the Control Plane, it is responsible for managing and maintaining the desired state of the cluster. It is composed of

several components that work together to ensure the proper functioning of the Kubernetes cluster.

In details:

- **kube-api-server:** It acts as the primary interface for the cluster and serves as the control plane's frontend. API Server exposes the Kubernetes API, which allows users and other components to interact with the cluster.
- **kube-scheduler:** It is responsible for assigning pods to worker nodes based on resource requirements, quality of service, and other policies. It ensures that pods are scheduled onto appropriate nodes to achieve optimal resource utilization and maintain high availability.
- **kube/clud-controller-manager:** It is a collection of controllers that monitor the cluster's state and perform actions to maintain the desired state. It includes controllers for replication, endpoints, services, nodes, and other resources. Each controller is responsible for a specific aspect of the cluster's operation.
- **etcd:** It s a distributed key-value store used by Kubernetes to store the cluster's configuration and state information. It provides a reliable and consistent data store for the control plane components to read from and write to.

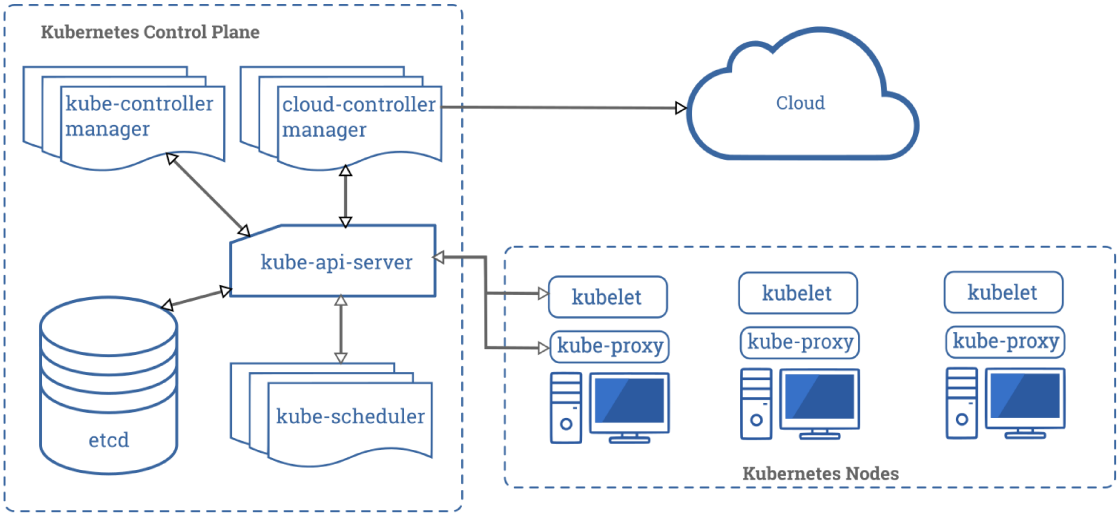


Figure 8.2: Docker containers.

9

Conclusions

The primary objective of this master's thesis was to enhance customer experience and satisfaction through the implementation of AI models. The thesis encompasses the complete development process of a language detection model and a ticket classification system, each targeting specific challenges and requirements.

In the domain of language detection, the focus was on optimizing performance for ChatBot messages, which often consist of short sentences. Various algorithms and neural networks were explored, and a meticulous analysis was conducted to identify critical areas for data cleaning and model performance improvement. The ultimate goal was to establish an effective preprocessing pipeline. Among the evaluated models, FastText emerged as the top performer in terms of both accuracy and speed.

To further enhance short text prediction, a comprehensive examination of the detection results was carried out. Subsequently, a hybrid algorithm was devised to post-process the detection outcomes and refine performance specifically for short text inputs. This algorithm combines character thresholding and confidence output, incorporating a dictionary check to enhance precision in the detection process.

By implementing these measures, the language detection model achieved significant improvements in accuracy and efficiency, particularly in the context of short text analysis. These enhancements directly contribute to improving customer interactions and overall user experience.

When it comes to the ticket classification model, the primary objective was

to develop a robust system capable of accurately predicting the correct application/class for each ticket. The aim was to enhance customer service by reducing misclassifications made by users, thereby improving the reliability of the system and response time service levels.

To achieve this, a specific dataset was utilized, and thorough analysis and preprocessing were performed. Addressing the issue of data imbalance played a crucial role in the preprocessing phase. Several techniques for generating synthetic data using Markov Chain were tested, and a decision was made to reduce the number of classes by selecting only those classes with a specific threshold of tickets. This approach facilitated a more effective training phase.

Various machine learning and deep learning algorithms were implemented, ranging from baseline models to more complex ones such as SVM, LSTM, BERT, and StarSpace. The performance of each model was evaluated based on precision, recall, F1 score, and accuracy. Additionally, the training and prediction times, as well as resource consumption, were taken into consideration. After careful analysis and testing, it was found that SVM, StarSpace, and BERT (specifically the Italian base XXL and GilBERTo pretrained models) demonstrated the best performance. However, considering the resource consumption, SVM was ultimately chosen as the preferred model.

Analyzing the results revealed the challenge of predicting minority classes due to the imbalanced dataset. To address this issue, an innovative hybrid system was implemented, leveraging user choice when the model's confidence fell below a certain threshold. This threshold was determined through a grid search method. By incorporating user feedback in this manner, substantial improvements in performance were achieved, leading to highly satisfying results.

9.1 Practical Implications

Both the language detection model and the ticket classification model will be implemented in PAT Group's solutions, specifically the CX Studio & Engagent's ChatBot for the language detector and HelpDesk Advanced (HDA) for the ticket classification model. The application of these models is closely intertwined.

Many of PAT group customers operate in international contexts, which necessitates the need for a ticket classification model that can handle multiple languages. To address this, a BERT multilingual pretrained model can be employed. However, it should be noted that while this approach offers multilingual

support, it may not achieve the same level of performance as language-specific models. In such cases, the language detection model can play a dual role. It can be used during the training phase to detect the language of the data, allowing for the creation of separate language-specific classification models. Additionally, during the prediction phase, the language detection model can be employed to determine the language of new tickets and subsequently submit them to the appropriate language-specific model for prediction. This integrated approach ensures accurate classification while leveraging the benefits of language-specific models.

