SCHOOL OF
MANAGEMENT AND
TECHNOLOGY
POLYTECHNIC OF
PORTO

M

MASTER
COMPUTER ENGINEERING

Optimising User Experience With Conversational Interfaces

António Miguel Gonçalves Costa

12/2019

António Miguel Gonçalves Costa. Optimising User Experience With Conversational Interfaces

12/2019

Optimising User Experience With
Conversational Interfaces
António Miguel Gonçalves Costa

M
MASTER
COMPUTER ENGINEERING

SCHOOL OF
MANAGEMENT AND
TECHNOLOGY
POLYTECHNIC OF
PORTO

SCHOOL OF
MANAGEMENT AND
TECHNOLOGY
POLYTECHNIC OF
PORTO

M

MASTER
COMPUTER ENGINEERING

Optimising User Experience With Conversational Interfaces

António Miguel Gonçalves Costa
Fábio André Souto da Silva, Marco Filipe Vieira Gomes

Abstract

User Experience is one of the main aspects that maintain a customer loyal to cloud based solutions or *SaaS* (Software as a Service). With the rise of the natural language processing techniques, the industry is looking at automated *chatbot* solutions to boost and expand their services. This thesis presents a practical case study of the implementation of a *chatbot* solution to complement a *CRM* (Customer Relationship Management) software called *FOXAIO*, and then quantify, following the most appropriate guides and solutions available, the User Experience *(UX)* optimisation.

In order to create a robust and scalable solution based on the constraints created by the company in the case, we reviewed the current deep learning techniques, tools and libraries available to help the development process. The most proven techniques in the field of *Natural Language Processing (NLP)* will be introduced.

To achieve the goals of this solution without "reinventing the wheel", we present possible architectures to use at the top of some open source and available tools on the market, with a special relief in the framework *RASA*. Also we discussed some of possible techniques to create the intent classifier, where we detail the better performance in the top of the *rasa tensorflow embedding* pipeline for this particular case.

The conversational system, also, required a channel to interact with the final user. To achieve that, we also implemented a basic *chat* interface created on the top of the socket protocol, which communicate with the conversation system. In any case, it would be possible to extend to the other channel's available on the market, like *messenger*, *slack*, *telegram*.

Finally, we detail with a few use cases, that's hypothetically possible to improve the user experience of an existing *software* system (*FOXAIO*) using a conversational interface on the top of that. Also, we achieved some highlights about the preference to use a conversational interface because of his simplicity, defended by a better score in the *SUS* scale, 70 against 58 to the traditional *UI*, and good indicatives by the *HEART* framework.

Keywords: *Conversational Interfaces*, *Deep Learning*, *Natural Language Processing*, *Chatbots*, *User Experience*, *CRM.*

Resumo

O *User Experience* é possivelmente um dos principais aspetos para fidelizar um cliente numa solução *cloud*, as chamadas soluções *SaaS (Software as a Service)*. O crescimento acentuado deste tipo de soluções aquece a rivalidade entre competidores e cada vez mais pretende-se oferecer as formas mais revolucionárias para premiar a qualidade de um serviço. Com o crescimento acentuado das técnicas na área do *NLP (Natural Language Processing)* a indústria começa a olhar para os *chatbots* como uma possível solução de automatizar, impulsionar e expandir as suas ofertas. A presente tese visa a apresentar uma implementação prática de um *chatbot* sobre um *software* com semelhanças de um *CRM (Customer Relationship Management)* existente intitulado por *FOXAIO*.

Com o objetivo de desenvolver uma solução robusta e escalável tendo em atenção as condições elaboradas pela empresa em questão, um longo e detalhado estudo foi elaborado sobre as mais diversas técnicas de *deep learning* usadas no ramo de *Processamento de Linguagem Natural (NLP)*. Atribuindo um particular ênfase às redes neurais recorrentes *(RNN)* e com a devida extensão *Long Short Term Memory (LSTM)* que juntas, formam e trabalham muito bem na resolução dos problemas de um sistema de inteligência artificial, como é o caso.

Para a sua implementação sobre um *software* já existente, foi necessário o desenvolvimento de uma pequena interface conversacional com o objetivo de mais tarde a complementar sobre a interface do utilizador do mesmo. Para esse efeito, foi implementado um canal sobre o sistema conversacional de comunicação em protocolo de *socket*, criando uma classe para o efeito que mais tarde seria útil para gerar *logs* de análise.

Durante a implementação do sistema conversacional foram feitas várias comparações sobre as variantes dos seus módulos desde o *Dialog Management (DM)* ao *Intent Classifier* onde várias arquiteturas foram expostas e comparadas com o intuito de corresponder à melhor solução possível para um *chatbot* de língua portuguesa em primeira instância, foi optado pela escolha de um *Dialog Management* híbrido face ao domínio e à existência de conversas contextuais contínuas onde, por exemplo, se torna bastante difícil de desenvolver sobre outros paradigmas. Quanto ao *Intent Classifier*, foi usada a técnica *rasa tensorflow embedding*, esta técnica (que treina palavras do princípio) usada obteve melhores resultados para o particular caso estudado na presente tese (CRM), do que por exemplo o uso um modelo de dados com palavras já treinadas.

Finalmente, conseguimos apresentar hipoteticamente, possíveis melhorias do *UX* no uso de uma interface conversacional sobre uma interface tradicional, usando as várias ferramentas de análise disponíveis, onde por exemplo com o auxílio da *framework HEART* (criada pelo *Google*), conseguimos obter indicativos bastante satisfatórios por 34 pessoas que fizeram os primeiros testes no *chatbot* desenvolvido.

Examinando o *feedback* desses mesmos utilizadores em ambiente de teste, conseguimos obter um resultado na escala de *SUS (System Usability Scale)* com um valor de 70, enquanto a interface tradicional arrecadou 58, notando então que as pessoas se sentiram mais capazes no uso do sistema conversacional.

Palavras-Chave: *CRM*, *Sistemas Conversacionais*, *Deep Learning*, *Processamento da Linguagem Natural*, *Redes Neuras Recorrentes*, *Experiência do Utilizador*.

Contents

List of Figures

List of Tables

Nomenclature

AI Artificial Intelligence

API Application Programming Interface

$CBOW$ Continuous Bag-of-Words

CRM Customer Relationship Management

DM Dialog Management

$GloVe$ Global Vectors for Word Representation

$LSTM$ Long Short-term Memory

ML Machine Learning

NLG Natural Language Generation

NLP Natural Language Processing

NLU Natural Language Understanding

RNN Recurrent Neural Network

$SaaS$ Software as a Service

SUS System Usability Scale

UI User Interface

UX User Experience

Chapter 1

Introduction

This chapter will briefly introduce the context of the present document and the idea in develop a conversational system at section 1.1. Then a brief review of the history of the conversational agents in section 1.2. Afterwards, in section 1.3 the motivation for this research theme will be explained, in section 1.4, we will highlight a brief description of the opportunity created by an implementation of a conversational interface, then in section 1.5, we will list the objectives that were agreed upon with the startup company. Finally in section 1.6, the chronological order of the structure to the present document will be summarized.

