FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Classifying Medical Specialty From Unstructured Text

André Abrantes Tavares Paiva Machado

FINAL REPORT



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. João Mendes Moreira

Supervisor at Glintt: Pedro Rocha

January 30, 2019

Classifying Medical Specialty From Unstructured Text

André Abrantes Tavares Paiva Machado

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof.
External Examiner: Prof.
Supervisor: Prof. João Mendes Moreira

January 30, 2019

Abstract

When a patient wants to schedule a medical appointment, there is already no need of being physically present in the clinic or hospital to make that booking. That same person can, in most cases, handle that process online.

The lack of contextual knowledge can induce doubts about the adequate specialty one should visit according to the symptoms presented. This can, consequently, lead to inadequate appointments that have costs for both the patients and the institutions.

This document proposes a conversational agent for medical appointments scheduling while focusing mainly on the topic of correctly classifying a specialty according to the frequency of the words that appear in patients' symptom descriptions.

We use a machine learning approach to solve this problem. First, we identify the company's business objectives regarding this problem, and then we make an analysis of the dataset provided and extract some knowledge from it. We ended up realizing that the data available wasn't adequate for the problem at hand, so we explain how this problem was faced.

After performing multiple operations on the new data collected, we ended up testing several classification algorithms and from that resulted a model that presented great validation results.

Given the results obtained we are confident that the developed model and its integration in a conversational agent deployed as a tool for medical appointments' scheduling may be valuable for both patients and clinical institutions. We also conclude that in order to improve these results a more significant effort from the part of the companies to collect valuable data is required.

Contents

1 Introduction **1**
1.1 Methodology . . . 1
1.2 Document Structure . . . 2

2 Business Understanding **5**
2.1 Business Objective . . . 5
2.2 Project Goals . . . 6

3 Basic Concepts **7**
3.1 Conversational Systems . . . 7
3.1.1 Rule-Based Systems . . . 8
3.1.1.1 ELIZA . . . 8
3.1.1.2 ALICE . . . 10
3.1.2 Corpus-Based Systems . . . 10
3.1.2.1 Information-Retrieval Systems . . . 10
3.1.2.2 Learning-Based systems . . . 10
3.1.3 Frame-Based Systems . . . 12
3.2 Machine Learning . . . 14
3.2.1 Regression Models . . . 15
3.2.2 Classification Models . . . 15
3.2.2.1 Logistic Regression . . . 15
3.2.2.2 Naive Bayes . . . 16
3.2.2.3 Support Vector Machines . . . 17
3.2.2.4 Neural networks . . . 18
3.2.2.4.1 Artificial Neurons . . . 18
3.2.2.4.2 Feedforward Neural Network . . . 22
3.2.2.4.3 Cost Function . . . 22
3.2.2.4.4 Gradient Descent . . . 23
3.2.2.4.5 L-BFGS . . . 23
3.2.3 Regularization . . . 24
3.2.3.1 Ridge Regression . . . 25
3.2.3.2 Lasso Regression . . . 25
3.2.3.3 Elastic Net . . . 26

4 Data Understanding **27**
4.1 Data Overview . . . 27
4.2 Understanding the data . . . 28
4.3 Collecting data . . . 31

4.4 Some Considerations . . . 32

5 Data Preparation **33**
5.1 Data Cleaning . . . 33
5.2 Data Pre-Processing . . . 34
5.2.1 Text Normalization . . . 34
5.3 Feature Extraction . . . 34
5.3.1 Term Ranking . . . 34
5.3.2 Feature Hashing . . . 35
5.3.3 Sparsity . . . 36

6 Modeling **37**
6.1 Model Structure . . . 37
6.1.1 Classification Algorithm . . . 38
6.2 Validation methods . . . 38
6.2.1 Holdout . . . 38
6.2.2 K-fold cross-validation . . . 39
6.3 Data splitting . . . 39
6.4 Validation metrics . . . 40
6.4.1 Accuracy . . . 41
6.4.2 Precision and Recall . . . 42
6.4.3 F-Score . . . 42
6.5 Validation Results . . . 42
6.6 Integration . . . 44

7 Evaluation **47**
7.1 Results Evaluation . . . 47
7.2 Process Review . . . 48
7.3 Next Steps . . . 49

A Parameter Tuning **51**
A.1 OvR SVM Tuning . . . 51
A.2 OvR LR Tuning . . . 51
A.3 MPC Tuning . . . 52
A.4 Train and Test times . . . 52

References **53**

List of Figures

1.1 Phases of the CRISP-DM reference model 2

3.1 An architecture representation of a simple seq2seq model 11
3.2 A representation of a multi-layered neural network with an attention mechanism as proposed by Luong *et al.* [LPM15] 12
3.3 Sigmoid function 16
3.4 A representation of a separable problem in a 2D space. The optimal hyperplane represents the margin of largest separation between the two classes [VC95]. . . . 18
3.5 Schematic representation of an artificial neuron. 19
3.6 A graphic representation of a set of possible Activation Functions. 20
3.7 A graphic representation of two sets of 2D points and the obtained classification regions using a ReLU activation function. 21
3.8 Neural network with one hidden layer applied to linearly inseparable data. 21
3.9 Representation of a Feedforward Neural Network 22
3.10 A graphic representation of the results obtained from two neural networks with and without regularization on the same dataset. (Only showing train data points) . 24

4.1 A basic overview of the dataset including age, gender and medical specialty counts. 28
4.2 Patients distribution through destinies according to gender. 29
4.3 A basic overview of the dataset regarding age distribution throughout the different specialties. 30

5.1 Graphic representation of a dimensionality reduction process via Feature Hashing. 35
5.2 Percentage of collisions with respect to the hash vector size using MurmurHash 3 algorithm on a dataset with exactly 1538 different terms. 36

6.1 Representation of the implemented pipeline. 37
6.2 Representation of the k-fold cross-validation method with k=3 39
6.3 Distribution of samples through each class on the new dataset. 40
6.4 Representation of the stratified sampling technique 40
6.5 Normalized confusion matrix resultant from Multilayer Perceptron Classifier, One vs. Rest SVM and One vs. Rest LR. 43
6.6 Train and test times for all four algorithms. 44

LIST OF FIGURES

List of Tables

3.1 An example of different possible responses to the same user utterance 11
3.2 An outline of a keyframe structure from GUS system proposed by Bobrow *et al.* . 13
3.3 A representation of the Domain, Intent and Entities extracted from the user utterance "Book me a flight to Cairo for tomorrow." 14

6.1 Confusion matrix representation for a binary classification problem 41
6.2 List of metrics used for evaluation. 41
6.3 Confusion matrix for the fraud classification example. 42
6.4 Evaluation results for the different tested classification algorithms (values in percentage). 43

A.1 Parameter tuning results for OvR SVM. 51
A.2 Parameter tuning results for OvR LR. 52
A.3 Parameter tuning results for MPC. 52
A.4 Train and test times for all four algorithms (in seconds). 52

LIST OF TABLES

Abbreviations

ACE	Artificial Conversational Entity
AI	Artificial Intelligence
AIML	Artificial Intelligence Markup Language
CLI	Command Line Interface
GUI	Graphical User Interface
GUS	Genial Understander System
IR	Information Retrieval
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
LR	Logistic Regression
LSTM	Long short-term memory
LSVM	Linear Support Vector Machine
MPC	Multilayer Perceptron Classifier
MT	Machine Translation
NLP	Natural Language Processing
NLU	Natural Language Understanding
OvR	One vs. Rest (One against All)
OWL-QN	Orthant-Wise Limited Memory Quasi-Newton
PDA	Personal Digital Assistants
RNN	Recurrent Neural Network
seq2seq	Sequence to Sequence
SMT	Statistical Machine Translation
SVM	Support Vector Machine
XML	Extensible Markup Language

Chapter 1

Introduction

This chapter introduces this dissertation, by first presenting the used methodology in section 1.1 and then, section 1.2 makes a brief description of how this document is structured.

1.1 Methodology

For the sake of a structured and organized software development process, it is essential to divide the work into several different and well-defined parts. This division and the requirements for each phase will depend on the methodology which should be adopted according to the nature of the project.

As the reader will notice this document describes the development of a data-mining project and, for that reason, its structure is based on the CRISP-DM reference model.

Cross-Industry Standard Process for Data Mining is an industry-proven methodology adequate for data mining projects. The model reference provides an overview of a project life cycle and consists of six main phases as represented in Img.1.1

The sequence between phases is not rigid, and one can switch between them as required. Furthermore, this methodology considers that the data mining process never ends, *i.e.*, when the solution is deployed new questions may arise that will give origin to new studies [CCK$^+$00].

We now present a brief explanation of each phase of this process:

1. Business Understanding - This is the initial phase of the data mining project. Here we should describe the context where the project is inserted and determine the business objectives. It is also crucial at this point to establish our goals, *i.e.*, what we want to achieve by the end of the project;

2. Data Understanding - Here we need to collect the data available, describe it and figure out what value we can extract from it. After this process we should be able to access the data regarding its quality;

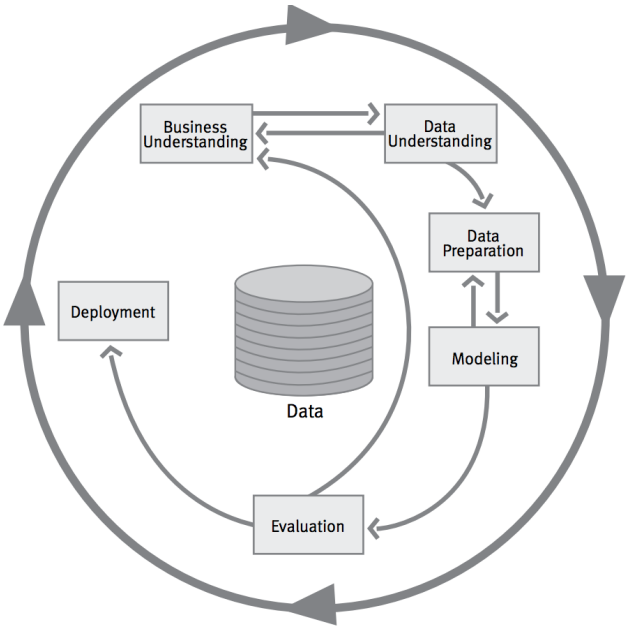


Figure 1.1: Phases of the CRISP-DM reference model

3. Data Preparation - In this phase we already have one or more datasets to work on. We now need to clean the data, obtain derived attributes that may be of value, merge all the data required and extract a set of features from it;

4. Modeling - This is when we select the modeling techniques we will be using, and the parameter tuning necessary for all models is performed. Here we also validate the models and assess each one of them.

5. Evaluation - This is the phase where we evaluate the results obtained not only concerning the models' results but also regarding the achievement of the business requirements. After this evaluation, we should make a review of the process and identify the next steps for future work.

6. Deployment - In this phase, we should create a plan that describes the deployment strategy and another one on how to handle future monitoring and maintenance that may be required, as well as a final report and a review of the project.

1.2 Document Structure

This document is structured following the previously referred methodology, *i.e.*, each chapter (excepting the current and chapter 3) describes one of the phases of the data mining process based on the CRISP-DM reference model.

Chapter 3 provides the reader with a brief explanation of the basic concepts one should grasp in order to understand the rest of the document.

Chapters 2, 4, 5, 6 and 7 describe the phases of Business Understanding, Data Understanding, Data Preparation, Modeling, and Evaluation respectively.

Notice that the phase concerning Deployment is not being considered in this document as most of the deployment process is supposed to be handled by the company.

Chapter 2

Business Understanding

This chapter focuses on the first phase of this data-mining project. In section 2.1 we try to explore the context in which this project is inserted and expose the business motivation for its development. After talking about the business objectives, in section 2.2 we define the project requirements while considering the previously referred business objectives.

2.1 Business Objective

When a patient calls a hospital or, in most cases, visits its website, he has the possibility to schedule an appointment without the need of being physically present in that same institution. When this happens, the patient may know the medical specialty he wants to attend, or he presents a set of symptoms and wants to make an appointment according to them.

The main problem with the latter option is that sometimes, the lack of contextual knowledge forbids the patient from making a correct decision, which can lead to an inadequate appointment and possibly to the need of making another one. This can have some consequences to the hospital, such as the unnecessary occupation of needed resources, and the patients, as their insurance may not be willing to cover two medical appointments.

There were some successful attempts to extract information from medically related texts [SMO+10] and several Machine Learning Methods that allow us to map patient's conditions according to their symptoms. With this approach is then possible to infer the medical specialty a patient should be redirected to. Some hospitals have already appointment scheduling systems capable of doing this by enforcing the user to, for example, fill out a form.

A system capable of performing the same task through a conversational process could be of extreme value as it would potentially be more intuitive, less prone to mistakes and, if integrated with a *Speech to Text* and *Text to Speech* services it could replace the medical appointment scheduling process via telephone.

Some startups[1][2][3][4] are already offering healthcare chatbots that perform assisted diagnosis. Although being conversational agents, the user is only allowed to select symptoms that the app provides via, for example, a drop-down menu. Despite its usefulness, this method presents some limitations as it cannot understand the patient's symptoms when presented in a different format than the one expected.

The following project will be developed at *Glintt*[5], a leading player on the healthcare IT services with the supervision of Pedro Rocha. The primary business objective with this project is to improve the current appointment scheduling systems employing a conversational system in order to mitigate the previously referred appointment problems.

The implemented system can later be used for example to replace or as first service line for healthcare facilities' call centers, possibly allowing them to reduce unnecessary costs.

2.2 Project Goals

This study proposes an approach to the development of a domain-specific chatbot for healthcare services, which is capable of recommending a medical specialty according to patients' symptoms.

The main challenge of this project is not the development of the conversational system as several frameworks facilitate that operation and, for that reason, the conception of the chatbot will not be extensively detailed in this document. This project will focus mainly on the problem of correctly classifying a medical specialty when a patient describes his symptoms in a free text format.