9.2 Future works

Nowadays, artificial intelligence applied to Natural Language Processing (NLP) is having a profound impact on various aspects of real life, and its influence is expected to grow even further in the future. Researches in this field are in continuous evolution.

In the last months *ChatGPT* [25] by OpenAI, that is based on Generative Pre-trained Transformer [38], has overwhelmingly taken the leading role in this field. Generative Pretrained Models are advanced deep learning models that have been trained on a large corpus of text data using unsupervised learning techniques. These models learn to predict the next word in a sentence, which helps them capture the statistical patterns and structures present in the training data. These models are called "generative" because they can generate coherent and contextually relevant text based on a given prompt or input. Generative pretrained models have achieved remarkable success in various natural language processing (NLP) tasks and have demonstrated impressive language understanding and generation capabilities.

In text classification, GPT can be used directly for classification, a common approach is to fine-tune these models on a specific classification task, the process is similar to the one used in this thesis. At the moment is still quite resources expensive to use it but in the future it could became a very powerful tools to improve the overall performances.

Another point on which I will personally focus on in the next future is related the issue faced during this master thesis, the class imbalance and the missing of quality data.

Models like BERT, GPT etc. are very powerful, but in a real world scenario

they suffer of this phenomenon, in this master thesis I have tried the Markov Chain method to solve this problem. The results achieved was not so satisfying because for some classes I have less than 20 tickets and the text generated based on Markov Chain will result on adding redundancy and it achieved inconsistent results. An alternative approach is to train the generator on the entire corpus, considering all the classes, and than use the model itself (by looking on the confidence) to classify the result and allocate it on the classes as synthetic data. After that, retrain the model. The idea is to augment classes in a more efficiency way.

There are more complex methods that can be used for text generation, for example an interesting one is the Relational Generative Adversarial Networks (RelGAN) [30]. A GAN is commonly used in computer vision field, usually it is used to generate fake images, in a nutshell it is composed of two neural networks, the generator, which try to generate fake image, and the discriminator which has to classify the image as real or fake, the goal of the generator is fooling the discriminator and generates an image that the discriminator is not able to detect as false.

The main objective of RelGAN is to address the limitations of traditional text generation models. RelGAN introduces a new adversarial training framework that incorporates relational knowledge into the generation process. The key components include a generator, a discriminator, and a relation discriminator. The addition of the relation discriminator helps in capturing the coherence and relation consistency in the generated responses, leading to more contextually relevant and coherent text generation. It enables RelGAN to generate diverse and contextually appropriate responses, enhancing the overall performance of text generation models.

9.3 Personal Growth and Accomplishments

Throughout this thesis, I have encountered a real-world scenario that has allowed me to effectively tackle and fulfill all the objectives set forth in this research. The successful completion of these goals can be attributed to the culmination of my dedicated years of study and the comprehensive competences I have acquired throughout my academic journey. The theoretical foundations in machine learning and algorithms, as well as the practical skills and critical thinking abilities honed during my studies, have played a pivotal role in the

proficient execution of this research. Engaging in this project has not only expanded my technical expertise but also sharpened my problem-solving skills, project management abilities, and critical evaluation capabilities. The outcomes achieved, from my perspective and that of the companies involved, are highly satisfactory.

Appendix: Detailed Model Performance Metrics

In this section will be displayed the results in terms of precision, recall and F1-score of every single classes for the most relevant models tested. The following tables refers to results achieved using the dataset on its final version, with preprocessing applied (table 7.15) and with the *Subject* field concatenated with the *Request*.

A.1 User

Class	Precision	Recall	F1-score	Support
4U	100%	100%	100%	6
ARCHIBUS	100%	100%	100%	4
ARCHIFLOW Protocollo	100%	100%	100%	39
AZURE DataBricks	100%	100%	100%	7
App DPI	60%	100%	75%	6
App Geocall Pronto I	100%	100%	100%	3
App Prevent, Lavori	100%	100%	100%	4
BEAM	100%	42%	59%	55

Creditflow-CVP	100%	100%	100%	4
DOC1	81%	100%	90%	22
DOCFLOW	100%	100%	100%	3
ESRI GIS	100%	80%	89%	10
EngageOne	33%	50%	40%	8
GEOCALL	88%	100%	93%	21
HDA	96%	96%	96%	23
HEROKU	0%	0%	0%	12
HRNext	100%	100%	100%	3
HRNext-MasterData	91%	100%	95%	10
HRNext-Payroll	100%	98%	99%	40
HRNext-Time	100%	100%	100%	24
LIMS	100%	97%	99%	39
MARKETING CLOUD	67%	67%	67%	3
MDM - Vendita	33%	100%	50%	3
MULESOFT	0%	0%	0%	3
MyAcademy	100%	100%	100%	3
OIG	100%	82%	90%	76
OPENTEXT	100%	100%	100%	4
ORACLE HCM	59%	100%	75%	19
Oracle Note Spese	100%	100%	100%	3
PITECO	86%	100%	92%	6
Portale Web Terzisti	84%	100%	91%	16
REMA	80%	100%	89%	4
RPA-HOPE	100%	100%	100%	3
RPA-HVAC	100%	100%	100%	7