1.1 Work Context

The idea of investing and studying the concept of developing a conversational interface in the top of a regular system using the *UI's* to communicate, was born due the proved market growth of the cloud services, more precisely, in the *SaSS* section.

The world market for cloud services is projected to grow by 17.5 % to a market value of 214.3 billion from 182 billion of dollar's by 2019 according to *Gartner, Inc.* This growth is detailed in table 1.1 [10].

"Cloud services are definitely shaking up the industry", according to Sid Nag, vice president researcher at *Gartner* [10]. Also, according to the latest *Gartner* surveys, more than a third of organizations look at cloud service investments as a top-priority investment, impacting on market offers. By the end of 2019, more than 30% of technology providers are expected to move from the cloud-first to the cloud-only methodology, meaning that these 30% intend to offer services entirely *cloud*, causing the license-based software consumption to go down, while subscription-based *cloud* will continue to rise [10].

	2018	**2019**	**2020**	**2021**	**2022**
Cloud Business Process Services (BPaaS)	45.8	49.3	53.1	57.0	61.1
Cloud Application Infrastructure Services (PaaS)	15.6	19.0	23.0	27.5	31.8
Cloud Application Services (SaaS)	80.0	94.8	110.5	126.7	143.7
Cloud Management and Security Services	10.5	12.2	14.1	16.0	17.9
Cloud System Infrastructure Services (IaaS)	30.5	38.9	49.1	61.9	76.6
Total Market	**182.4**	**214.3**	**249.8**	**289.1**	**331.2**

Table 1.1: Cloud World Public Services Revenue Forecast (Billions of dollars) [10]

With this rapid growth of cloud services, more precisely on the *SaaS* applications, the way customers communicate and perform their tasks on that service becomes increasingly an important element in their loyalty to the service. This opens up an opportunity to enhance the user experience using different solutions, breaking away from traditional ones.

Exposing that, the conversational interfaces system's can increasingly be taken as a solution to apply over the *SaaS* applications in order to improve the overall user experience of the product.

1.2 A Brief Summary of the Conversational Systems

The idea of building a computer or program capable of communicating with a person came about in the 1950s, when *Alan Turin*, a mathematician, proposed the Turin test, which aims to determine if a machine is capable of exhibiting human-equivalent behaviour. This test is still used for the same purposes these days [26].

One of the first *chatbots* created was the *ELIZA* created by *Weizenbaum* in 1966, and it was also one of the first to pass the *Turing Test*. *ELIZA* simply answers the user's questions with other vague questions, sometimes rudely, thus trying to mislead the user into believing they would be interacting with another human [26].

Developments in automatic voice recognition arouses the interest of academics and the industry began to emerge in the 1980s. Conversation then jumped from the context of text to voice, which presumably became more natural and easier to interact with. *ATIS*, a phone scheduled to book flights, was founded by *DARPA* in 1990 [26].

The 90's are marked by the success of chat browsers, started by *IRC*. By the end of the decade, these were extended by AOL Instant Messaging, *Yahoo! Messenger* and *MSN Messenger*, in which *chats* it was already possible to add *bots* as contacts, and exchange simple messages [26].

Only recently the smart assistants began to gain public attention. According to *McTear*, different reasons have influenced this success, the advance in assistive technologies of artificial intelligence, such as speech and image recognition, the emergence of the term *semantic web*, the interest of the major technology *players* worldwide, launching wizards like *Siri* a product from the *Apple* (the first smart assistant with voice), *Microsoft Cortana*, *Google Now*, and *Amazon Alexa* are the main actors of the last five years in the convergence of conversational interfaces [26].

The last few years have been marked by the strong growth of the conversational systems, with the same characteristics of their predecessors, where in addition to gaining some capabilities, they are also developed in the cloud. In 2014 several online text messaging systems (*KIK*, *Telegram*, *Wechat*) opened their services to third parties by exposing their *APIs* to other programmers, allowing them to build their own *chatbots* with access to various high level services (online payments, messaging, authentication, etc.) as well as graphics (images, locations, etc), thus offering the possibility to build innovative services through conversational interfaces [26].

As shown in this summary, a lot of progress has been made since the early days of *NLP*. However, in the real user's perception, this does not imply that current solutions are perfect, as we present in section 1.3.

1.3 Motivation

In a recent study made by *Drift* in collaboration with *SurveyMonkey Audience*, *Salesforce* and *Myclever*, they asked over 1,000 people to try to identify the biggest frustrations in services *web*, *websites* and mobile applications in the last month [12].

The following conclusions can be drawn from the same study.

- 34 % of the users consider the *websites* / applications hard to navigate.
- 31 % of the users say it is difficult to get answers to simple questions.
- 28 % of the users consider the basic details about a business, like address, hours of operations, and phone number being hard to find.

Still on the same survey, the following question was asked, "If *chatbots* were available (and effectively) for the online services that you use, which of these benefits would you expect to enjoy?"

From this question we can highlight the following points.

- 64 % of the users would enjoy from a 24 hour service.
- 55 % highlights the opportunity to get instant answers.
- 55 % also highlights the possibility of getting answers to simple questions.

In short, users need's for web solutions are not being matched by the current traditional solutions. Also, the users see the advantages of using a *chatbot* over other solutions.

Although they can identify these advantages, another study by *Chatbots.org* [9], which is a community focused on researching, technology and innovation to drive the success of the developers, academics and users, points out that most users still do not find *chatbots* properly effective, as they should be.

Summarizing this survey, across all generations, 53 % of consumers find the *chatbots* "*not effective*" or just "*somewhere effective*". Further discriminating these results, it appears that younger generations are more optimistic in using conversational interfaces, according to the study, the "*millienials*" and "*generation Z*" generations find *chatbots* 56 % and 53 % respectively as "*effective*" or "*very effective*", while generations called "*boomers*" and "*silent generation*" only assign 38 % and 49 % respectively.

Given the following studies and surveys, it is clear that the population still does not see conversational interfaces as an efficient solution, so all research and advances in this subject will be essential for *chatbots* to be able to offer a conversational system as an effective and efficient service.

1.4 The opportunity

As we already presented in the section 1.1, the online cloud services are growing from year to year, and in order to provide the best possible experience in a paid service, we should consider all the alternatives on the way we want to give that service.

For a better context, we are introducing the *FOXAIO*[1] which is the software that we are going use to create the conversational interface in to retrieve the in formations of the intents of their users. This is a software with a virtue of a *CRM*, where we can manage the opportunities, third-parties, contacts, etc. But where is also possible to track other processes inherent on the track of a customer, such as route management, contracts, tasks/tickets. This same software is available to be accessed everywhere through any modern browser or via the smartphone app. There‘s an authentication system obviously, implemented on the top of *OAuth2* paradigm, which is an authorization protocol that allow third-party access to limited system resources [14].