Having all this in consideration, the goals for the project to be completed consist of a final application:

1. Where a user can describe his symptoms through a text-based natural language conversation;
2. That based on the information provided on the dialog can infer the adequate medical specialty for the patient;
3. That allows the user to express himself through a fluid, natural language dialog;
4. Developed with and according to the state of the art technologies and practices.

[1] `https://www.your.md/`
[2] `http://www.sensely.com/`
[3] `https://infermedica.com/`
[4] `https://www.florence.chat/`
[5] `https://www.glintt.com/`

Chapter 3

Basic Concepts

This chapter exposes some basic concepts required to understand the rest of this document. Section 3.1 makes a brief introduction about Conversational Systems since they were first introduced in the mid 60s until today. Then, in section 3.2 we introduce some machine learning methods and algorithms that will be useful for the development of this project, with a more significant focus on classification models.

3.1 Conversational Systems

In the last decades, there was a massive revolution in the way users interact with computers. In the mid-1960s the CLI (Command Line Interface) was the primary mean of interaction with a computer, and it continued to be used throughout the 1970s and 1980s.

In 1979 the PERQ workstation was released into the market. This was the first commercially produced computer with a GUI (Graphical User Interface). This type of interface uses visual representations of files, programs, windows and actions leveraging many of the mental models that users already have from the real world. This made the interaction with the machine more natural, and they are today's primary way of interacting with a computer.

Nowadays digital assistants driven by Artificial Intelligence like *Siri (Apple)*[1], *Google Assistant (Google)*[2], *Cortana (Microsoft)*[3] and *Alexa (Amazon)*[4] are gaining popularity, and a relatively new way of interacting with a computer is arising - the Conversational User Interfaces.

Before going in-depth into Conversational Systems, it is important to give a brief definition of what a conversation is. According to the Cambridge dictionary a conversation refers to "(a) talk between two or more people in which thoughts, feelings, and ideas are expressed, questions are asked and answered, or news and information are exchanged".

[1] `https://www.apple.com/ios/siri/`
[2] `https://assistant.google.com/`
[3] `https://www.microsoft.com/cortana/`
[4] `https://developer.amazon.com/alexa-voice-service`

This definition reveals that for a conversation to happen, at least two entities must exist and the initiative to start it can come from each one of them, something that Filip Radlinski and Nick Craswell call mixed initiative [RC17]. Besides that, they also defend that a conversational system should have the following five properties:

1. User Revealment - The system helps the user express (potentially discover) their true information need, and possibly also long-term preferences;
2. System Revealment - The system reveals to the user its capabilities and corpus, building the user's expectations of what it can and cannot do;
3. Mixed Initiative - The system and user both can take the initiative as appropriate;
4. Memory - The user can reference past statements, which implicitly also remain true unless contradicted;
5. Set Retrieval - The system can reason about the utility of sets of complementary items;

One of the most popular applications of this kind of systems is what we call the Chatbots also known as conversational agents, chat robot, talk bot, chatterbot, among others.

Chatbots are systems designed to simulate human conversations. They take user's sentences as an input and return a textual response that makes some sense in the conversation context. Like almost everything else in language processing, conversational agent architectures fall into one of these two classes: Rule-Based and Corpus-Based Systems.

3.1.1 Rule-Based Systems

Rule-Based (or Expert) Systems are the simplest form of Artificial Intelligence [GA11]. These systems represent knowledge as a set of rules that conduct the program in a certain way depending on the different possible situations. This type of architecture may be useful in cases where the domain is very specific and where the problem area can be expressed as a set of *if-then* rules.

3.1.1.1 ELIZA

With the goal of showing the superficiality of communication between man and machine, Joseph Weizenbaum developed ELIZA from 1964 to 1966 [Wei66]. This was one of the first chatbots ever created and was one of the first programs capable of attempting the Turing Test.

To make this possible Weizenbaum defined those who would be "the fundamental technical problems" that the system would have to face:

1. The identification of the "most important" keyword occurring in the input message;
2. The identification of some minimal context within which the chosen keyword appears;
3. The choice of an appropriate transformation rule and the making of the transformation itself;
4. The provision of a mechanism that will permit ELIZA to respond when no keywords are detected in the input text;

5. The provision of machinery that facilitates the editing of the ELIZA's script.

ELIZA starts by reading the text in search of specific keywords. If at least one is found, a response is constructed according to a rule associated with the "most important" keyword. If none is detected, a content-free sentence or a previous transformation is retrieved [Wei66]. This set of transformation rules and keywords consist of ELIZA's script. It's important to notice that this script is nothing more than data, allowing the bot to be adaptable to recognize different patterns and to generate different responses [Wei66]. Its most famous script simulated a Rogerian psychotherapist, the DOCTOR, that always responded to user inputs with non-directional sentences.

Let us take the following example:

HUMAN: Men are all alike.
ELIZA: In what way?
HUMAN: They're always bugging us about something.
ELIZA: Can you think of a special example?
HUMAN: Well, my boyfriend made me come here.
ELIZA: Your boyfriend made you come here?
HUMAN: He says I'm depressed much of the time.
ELIZA: I'm sorry to hear that you are depressed much of the time.
HUMAN: It's true. I'm unhappy.
ELIZA: Do you think coming here will help you not to be unhappy?
HUMAN: I need some help, that much seems certain.

ELIZA uses simple pattern-matching techniques to recognize user inputs to parse sentences like "He says I'm depressed much of the time" into something like "(A) I'm (B)". The system then creates an appropriate response according to the rule associated with the "most important" keyword. In this case, it generates the sentence "I'm sorry to hear that you are (B).". This simple technique succeeds in this domain because ELIZA does not need to know anything to mimic a Rogerian psychotherapist [JM17]. The system architecture can be summarized as follows [JM17]:

```
1  function ELIZA GENERATOR(user sentence) returns response
2    Find the word w in sentence that has the highest keyword rank
3    if w exists
4        Choose the highest ranked rule r for w that matches sentence
5        response <- Apply the transform in r to sentence
6        if w = 'my'
7            future <- Apply a transformation from the 'memory' rule list to sentence
8            Push future onto memory stack
9    else (no keyword applies)
10 x     either
11           response <- Apply the transform for the NONE keyword to sentence
12       or
13           response <- Pop the top response from the memory stack
14   return(response)
```

3.1.1.2 ALICE

ALICE (Artificial Linguistic Internet Computer Entity) is a 3-time winner of the Loebner Prize chatbot. It's fair to classify ALICE as an extension of ELIZA regarding its stimulus-response architecture [Wal09]. The main improvement over its predecessor is that ALICE uses AIML, an XML dialect meant to store knowledge of artificial intelligent applications.

This system uses a Supervised Learning model as it requires a "botmaster" that is responsible for the creation of new AIML content.

3.1.2 Corpus-Based Systems

Instead of using a set of built-in rules, corpus-based chatbots derive a pool of abstract rules extracted from several human-human or human-machine conversation texts. The "recent" explosion of users public dialogs on platforms such as Twitter or Facebook presented new opportunities to apply data-driven techniques in these types of systems [RCD11]. Due to the type of data that is used to train them, this type of chatbots often have some similarity with question-answering systems [JM17] applied for non-specific purposes, as they tend to ignore the conversation's context and goals. Corpus-based chatbots can be divided into two categories: Information-Retrieval (IR) and Learning-Based (like transduction models) systems.

3.1.2.1 Information-Retrieval Systems

Information retrieval systems like Google use natural language processing (NLP) techniques based on keyword matching to represent, index, and retrieve texts [PML$^+$99]. IR chatbots can, in a certain way, be compared with these systems as they respond to a user's input with the most "appropriate" text available in the bot corpora. IR agents differ from one another's by the way they choose the "appropriate" answer to an user input and by the corpus of natural text that each bot possesses.

3.1.2.2 Learning-Based systems

Ritter *et al.* (2011) proposed the first machine learning phrase-based model that used a linguistic corpus to transduce a user's stimuli into an answer. In his paper, it is described how they used statistical machine translation as an approach for response generation [RCD11]. They start by identifying the two main challenges in adapting SMT for response generation:

1. In a translation system, stimulus-response pairs are usually equivalent. The same doesn't happen in the response generation case, where a vast pool of responses may exist for a given stimulus;
2. In stimulus-response pairs, some unaligned words may exist. Consequently, large portions of the stimulus may not be referenced in the response and vice versa.

Although being able to tackle these challenges with some success, they concluded that the system developed was still far from human-level performance. Nevertheless, the results obtained were still better when compared to an IR System.

It became evident that despite being a generative response method, SMT-based systems where unfit for dialog generation because a user stimulus is not semantically equivalent to the response as in translation so inputs can have several completely different responses [LGB+15] as shown in table 3.1.

Input	I bought a new car
User 1	What kind of car is it?
User 2	Do you have a driving license?
User 3	Nice! Now you can take me to work.

Table 3.1: An example of different possible responses to the same user utterance

Sequence to sequence (seq2seq) models employing Recurrent Neural Networks (RNNs) have a notable success in different areas such as speech recognition, text summarization, and machine translation.

A simple seq2seq model consists of two RNNs: an encoder that processes a given input and a decoder that produces the output. Image 3.1 represents the underlying architecture of this model.

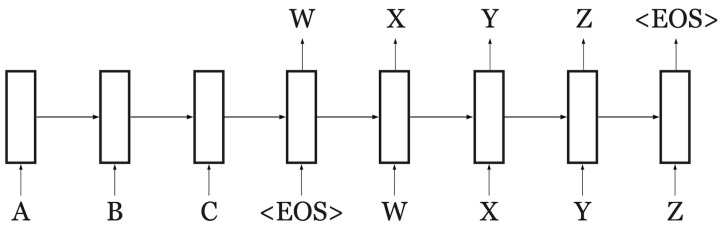


Figure 3.1: An architecture representation of a simple seq2seq model

In the basic model represented above in fig. 3.1 every square represents a cell of the Recurrent Neural Network, and the input sequence needs to be mapped to a fixed-sized vector as it is the only thing that's passed to the decoder [SVL14]. This may cause some difficulties for Neural Networks to cope with large inputs, especially if the sentences are more extensive than those available in the training corpus [CvMBB14].

To tackle this issue, a model that does not encode a whole input sentence into a single fixed-length vector was proposed [BCB14]. It consists of a multi-layer seq2seq network with LSTM cells and an attention mechanism in the decoder. By adding this mechanism to the decoder, it is possible to derive a context vector that will capture relevant information helping in the prediction of the output word [LPM15] relieving the encoder from the task of encoding all information into a vector with a fixed length.

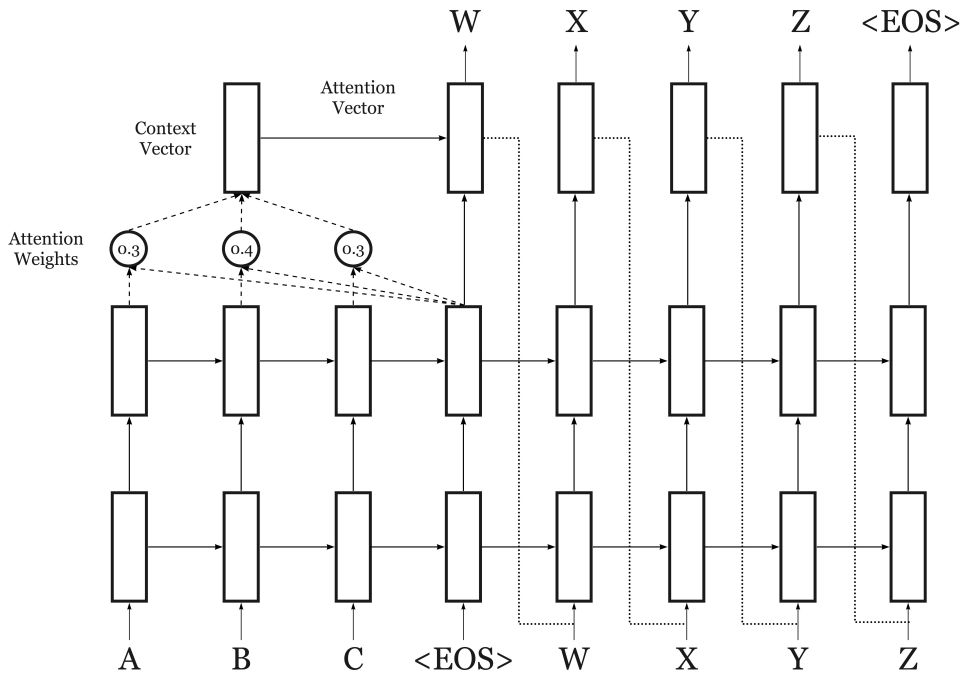


Figure 3.2: A representation of a multi-layered neural network with an attention mechanism as proposed by Luong *et al.* [LPM15]

All the seq2seq models presented above were applied for MT problems. In 2015 a neural network-based response generator for Short-Text Conversation approach was proposed. It consisted of a probabilistic model that employed a neural encoder-decoder with an attention mechanism [SLL15]. With his work, Li proved that these models could be adapted for Response Generation systems, when trained with a reasonable amount of data, despite the difficulty of the task.

Despite these models' success to generate human-like responses to a given user input, they are still not that useful for commercial purposes since the amount of training data needed to train a system for a specific area may not be easy to get or may be too expensive.

For task-oriented dialog generation, a different kind of model is required. The next section will introduce the frame-based architecture (or GUS architecture) that is used in almost every modern task-based dialog system [JM17].

3.1.3 Frame-Based Systems

Most modern-day commercial dialog systems are centered on a domain-specific topic. These types of systems define one or more frames as a form of knowledge representation which are used

to map information at different levels within the system. That can go from the sequence of a dialog to the attributes of a data structure. [BKKW77]

This architecture was proposed in 1977 by Bobrow *et al.* in his GUS system for travel planning, a tool that attempts to model an airline booking system in which the communication with the user is made through natural dialog.

Frames (like the one represented in table 3.2) are used to control the conversation flow. First, an instance dialog frame is created and then it goes through the slots of that instance and requests information to the user in order to fill them. By doing this, GUS can have an initiative in the dialog.