SALESFORCE GWM	100%	100%	100%	22
SALESFORCE ***VENDI	78%	98%	87%	164
SAP	0%	0%	0%	11
SAP-BI	57%	74%	65%	120
SAP-BO	36%	100%	53%	5
SAP-BRIM	100%	33%	50%	3
SAP-BW	79%	87%	83%	31
SAP-BW/4 HANA	95%	92%	94%	104
SAP-CO	100%	100%	100%	5
SAP-CRM ***tech	75%	100%	86%	6
SAP-CRM/Nuovo CCT	86%	67%	75%	9
SAP-DBO	56%	85%	68%	133
SAP-DM Commerciale	54%	48%	51%	31
SAP-DM Operations	55%	31%	40%	141
SAP-Data Service	100%	77%	87%	13
SAP-EBDM	96%	74%	84%	31
SAP-EHS DPI	100%	50%	67%	14
SAP-FI	63%	56%	59%	9
SAP-FI/CA	80%	75%	77%	75
SAP-FO	86%	67%	75%	9
SAP-HR	83%	67%	74%	15
SAP-MD	56%	30%	39%	30
SAP-MDG	82%	35%	49%	26
SAP-MM	90%	82%	86%	34
SAP-PI	15%	100%	26%	3
SAP-PM	61%	33%	43%	33

SAP-SD	100%	73%	85%	15
SAP-SDD	92%	63%	75%	19
SAP-SRM	88%	70%	78%	10
SAP-SRP	100%	100%	100%	17
SAP-SSA	94%	83%	88%	53
SAP-VIM	100%	100%	100%	5
SAP-WASTE	75%	69%	72%	13
SAP-WM	64%	84%	73%	38
SIEBEL	90%	79%	84%	132
STAMPE$_I SU$	93%	44%	60%	61
Sac Ambiente	83%	100%	91%	5
ServiziOnLine	100%	100%	100%	6
Sistema Comuni	100%	100%	100%	7
Sistema DUALE	100%	100%	100%	4
Sito Gruppo***	100%	100%	100%	51
Sito ***Comm	100%	100%	100%	18
Sito In***	91%	100%	95%	10
YUBIK LEGALE	100%	100%	100%	12
YuBSC	100%	100%	100%	25
xxx ALTRO	0%	0%	0%	10
OTHERS	0%	0%	0%	0
Accuracy	**77%**	**77%**	**77%**	**2081**
Macro avg	**71%**	**70%**	**69%**	**2081**
Weighted avg	**80%**	**77%**	**77%**	**2081**

Table A.1: The following table provides the classification report based on user choices. It utilizes the same test dataset used for the models. The class labeled as "OTHERS" encompasses categories that are outside the context of the classification. As this classification is performed by a human, there may be instances where misclassifications occur, particularly for classes that were removed during the preprocessing stage.

A.2 SVM

Class	Precision	Recall	F1-score	Support
4U	86%	100%	92%	6
ARCHIBUS	100%	75%	86%	4
ARCHIFLOW Protocollo	93%	97%	95%	39
AZURE DataBricks	70%	100%	82%	7
App DPI	67%	67%	67%	6
App Geocall Pronto I	100%	33%	50%	3
App Prevent. Lavori	75%	75%	75%	4
BEAM	87%	98%	92%	55
Creditflow-CVP	100%	100%	100%	4
DOC1	95%	95%	95%	22
DOCFLOW	100%	33%	50%	3
ESRI GIS	82%	90%	86%	10
EngageOne	57%	50%	53%	8
GEOCALL	60%	86%	71%	21
HDA	86%	78%	82%	23
HEROKU	50%	50%	50%	12
HRNext	75%	100%	86%	3
HRNext-MasterData	64%	70%	67%	10
HRNext-Payroll	95%	93%	94%	40
HRNext-Time	91%	83%	87%	24
LIMS	97%	100%	99%	39
MARKETING CLOUD	17%	33%	22%	3
MDM - Vendita	0%	0%	0%	3
MULESOFT	0%	0%	0%	3

MyAcademy	100%	100%	100%	3
OIG	93%	83%	88%	76
OPENTEXT	100%	50%	67%	4
ORACLE HCM	82%	47%	60%	19
Oracle Note Spese	67%	67%	67%	3
PITECO	100%	100%	100%	6
Portale Web Terzisti	73%	47%	57%	17
REMA	100%	75%	86%	4
RPA-HOPE	75%	100%	86%	3
RPA-HVAC	88%	100%	93%	7
SALESFORCE GWM	100%	95%	98%	22
SALESFORCE ***VENDI	82%	98%	89%	164
SAP	0%	0%	0%	11
SAP-BI	70%	72%	71%	120
SAP-BO	67%	40%	50%	5
SAP-BRIM	75%	100%	86%	3
SAP-BW	79%	74%	77%	31
SAP-BW/4 HANA	90%	91%	90%	104
SAP-CO	50%	40%	44%	5
SAP-CRM ***tech	100%	67%	80%	6
SAP-CRM/Nuovo CCT	64%	78%	70%	9
SAP-DBO	74%	81%	78%	133
SAP-DM Commerciale	54%	48%	51%	31
SAP-DM Operations	71%	74%	73%	141
SAP-Data Service	100%	85%	92%	13
SAP-EBDM	83%	81%	82%	31

SAP-EHS DPI	83%	71%	77%	14
SAP-FI	44%	44%	44%	9
SAP-FI/CA	78%	75%	76%	75
SAP-FO	88%	78%	82%	9
SAP-HR	94%	100%	97%	15
SAP-MD	48%	33%	39%	30
SAP-MDG	63%	85%	72%	26
SAP-MM	91%	85%	88%	34
SAP-PI	50%	33%	40%	3
SAP-PM	68%	52%	59%	33
SAP-SD	85%	73%	79%	15
SAP-SDD	77	53%	62%	19
SAP-SRM	89%	80%	84%	10
SAP-SRP	100%	94	97%	16
SAP-SSA	83%	85%	84%	53
SAP-VIM	67%	80%	73%	5
SAP-WASTE	100%	62%	76%	13
SAP-WM	83%	76%	79%	38
SIEBEL	89%	94%	91%	132
STAMPE_ISU	79%	87%	83%	61
Sac Ambiente	71%	100%	83%	5
ServiziOnLine	100%	83%	91%	6
Sistema Comuni	100%	86%	92%	7
Sistema DUALE	100%	100%	100%	4
Sito Gruppo***	92%	94%	93%	51
Sito ***Comm	89%	89%	89%	18

Sito In***	82%	90%	86%	10
YUBIK LEGALE	100%	83%	91%	12
YuBSC	96%	100%	98%	25
xxx ALTRO	75%	30%	43%	10
Accuracy	**81%**	**81%**	**81%**	**2081**
Macro avg	**78%**	**74%**	**75%**	**2081**
Weighted avg	**81%**	**81%**	**80%**	**2081**

Table A.2: SVM complete classification report.