Since the *FOXAIO* system can work as a *CRM*, there are some tasks and processes that we repeat every day which can be automated. The purpose of the implementation of a conversational interface in the top of the *FOXAIO* is to speed up some of these "boring" tasks on the system, which the user does daily or almost, like search for a specific customer to check the current balance value, search again for a specific customer to search a specific task, sending an email to a specific customer with the overdue bills, and the processes go on.

The idea is to create for the first version a *chatbot* widget on the current *UI* where the client of the software can use freely, optimising the execution of these same repetitive processes and renouncing from a psychical operator.

1.5 Goals

The high-level view of the main goal is to implement a scalable, easily maintainable *chatbot* which will work in Portuguese at the first instance, but with the possibility of having other languages in the future. It should interact with the current *software* system called *FOXAIO* which we already introduced, and in short, it should have the possibility of answer correctly on a few use cases at the start, and then it should be possible to grow in the features. Obviously it

[1]Startup company related to the project (www.foxaio.com)

should never disable the main features of the software itself, but it should complement the way the user do the tasks on it.

Ideally the conversational interface should satisfy the following list:

- Recognize utterances the most as possible to the specific domain intent correctly.
- Should provide a way to interact with the failed intents messages, in order to optimise these intents.
- Have an appropriate format data to train the machine learning solution.
- Be implemented in a programming language that can be deployed on multiple platforms with little effort.
- Be able to reply to the users in real-time.
- The interaction with the *API* should always be with the appropriate authentication of the user.
- Ask the user for more details if the intended message is not clear enough.
- Allow to customize the *chatbot* responses.
- Support the Portuguese language in the first instance, but with the possibility to grow and extend to the other languages.
- Have a small widget on the current software to interact with the conversational interface which is accessed only after the correctly authentication of the system, but also, with the possibility of employment in another message platform in the future, such system's like *Slack*, *Facebook Messenger*, etc.

On the other way, the implementation of the *conversational agent* must also follow some guidelines implemented by the company in question.

- Should be easy to understand the code, so in the future, other developers can easily pick up this project and extend with other features or just maintain.

- Should be implemented with active *tool-kits* and *frameworks* which should be well documented and not obsolete in the present year, they must be *open source* if possible and allow to be deployed on any server.

- Should have, the less errors as possible, even if it should start with only one or two features, and should also be possible to have a reasonable fallback response if the *chatbot* is not trained yet to a specific user's intent.

- The responses should be fully controlled by the maintainers or the developers.

In resume of the previous extensive list, we should consider the most proper technologies, *frameworks* and tools to create the solution, however, it may possibly not corresponding to every point, but without compromising an extendable and scalable solution which can be deployed in any server at the moment.

1.6 Document Structure

It will be important to highlight now the main sections that constitute the document to be noted the evolution of the concept of the conversational interface.

This document will consist of a section of literature review, with the theoretical foundation of conversational interfaces, such as the state-of-art of *machine learning* techniques practiced in the conception of *chatbots*, and how it should interact with the user experience.

We now enter into sections on the conversational agent design, where we will introduce the main features and the use case of each feature detailed, after, we will feature a discussion on software architecture and implementation, the main programming languages, libraries or frameworks which were used for the conversational agent implementation. A description will be presented of the service flow, since the first message from a client to *chatbot*, as well as a class diagram where each agent module will be described in detail, and the steps taken to provide a correct implementation based on the requirements.

Once the conversational agent is implemented, analysis of the results is presented, firstly a subsection of the experimental phase, where various methodologies for predicting the intentions of the users was used and which includes an extensive justification of the best option chosen.

Moreover, there is a section with the results and intent's prediction performance, used in the real environment of the agent system.

With the analysis and the tests of the conversational system described, we will have, then, a section to discuss the possible progress in optimizing the overall user experience and present the obtained feedback from the real users based on the most appropriated tools/methods to quantify.

Finally, there will be a concluding section that will serve to recap the progress made, as well as an overview of the service / system implemented and the main points of development for the future work will be presented.

Chapter 2

Literature Review

In this chapter, we will formally introduce the theory behind the conversational interfaces and how it can interfere with the optimisation of the user experience in the current era, as well as the main techniques and their state of the art in the field of artificial intelligence, more precisely the *machine learning* techniques used in the conception of conversational agents.

One area of improvement in the field of conversational interfaces we can identify, is the performance assessment and metrics used to quantify the quality of a *chatbot*. The article "*How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation*" gives the focus on how uncorrelated standard metrics in the field tend to be against the human judgment, The fundamental issue is that speech and eloquence is subjective and it is therefore very difficult to quantify it [34].

2.1 High-level schematic view of a conversational interface

Conceptually, a conversational interface is composed of several components working together to accomplish a common goal, in this case to provide the most correct response. Figure 2.1 which is inspired by the article in [23], should visually summarize the relationships of these conversational system's components.

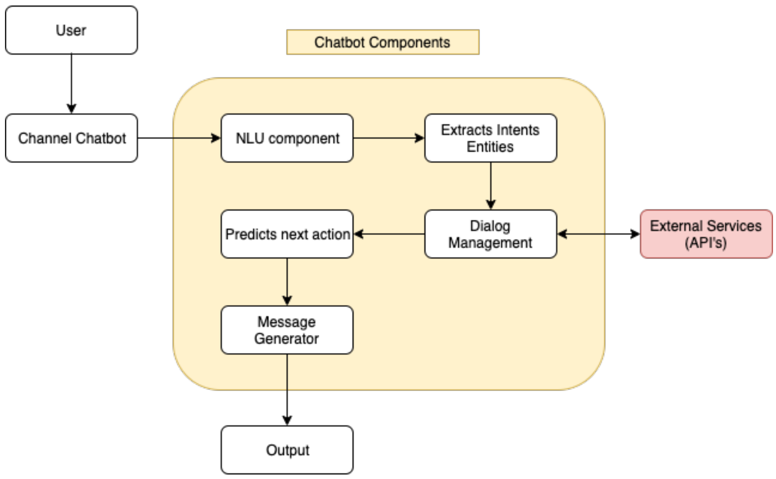

Figure 2.1: High-level schematic flow of a conversational interface

Note that a message received by a conversational interface always originates from a specific system-available channel. Once this message is received, it should be rerouted to a Natural Language Understanding (*NLU*) component, as this component is responsible for assigning the meaning of the text, extracting its intentions and entities.

After the *NLU* component, the message should then be handled by the *DM*, which is the most important and unique component of a conversational interface, as the whole conversational interface experience depends on its performance. In order to maintain a good conversational experience, the *DM* should manage the conversation context and user profile and preferences [22].

The same *DM* will also be responsible for predicting the next action or sequence of actions to be performed. For example a *chatbot* may answer with a question if the intention is not entirely clear, or it may objectively answer the message formulated by the user. This same component may also request external services to complement and enrich the response. There are several types of *DM* that differ in the way the context of the conversation flows, which will be introduced later.