Slots	**Fillers**	**Servants**	**Demons**
Dialog			
(1) CLIENT	**Person**	Create	Link to TRAVELLER
(2) NOW	**Date**	GetDate	
(3) TOPIC	**Trip Specification**	Create	
TripSpecification			
(4) HOMEPORT	**City**	Default - Palo Alto	
(5) FOREIGNPORT	**City**		Link to OUTWARDLEG, AWAYSTAY, INWARDLEG
(6) OUTWARDLEG	**TripLeg**	Create	
(7) AWAYSTAY	**PlaceStay**		
(8) INWARDLEG	**TripLeg**	Create	
TripLeg			
(9) FROMPLACE	**City**	FindFrom Homeport	
(10) TOPLACE	**City**	AskClient	
(11) TRAVELDATE	**Date**	AskClient	
(12) DEPARTURESPEC	**TimeRange**	AskClient	Propose-Flight-By-Departure
(13) ARRIVALSPEC	**TimeRange**		Propose-Flight-By-Arrival, Link to DEPARTURESPEC
(14) PROPOSEDFLIGHTS	**(SetOfFlight)**		
(15) FLIGHTCHOSEN	**Flight**	AskClient	
(16) TRAVELLER	**Person**	AskClient	

Table 3.2: An outline of a keyframe structure from GUS system proposed by Bobrow *et al.*

Most conventional task-oriented dialog systems consist of four modules: [You06, ZE16]

1. Natural Language Understanding - That maps natural language user inputs into semantic frames;

2. Dialog State Tracker - Processes the information provided by the NLU module and keeps track of the dialog state;
3. Dialog Manager - Is responsible for mapping the current dialog state into a system action;
4. Natural Language Generator - Maps the system action selected by the Dialog Manager into a natural language response.

The key component of the understanding system is the NLU module, where its three main tasks are domain classification, *i.e.* determines which domain the user input belongs to, intent determination where the task or input goal is resolved, and finally, slot filling where the slots and entities of the sentence are extracted [MDY+15]. For the user utterance "Book me a flight to Cairo for tomorrow" a possible system representation could be something like what is represented in table 3.3.

Domain		AirTravel
Intent		BookFlight
Entities	DestCity	Cairo
	Time	Tomorrow

Table 3.3: A representation of the Domain, Intent and Entities extracted from the user utterance "Book me a flight to Cairo for tomorrow."

As stated before these types of systems are domain-specific, *i.e.*, one particular application usually fits in only one domain. In this case, domain classification would be redundant, nevertheless, although being most use-cases like this, multi-domain support is a standard in most systems. [JM17]

3.2 Machine Learning

"Machine learning research is part of research on artificial intelligence, seeking to provide knowledge to computers through data, observations and interacting with the world. That acquired knowledge allows computers to correctly generalize to new settings."

Dr. Yoshua Bengio, Université de Montréal

Several other definitions of Machine Learning exist, some more technical and others more focused on the business point of view. To find a definition of Machine Learning is not of relevance for this document. What we need to consider is that a Machine Learning task learns and retrieves valuable information from the data provided. Most of these learning tasks are usually classified as Supervised or Unsupervised.

In an Unsupervised learning task, we only have the observations, but no information about the outcome of those inputs is provided, so the goal is usually to describe how the data is organized.

Supervised Learning is usually done in the context of Classification, where we want to map our observations into a set of classes, or Regression, where our goal is to predict a continuous value based on the input provided. This type of problem id often called Supervised because there is a dependent variable guiding the learning process.

Due to the nature of the problem presented in this document this part of the chapter will focus mainly on providing information about Classification methods.

3.2.1 Regression Models

There are many regression methods that may be more or less adequate depending on one's available data and goals. One of the most popular and probably simplest methods is Multivariate Linear Regression. This method takes a set of continuous inputs to find a p-dimensional function that represents a "line" where the distance of the points to that line is minimized according to a specific loss function.

Let Y denote a quantitative dependent variable that is linearly related to a vector X of p independent variables $X=[X_1, X_2, ..., X_p]$ thought the regression coefficients $\beta=[\beta_1, \beta_2, ..., \beta_p]$. Let also ε represent a random error and β_0 the Y-Intercept also known as the model *bias*. A multiple linear regression model can be expressed with the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \varepsilon$$

Which is equivalent to:

$$Y = \beta_0 + \sum_{j=1}^{p} (\beta_j X_j) + \varepsilon$$

By including β_0 in the regression coefficients array and the value "1" in the vector of independent variables we can represent the previous function in the vector form:

$$Y = X^T \beta + \varepsilon$$

Where X^T denotes the transpose of X.

3.2.2 Classification Models

3.2.2.1 Logistic Regression

Performing a linear regression only makes sense if the relationship between the independent variables is linear and the expected output is a continuous value. If we are dealing with a binary classification problem, we could define a threshold and say that every point that falls beneath that value has a specific classification and those who do not will have the opposite classification.

However, when dealing with binary classification problems, this method was proven to be less than ideal, so other methods for the prediction of a dichotomous outcome are preferred.

Logistic regression arose from the need of classifying binary outcomes via a linear function while ensuring that the predicted result would be constrained to the interval [0,1]. First introduced by Cox [Cox58], the logit model can be represented in the form:

$$P(Y \mid X) = \frac{1}{1 + e^{-X^T \beta}}$$

This function varies from [-∞, ∞] in X while its result P is restricted to [0,1] as plotted in Fig.3.3. Although being a regression model, the step-like behavior of the logit function makes the logistic regression more adequate for binary classification problems.

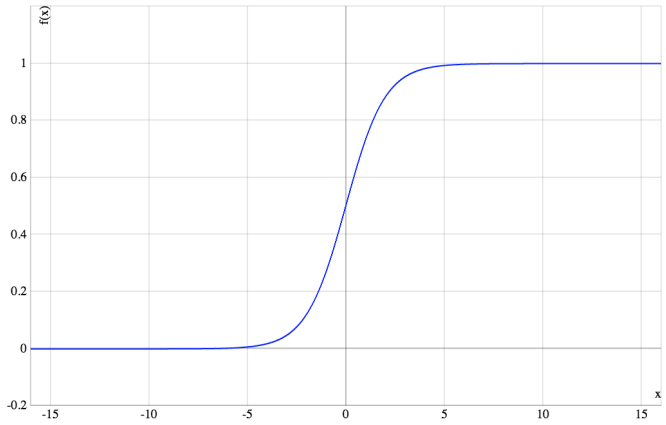


Figure 3.3: Sigmoid function

3.2.2.2 Naive Bayes

Naive Bayes is a classification algorithm based on Bayes theorem named after the man who first proposed it, Reverend Thomas Bayes. This theorem describes the probability of an event A based on *a priori* knowledge of an event B:

$$P(A \mid B) = \frac{P(B \mid A)\,P(A)}{P(B)}$$

In a classification problem, we want to calculate the probability of a specific dependent variable Y_j from k possible classes $[Y_1, \ldots, Y_j, \ldots, Y_k]$, given a set of independent variables $X=[X_1, X_2, \ldots, X_p]$:

$$P(Y_\mathrm{j} \mid X) = \frac{P(X \mid Y_\mathrm{j})\,P(Y_\mathrm{j})}{P(X)}$$

$$P(Y_\mathrm{j} \mid X_1, X_2, \ldots, X_\mathrm{p}) = \frac{P(X_1, X_2, \ldots, X_\mathrm{p} \mid Y_\mathrm{j})\,P(Y_\mathrm{j})}{P(X_1, X_2, \ldots, X_\mathrm{p})}$$

In Bayes theorem, the predictor prior probability is used for normalization, and as we usually cannot compute $P(X_1, X_2, ..., X_p)$ we can remove this term and say that:

$$P(Y_j \mid X) \propto P(X_1, X_2, ..., X_p \mid Y_j) P(Y_j)$$

$P(Y_j)$ is fairly easy to calculate as it is simply the proportion of the dataset that falls into the class Y_j, however calculating the term $P(X_1, X_2, ..., X_p \mid Y_j)$ can be computationally expensive. To minimize this computation cost we assume that given Y_j, all the independent variables in X are *conditionally independent* (which may not be necessarily true), hence the word *naive* in the name of this classifier:

$$P(X_1, X_2, ..., X_p \mid Y_j) = P(X_1 \mid Y_j) \times P(X_2 \mid Y_j) \times ... \times P(X_p \mid Y_j)$$

$$P(X_1, X_2, ..., X_p \mid Y_j) = \prod_{i=1}^{p} P(X_i \mid Y_j)$$

Making this assumption of *conditional independence* we can write the posterior probability of class Y_j given the predictor $X = [X_1, X_2, ..., X_p]$ as:

$$P(Y_j \mid X_1, X_2, ..., X_p) \propto P(Y_j) \prod_{i=1}^{p} P(X_i \mid Y_j)$$

Assuming that no relation exists between the independent variables is a massive simplification for the classification process and, although not always being an accurate assumption, it does not seem to affect the posterior probabilities calculation significantly, thus not having a great effect on the classification task results. [Ris01]

There are different Naive Bayes classifiers, and they differ mainly on the assumptions they make about the data distribution, affecting the way the class conditional densities $P(X_i \mid Y_j)$ are calculated. In Gaussian Naive Bayes for example, is assumed that the features are distributed according to a *Normal distribution*. In the context of text classification, the data usually doesn't follow this type of distribution. In this case, the *Multinomial Naive Bayes* which proved to be successful in several related cases [GWG94, LG94, YL02], is usually the preferred classifier.

3.2.2.3 Support Vector Machines

Even though they can be used for both classification and regression, Support Vector Machines (SVMs), first introduced by Vladimir Vapnik [VC95], are mainly used in classification problems. The primary goal of SVMs is to fit a hyperplane in an N-dimensional space (where N equals the number of features) that correctly classifies the data points while maximizing the distance between the points of both classes.

We are dealing with a constraint optimization problem. A strategy for finding the maximum and minimum of such problems is using the Lagrange Multipliers technique. This technique is explained with detail in [Ber14].

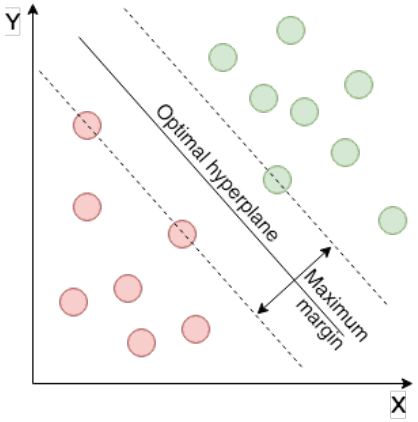


Figure 3.4: A representation of a separable problem in a 2D space. The optimal hyperplane represents the margin of largest separation between the two classes [VC95].

The case displayed in Img.3.4 represents a Linear Support Vector Machine (LSVM) as both classes are linearly separable. If we are dealing with non-linearly separable data, we can apply a function to transform our data into a higher-dimensional space. This operation, however, is computationally expensive, and, to reduce this cost we can use a kernel function to transform a non-linear space into a linear space. For a more complete overview of Support Vector Machines, we refer the reader to [Vap95]

3.2.2.4 Neural networks

Artificial neural networks are an information processing paradigm proposed by the neurophysiologist Warren McCulloch and the logician Walter Pit, and it was inspired by the way the brain works in order to process information [MP43].

The expensive computational power required to emulate these types of networks was for years a considerable setback for research in this area. Recently the neural network field is having a resurgence of interest and consequently an increase in research and funding in such a way that companies like Google[5] and Amazon[6] are developing their libraries.

These networks are especially attractive because they allow us to have good results when creating a predictive system with input data that is hard to understand. They also don't require extensive knowledge of the system we are modeling as we can treat it as a "black box" where only the dependent and independent variables are known.

3.2.2.4.1 Artificial Neurons

The fundamental computational unit in an Artificial Neural Network is the neuron, also called by unit or node. The node computes the weighted sum of its inputs (x) with the respective weights

[5] `https://www.tensorflow.org/`
[6] `https://github.com/amzn/amazon-dsstne`

(w) plus some bias (b). The unit output (o) is the result of that calculation when passed by a specific activation function (f) as represented in Fig.3.5.

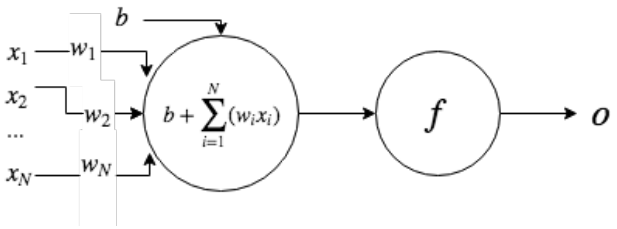


Figure 3.5: Schematic representation of an artificial neuron.