A.3 LSTM

Class	Precision	Recall	F1-score	Support
4U	56%	83%	67%	6
ARCHIBUS	100%	50%	67%	4
ARCHIFLOW Protocollo	97%	85%	90%	39
AZURE DataBricks	36%	57%	44%	7
App DPI	71%	83%	77%	6
App Geocall Pronto I	0%	0%	0%	3
App Prevent. Lavori	100%	50%	67%	4
BEAM	78%	91%	84%	55
Creditflow-CVP	100%	100%	100%	4
DOC1	87%	91%	89%	22
DOCFLOW	0%	0%	0%	3
ESRI GIS	17%	10%	12%	10
EngageOne	100%	38%	55%	8
GEOCALL	50%	81%	62%	21
HDA	80%	70%	74%	23
HEROKU	0%	0%	0%	12
HRNext	50%	33%	40%	3
HRNext-MasterData	60%	30%	40%	10
HRNext-Payroll	90%	93%	91%	40
HRNext-Time	68%	88%	76%	24
LIMS	90%	95%	92%	39
MARKETING CLOUD	0%	0%	0%	3
MDM - Vendita	0%	0%	0%	3
MULESOFT	0%	0%	0%	3

MyAcademy	100%	67%	80%	3
OIG	90%	75%	82%	76
OPENTEXT	100%	25%	40%	4
ORACLE HCM	45%	47%	46%	19
Oracle Note Spese	100%	67%	80%	3
PITECO	60%	50%	55%	6
Portale Web Terzisti	38%	18%	24%	17
REMA	67%	100%	80%	4
RPA-HOPE	33%	67%	44%	3
RPA-HVAC	100%	100%	100%	7
SALESFORCE GWM	78%	95%	86%	22
SALESFORCE ***VENDI	76%	98%	86%	164
SAP	0%	0%	0%	11
SAP-BI	59%	71%	64%	120
SAP-BO	50%	20%	29%	5
SAP-BRIM	0%	0%	0%	3
SAP-BW	56%	61%	58%	31
SAP-BW/4 HANA	92%	87%	89%	104
SAP-CO	25%	20%	22%	5
SAP-CRM ***tech	75%	50%	60%	6
SAP-CRM/Nuovo CCT	67%	44%	53%	9
SAP-DBO	78%	68%	73%	133
SAP-DM Commerciale	38%	35%	37%	31
SAP-DM Operations	67%	72%	69%	141
SAP-Data Service	92%	85%	88%	13
SAP-EBDM	71%	81%	76%	31

SAP-EHS DPI	67%	57%	62%	14
SAP-FI	17%	33%	22%	9
SAP-FI/CA	74%	69%	72%	75
SAP-FO	100%	56%	71%	9
SAP-HR	86%	80%	83%	15
SAP-MD	36%	13%	20%	30
SAP-MDG	64%	54%	58%	26
SAP-MM	86%	71%	77%	34
SAP-PI	33%	33%	33%	3
SAP-PM	45%	30%	36%	33
SAP-SD	35%	60%	44%	15
SAP-SDD	53%	53%	53%	19
SAP-SRM	80%	80%	80%	10
SAP-SRP	100%	94%	97%	16
SAP-SSA	72%	74%	73%	53
SAP-VIM	14%	20%	17%	5
SAP-WASTE	37%	54%	44%	13
SAP-WM	53%	66%	59%	38
SIEBEL	86%	89%	87%	132
STAMPE_ISU	82%	84%	83%	61
Sac Ambiente	38%	60%	46%	5
ServiziOnLine	60%	50%	55%	6
Sistema Comuni	83%	71%	77%	7
Sistema DUALE	75%	75%	75%	4
Sito Gruppo***	81%	84%	83%	51
Sito ***Comm	77%	94%	85%	18

Sito In***	100%	60%	75%	10
YUBIK LEGALE	88%	58%	70%	12
YuBSC	96%	96%	96%	25
xxx ALTRO	17%	10%	12%	10
Accuracy	**72%**	**72%**	**72%**	**2081**
Macro avg	**62%**	**57%**	**57%**	**2081**
Weighted avg	**72%**	**72%**	**71%**	**2081**

Table A.3: LSTM complete classification report.

A.4 StarSpace

Class	Precision	Recall	F1-score	Support
4U	83	83	83	6
ARCHIBUS	67%	50%	57%	4
ARCHIFLOW Protocollo	84%	92%	88%	39
AZURE DataBricks	71%	71%	71%	7
App DPI	83%	83%	83%	6
App Geocall Pronto I	50%	33%	40%	3
App Prevent. Lavori	67%	50%	57%	4
BEAM	83%	100%	91%	55
Creditflow-CVP	100%	100%	100%	4
DOC1	88%	95%	91%	22
DOCFLOW	0%	0%	0%	3
ESRI GIS	62%	50%	56%	10
EngageOne	50%	50%	50%	8
GEOCALL	48%	76%	59%	21
HDA	82%	78%	80%	23
HEROKU	32%	58%	41%	12
HRNext	33%	33%	33%	3
HRNext-MasterData	67%	60%	63%	10
HRNext-Payroll	88%	93%	90%	40
HRNext-Time	90%	79%	84%	24
LIMS	90%	97%	94%	39
MARKETING CLOUD	100%	33%	50%	3
MDM - Vendita	0%	0%	0%	3