Finally, having an action already indicated intention of the message, it should have a proper format based on the channel that the user is using to communicate with the conversational interface, this formatting may vary according to the agent's environment, for example, this response can be specifically formatted if you are communicating with this *chatbot* via the *Messenger* platform or a chat from a website.

2.2 Machine Learning Techniques

The *chatbots* come in two flavors, rule-based and *AI* bots. Rule-based *bots* answer to questions based on the predefined rules developers embed into them, unfortunately rule-based bots aren't able to answer questions that exhibit patterns for which these bots weren't designed, that's why machine learning techniques holds such a potential in the area of *chatbots*, these machine learning techniques are helping *chatbots* get closer where the customers will find it difficult to distinguish between a human and a bot [16], these techniques should unlock the possibility of learning from the interactions of the end users.

This section will briefly introduce the most promising deep learning techniques used in *NLP*, particularly techniques such as word embedding, a new *tensorflow embedding pipeline* technique introduced by the framework *Rasa*, and *RNN*. The first and last techniques are present in almost all frameworks in order to provide a decent ecosystem to start the development of new conversational system's.

2.2.1 *Word embedding*

There are several definitions to the term *word embedding*, but in a general notion it can be defined as the numerical representation of words, usually in a vector form. Being more specific, these are word representation vectors, where relative similarities correlate with semantic similarities [39].

Afterwards, these vectors could be a resource for feeding a machine learning algorithm. There are two popular methods for the described operation, *GloVe* which is a unsupervised learning method (a method which should have an input (X) data and no corresponding output variables), and *Word2Vec*, which is an efficient predictive model for learning word embedding of raw text [39].

GloVe, introduced by *Pennington* is an famous word embedding method which is essentially a "count-based" model. The word co-occurrence count matrix is pre-processed by normalizing the counts and log-smoothing operation, then the matrix is factorized to get the lower dimensional representation. [7] A detailed explanation of this method can be found in the article "Glove: Global Vectors for Word Representation".

The *Word2Vec* is of particular interest to the present thesis, as it has already proven to work efficiently in the field of conversational interfaces. More specifically, this technique comes with two models, the Continuous Bag-of-Words model *(CBOW)* and the *skip-gram model.* The *skip-gram model* technique is present in the library *Keras* [20], which is used in various *frameworks* to create conversational agents.

2.2.1.1 *Continuous Bag-of-Words Model*

The objective of the *CBOW* technique is pretty simple: computing the conditional probability of a target word given the context words surrounding it across a window of size k [7]. On that technique, the the non-linear hidden layer is removed and the projection layer is shared for all words, thus, all words get projected into the same position (their vectors are averaged). We call this architecture a bag-of-words model as the order of words in the history does not influence the projection [11].

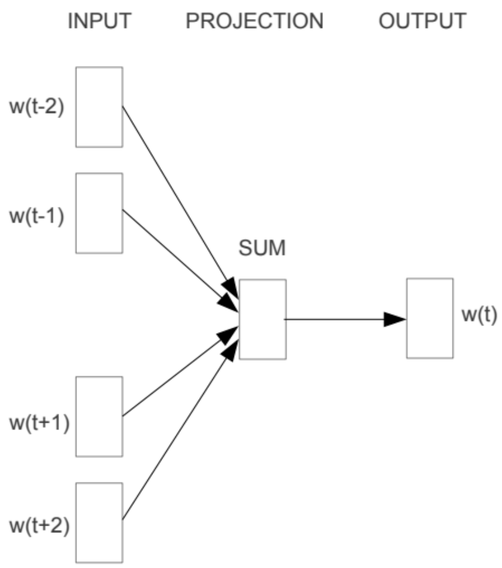

Figure 2.2: *Continuous Bag-of-words Model* architecture, source: Word Embeddings and Their Use In Sentence Classification Tasks [39]

2.2.1.2 *Skip-gram Model*

Initially introduced in *Efficient Estimation of Word Representations in Vector Space* [11], where the under-laying principle is simple, predicting the surrounding context words given the central target word. which is the exact opposite of the *CBOW* technique.

More precisely, each word is used as an input to a log-linear classifier with a continuous projection layer, and then words are predicted within a range before and after the current word [11]. Increasing the range improves the quality of the applied word vectors, but also increases the computational complexity. As the most distant words are less related to the current word than the words closest to it [11].

The figure 2.3, its a schematic image of the model architecture *skip-gram*.

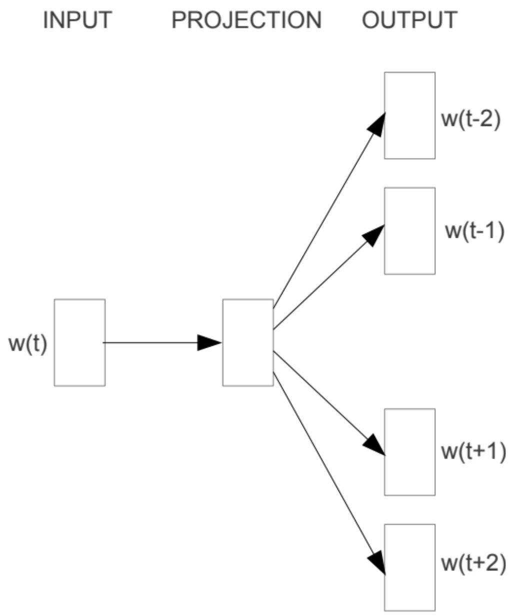

Figure 2.3: *Skip-gram Model* architecture, source: Word Embeddings and Their Use In Sentence Classification Tasks [39]

Referring to the figure 2.3, essentially artificial training examples are generated in the form of *(wt,[wt-2,wt-1,wt+1,wt+2])*, considering the window size is 5. The principle goes through the sequence of words, the middle is the size target word (input) and the words that proceed it, while the following words then form the context of the target word. The main purpose is to extract the inner layer and use as a vector representation of the word for the trained vocabulary, if the words are in *one-hot encoding* (a process by which categorical variables are converted into a form that could be provided to *ML* algorithms to do a better job in prediction), then it will serve only as a lookup table for the vector representations.

In practice, the vocabulary is converted to index sequences first, then the *skip-gram model* is trained on those same sequences and finally these index sequences are converted to vector sequences, so the final algorithm can be trained.

2.2.2 *Tensorflow Embedding by RASA*

The bag-of-words approach proved to be a good baseline, we can find some results in the following article "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification", but it can have some limitations on the practice case, lack of vector words of some important words for example, is one of the better known problems, that's especially true if we work with languages other than English [29].

Meanwhile, the *framework RASA*, introduced a new technique, called *Tensorflow Embedding Pipeline*, which instead of using pre-trained embeddings and training a classifier on top of that, it trains word embeddings from scratch. It is typically used with a bag-of-words technique to count how often distinct words of the training data appear in a message, and provides that as an input for the classifier, later. The figure 2.4 explains how the count vectors would differ [45].