Generally, the output of the node with N inputs can be written as:

$$o = f(b + \sum_{i=1}^{N}(w_i x_i))$$

Activation functions

An artificial neuron calculates the weighted sum of its inputs with the respective weights, adds a bias value and then it decides whether it should be fired or not. The problem with this sum is that its result ranges in $[-\infty, \infty]$, so we need a function to map those results into a more "classifiable" range. Several different activation functions exist:

Heaviside function: is an often used *step function* which returns 0 for negative numbers, 1 for positive numbers and 0.5 when the input is 0. (Fig. 3.6a)

$$f(x) = \begin{cases} 0 & x < 0 \\ \frac{1}{2} & x = 0 \\ 1 & x > 0 \end{cases}$$

Sigmoid function: is a "step-like" function that will tend to 1 when the input is a large positive number or 0 when it's a large negative number, and it is strongly sensitive when the input is close to zero. (Fig. 3.6b)

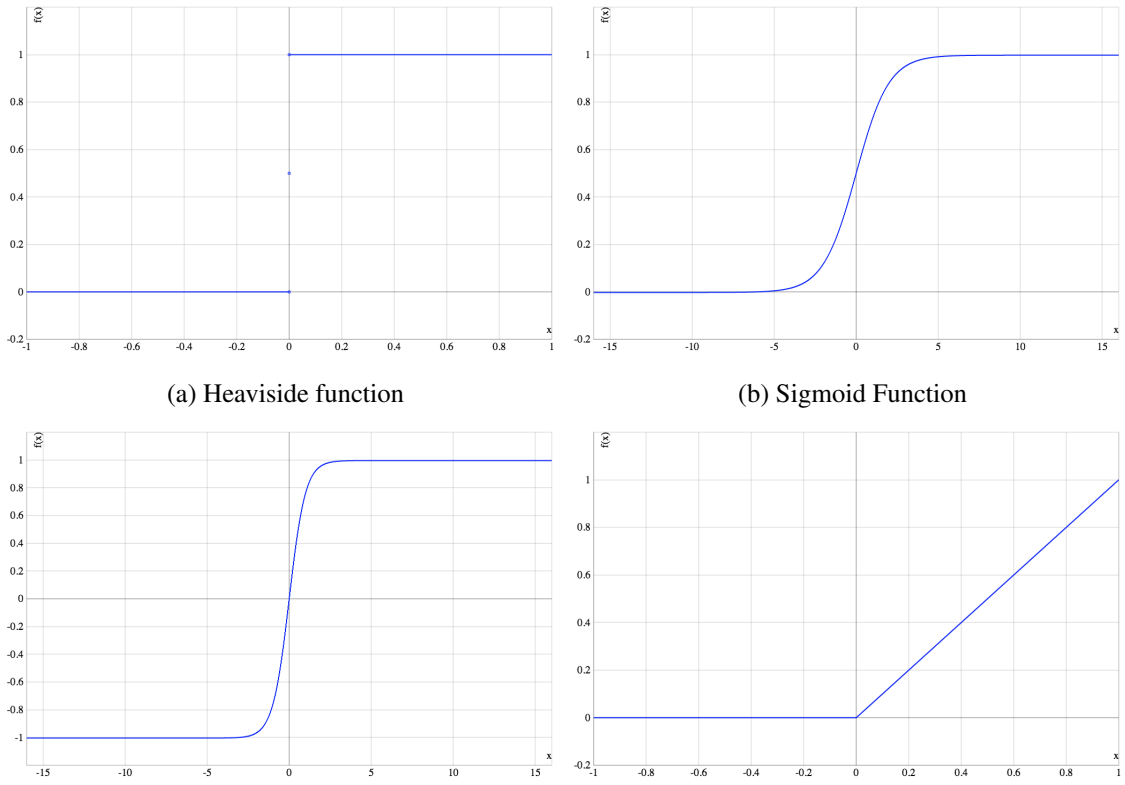
$$f(x) = \frac{1}{1 + e^{-x}}$$

Tanh function: is similar to the sigmoid function but returns a value between $[-1, 1]$. (Fig. 3.6c)

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

ReLU function: sets all negative inputs to zero, returning a value in the range $[0, \infty]$. (Fig. 3.6d)

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$



(a) Heaviside function

(b) Sigmoid Function

(c) Tanh Function

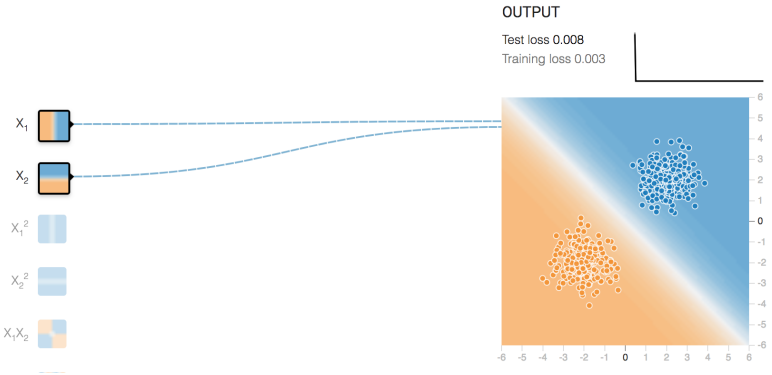
(d) ReLU Function

Figure 3.6: A graphic representation of a set of possible Activation Functions.

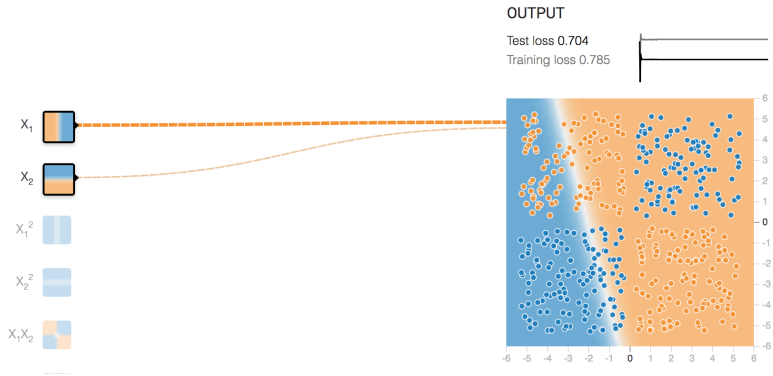
Perceptrons representational power and limitations

Let us consider a *Binary Classification* problem where we want to obtain one of two classes: C_1 if $o \leq 0.5$ and C_2 if $o > 0.5$, being o the perceptron output as defined in Fig.3.5.

A single perceptron will represent a hyperplane on the space defined by the input variables $X = [x_1, x_2, ..., x_N]$ and consequently the N-dimensional space will be divided into two regions that correspond to each side of the plane. To get a zero-error classification the input data must be *linearly separable*, meaning that we should be able to draw an N-dimensional plane that correctly separates all C_1 points from C_2 ones [MP88]. (See Fig. 3.7)



(a) Linearly separable input data



(b) Linearly inseparable input data

Figure 3.7: A graphic representation of two sets of 2D points and the obtained classification regions using a ReLU activation function.

However, it is possible to separate linearly inseparable data by adding more levels to the neural network. These levels are called *hidden layers*. Note that by adding just one hidden layer to the example provided in Fig.3.7b it is possible to better fit the classification regions to the same data points (Fig.3.8). These images were possible thanks to Tensorflow Playground [7]

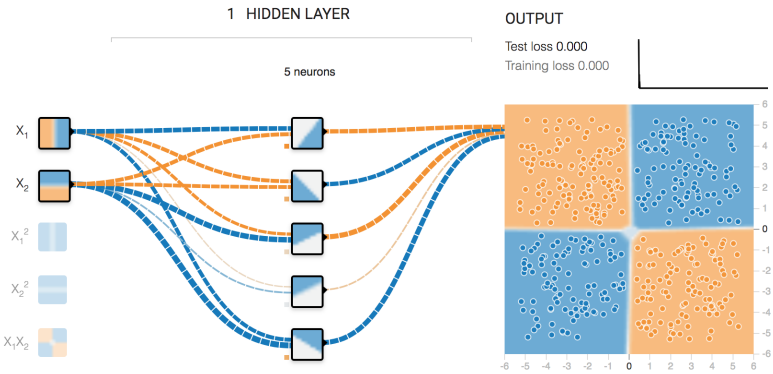


Figure 3.8: Neural network with one hidden layer applied to linearly inseparable data.

[7] http://playground.tensorflow.org/

3.2.2.4.2 Feedforward Neural Network

A Feedforward Neural Network is the simplest type of ANN. It consists in a set of nodes arranged in layers where the information flows in one way only (no loops exist) from the input layer, through the hidden ones (if any exist), to the output layer.

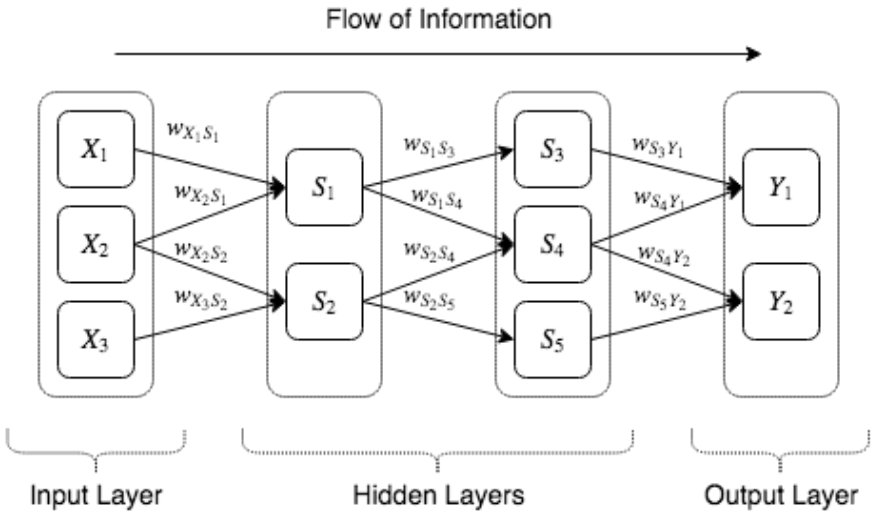


Figure 3.9: Representation of a Feedforward Neural Network

Now that we have a basic idea on how a neural network is structured, our goal is to find a combination of nodes with specific weights and bias that, given a set of inputs, can make an approximation of the desired output. In order to do that is necessary to define a cost (or error) function (E) that will compute the difference between the network output (o) and the expected result (y).

3.2.2.4.3 Cost Function

Several cost functions exist, and the correct choice of which to use depends on the specific problem. Let us consider the Logistic Loss Function (log-loss) which is mainly used with logistic regressions and extensions like neural networks. For multiclass classification problems with a total of M classes, the log-loss function can be defined as:

$$C(y,o) = -\frac{1}{N}\sum_{i=1}^{N}(\sum_{j=1}^{M}(y_{i,j}ln(o_{i,j})))$$

where:

$y_{i,j}$ = Binary indicator of whether or not class j is the correct classification for the instance i
$o_{i,j}$ = Predicted probability of instance i being classified as class j.

So, as we want to get a value of o similar to y we need to find a way to minimize this cost function.

When we first create a neural network all the weights an bias are usually random, so we need a training process to adjust those values. We need to iteratively adjust the weights and biases values by minimizing the cost function. For each iteration, the error (log-loss in this case) is given by:

$$E(y,o) = -\sum_{j=1}^{M}(y_j ln(o_j))$$

3.2.2.4.4 Gradient Descent

A simple and well-accepted method for updating the weights and bias is using the gradient descent:

$$w_{new} = w - \alpha\frac{\partial E}{\partial w} = w - \alpha\nabla E(w)$$

$$b_{new} = b - \alpha\frac{\partial E}{\partial b} = b - \alpha\nabla E(b)$$

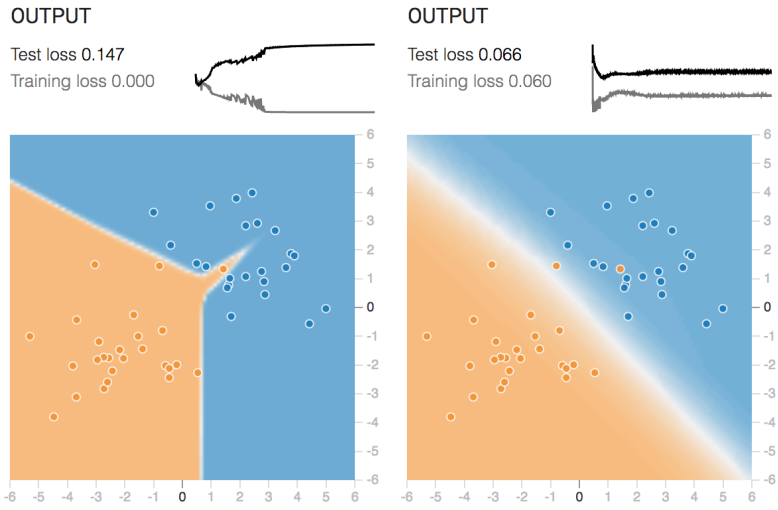
where:

w = The current weights.
b = The current bias.
w_{new} = The updated weights.
b_{new} = The updated bias.
α = Learning rate.
$\frac{\partial E}{\partial w}$ = Partial derivative of the cost function with respect to the weights.
$\frac{\partial E}{\partial b}$ = Partial derivative of the cost function with respect to the bias.

This type of descent gradient methods is often used in several machine learning problems as they are easy to implement and relatively fast for problems with several training instances. However, the most significant disadvantage of this method is that it requires much manual tuning of optimization parameters such as learning rates and convergence criteria. This is usually done by trial and error, something that makes this method unfeasible when the optimization procedure is computationally expensive.

3.2.2.4.5 L-BFGS

The Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) is a method which is more stable to train and easier to check for convergence [VNC+11], and it is used in minimization problems where the objective function and its gradient can be computed. L-BFGS is an optimization algorithm in the family of Quasi-Newton methods. Like the BFGS method, it is used to solve a minimization problem by computing approximations to the Hessian matrix of the objective function. However, instead of storing a fully dense $n \times n$ Hessian approximation matrix (where n represents the number of variables in the problem), L-BFGS method only stores some vectors



(a) Overfitted Neural Network

(b) Neural Network with L2 Regularization

Figure 3.10: A graphic representation of the results obtained from two neural networks with and without regularization on the same dataset. (Only showing train data points)

with length n that implicitly represent the approximations. For a more in-depth reading about Quasi-Newton methods with limited storage we refer the reader to [Noc80]

3.2.3 Regularization

The main advantage of Artificial Neural Networks is their ability to learn nonlinear decision boundaries by taking advantage of the connection between several simple linear classifiers. However, when these networks become too complex and when *outliers* exist in the training data (which they usually do), the model may also try to learn that nonlinearity introduced by the noise. This will present a good performance of the model during the training process but will lead to bad results when exposed to new unseen data - a phenomenon called *overfitting*. We can notice this in Fig.3.10a where the model fits perfectly with the training data but doesn't perform as well with the test dataset.

There are several methods to prevent overfitting. A good universal one that works well most of the times is to get more training data. However, as it is not always possible to do so, other more complex methods like regularization exist. Some of the most popular regularization methods are *L1* and *L2* regularization.