MULESOFT	0%	0%	0%	3
MyAcademy	100%	67%	80%	3
OIG	91%	84%	88%	76
OPENTEXT	100%	50%	67%	4
ORACLE HCM	65%	58%	61%	19
Oracle Note Spese	100%	67%	80%	3
PITECO	100%	100%	100%	6
Portale Web Terzisti	68%	88%	77%	17
REMA	100%	75%	86%	4
RPA-HOPE	75%	100%	86%	3
RPA-HVAC	100%	100%	100%	7
SALESFORCE GWM	95%	95%	95%	22
SALESFORCE ***VENDI	84%	89%	87%	164
SAP	0%	0%	0%	11
SAP-BI	70%	65%	67%	120
SAP-BO	50%	40%	44%	5
SAP-BRIM	100%	67%	80%	3
SAP-BW	77%	74%	75%	31
SAP-BW/4 HANA	93%	88%	91%	104
SAP-CO	60%	60%	60%	5
SAP-CRM ***tech	100%	67%	80%	6
SAP-CRM/Nuovo CCT	60%	67%	63%	9
SAP-DBO	73%	77%	75%	133
SAP-DM Commerciale	51%	61%	56%	31
SAP-DM Operations	76%	71%	74%	141
SAP-Data Service	100%	85%	92%	13

SAP-EBDM	78%	81%	79%	31
SAP-EHS DPI	73%	79%	76%	14
SAP-FI	20%	11%	14%	9
SAP-FI/CA	72%	76%	74%	75
SAP-FO	71%	56%	63%	9
SAP-HR	94%	100%	97%	15
SAP-MD	39%	40%	39%	30
SAP-MDG	75%	81%	78%	26
SAP-MM	78%	82%	80%	34
SAP-PI	50%	33%	40%	3
SAP-PM	61%	58%	59%	33
SAP-SD	100%	67%	80%	15
SAP-SDD	63%	63%	63%	19
SAP-SRM	88%	70%	78%	10
SAP-SRP	88%	94%	91%	16
SAP-SSA	93%	72%	81%	53
SAP-VIM	100%	60%	75%	5
SAP-WASTE	100%	46%	63%	13
SAP-WM	75%	71%	73%	38
SIEBEL	82%	91%	86%	132
STAMPE_ISU	89%	89%	89%	61
Sac Ambiente	80%	80%	80%	5
ServiziOnLine	50%	83%	62%	6
Sistema Comuni	88%	100%	93%	7
Sistema DUALE	75%	75%	75%	4
Sito Gruppo***	91%	96%	93%	51

Sito ***Comm	84%	89%	86%	18
Sito In***	82%	90%	86%	10
YUBIK LEGALE	83%	83%	83%	12
YuBSC	100%	100%	100%	25
xxx ALTRO	29%	20%	24%	10
Accuracy	**78%**	**78%**	**78%**	**2081**
Macro avg	**73%**	**69%**	**70%**	**2081**
Weighted avg	**78%**	**78%**	**78%**	**2081**

Table A.4: StarSpace complete classification report.

A.5 BERT: Italian based XXL

Class	Precision	Recall	F1-score	Support
4U	100%	100%	100%	6
ARCHIBUS	50%	50%	50%	4
ARCHIFLOW Protocollo	91%	82%	86%	39
AZURE DataBricks	50%	86%	63%	7
App DPI	0%	0%	0%	6
App Geocall Pronto I	0%	0%	0%	3
App Prevent. Lavori	100%	100%	100%	4
BEAM	88%	91%	89%	55
Creditflow-CVP	100%	100%	100%	4
DOC1	95%	95%	95%	22
DOCFLOW	0%	0%	0%	3
ESRI GIS	86%	60%	71%	10
EngageOne	75%	38%	50%	8
GEOCALL	68%	90%	78%	21
HDA	74%	74%	74%	23
HEROKU	42%	67%	52%	12
HRNext	100%	33%	50%	3
HRNext-MasterData	43%	30%	35%	10
HRNext-Payroll	77%	93%	84%	40
HRNext-Time	94%	67%	78%	24
LIMS	100%	100%	100%	39
MARKETING CLOUD	100%	33%	50%	3
MDM - Vendita	50%	33%	40%	3
MULESOFT	0%	0%	0%	3

MyAcademy	100%	100%	100%	3
OIG	84%	84%	84%	76
OPENTEXT	67%	50%	57%	4
ORACLE HCM	100%	58%	73%	19
Oracle Note Spese	75%	100%	86%	3
PITECO	100%	100%	100%	6
Portale Web Terzisti	56%	82%	67%	17
REMA	100%	50%	67%	4
RPA-HOPE	100%	100%	100%	3
RPA-HVAC	88%	100%	93%	7
SALESFORCE GWM	92%	100%	96%	22
SALESFORCE ***VENDI	85%	91%	88%	164
SAP	0%	0%	0%	11
SAP-BI	64%	80%	71%	120
SAP-BO	100%	40%	57%	5
SAP-BRIM	100%	100%	100%	3
SAP-BW	84%	68%	75%	31
SAP-BW/4 HANA	87%	93%	90%	104
SAP-CO	50%	20%	29%	5
SAP-CRM ***tech	75%	50%	60%	6
SAP-CRM/Nuovo CCT	42%	56%	48%	9
SAP-DBO	72%	80%	76%	133
SAP-DM Commerciale	76%	52%	62%	31
SAP-DM Operations	77%	77%	77%	141
SAP-Data Service	100%	92%	96%	13
SAP-EBDM	79%	87%	83%	31

SAP-EHS DPI	60%	86%	71%	14
SAP-EHS DPI	50%	11%	18%	9
SAP-FI/CA	67%	83%	74%	75
SAP-FO	100%	67%	80%	9
SAP-HR	100%	100%	100%	15
SAP-MD	53%	27%	36%	30
SAP-MDG	65%	77%	70%	26
SAP-MM	79%	79%	79%	34
SAP-PI	100%	33%	50%	3
SAP-PM	71%	30%	43%	33
SAP-SD	56%	60%	58%	15
SAP-SDD	91%	53%	67%	19
SAP-SRM	75%	90%	82%	10
SAP-SRP	94%	94%	94%	16
SAP-SSA	93%	72%	81%	53
SAP-VIM	75%	60%	67%	5
SAP-WASTE	100%	62%	76%	13
SAP-WM	69%	76%	72%	38
SIEBEL	88%	92%	90%	132
STAMPE_ISU	81%	82%	81%	61
Sac Ambiente	100%	60%	75%	5
ServiziOnLine	100%	83%	91%	6
Sistema Comuni	100%	100%	100%	7
Sistema DUALE	100%	75%	86%	4
Sito Gruppo***	96%	90%	93%	51
Sito ***Comm	89%	89%	89%	18