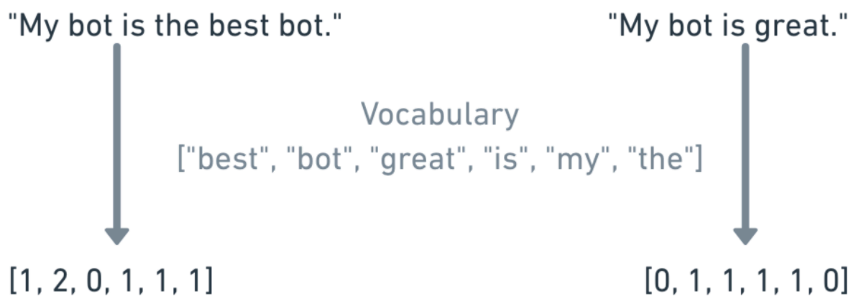

Figure 2.4: *RASA* tensorflow embedding in depth, source: Tobias Wochinger, Rasa NLU in Depth [45]

Furthermore, another count vector is created for the intent label. This method learns separate embeddings for feature and intent vectors, and both vectors have the same dimensions, which makes it possible to measure the vector distance between the embedded features and the embedded intent labels using cosine similarity [45].

This new technique presented in 2018 was developed by the team *RASA*, but was inspired by the paper "StarSpace: Embed All The Things!".

The *StarSpace* is a general neural model purpose with the objective of efficient learning entity embeddings for solving a wide variety of the problems, one of the case that can be applied is in intent classification for a conversational *AI* system [32].

The *StarSpace* model, created by *Facebook*, consists of learning entities, each of which is described by a set of discrete features (bag-off-features) coming from a fixed-length dictionary, where an entity such as a document or a sentence can be described by a *bag-of-words* or *n-grams*, an entity such a user can be described by the bag of documents, movies or items they have liked. The model is free to compare entities of different kinds. A user entity can be compared with an item entity (recommendation), or a document entity with label entities (text classification) and so on. This is possible by learning to embed them in the same space such that comparisons are meaningful [44].

In short *StarSpace* embeds entities of different types into a vectorial embedding space, so the word *star* (*), meaning all the types and the *space* name, in that common space compares them against each other [32].

For an *AI* conversation agent, the embedding intent classifier, embeds user inputs and intent labels into the same space as already explained, the user inputs can be considered by a *bag-of-words*. Also during the training model user inputs should be compared and the following loss should be minimized [32].

$$\sum L^{batch}(sim(a,b), sim(a,b_1^-), ..., sim(a,b_k^-)) \tag{2.2.1}$$

In the equation 2.2.1 we can identify the following points, if applied in a *AI* conversational system [32].

- a are the documents (*bag-of-words*).
- b are the labels (intents) from the training set.
- Negative entities b are sampled from the set of possible labels.

- (a, b) is positive entity pairs, comes directly from a training set of labeled data specifying (a, b) pairs.

- $sim(\cdot, \cdot)$ is the similarity function. In the case the *Rasa* default uses cosine similarity.

- L is the loss function that compares the positive pair (a, b) with the negative pairs.

2.2.3 *Recurrent Neural Networks*

The *RNNs* or Recurrent Neural Network is a neural network model proposed in the 80's by *(Rumelhart et al., 1986; Elman, 1990; Werbos, 1988)* for modelling time series [2]. The main feature of this type is that it can retain past information from received input, allowing to discover temporal correlations between events that could possibly be distant from each other input in the data. It can be also said, they're a neural network with loops allowing the information to persist.

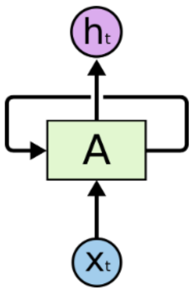

Figure 2.5: A chunk of an *RNN*, source: Cristopher Olah, Understanding LSTM Networks [31]

In figure 2.5, A looks at some input, in this case, X_t and outputs a value h_t. This loop allows the information to be passed from one step of the network to the next. A *RNN* can be thought of as multiple copies of the same network, each passing a message to the successor [31].

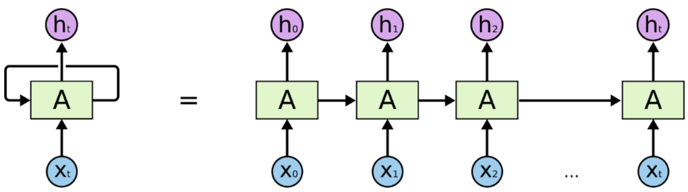

Figure 2.6: Unrolled loop of an *RNN*, source: Cristopher Olah, Understanding LSTM Networks [31]

Looking at the figure 2.6 which depicts an unrolled loop of an *RNN*, we can realize the nature of that neural network is related to the sequences and lists. They're the natural architecture of the neural network to use for such data.

In recent years there has been a quite good success applying *RNNs* in the most variable problems, speech recognition, translation, image captioning [31].

This success comes from the use of Long Short Term Memory Networks (LSTMs), which are a type of *RNNs* that work in many cases, much better than the standard model, almost all exciting results gained from the use of *RNNs* are complemented by their use.

2.2.4 *Long Short Term Memory Networks*

The Long Short Term Memory Networks are used extensively in *NLP*, they were first introduced in 1997 by *Hochreiter and Schmidhuber* [38]. The *LSTMs* are designed to avoid a problem present in the *RNN* version already explained in the section 2.2.3, the long-term dependency problem, in a nutshell, the simple *RNNs* can contain a gap on transporting the contextual information between the networks, this problem has been explored in depth by *Bengio* [3], who found some pretty fundamental reasons why it must be difficult.

If we compare the repeating module from the standard *RNN* as the *Cristopher Colah* [31] mentions, we can observe such a simple structure, with a single *tanh* layer, reffering in the image 2.7 .

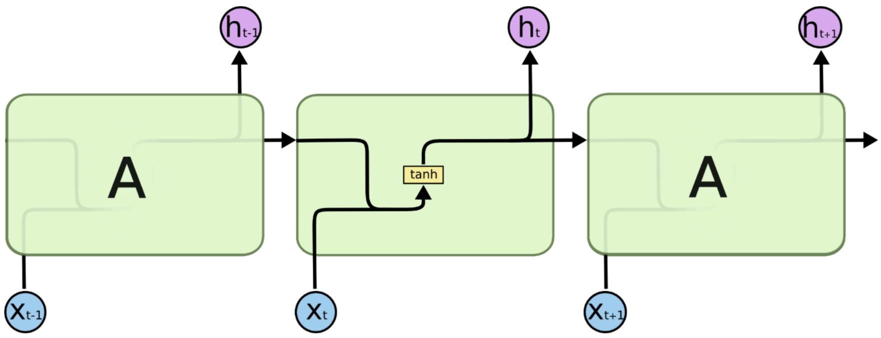

Figure 2.7: Repeating module in a standard *RNN*, source: Cristopher Olah, Understanding LSTM Networks [31]

However, if we compare it for the chain structure of the *LSTMs*, we can observe the differences on the repeating module, instead of having a single neural network layer, there are four, interacting in a very special way, those layers are presented in figure 2.8.