Even though we have been providing examples of Artificial Neural Networks, it is important to notice that these regularization methods can be applied to other algorithms such as Linear or Logistic Regressions. The main idea behind this technique is to add a regularization term to the cost functions in order to shrink or regularize the coefficient estimates towards zero. Let us consider the following example:

As seen before the following equation can express a multiple linear regression model:

$$Y = \beta_0 + \sum_{j=1}^{p} (\beta_{\mathrm{j}} X_{\mathrm{j}})$$

The fitting process requires a loss function that we want to minimize. In this case, let us consider the *Residual Sum of Squares* to measure the discrepancy between the data and the results provided by the model:

$$RSS = \sum_{i=1}^{n} (y_i - \beta_0 + \sum_{j=1}^{p} (\beta_{\mathrm{j}} X_{\mathrm{i,j}}))^2$$

The regularization term is then added to the loss function.

3.2.3.1 Ridge Regression

A Ridge regression performs L2 regularization by adding a penalty to the loss function equivalent to the square of the coefficients magnitude.

$$\sum_{i=1}^{n} (y_i - \beta_0 + \sum_{j=1}^{p} (\beta_{\mathrm{j}} X_{\mathrm{i,j}}))^2 + \lambda \sum_{j=1}^{p} (\beta_j^2) = RSS + \lambda \sum_{j=1}^{p} (\beta_j^2)$$

Where λ is the *tuning parameter* that will control how much the penalization will affect our model. Not that if $\lambda = 0$ we get the RSS, while if λ is a very high number our coefficients will tend to zero and the model will be *underfitted*.

By including this regularization term in the cost function, we can notice that all other things being equal, the algorithm will prefer to learn small weights. Large values for the coefficients will only be accepted if they considerably improve the first part of the loss function.

We can now use an optimization algorithm like the Gradient Descent to find the minimum of this function and with it update the regression coefficients.

3.2.3.2 Lasso Regression

A Lasso regression performs L1 regularization by adding a penalty to the loss function equivalent to the absolute value of coefficients magnitude.

$$\sum_{i=1}^{n} (y_i - \beta_0 + \sum_{j=1}^{p} (\beta_{\mathrm{j}} X_{\mathrm{i,j}}))^2 + \lambda \sum_{j=1}^{p} (|\beta_j|) = RSS + \lambda \sum_{j=1}^{p} (|\beta_j|)$$

The *tuning parameter* has the same effect on the penalization control. However, it is important to notice that the L1-term used in Lasso regression cannot be differentiated at $x = 0$. This will require us to use a different optimization algorithm such as *Coordinate descent* (for more information refer to [TL08]) which will set some of the coefficients to zero, making the lasso method also perform *variable selection* [Tib96].

3.2.3.3 Elastic Net

When dealing with a big feature space, we may end up with many coefficients. This is something we do not want as it can complicate the model's training, execution, and interpretability. By applying a Lasso regularization, we can push as much of the coefficients down to zero as we possibly can, allowing us to reduce the number of variables that are actually going into the model. The problem with this method is that sometimes we may get rid of variables we actually don't want to get rid of. In this case, we might want to apply a Ridge Regression which will push the coefficients down to zero but not all the way down.

Elastic nets in some way try to get the "best of both worlds" by combining both L1 and L2 penalties. Zou *et al* [ZH05], proposed this method that mathematically is defined as the convex combination of both L1 and L2 regularization terms:

$$\text{Elastic Net Penalty} = \alpha \times \text{Ridge Penalty} + (1-\alpha) \times \text{Lasso Penalty}$$

Where $\alpha \in [0,1]$. Note that if the α parameter is set to zero, we obtain the Lasso model while if it is set to one the model is reduced to a Ridge regression.

Chapter 4

Data Understanding

In this chapter, we will try to understand what value we can extract from the available data. In section 4.1 we'll make an overview of the dataset provided by the company. After that, in section 4.2 we will make a more in-depth analysis to figure out if there is a pattern that stands out. After realizing that the available dataset isn't adequate for the current project, in section 4.3 we present our solution to collect more data. Finally, in section 4.4 we point out some considerations that we had to make in order to continue the development process.

4.1 Data Overview

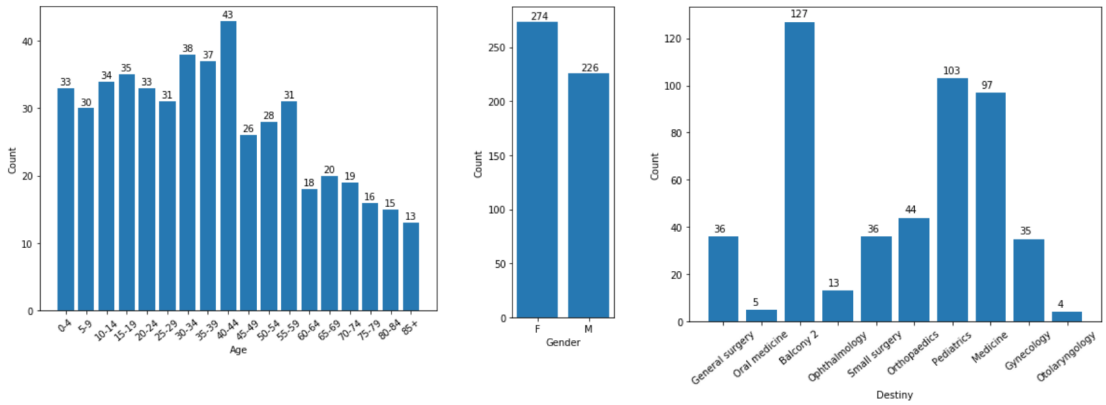
The triage process is used to determine the priority of patients treatment and the medical specialty they should be referred to. This operation is made by a clinical that then inputs that triage information into the hospital software.

As stated before this project was proposed by Glintt, a multinational that offers technological solutions to several hospitals and clinics. The company was to able gather data from 500 triages performed on October 23, 2018, on a single hospital, which contained the following information:

1. Patient's date of birth;
2. Patient's gender;
3. Symptom description;
4. Medical specialty the patient was referred to.

Through the date of birth, we can obtain the patient's age on the triage's day. A first overview shows us that patient's age and gender may be well represented in this dataset; however, some problems regarding the specialty distribution may arise.

It is important to notice that everything related with to the datasets presented in this document was translated from Portuguese for textual coherence reasons.



(a) Age distribution

(b) Gender distribution

(c) Medical Specialty distribution

Figure 4.1: A basic overview of the dataset including age, gender and medical specialty counts.

4.2 Understanding the data

Our task is to automatically classify to which medical specialty a patient should be sent to depending on its symptoms. However, in order to obtain a more accurate classification, it is essential to discover if other factors like patient's age or gender have any effect on this decision process.

Taking a look at the data obtained from the hospital software, we can notice that after the triage process the patients were redirected to one of the ten following locations:

1. General surgery;
2. Oral medicine;
3. Balcony 2;
4. Ophthalmology;
5. Small surgery;
6. Orthopedics;
7. Pediatrics;
8. Medicine;
9. Gynecology;
10. Otolaryngology.

A few things are to notice: A victim of an accident, for example, after the triage process can be sent directly to surgery; however, a patient cannot schedule such intervention without first being seen by a clinical. This means that our system should not be able to redirect the users to small or general surgeries.

We can observe from Fig.4.1c that the destiny with the most affluence was *Balcony 2*. This doesn't refer to any specific medical specialty, and it is probably related to the hospital internal operation. This is something we should take into account when analyzing this dataset.

Besides that, we can also see from Fig.4.1c that the classes *Oral medicine*, *Otolaryngology*, and *Ophthalmology* may not be representative due to their low frequency on the dataset.

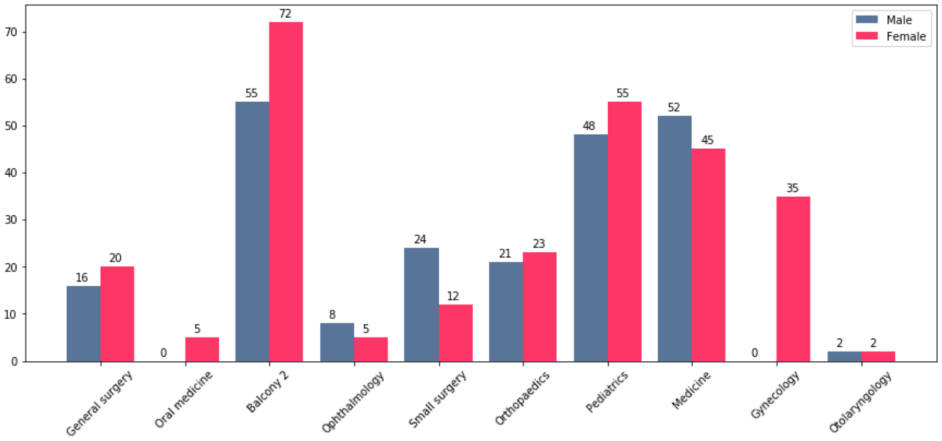
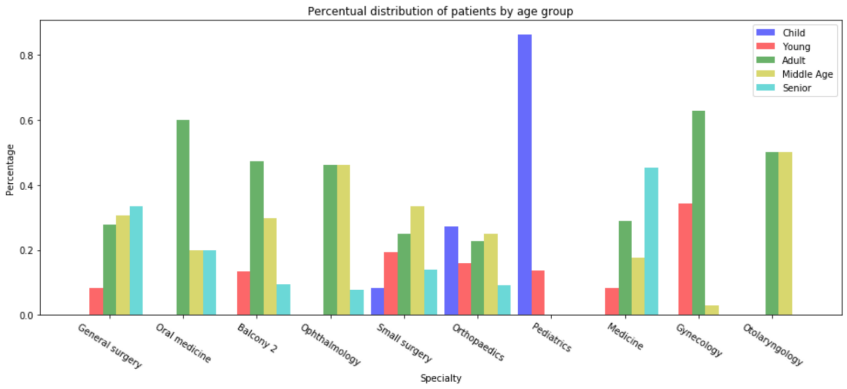
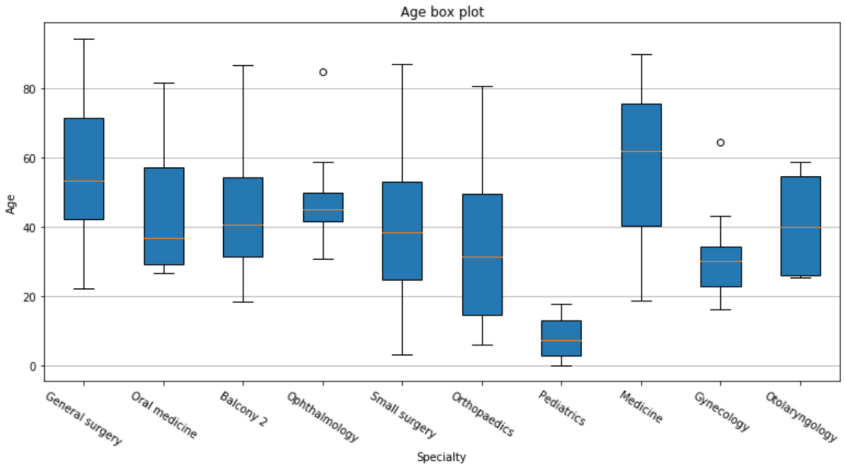


Figure 4.2: Patients distribution through destinies according to gender.

Regarding gender distribution through the classes we can notice from Fig.4.2 that, as it should be expected, only women are sent to *Gynecology*. Besides , we can see that twice more male patients are sent to *Small surgery* than female ones and that *Oral medicine* is only attended by female patients. However, as stated before our system should not be able to redirect the users to small or general surgeries, so the analysis of both these classes isn't of relevance for this problem. Furthermore, although being *Oral medicine* only frequented by women, as said before, we cannot draw any conclusion due to its low frequency on the dataset.



(a) Percentual distribution of patients (by age group), throughout the different specialties.



(b) Age by specialty

Figure 4.3: A basic overview of the dataset regarding age distribution throughout the different specialties.

In order to analyze the patient's age distribution on the dataset, we first categorized it into five different groups according to the provisional guidelines on standard international age classifications [Uni82] for health services (low detail). The categorization is as follows:

1. *Child* - Under 15 years old;
2. *Young* - Between 15 and 24 years old;
3. *Adult* - Between 25 and 44 years old;
4. *Middle Age* - Between 45 and 64 years old;
5. *Senior* - 65 years old or more.

Note: Children with less than one year are categorized as *Infant*. We are considering this category as belonging to *Child*.

By looking at Fig.4.3a and excluding the non-representative classes stated before we can notice that two medical specialties pop-up: *Pediatrics* and *Gynecology*. As expected *Pediatrics* is

attended only by patients categorized as *Minor* and *Young*, more specifically by patients with less than 18 years old as seen in Fig.4.3b. In addition, *Gynecology* is mainly but not exclusively attended by women categorized as *Young* or *Adult*.

Besides that, no other relevant pattern was found.

It is expected that when it comes to predicting a medical specialty according to a patient's overall condition, that the symptoms described by him play an essential role in the decision process. This dataset, however, contains only a brief description of the condition as reported by the clinical. This information is, most of the times, not representative of how someone would describe their symptoms as it usually just contains one or two terms often containing technical language or acronyms.

The non-representative nature of the symptoms description data-points invalidates the use of this dataset to build our classification model. However, it gave us insights, ones more evident than others, on some rules or details our system should be aware of:

1. Only female patients should be sent to gynecology;
2. Women attending gynecology are mainly between 22 and 35 years old (1^{st} and 3^{rd} quartile respectively) as seen in Fig.4.3b
3. Pediatrics is only attended by patients with less than 18 years old.

This dataset was useful to show those insights, however, as stated before, nurse clinicians created it, so symptoms are described with technical terms and acronyms. As this type of data is not representative of how a patient without medical training would explain their condition, a different approach was required.