Sito In***	77%	100%	87%	10
YUBIK LEGALE	91%	83%	87%	12
YuBSC	86%	100%	93%	25
xxx ALTRO	33%	30%	32%	10
Accuracy	**79%**	**79%**	**79%**	**2081**
Macro avg	**76%**	**68%**	**70%**	**2081**
Weighted avg	**79%**	**79%**	**78%**	**2081**

Table A.5: BERT Italian based XXL pretrained model (pooler output) complete classification report.

A.6 GilBERTo

Class	Precision	Recall	F1-score	Support
4U	100%	100%	100%	6
ARCHIBUS	43%	75%	55%	4
ARCHIFLOW Protocollo	91%	77%	83%	39
AZURE DataBricks	67%	86%	75%	7
App DPI	60%	100%	75%	6
App Geocall Pronto I	0%	0%	0%	3
App Prevent. Lavori	100%	100%	100%	4
BEAM	83%	91%	87%	55
Creditflow-CVP	100%	100%	100%	4
DOC1	95%	95%	95%	22
DOCFLOW	100%	67%	80%	3
ESRI GIS	100%	60%	75%	10
EngageOne	42%	62%	50%	8
GEOCALL	67%	86%	75%	21
HDA	100%	70%	82%	23
HEROKU	44%	67%	53%	12
HRNext	67%	67%	67%	3
HRNext-MasterData	70%	70%	70%	10
HRNext-Payroll	93%	95%	94%	40
HRNext-Time	95%	83%	89%	24
LIMS	100%	100%	100%	39
MARKETING CLOUD	50%	33%	40%	3
MDM - Vendita	0%	0%	0%	3
MULESOFT	0%	0%	0%	3

MyAcademy	100%	100%	100%	3
OIG	87%	88%	88%	76
OPENTEXT	100%	50%	67%	4
ORACLE HCM	67%	63%	65%	19
Oracle Note Spese	100%	67%	80%	3
PITECO	100%	100%	100%	6
Portale Web Terzisti	52%	82%	64%	17
REMA	100%	75%	86%	4
RPA-HOPE	100%	100%	100%	3
RPA-HVAC	100%	100%	100%	7
SALESFORCE GWM	81%	95%	88%	22
SALESFORCE ***VENDI	88%	92%	90%	164
SAP	0%	0%	0%	11
SAP-BI	71%	75%	73%	120
SAP-BO	100%	40%	57%	5
SAP-BRIM	100%	100%	100%	3
SAP-BW	72%	68%	70%	31
SAP-BW/4 HANA	87%	93%	90%	104
SAP-CO	50%	40%	44%	5
SAP-CRM ***tech	80%	67%	73%	6
SAP-CRM/Nuovo CCT	56%	56%	56%	9
SAP-DBO	77%	82%	79%	133
SAP-DM Commerciale	49%	61%	54%	31
SAP-DM Operations	73%	72%	73%	141
SAP-Data Service	92%	85%	88%	13
SAP-EBDM	76%	84%	80%	31

SAP-EHS DPI	100%	71%	83%	14
SAP-EHS DPI	100%	44%	62%	9
SAP-FI/CA	86%	81%	84%	75
SAP-FO	100%	67%	80%	9
SAP-HR	94%	100%	97%	15
SAP-MD	38%	17%	23%	30
SAP-MDG	63%	65%	64%	26
SAP-MM	74%	74%	74%	34
SAP-PI	50%	33%	40%	3
SAP-PM	60%	55%	57%	33
SAP-SD	56%	60%	58%	15
SAP-SDD	86%	63%	73%	19
SAP-SRM	71%	100%	83%	10
SAP-SRP	100%	94%	97%	16
SAP-SSA	78%	79%	79%	53
SAP-VIM	75%	60%	67%	5
SAP-WASTE	67%	77%	71%	13
SAP-WM	64%	76%	70%	38
SIEBEL	88%	89%	89%	132
STAMPE_ISU	87%	85%	86%	61
Sac Ambiente	50%	20%	29%	5
ServiziOnLine	83%	83%	83%	6
Sistema Comuni	86%	86%	86%	7
Sistema DUALE	100%	100%	100%	4
Sito Gruppo***	94%	92%	93%	51
Sito ***Comm	71%	83%	77%	18

Sito In***	82%	90%	86%	10
YUBIK LEGALE	100%	67%	80%	12
YuBSC	86%	100%	93%	25
xxx ALTRO	67%	20%	31%	10
Accuracy	**80%**	**80%**	**80%**	**2081**
Macro avg	**76%**	**72%**	**73%**	**2081**
Weighted avg	**80%**	**80%**	**79%**	**2081**

Table A.6: GilBERTo (4 layers concatenation) complete classification report.