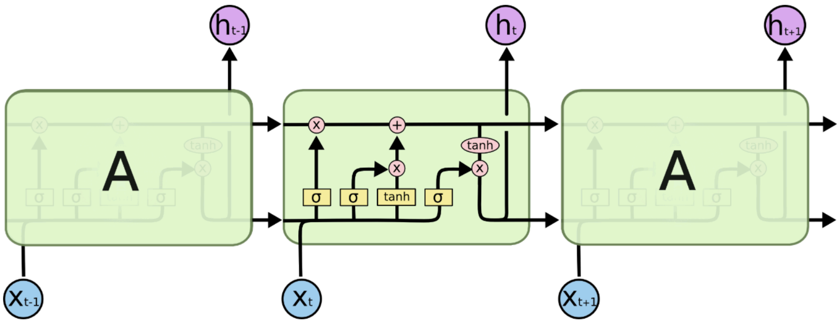

Figure 2.8: Repeating module in the *LSTMs*, source: Cristopher Olah, Understanding LSTM Networks [31]

The main components of the *LSTMs* are the *forget gate*, *input gate*, *cell state* and *output gate*, all the them will be explained on this section for a better understanding of the flow at *LSTMs* neural network's.

The first state, called as forget gate layer, which is responsible to decide what information should throw away from the cell state, and its provided by the equation 2.2.2.

$$f_t = (W_f \cdot [h_t{-}1, x_t] + b_f) \tag{2.2.2}$$

It looks at $^ht{-}1$ and the x_t and then outputs a number between 0 and 1 for each number in the cell state $^Ct{-}1$. [31] Logically the number 1 represents to keep the value, while the number 0 represents to forget the value. The figure 2.9 illustrates the process.

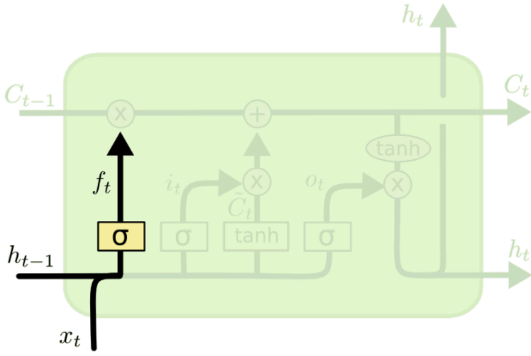

Figure 2.9: *LSTM* forget gate unit, source: Cristopher Olah, Understanding LSTM Networks [31]

The *input gatet́s* will decide what information should store in cell state, which is segmented by two sections. The first which its called by *input gate layer* decides which values should update, provided by the equation 2.2.3, denoted by $\tilde{C}_t$ scaled by i_t as the figure 2.10 shown [31].

$$i_t = (Wi \cdot [h_t{-}1, x_t] + b_i) \tag{2.2.3}$$

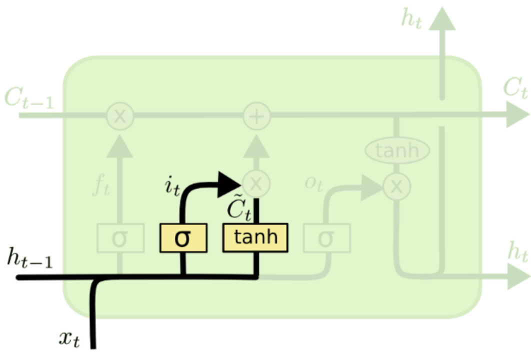

Figure 2.10: *LSTM* input gate layer, source: Cristopher Olah, Understanding LSTM Networks [31]

The old cell state is then updated according to the equation 2.2.4, donated by $\tilde{C}_t{-}1$ into the new state donated by C_t. The previous steps already decided it, so we should only execute it. It should multiply the old state by f_t, forgetting the values we decided at first step, then we add $i_t{*}C_t$. This should be the new values of the *cell state*. The connections of this update are highlighted in figure 2.11 [31].

$$C_t = f_t C_t{-}1 + i_t tanh(W_C \cdot [h_t{-}1, x_t] + b_C) \tag{2.2.4}$$

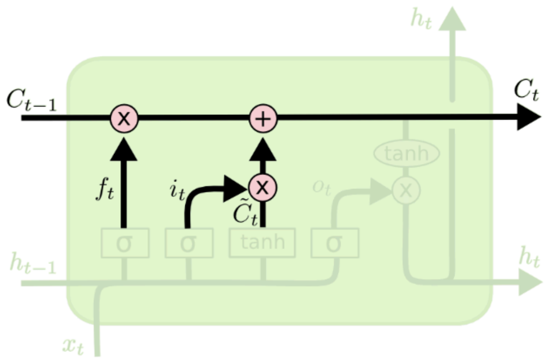

Figure 2.11: *LSTM* cell state update, source: Cristopher Olah, Understanding LSTM Networks [31]

In the final step, it should decide what the output should be. The output is based on the current *cell state*, but a filtered version. It runs a sigmoid layer which decides what parts of the state it should output, and then, the cell state should be putted through *tanh* and multiply it by the output of the sigmoid gate function. It is reveled by the equations 2.2.5, 2.2.6 and the figure 2.12 [31].

$$o_t = (W_o \cdot [h_t{-}1, x_t] + b_o) \tag{2.2.5}$$

$$h_t = o_t * tanh(C_t) \tag{2.2.6}$$

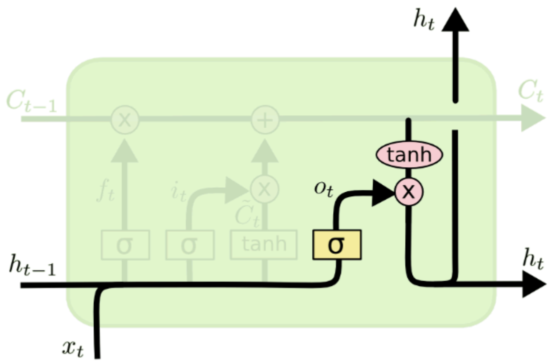

Figure 2.12: *LSTM* cell output gate, source: Cristopher Olah, Understanding LSTM Networks [31]

2.3 Intent Classification

Once received a new message input, the conversational interface should be able to identify the goal the user is trying to accomplish. This is usually modelled as a multi-classification problem where we have labels as the names of the possible user intentions. There are several techniques to solve this problem. Can vary from a simple keyword extraction to an Bayesian inference in order to determine the users request based on multiple messages. The *LSTMs* networks are proven to work well [18], however, the new *RASA tensorflow classifier* also provides such good metric results that can be detailed observed at publication "*Supervised Word Vectors from Scratch in Rasa NLU*" [29], which will be used at the conception of the *chatbot*.