We are now facing a data-mining problem were no data is available. As so, we need to collect data from a different source.

4.3 Collecting data

To tackle this issue a questionnaire (in the format of a chatbot) was created where the users were challenged to impersonate a patient with a particular gender and age and with a specific symptom. They were then asked how they would respond in one sentence to the question "What are you feeling?".

BOT: You are a 68-year-old woman with: Breathing troubles.
BOT: How would you answer in one sentence to the question: "What are you feeling?"
USER: I find it hard to breathe.

The questionnaire was sent to approximately 23000 students of Porto University, and with it, we were able to collect 1302 entries consisting of symptom, symptom description, gender, age and medical specialty for a total of 42 different symptoms and eight medical specialties.

As stated before, except for Gynecology and Pediatrics, no significant correlation between patient's age and the destination was found. For that reason, and because we will be considering other specialties in this new dataset, we will not be using it as a possible feature. It was only included in the questionnaire dialogues so that the users could play the role of a patient with a certain age.

4.4 Some Considerations

The triage for clinical appointment's booking isn't always a straightforward process. Sometimes the symptoms description provided by the patients is not enough, and some inferred questions can come up that may be necessary clarify in order to do a correct assessment of the situation.

In this document, regarding the appointment's booking, we consider that the patient's gender and description of a single specific symptom is information enough to schedule an appointment.

It is also assumed that patients, with 18 years or more, with the same symptoms and gender, should be referred to the same medical specialty regardless of their age.

Chapter 5

Data Preparation

In this chapter, we will describe all the transformations performed on the dataset from their cleaning in section 5.1 to all the required pre-processing we had to execute as referred in section 5.2. After applying all necessary transformations to the data, we were able to extract a set of features as explained in section 5.3.

5.1 Data Cleaning

The first operation to do after collecting and understanding the dataset is to detect and correct (or remove) all the inaccurate, corrupt or missing data points.

The data collection process was made in a controlled environment. A user was asked to answer a question as explained in Section 4.3, which answer would then be saved along with the patients' gender and the medical specialty he should be sent to. This controlled scenario made the users answers our only concern regarding data cleaning as no cases of missing data-points existed.

By taking a first glance at the dataset, we noticed that some users intentionally gave answers that were not related to the question. The process of removing these outliers was manual, and a total of 22 entries were removed.

As a *Stemming* operation is to be performed as explained in subsection 5.2.1, it is also essential to perform contextual grammar and spell checking on the users' answers as well as to filter all non-alphabetical characters. This process of making orthography and syntax correction was made using *Bing Spell Check (API v7)*[1]

By making a simple count of word occurrences in the dataset is possible to notice that the more frequent ones are connectors or other common words. Ideally, we remove all but the significant vocables for the classification process, however as significance is difficult to predict, it is more practical to remove all undoubtedly non-significant tokens, with the risk of accepting words of questionable relevance [P. 60]. These insignificant words are often called *Stop Words*. To remove we created a *Stop List* containing specific tokens that were then removed from the dataset. Besides that, all the words that appeared less than three times were also removed (this elimination occurred

[1] `https://azure.microsoft.com/en-us/services/cognitive-services/spell-check/`

only after the text normalization process described in subsection 5.2.1). The removal of these vocables may significantly increase the classification performance.

5.2 Data Pre-Processing

5.2.1 Text Normalization

Text normalization is one of the most relevant steps of the data transformation process, and it consists in applying linguistic reductions through *Stemming*, *Lemmatization* or other standardization technique.

According to the Oxford Dictionary, *inflection* is defined as "a change in the form of a word (typically the ending) to express a grammatical function or attribute such as tense, mood, person, number, case, and gender.".

The *Stemming* process aims to reduce the words inflection to their root forms mapping a group of words to the same stem (root) even if the obtained stem is not a valid word by itself. *Lemmatization*, on the other hand, reduces the inflected words properly, mapping them to a *lemma*, a root word that belongs to the language.

Even though *lemmatization* can yield better results than *stemming*, it can be a quite intensive and slow process. For that reason, the text normalization process was made using the Stemming algorithm for the Portuguese language proposed by Viviane Orengo [OH01].

5.3 Feature Extraction

After cleaning the dataset, we need to extract features from it in a format that can be used by machine learning algorithms.

5.3.1 Term Ranking

Now that we normalized the text information it is essential to determine the importance of each token for a document (utterance) in the dataset. TF-IDF (Term Frequency - Inverse Document Frequency) is the most popular weighting scheme (70%) for text-based recommender systems where the scheme was specified [BGLB16].

Two terms compose the TF-IDF weight:

1. *TF* - Measures the frequency of a term in a specific document;
2. *IDF* - Measures how important a term is by weighing down the frequent terms and increasing the value of the rare ones.

The TF-IDF weight for a term t in a document can be calculated according to the following equations:

$$\mathrm{TF}(t) = \frac{\text{Number of times term } t \text{ appears in the document}}{\text{Total number of terms in the document}}$$

$$\text{IDF}(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}}\right)$$

$$\text{TF-IDF}(t) = \text{TF}(t) \times \text{IDF}(t)$$

5.3.2 Feature Hashing

For the moment all we have is a collection of text streams. To convert it into a set of features represented as integers that can later be used in a machine learning algorithm, we applied a dimensionality reduction process via *Feature Hashing*. This way, instead of building a table with all features found during the training process, a hash function is applied to each one of the m terms (T) in order to determine their index in the matrix directly, as represented in Fig.5.1.

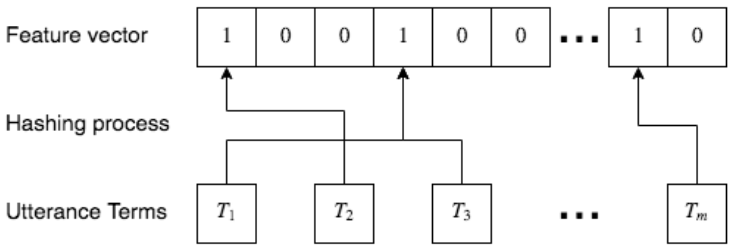


Figure 5.1: Graphic representation of a dimensionality reduction process via Feature Hashing.

In Fig.5.1, each index of the feature vector represents the occurrence of a word (1 if the term appeared, 0 otherwise). Although being this a possible approach, is essential to represent the significance of each word in the documents somehow. For that reason, instead of using the value 1 to represent the occurrence of a term we will use the *TF-IDF* weight to represent its pertinence.

In this case, we are using a *Bag of Words* representation, where the features represent the frequency of the term's occurrence. It is important to notice that this type of representation does not preserve nor describes the order of the words in the text.

By using this hashing method, it is possible to increase the model speed significantly and reduce its memory usage. However, as we can see in Fig.5.1, it comes at the expense of a bigger inspectability as the hash function may cause collisions between different words, as illustrated by terms T_2 and T_3.

To reduce the previously referred inspectability, it is necessary to use a feature vector with a number of elements considerably more significant than the number of different terms used in the training process in order to avoid collisions during the hash process.

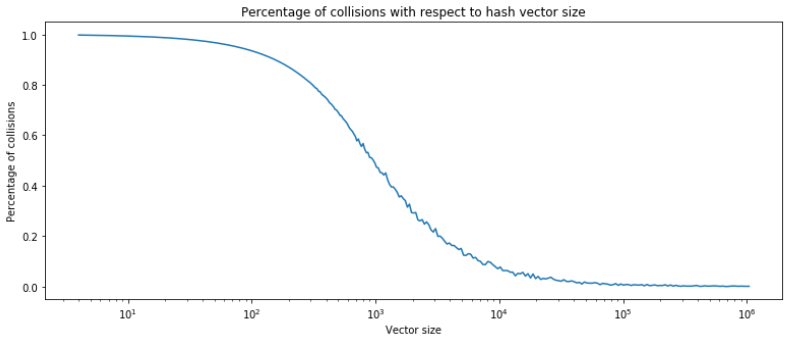


Figure 5.2: Percentage of collisions with respect to the hash vector size using MurmurHash 3 algorithm on a dataset with exactly 1538 different terms.

We will be using Austin Appleby's MurmurHash 3 algorithm to calculate the hash codes of a total of 1538 different terms. Fig.5.2 shows the percentage of collisions in function of that vector size. We can see that the number of collisions starts to tend to 0 when the vector has more than 10^5 elements. However, since a modulus is used to transform the hash function into an index of the vector, it is recommended to use a vector with a power of size two. For that reason, we will be using an array of size 2^{17}. It is important to notice that for a bigger dataset an increase of the vector size may be required.

5.3.3 Sparsity

The text features (TF-IDF weights) described in the previous subsection could be stored in a dense $m \times n$ matrix, where m represents the total number of sentences in the dataset (documents) and n the size of the previously computed hashing vector.

Assuming that no collisions exist, a sentence with 10 terms would produce an array where only 10 out of 2^{17} values are different from zero ($< 0.01\%$).

To store such sparse data in memory and also to speed up algebraic operations performed on these features a sparse matrix representation is used.

Chapter 6

Modeling

In this chapter, we describe all the models that were implemented and some details regarding their validation. In section 6.1 we present the general structure of the implemented models along with the list of classification algorithms that were used. Section 6.2 describes two validation methods and presents the one that will be used to validate our models and section 6.3 explains how the data will be split for training and testing. Section 6.4 unravels the metrics that will be used for the previously referred validations and finally section 6.5 displays those results.

Section 6.6 makes just a brief reference to the environment where the best model obtained was integrated.

6.1 Model Structure

Now that we know how to extract our feature vector it's time to start building our model. As this will be integrated with a conversational agent, we needed a framework powerful enough to process and classify user's inputs in real time. For that reason, we will be using *Apache Spark*[1] to develop our model. This whole process requires us to run a sequence of algorithms to process and learn from data, as so we will be constructing a *Pipeline* with a structure as represented in Fig.6.1.

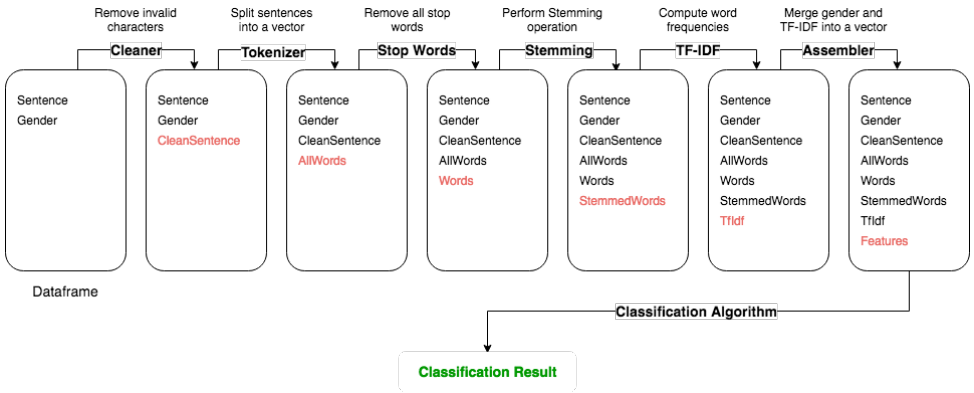


Figure 6.1: Representation of the implemented pipeline.

[1] https://spark.apache.org/

Pipelines allow us to perform operations on dataframes sequentially. These operations can be either *Transformers* or *Estimators*. The pipeline from Fig.6.1 consists of seven stages where the first six are transformers, and the last one (Classification Algorithm) is an Estimator.

6.1.1 Classification Algorithm

In order to obtain a classification result, we still have to choose which classification algorithm we will use. To make this selection, we will perform tests with four different algorithms:

The first one will be *Multinomial Naive Bayes*, a simple probabilistic algorithm that as stated in subsubsection 3.2.2.2 proved to be successful in the context of text classification.

Secondly we wil be testing the *Logistic Regression* algorithm with a *One vs Rest* approach (OvR LR). This logistic regression uses L-BFGS as a solver and elastic net for coefficient penalization.

We will then test a *Linear Support Vector Machine* also with a *One vs Rest* approach (OvR LSVM). This algorithm optimizes the *Hinge Loss* using the Orthan-Wise Limited Memory Quasi-Newton method (OWL-QN), an extension of L-BFGS for solving L1 regularized sparse learning problems [GY15].

Finally we will test a *Multilayer Perceptron Classifier* (MPC), *i.e.*, a fully connected feed-forward neural network with one hidden layer. It uses backpropagation for learning the model, the logistic loss as cost function and L-BFGS as an optimization routine. The nodes in the output layer use the soft-max function.

6.2 Validation methods

One of the most important parts of any data mining development process is the model test step. This procedure helps us figure to out how well our model fits the data available and also gives us an idea of how it will behave when exposed to new unseen data.

6.2.1 Holdout

Using the same dataset for training and testing our model is a common methodological mistake as it will produce overoptimistic and overfitted models. A possible solution could be to apply the *Holdout* method, where we would split our dataset into training and test sets making sure that none of the test features was used in the training process.

This method, however, when doing hyperparameter tuning, can tend to overfit on the test set since these parameters can be changed until the model performs optimally. To solve this, we can split the data into three sets: The ones previously referred and a third one called the *validation set*. This way, we can perform the training with the train set, see how the model fits that data while tuning its hyperparameters with the validation set and then perform an unbiased evaluation of the final model with the test dataset.

It is usually recommended to make a random split of the data in order to capture all data properties in each subset. However, when performing multiple tests, it is possible to obtain results with a high variance as each observation will be dependent of the test dataset, especially if we have a relatively small amount of data (which is our case). That is the reason why we will be using *K-fold cross-validation* method to evaluate our models.

6.2.2 K-fold cross-validation

Cross-validation consists in splitting our dataset into k parts (folds). One of those sets will be used as the test dataset while the remaining will be used for training. This process will be repeated k times until all folds have been used for testing as represented in Fig.6.2. The performance measure will be the average of the obtained results in each iteration of the validator. This has been proven to be a more reliable method for performance analysis than holdout [Koh01].