A.7 Hybrid System - SVM based

Class	Precision	Recall	F1-score	Support
4U	100%	83%	91%	6
ARCHIBUS	100%	100%	100%	4
ARCHIFLOW Protocollo	98%	100%	99%	39
AZURE DataBricks	100%	100%	100%	7
App DPI	57%	67%	62%	6
App Geocall Pronto I	67%	67%	67%	3
App Prevent, Lavori	80%	100%	89%	4
BEAM	89%	93%	91%	55
Creditflow-CVP	100%	100%	100%	4
DOC1	85%	100%	92%	22
DOCFLOW	100%	100%	100%	3
ESRI GIS	100%	90%	95%	10
EngageOne	36%	50%	42%	8
GEOCALL	80%	95%	87%	21
HDA	96%	96%	96%	23
HEROKU	80%	33%	47%	12
HRNext	100%	67%	80%	3
HRNext-MasterData	90%	90%	90%	10
HRNext-Payroll	100%	98%	99%	40
HRNext-Time	96%	100%	98%	24
LIMS	98%	100%	99%	39
MARKETING CLOUD	0%	0%	0%	3
MDM - Vendita	100%	100%	100%	3
MULESOFT	0%	0%	0%	3

MyAcademy	100%	100%	100%	3
OIG	91%	93%	92%	76
OPENTEXT	80%	100%	89%	4
ORACLE HCM	68%	79%	73%	19
Oracle Note Spese	100%	100%	100%	3
PITECO	86%	100%	92%	6
Portale Web Terzisti	73%	69%	71%	16
REMA	60%	75%	67%	4
RPA-HOPE	75%	100%	86%	3
RPA-HVAC	100%	100%	100%	7
SALESFORCE GWM	96%	100%	98%	22
SALESFORCE ***VENDI	83%	95%	89%	164
SAP	0%	0%	0%	11
SAP-BI	74%	81%	77%	120
SAP-BO	83%	100%	91%	5
SAP-BRIM	60%	100%	75%	3
SAP-BW	88%	90%	89%	31
SAP-BW/4 HANA	97%	97%	97%	104
SAP-CO	75%	60%	67%	5
SAP-CRM ***tech	67%	100%	80%	6
SAP-CRM/Nuovo CCT	86%	67%	75%	9
SAP-DBO	79%	86%	82%	133
SAP-DM Commerciale	57%	42%	48%	31
SAP-DM Operations	79%	78%	78%	141
SAP-Data Service	100%	92%	96%	13
SAP-EBDM	97%	90%	93%	31

SAP-EHS DPI	83%	71%	77%	14
SAP-FI	50%	56%	53%	9
SAP-FI/CA	84%	83%	83%	75
SAP-FO	78%	78%	78%	9
SAP-HR	88%	93%	90%	15
SAP-MD	70%	47%	56%	30
SAP-MDG	75%	58%	65%	26
SAP-MM	86%	94%	90%	34
SAP-PI	20%	33%	25%	3
SAP-PM	69%	61%	65%	33
SAP-SD	88%	93%	90%	15
SAP-SDD	100%	63%	77%	19
SAP-SRM	100%	80%	89%	10
SAP-SRP	94%	100%	97%	17
SAP-SSA	82%	87%	84%	53
SAP-VIM	100%	100%	100%	5
SAP-WASTE	86%	46%	60%	13
SAP-WM	74%	82%	78%	38
SIEBEL	92%	91%	91%	132
STAMPE_ISU	96%	82%	88%	61
Sac Ambiente	80%	80%	80%	5
ServiziOnLine	100%	100%	100%	6
Sistema Comuni	86%	86%	86%	7
Sistema DUALE	100%	75%	86%	4
Sito Gruppo***	100%	96%	98%	51
Sito ***Comm	95%	100%	97%	18

Sito In***	83%	100%	91%	10
YUBIK LEGALE	100%	100%	100%	12
YuBSC	100%	100%	100%	25
xxx ALTRO	33%	10%	15%	10
Accuracy	**85%**	**85%**	**85%**	**2081**
Macro avg	**81%**	**81%**	**80%**	**2081**
Weighted avg	**85%**	**85%**	**85%**	**2081**

Table A.7: Hybrid System SVM based complete classification report.

References

[1] Martn Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *arXiv preprint arXiv:1603.04467* (2015).

[2] Suphamongkol Akkaradamrongrat, Pornpimon Kachamas, and Sukree Sinthupinyo. "Text Generation for Imbalanced Text Classification". In: (2019).

[3] Thomas Bayes. "An Essay towards solving a Problem in the Doctrine of Chances". In: (1763).

[4] P. Billingsley. "The Annals of Mathematical Statistics". In: (1961).

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Toolkit (NLTK)*. The NLTK Project, 2009. URL: `http://www.nltk.org/`.

[6] *ClearNLP Constituent-to-Dependency Conversion*. URL: `https://github.com/clir/clearnlp-guidelines/blob/master/md/components/dependency_conversion.md` (visited on 2023).

[7] *Common crawl corpus*. URL: `https://commoncrawl.org/` (visited on 2023).

[8] Alexis Conneau et al. "Unsupervised Cross-lingual Representation Learning at Scale". In: *CoRR* (2019). URL: `http://arxiv.org/abs/1911.02116`.

[9] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[10] Docker. *Docker - Build, Ship, and Run Any App, Anywhere*. URL: `https://www.docker.com/` (visited on 2023).

[11] Python Software Foundation. *Python*. 2001. URL: `https://www.python.org/`.

[12] The Apache Software Foundation. *Groovy: a powerful multi-faceted programming language for the JVM platform*. 2003. URL: `https://groovy-lang.org/`.

[13] *GilBERTo: An Italian pretrained language model based on RoBERTa*. https://github.com/idb-ita/GilBERTo. 2019.

[14] *Google Colaboratory*. URL: https://colab.research.google.com/ (visited on 2023).

[15] Charles R. Harris et al. *NumPy: array processing for numbers, strings, records, and objects*. 2006. URL: https://numpy.org/.

[16] Hochreiter and Schmidhuber. "Long Short-Term Memory". In: (1997).

[17] Matthew Honnibal and Ines Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing". In: *To appear* 999 (2017), pp. 1–21. DOI: 10.1016/j.jocs.2017.11.006.

[18] J. D. Hunter. *Matplotlib: A 2D Graphics Environment*. 2007. URL: http://matplotlib.org.

[19] JetBrains. *JetBrains*. https://www.jetbrains.com/.

[20] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *arXiv preprint arXiv:1607.01759* (2016).

[21] *Kaggle*. URL: https://www.kaggle.com/ (visited on 2023).

[22] Kubernetes. *Kubernetes - Production-Grade Container Orchestration*. URL: https://kubernetes.io/ (visited on 2023).

[23] *Language detection dataset*. URL: https://www.kaggle.com/datasets/basilb2s/language-detection (visited on 2023).