2.3.1 Metrics

To measure as possible, the performance of the existed intents at the conversational system it's necessary to apply some metrics. We can identify the following important metrics for that, **precision**, **recall** and **F1-Score**, **macro-average measure** and finally the **micro-average measure**. The calculation of these metrics is based upon four categories, true positive, true negative, false positive and false negative, which will be explained in the next section [33].

2.3.1.1 True/false positives, true/false negatives

The positive/negative identifies the declared solution for the data point. For example a data point is declared positive if the system declares they have the condition, while the true/false refers to the success or failure of the prediction.

For example, the true negative means that the data point we classified doesn't belong to the class and that's exactly what was predicted. For a better visualization of the explanation we can observe the figure 2.13 [33].

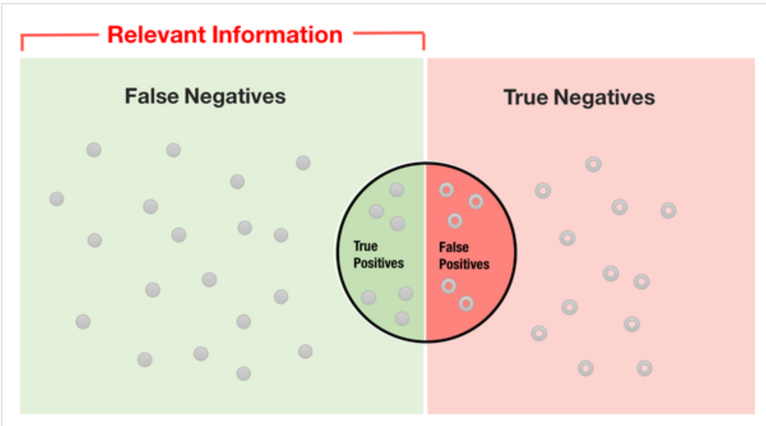

Figure 2.13: *True/false positive and true/false negatives* visualization, source: [35]

2.3.1.2 Precision

The precision identifies the frequency of the correct answers, when the prediction is A, we can imagine as the answer of the question, "In all of the predictions to A, how many were correct?", this is defined by the equation 2.3.1, where we can identify tp for the true positives and fp to false positives [33].

$$Precision = \frac{tp}{tp + fp} \tag{2.3.1}$$

2.3.1.3 Recall

The recall identifies the frequency of detecting A out of all examples to A in the reality, it answers to "out of all examples in A, how many were detected?", which is given by the following equation 2.3.2 where the tp is the true positives, and fn is the false negatives respectively [33].

$$Recall = \frac{tp}{tp + fn} \tag{2.3.2}$$

2.3.1.4 F1-Score

Finally the *F1-Score*, calculates the harmonic mean (type of numerical average, it is calculated by dividing the number of observations by the reciprocal of each number in the series) of the **precision** and **recall**, in short, it should answer to the question, "What is the global performance of prediction with respect to the class A ?", which can be defined by the equation 2.3.3 citeclassification-metrics.

$$F1Score = 2 \cdot \frac{Precision * Recall}{Precision + Recall} \tag{2.3.3}$$

2.3.1.5 Confusion matrix

The confusion matrix is a table that helps to recognize more precisely the issues in the prediction on the classifier, it‘s based on the metrics which were explained before [33].

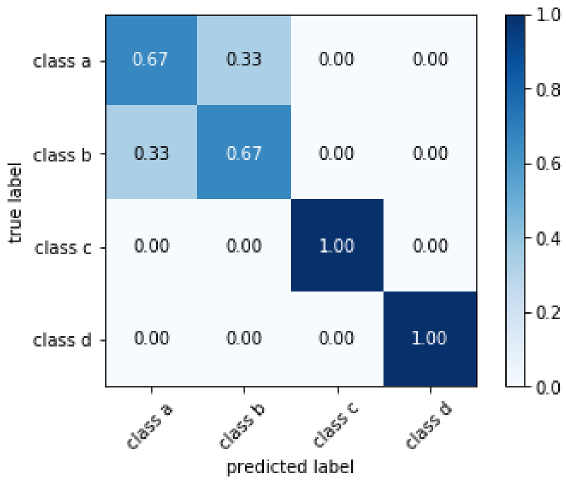

Figure 2.14: *Confusion Matrix* example, source: [36]

Reading the confusion matrix is easy, we can observe in figure 2.14, it‘s a simple table with rows and columns, the rows represent the true labels while the columns represent the detected labels. The perfect confusion matrix should be brightly colored at main diagonal without scores in any other areas [33].

2.3.1.6 Macro-average measure

In the *macro-average* measure we sum the individual true positives, true negatives, false positives and false negatives of the system and then we apply them for getting the statistics. *Macro-average* gives the equal weight to each class. Considering the equation 2.3.4, where the $B(tp, tn, fp, fn)$ is the binary evaluation measure, that is calculated based on the number of true positives, true negatives, false positives and false negatives, let tp_λ, fp_λ, tn_λ and fn_λ be the number of true positives, false positives, true negatives and false negatives after binary evaluation for a label λ [1].

$$B_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q} B(tp_\lambda, fp_\lambda, tp_\lambda, fn_\lambda) \tag{2.3.4}$$

2.3.1.7 Micro-average measure

While *Macro-average* gives equal weight to each class, *Micro-average* gives equal weight to each per-document (aggregate the contributions of all classes) classification decision, because, the *F1* measure ignores true negatives and its magnitude is mostly determined by the number of true positives, generally the *micro-average* measures the effectiveness on the large classes for a test collection (will give a closer number to the highest number of the collection) while the macro-average should measure the sense of effectiveness of the small classes. Considering the equation to compute the *micro-average* in 2.3.5 which uses the same expressions in the formula at 2.3.1.6 [1].

$$B_{micro} = B\left(\sum_{\lambda=1}^{q} tp_\lambda, \sum_{\lambda=1}^{q} fp_\lambda, \sum_{\lambda=1}^{q} fn_\lambda, \sum_{\lambda=1}^{q} tn_\lambda\right) \tag{2.3.5}$$

2.3.2 Metrics Implementation

All of these metrics gives different insights about the performance of each intent, as well as the *chatbot* as a whole (since the *NLU* component is considered in the performance of the solution).

It should be separated calculated over each intent and combined in order to correctly measure the performance of a conversation system. To detect the issues with more precision, the intent confusion matrix should be considered, it should work as a visual tool to identify problems and inconsistencies at intent classification presented on a *chatbot*.

2.4 Dialog Management

In order to communicate, a conversational interface should have the ability to reply. The conversational agent receives requests from users, either through spoken language or direct text input and outputs also a text or vocal response [17].

2.4.1 High-level view of the *DM*

The management of the responses generated by a conversational interface is done from the *Dialog Management (DM)* component, for a better understanding of the flow at the *DM* present on a conversational interface we can observe figure 2.15.