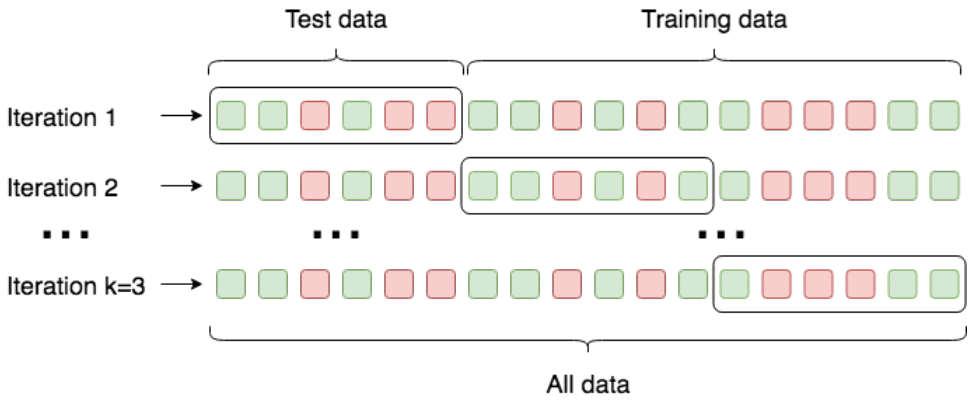


Figure 6.2: Representation of the k-fold cross-validation method with k=3

However, it is important to notice that this approach can be computationally expensive as we will need to do k evaluations to analyze the model performance. This detail can make this method unfeasible when dealing with large amounts of data.

6.3 Data splitting

When performing any type of evaluation some problems may arise from randomly splitting the data, especially if an imbalance in the distribution of the target classes exists. As we can see from Fig.6.3, our data isn't perfectly balanced.

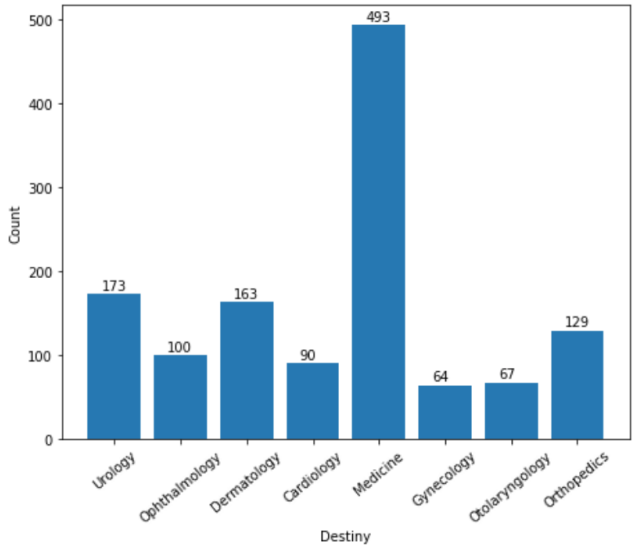


Figure 6.3: Distribution of samples through each class on the new dataset.

In this cases, it is recommended to use *stratified sampling* to ensure that the train and test subsets have the same percentage of samples of each target class as the original dataset as represented in Fig.6.4.

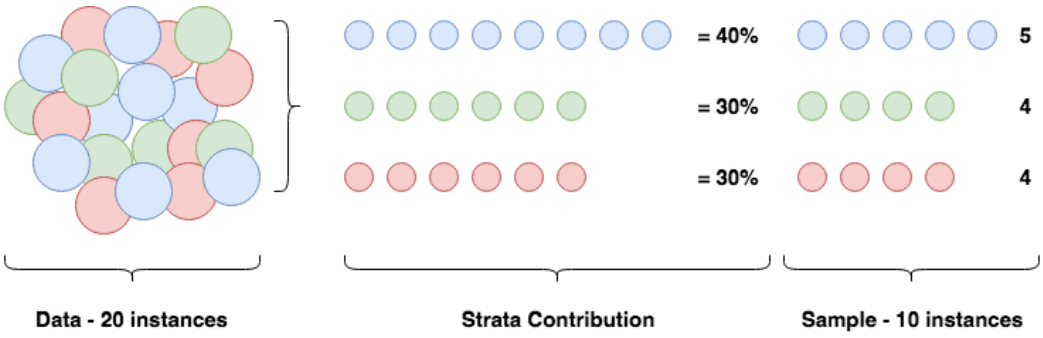


Figure 6.4: Representation of the stratified sampling technique

6.4 Validation metrics

In order to see how well our model is performing, we need to specify the metrics used for that assessment. To better comprehend the metrics used it is important first to understand four different terms that will be used to compute our evaluation measurements. These terms are:

1. *True Positive* - Positive class is correctly classified as positive;
2. *True Negative* - Negative class is correctly classified as negative;
3. *False Positive* - Negative class is wrongly classified as positive;
4. *False Negative* - Positive class is wrongly classified as negative.

		Predicted Class		
		yes	**no**	**total**
Actual Class	**yes**	*True Positive (tp)*	*False Negative (fn)*	*p*
	no	*False Positive (fp)*	*True Negative (tn)*	*n*
	total	*p'*	*n'*	*p+n*

Table 6.1: Confusion matrix representation for a binary classification problem

The metrics used to evaluate our model and the way to calculate them are presented below in Table 6.2

Metric	**Formula**
Accuracy (Recognition Rate)	$\frac{tp+tn}{p+n}$
Precision	$\frac{tp}{tp+fp}$
Recall (Sensitivity / True Positive Rate)	$\frac{tp}{p}$
F-Score (F_β)	$\frac{(1+\beta^2)\times precision\times recall}{\beta^2\times precision+recall}$

Table 6.2: List of metrics used for evaluation.

6.4.1 Accuracy

One of the most popular and simple metrics for performance analysis is *accuracy*, which can be computed by dividing the number of correct predictions by the size of the test set. This metric, however, has been proven to be less than ideal for skewed data [WP06] as it will tend to bias the majority class data. Let us consider a fraud detection example where we are trying to classify an occurrence as *fraud* (yes) or *not-fraud* (no). The results of this classifier are presented in Table 6.3.

		Predicted Class		
		yes	**no**	**total**
Actual Class	**yes**	*tp=2*	*fn=10*	*p=12*
	no	*fp=0*	*tn=988*	*n=988*
	Total	*p'=2*	*n'=998*	*p+n=1000*

Table 6.3: Confusion matrix for the fraud classification example.

We can see that this model gives us an accuracy of 99.0%, which seems to an excellent result. However, we can see that it was only able to detect 2 out of 12 fraud cases.

6.4.2 Precision and Recall

Other notable performance metrics are Precision, which as we can see in table 6.2 tells us what proportion of positive identifications are correct, and recall, which tells us the proportion of positives that were correctly classified.

From the previous example we can obtain a Precision value of 100% meaning that all the instances classified as fraud are actually frauds; however, we get a Recall value of only 16.67% that reflects the significant amount of false negatives.

6.4.3 F-Score

To evaluate the performance of the classification model, it is essential to examine both precision and recall. These values, however, are usually in tension, meaning that increasing the precision value will often result in a decrease of the recall and vice versa.

F-Score is a metric that combines both of these measures. The most used is the *Harmonic Mean between Precision and Recall* where the β value equals 1. A harmonic mean is usually preferred to arithmetic as its results fall closer to the lower of the two values.

For the example previously presented we obtain a harmonic mean (F_1-Score) of approximately 28.6% and an arithmetic mean value of 58.3% that can be achieved with high Precision at the expense of low Recall.

To evaluate our model we will be using the F_1-Score; however, if we wanted to give more importance to the recall value we could increase β and use for example the F_2-Score or, reduce that value if a high precision was more important, using for example, the $F_{0.5}$-Score.

6.5 Validation Results

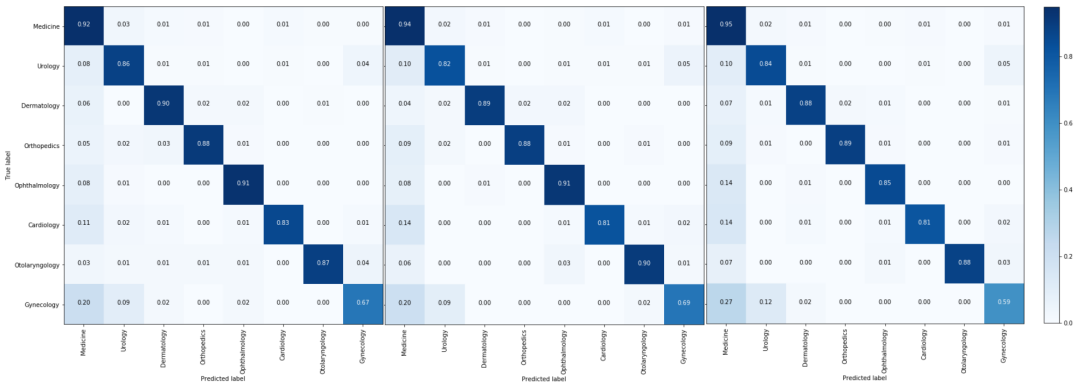
From this point on the validation results obtained from the several models will be compared and analyzed. Before extracting these results, it was necessary to tune every parameter of each one of the algorithms used. It is important to notice that due to hardware and time constraints some algorithms have been subjected to more exhaustive tuning than others. That being said, the

models used for analysis are the ones that, for each classification algorithm, obtained the best F_1 score during the hyper-parameter tuning process.

		Weighted Average			Macro Average		
	Accuracy	Precision	Recall	F_1Score	Precision	Recall	F_1Score
Naive Bayes	80.193	81.510	80.193	80.103	79.041	75.664	76.207
OvR LR	88.160	88.836	88.160	88.107	88.885	83.678	85.835
MPC	88.439	88.924	88.439	88.375	88.516	85.364	86.522
OvR SVM	88.523	89.016	88.523	88.491	88.598	85.469	86.637

Table 6.4: Evaluation results for the different tested classification algorithms (values in percentage).

Table 6.4 displays the five folds average of the results obtained for each model in the validation process. From these results, we can notice that for this application all algorithms outperformed the Naive Bayes approach, obtaining very similar results. We can also notice that in all the tests the Weighted Average results were higher than the Macro. This means that the incorrect classifications are tending to the majority class as we can see in Fig.6.5.



(a) Normalized confusion matrix resultant from Multilayer Perceptron Classifier.

(b) Normalized confusion matrix resultant from One vs. Rest SVM.

(c) Normalized confusion matrix resultant from One vs. Rest LR.

Figure 6.5: Normalized confusion matrix resultant from Multilayer Perceptron Classifier, One vs. Rest SVM and One vs. Rest LR.

Figure 6.6 displays the average of ten executions and train instances for each one of the algorithms. As the model is expected to respond in real time when considering the results displayed in Table 6.4, the Logistic Regression with a One vs. Rest approach may be the more adequate algorithm.

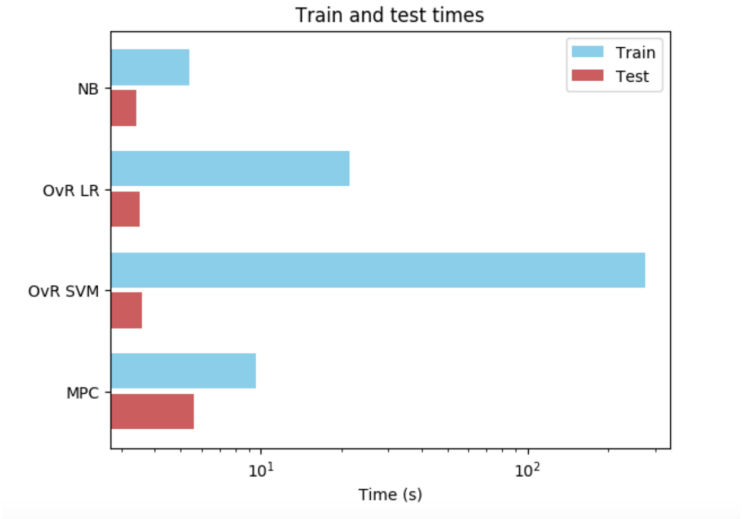


Figure 6.6: Train and test times for all four algorithms.

It is important to notice that that the infrastructure supporting the developed model was not taken into account in this document. The project was developed in a single machine with 16GB of RAM and an Intel Core i7-8550U CPU (1.8GHz-2.0GHz). Better time results can be achieved by setting up a proper infrastructure to support this type of applications.

6.6 Integration

As stated several times before, the best model found in the previous chapter is to be integrated into a specific domain chatbot.

To develop this conversational agent we used *Microsoft's Bot Framework (v4)* to build it and *LUIS* service for intent recognition. This framework speeds up development by providing a specific environment for conversational agents development, test, and deployment.

The bot conversation had to be guided towards specific goals: discover the patient's age, gender, and symptoms. This data would then be passed to the selected model who would answer with the medical specialty the patient should go in order to make the appointment. The following (translated) dialogue is an example of a possible conversation with the developed bot:

1 : BOT: Hello! Do you want to schedule an appointment?
2 : USER: Yes
3 : BOT: Ok, can you tell me your name?
4 : USER: My name is Mario (Gender inferred)
5 : BOT: Nice to meet you Mario!
6 : BOT: The appointment you're booking is for you or for someone else?
7 : USER: It's for my daughter. (Gender inferred)
8 : BOT: Ok, is your daughter of age?
9 : USER: Yes she is. (Age inferred)

10: BOT: Can you tell me her symptoms?
11: HUMAN: She's having huge headaches.
12: BOT: Ok then, let's book an appointment in: Medicine

Unfortunately, no data about this type of dialogues was available, so the bot training process had to be done manually. For that reason, this topic was not approached with more detail.

Chapter 7

Evaluation

To make a complete evaluation of this data-mining project the models and corresponding results obtained and presented in Section 6.5 are extremely important, however, it is far more relevant to see if those results and if the developed solution satisfy the business requirements. In section 7.1 we assess if the developed solution satisfies the business requirements. In section 7.2 we make a review of everything we did in this project and, to conclude this document, in section 7.3 we point out the potential next steps for future work.