[24] Daniele Licari and Giovanni Comandè. "ITALIAN-LEGAL-BERT: A Pretrained Transformer Language Model for Italian Law". In: (2020).

[25] Yiheng Liu et al. "Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models". In: *arXiv preprint arXiv:2304.01852* (5/2023).

[26] MarketsandMarkets. *Natural Language Processing (NLP) Market by Component (Solutions & Services), Application (Sentiment Analysis, Social Media Monitoring), Technology (IVR, OCR, Auto Coding), Vertical (BFSI, Retail & eCommerce, IT & ITES) & Region - Global Forecast to 2027*. https://www.marketsandmarkets.com/Market-Reports/natural-language-processing-nlp-825.html. 2022.

[27] *Markovify*. URL: https://github.com/jsvine/markovify (visited on 2023).

[28] Wes McKinney. *Pandas: powerful Python data analysis toolkit*. 2010. URL: https://pandas.pydata.org/.

[29] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint arXiv:1301.3781* (2013).

[30] Weili Nie, Nina Narodytska, and Ankit B. Patel. "Relational Generative Adversarial Networks For Text Generation". In: *ICLR* (2019). URL: https://openreview.net/pdf?id=rJedV3R5tm.

[31] *OntoNotes 5*. URL: https://catalog.ldc.upenn.edu/LDC2013T19 (visited on 2023).

[32] *OPUS corpus*. URL: https://opus.nlpl.eu/ (visited on 2023).

[33] *OSCAR corpus*. URL: https://oscar-project.org/ (visited on 2023).

[34] *Papaluca dataset*. URL: https://huggingface.co/datasets/papluca/language-identification (visited on 2023).

[35] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Stanford University* (2014).

[36] Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. "Mimicking Word Embeddings using Subword RNNs". In: (Sept. 2017). URL: https://aclanthology.org/D17-1010.

[37] Marco Polignano et al. "ALBERTO: Italian BERT Language Understanding Model for NLP Challenging Tasks Based on Tweets". In: (2019).

[38] Alec Radford et al. "Improving Language Understanding by Generative Pre-Training". In: *Open AI* (2018).

[39] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* (1986).

[40] *SETimes corpus*. URL: http://nlp.ffzg.hr/resources/corpora/setimes/ (visited on 2023).

[41] Nakatani Shuyo. *Language Detection Library for Java*. 2010. URL: http://code.google.com/p/language-detection/.

[42] *Simplemma*. URL: https://pypi.org/project/simplemma/ (visited on 2023).

[43] *spaCy model's hub*. URL: https://spacy.io/models (visited on 2023).

[44] Statista. *Revenues from the natural language processing (NLP) market worldwide from 2017 to 2025*. https://www.statista.com/statistics/607891/worldwide-natural-language-processing-market-revenues. 2018.

[45] Chi Sun et al. "How to Fine-Tune BERT for Text Classification?" In: *arXiv preprint arXiv:1905.05583* (2020).

[46] *Tatoeba corpus*. URL: https://tatoeba.org/it/ (visited on 2023).

[47] *UmBERTo: an Italian Language Model trained with Whole Word Masking*. https://github.com/musixmatchresearch/umberto. 2019.

[48] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems* (2017).

[49] Sida Wang and Christopher D. Manning. "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification". In: *Stanford University* (2012).

[50] *Wikipedia*. URL: https://en.wikipedia.org/wiki/Main_Page (visited on 2023).

[51] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: (2020). URL: https://aclanthology.org/2020.emnlp-demos.6.

[52] *WordNet 3.0 database*. URL: https://wordnet.princeton.edu/ (visited on 2023).

[53] Ledell Wu et al. "StarSpace: Embed All The Things!" In: *Facebook AI Research* (2017).

Ringraziamenti

Innanizi tutto ringrazio l'università di Padova che mi ha permesso di intraprendere questo percorso e ha facilitato il mio ingresso nel mondo del lavoro dandomi opportunità e competenze, ringrazio il mio relatore, il professor Michele Rossi per aver reso possibile questa intership, e il mio tutor aziendale il dottor Davide Bastianetto, che mi ha seguito durante lo stage trasmettendomi parte della sua esperienza.

Ringrazio mamma e papà, che durante questi anni di studio mi hanno sempre supportato, incoraggiato e spronato, anche quando io stesso non ci credevo più e l'idea di abbandonare aveva bussato alla mia porta.

Ringrazio tutta la mia famiglia in generale, Monica e Matteo, Manuel e Elena, zii e zie, che negli anni li ho sempre sentiti vicini appoggiando a pieno il percorso che stavo intraprendendo.

Grazie ai miei ex compagni di corso del "Grupo Cojones", per gli anni passati assieme tra le aule del DEI, grazie perchè avete reso fantastico questo percorso, dalle prime partite a curve crash fino alla fine, anche se le nostre strade hanno preso direzioni diverse, siamo sempre rimasti uniti, formando un gruppo solido che dura nel tempo, ancora oggi che non possiamo più definirci compagni di corso, ma semplicemente grandi amici.

Grazie alla "Trattoria da Cioro" che negli anni dell'università mi ha accolto come una seconda famiglia, dandomi lavoro e facendomi sentire a casa, non dimenticherò mai il tempo passato in quella cucina.

Per ultimi, ma non meno importanti, i miei amici di sempre, citarvi tutti è impossibile, ma voglio ringraziarvi per essermi stati accanto sempre, facendomi passare gli anni migliori della mia vita. Anche con chi ha preso scelte diverse dopo le superiori c'è sempre stato rispetto reciproco, e siete sempre riusciti a comprendere anche i momenti più difficili e stressanti che comporta una sessione d'esami.

Grazie a tutti, perchè anche nei momenti più difficili della mia vita personale fino ad ora, quando più volte sono caduto, ci siete sempre stati a rallegrare le mie giornate e darmi forza. Permettendomi cosi, di riuscire a portare a termine questo percorso di studi.

Grazie!

Francesco Fregona