7.1 Results Evaluation

Our primary investigation goal regarding this data-mining project was to figure out if the frequency of certain words in the symptom description, provided by a patient, could be of relevance for the discovery of the medical specialty he or she should be redirected to. The conducted study shows, based on the premise that a single symptom is sufficient for the discovery process, that a relationship between these two variables exists.

While this assumption may hold for some real cases, *e.g.*, a woman having a pregnancy-related complication should be redirected to Obstetrics, it is not that linear most of the times. Several symptoms are not specific enough or don't provide the information required to make this type of decisions, consequently more data may be required in order to provide a better and more accurate service to the patients.

A dull headache may be resultant of several completely different conditions; thus, a person presenting this symptom is a potential patient for a range of distinct specialties. For these more ambiguous situations, when no more information is available, it is always possible to redirect the patient to a more overarching specialty like Medicine, where later, after being attended by a doctor, he or she can be counseled to visit a specific specialist in case of need.

Metrics like the ones presented and used to evaluate our model in Chapter 6 don't properly reflect the cost of misclassifications, in the sense that wrongly redirecting a patient to Medicine may be way less problematic than erroneously sending him to any one of the other specialties. Having this in consideration, in terms of final application objectives, our results may be more optimistic

than they already appear to be since, as we can see from Fig. 6.5 most of the misclassifications are falling into Medicine, the most representative class.

Unfortunately, as we were limited on time and as no data was available, it was not possible to test nor collect user's feedback regarding the functionality of the developed conversational agent.

We believe, given the obtained results and the previous considerations, that the developed model and its integration in a conversational system may be valuable to reduce some cases in the triage process as it can confidently redirect patients that present a specific single symptom. However, it is important to point out that for even considering to fully automate this triage process a much more extensive research effort in this area is required. In addition to that, and not least important, a more significant focus especially from the part of the companies on collecting (relevant) data is necessary.

7.2 Process Review

At this point, we obtained a valid model that satisfies the initially established requirements. We will now summarize the data-mining process and try to point out some details that may have been overlooked, or that could be somehow improved.

Our main obstacle, the quality of the available data, presented itself at the very beginning of this project execution. This referred quality is closely tied with the value we can extract from it. To understand the full extent of it, a more intensive analysis and a broader knowledge of its domain would be required. For this particular case, we are labeling it as low-quality data in a sense that it does not meet the requirements for this specific project.

At this point, there wasn't much we could do. We were dealing with a problem where the data available wasn't suitable for its resolution. After discussing and testing some other different approaches, we concluded that the best solution would be to try to collect data in a format as similar as possible to the one the application would receive.

To achieve this goal we developed a conversational system to aid us, however, several things could have been improved here. Approximately 23000 users with ages mostly ranging from 18 to 25 years old were asked to interpret a patient with a particular gender, age, and a specific symptom. From this, we obtained 1302 responses where we noticed that some users dedicated some effort in performing that interpretation. This could have contributed to add some bias especially towards the preconceived idea young people have on the way that others, from a different age group, speak.

A solution to this issue would be to keep the data collection channel opened to a more diverse audience for a more extended period. This process would allow us not only to tackle the previously referred issue but also to collect more data to train and validate our models, but unfortunately, without the ability to stop or slow down time, some compromises had to be made. We still believe however that the quality and amount of the data obtained is good for the project in hands while it is true that it could be improved.

After having all available data collected the cleaning process began. The first step of this stage was to remove all entries that were out of the scope of what was requested. Unfortunately, this

process had to be done manually, and possibly a repetition of this procedure would still be valuable to exclude some outliers.

There is not much more to say about data cleaning as it was a very straightforward process, except that it would probably be better to make a more in-depth analysis of the frequency of the words in the corpus to check if something should be added to the stop-words list.

When the cleaning stage was done we had to normalize the text from our dataset, and for that, we used the *stemmer* proposed by Viviane Orengo [OH01] as it was, to the full extent of our knowledge, the only proposed method for Portuguese text normalization available. Even though it falls out of the scope of this project, we believe that the investigation and development of a *lemmatization* process for this specific language could benefit this particular research and others to come.

There were several ways of approaching this problem and; consequently, several different features could have been extracted. The proposed method involves describing each sentence as a bag of words representation, where the features are the frequency of occurrence of each term represented in a sparse vector of size 2^{17}, plus 0.0 or 1.0 if the patient is male or female, respectively. The main problem with this representation is that it does not preserve nor describes the order of the words in the text. A simple way to illustrate some relation between terms would be to use an *n-gram* representation ($n > 1$). This would be an option if we had a more significant amount of data samples or documents with a more substantial amount of terms. However, as we had a relatively small dataset with short sentences, we ended up excluding this option.

Regarding the classification algorithms, several others could have been tried, however when considering the validation results obtained, we believe that, although being worth trying, experimenting with other algorithms would be less valuable than properly tuning the ones presented in this document as, due to time and hardware constraints, a better adjustment of the algorithms' parameters could have been made.

Even though all requirements were successfully satisfied, as stated before, several parts of the process can still be improved, and different other approaches can be studied. Overall we consider the process was adequate and that good results were obtained especially when considering the data and time available for this project.

7.3 Next Steps

As a result of this project, we obtained a fully working conversational agent capable of requesting data and recommending a medical specialty in accordance with it. Even though we have successfully produced a prototype that proves that it is possible to automate this medical appointment scheduling process it is still not sturdy enough to be launched to a production environment.

We hope that this study and the prototype developed will open some doors to the company in this area of research. If it is decided to continue this project and move on to deployment, we present a list of the potential further actions ordered by priority:

1. Make a more intensive study regarding the impact and costs of this project - We showed that it is possible to automate this process with a high level of confidence. Now, before making a more significant investment it is necessary to analyze if it is actually beneficial, *i.e.*, compare the costs of the current triage process with the development and maintenance costs of this new method.
2. Gather more data - As stated several times before, one of our main concerns was about the datasets used. It is necessary to make a study about the data available (other than the provided) to see what value we can extract from it. If none is available, it is necessary to know which type of data could be valuable for this specific application and then find a way to collect it.
3. Study other possible approaches - One of the most relevant setbacks of the studied and implemented approach is its inability to know when more information is needed to make a more accurate classification. These other possible approaches will, of course, depend on the data available.

Appendix A

Parameter Tuning

This appendix presents the parameter tuning results obtained using grid search with cross-validation for the algorithms presented in chapter 6.

The values displayed in each table represent the 5-fold Average of the Weighted F_1-Score for the corresponding set of parameters.

A.1 OvR SVM Tuning

- λ - Regularization Parameter
- n - Maximum number of iterations taken for the solver to converge.

		λ					
		0	0.0001	0.001	0.01	0.1	1
n	1	*79.393*	*79.393*	*79.393*	*79.393*	*79.393*	*61.302*
	2	*81.675*	*81.783*	*81.783*	*81.704*	*81.609*	*84.168*
	5	*87.645*	*87.590*	*87.590*	*87.126*	*87.123*	*87.590*
	8	*88.157*	*88.314*	*88.169*	*88.252*	*88.181*	*85.343*
	12	*87.870*	*88.039*	*87.804*	*87.884*	***88.491***	*85.209*
	20	*87.955*	*88.034*	*87.950*	*88.137*	*88.135*	*84.702*

Table A.1: Parameter tuning results for OvR SVM.

A.2 OvR LR Tuning

- λ - Regularization Parameter
- α - Constant that multiplies the penalty terms (Elastic Net parameter)

		λ					
		0	0.0001	0.001	0.01	0.1	1
α	0.0	*85.470*	*86.333*	*86.580*	***88.021***	*85.658*	*43.328*
	0.2		*86.760*	*87.271*	*87.501*	*80.055*	*21.613*
	0.4		*86.687*	*87.212*	*87.145*	*75.321*	*21.613*
	0.6		*86.668*	*87.114*	*86.918*	*70.298*	*21.613*
	0.8		*86.619*	*87.428*	*86.521*	*57.761*	*21.613*
	1.0		*86.596*	*87.012*	*86.218*	*55.423*	*21.613*

Table A.2: Parameter tuning results for OvR LR.

A.3 MPC Tuning

- l - Number of neurons in the hidden layer.
- n - Maximum number of iterations taken for the solver to converge.

		l			
		10	16	25	50
n	1	*26.147*	*21.613*	*21.613*	*21.613*
	2	*21.613*	*21.613*	*21.613*	*21.613*
	5	*72.870*	*68.143*	*46.586*	*21.765*
	8	*87.826*	*85.707*	*83.640*	*60.006*
	12	*87.520*	*87.881*	*88.241*	***88.375***
	20	*85.659*	*86.777*	*86.231*	*87.832*
	30	*84.870*	*85.165*	*84.876*	*85.897*
	40	*84.176*	*84.738*	*84.193*	*85.296*
	50	*84.420*	*84.258*	*84.373*	*85.668*

Table A.3: Parameter tuning results for MPC.

A.4 Train and Test times

	Train	Test
Naive Bayes	5.4	3.4
OvR LR	21.5	3.5
OvR SVM	274.5	3.6
MPC	9.6	5.6

Table A.4: Train and test times for all four algorithms (in seconds).

References

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. pages 1–15, 2014.

[Ber14] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[BGLB16] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Research-paper recommender systems : a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, 2016.

[BKKW77] G Bobrow, Ronald M Kaplan, Martin Kay, and Terry Winograd. GUS , A Frame-Driven Dialog System. pages 155–173, 1977.

[CCK+00] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. *Crisp-dm Step-by-step data mining guide*. SPSS, 2000.

[Cox58] D. R. Cox. The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242, 1958.

[CvMBB14] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. 2014.

[GA11] Crina Grosan and Ajith Abraham. *Intelligent Systems- A Modern Approach*. 2011.

[GWG94] Louise Guthrie, Elbert Walker, and Joe Guthrie. Document Classification by Machine: Theory and Practice. *Proceedings of the 15th conference on Computational linguistics*, 2:1059–1063, 1994.

[GY15] Pinghua Gong and Jieping Ye. A modified orthant-wise limited memory quasi-newton method with convergence analysis. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 276–284. JMLR.org, 2015.

[JM17] Daniel Jurafsky and James H Martin. *Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3 edition, 2017.

[Koh01] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. 14, 2001.

[LG94] David D Lewis and William A Gale. A Sequential Algorithm for Training Text Classiers. *Proceeding SIGIR '94 Proceedings of the 17th annual international ACM*

SIGIR conference on Research and development in information retrieval, 1:3–12, 1994.

[LGB+15] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A Diversity-Promoting Objective Function for Neural Conversation Models. 2015.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. 2015.

[MDY+15] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-tur, and Xiaodong He. Using Recurrent Neural Networks for Slot Filling in Spoken Language Understanding. *IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 23(3):530–539, 2015.

[MP43] Warren S. McCulloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP88] Marvin L Minsky and Seymour A Papert. *Perceptrons, An Introduction to Computational Geometry*. 1988.

[Noc80] Jorge Nocedal. Updating Quasi-Newton Matrices With Limited Storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[OH01] Viviane Moreira Orengo and Christian Huyck. A Stemming Algorithm for the Portuguese Language. 2001.

[P. 60] H P. Luhn. Keyword-in-Context Index for Technical Literature (KWIC Index). *American Documentation*, 11:288–295, 1960.

[PML+99] Woojin Paik, Mary E Mckenna, Ming Li, Related U S Application Data, Primary Examiner, and Jack M Choules. US5963940, 1999.

[RC17] Filip Radlinski and Nick Craswell. A Theoretical Framework for Conversational Search. *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval - CHIIR '17*, pages 117–126, 2017.

[RCD11] Alan Ritter, Colin Cherry, and William B. Dolan. Data-driven response generation in social media. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'11)*, pages 583–593, 2011.

[Ris01] Irina Rish. An Empirical Study of the Naïve Bayes Classifier. *IJCAI 2001 Work Empir Methods Artif Intell*, 3, 2001.

[SLL15] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural Responding Machine for Short-Text Conversation. 2015.

[SMO+10] Guergana K. Savova, James J. Masanz, Philip V. Ogren, Jiaping Zheng, Sunghwan Sohn, Karin C. Kipper-Schuler, and Christopher G. Chute. Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): Architecture, component evaluation and applications. *Journal of the American Medical Informatics Association*, 17(5):507–513, 2010.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58:267–288, 1996.

[TL08] Tong Tong Wu and Kenneth Lange. Coordinate descent algorithms for Lasso penalized regression. *The Annals of Applied Statistics*, 2:224–244, 2008.

[Uni82] United Nations. Statistical Office. *Provisional Guidelines on Standard International Age Classifications*. UN, 1982.

[Vap95] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg, 1995.

[VC95] Vladimir Vapnik and Corinna Cortes. Support-vector networks. *Machine Learning*, 20(3):273–297, sep 1995.

[VNC+11] Quoc V. Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y. Ng. On Optimization Methods for Deep Learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, volume 2011, pages 265–272, 2011.

[Wal09] Richard S. Wallace. The Anatomy of A.L.I.C.E. In Robert Epstein and Gary Roberts, editors, *Parsing the Turing Test*, chapter 13, pages 181–210. Springer Science + Business Media, 1 edition, 2009.

[Wei66] Joseph Weizenbaum. ELIZA — A Computer Program For the Study of Natural Language Communication Between Man And Machine. *Communications of the ACM*, 9(1):36–45, 1966.

[WP06] Cheng G Weng and Josiah Poon. A New Evaluation Measure for Imbalanced Datasets. 2006.

[YL02] Kenji Yamanishi and Hang Li. Document Classification Using a Finite Mixture Model. 2002.

[You06] Steve Young. Using POMPDPs for Dialog Management. *2006 IEEE Spoken Language Technology Workshop*, 1:1–6, 2006.

[ZE16] Tiancheng Zhao and Maxine Eskenazi. Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning. 2016.

[ZH05] Hui Zou and Trevor Hastie. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2):301–320, 2005.