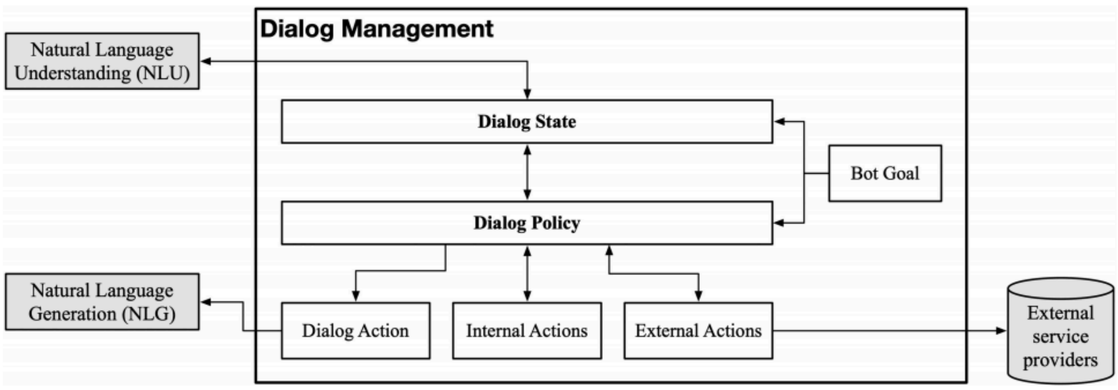

Figure 2.15: *Dialog Management* system architecture, source: Approaches for Dialog Management in Conversational Agents [17]

The first step on the flow of a conversational agent should be converting the user input message received to a user action, also called by intent. This step is processed by the *NLU* component of the agent. The *NLU* output can also carry data fields, also called slots or entities. For a better understanding, lets imagine the following example, the Dialog System is a travel planning service, and if that service receives an input of user like (e.g, "Book a flight to Oporto"), it should be labeled with an input name at the first step, (e.g., "*FlightBook*"), the slots or entities on that intent should be the (e.g., "Oporto") [17].

It's important to refer the *NLU* and *DM (Dialog Management)* are separate components, but they influence each other's performance.

The *Dialog State* should be tracking any information throughout the conversation, which is provided for the *NLU* output's and it should form the foundation for identifying the next action to take and interpreting the conversation. Also, The dialog state can also be influenced by the goal of the dialog agent itself, for example, if we want to sell any specific product on the dialog system, the dialog state can be influenced by these proposals of the dialog agent [17].

When the *Dialog State* is updated, the *Dialog Policy* should be triggered which takes a new state and decide the next action or actions the dialog agent should execute. The *Dialog Policy* is the central part of the *DM*, it should build the bridge between, the conversation context (*NLU* output's), third-party services and the dialog agent response's [17]. Also, the *Dialog Policy* should pick from the dialog, internal and external actions. The dialog corresponds to a message output to the user, which can be a template (e.g, "There's a flight at {departure_time}") or in the more robust and complex systems, a dialog act, for example, in order to inform the user (e.g., inform(departure_time= "1 PM")), and then this output will be converted by the *Natural Language Generation (NLG)* component to a textual or voice response for the user. While the internal action is one that the agent orchestrates with the objective to modify his behaviour or improve the performance, for example, improve the policy by retraining the system or seeking for external information for performance improvement. The external action should interact with a service provider *(API)* in order to provide a useful response to the user's request, by requesting data or triggering some other application event [17].

2.4.2 *Approach's and Tools to the DM*

In the document "*Approaches for Dialog Management in Conversational Agents*" [17], there are 3 referred types of approach's to create a *Dialog Management System.*

- Handcrafted (rule-based) approach.

- Probabilistic (statistical) approach.
- Hybrid approach.

The *handcrafted approach* is designed by the handcrafted dialog managers. They define the state of the system and the policy by a set of rules which are created by the developers and the domain experts. It's called the simplest subset of the dialog system which is modelled by a *finite-state automata* where the conversations are always one of defined states of the conversation at the time, also each state has a fixed number of transitions to the other state. Such dialogues have system-directed initiative, so the system can ask information from the user step-by-step [17].

The *probabilistic approach*, instead of defining the set of the rules to the dialogue system, probabilistic the *DM* learns for the rules from the actual conversations. The example-based systems learn the appropriate answers from a large corpus by matching the last query in an example of the training *dataset*, also extracts the response's for the training *dataset*. Imagine we have a training corpus like [user: "Hello", system: "Hi, how are you?"], if the user says "Hello", then the system will reply with the other message. However, this approach has several limitations in the terms of error handling [17].

The *hybrid approach* is next to the purely rule or statistical approach, but since that some work has been done to combining the advantages of the both approach's. These hybrid approaches are an important step toward introducing data-driven elements into available dialogue agents. This approach should use a neural network combined with coded constraints and rules. The intuition is that some parts of goal-oriented dialogues, like sorting the data returned by external service providers, is very hard to learn by example dialogues, so it should be much simpler to implement in a few lines of code [17].

The next figure 2.16 , from the article [17], it will suggests the available tools/frameworks to work on each approach already described.

High-level category	Low-level category	Tools	Comparison (dimensions)						
			Dialog structure	Learning	Error handling	Dependencies	Control	Domain independence	Tool availability
Handcrafted	Finite-state	Flow.xo	Tree	manual iterative	escalation message	none	complete control	none	Hosted, Visual
	Frame-based	DialogFlow	Directed acyclic graph	manual iterative	escalation message	none	highly controlled	sub-modules reusable	Hosted, Visual
	Model-based	TrindiKit (Information State)	Undirected graph	manual iterative	rule-based recovery	conversation model	highly controlled	model party reusable	Prolog/Python
Probabilistic	Example-based	Chatterbot	random	no learning	ignored	Large corpus	uncontrolled	examples reusable	Python
	MDP-based	PyDial	Undirected graph	runtime, iteratively	probabilistic recovery	large conversation collection (>1000)	semi-controlled	independent state tracker and policy	Python
	Neural Networks	Memory Networks	Undirected graph	runtime	-	~500 conversation collection	uncontrolled	-	-
Hybrid		OpenDial	Undirected graph	runtime	probabilistic recovery	conversation collection (>100)	semi-controlled	independent state tracker and policy	Java
		Rasa Core	Directed acyclic graph	manual runtime (through interaction)	probabilistic recovery	conversation examples (>10)	semi-controlled	sub-modules reusable	Python

Figure 2.16: *Dialog Management* available tools based on different approach's, source: Approaches for Dialog Management in Conversational Agents [17]

The figure 2.16, suggests an overall comparison between the different available approach's with respective *frameworks* or *tool-kits*, with special reference to the *dialog structure* which influences the complexity of the possible dialogs and the naturalness because rigid dialogs are also repetitive [17]. We highlight also, the *domain independence* referred, which is one of the influencing factors for scalability of the overall dialog system [17], it define the possibility of the sequences to the messages. Overall, the figure 2.16 details the best available tools to create a modern conversational system, but we will keep the focus on the *Rasa* tool, since it corresponds to the all the *constraints* cited in 1.5.

2.5 Responses Generation

In order to define the appropriate response chosen by the *Dialog Management* system it's necessary to have the properly method or service which will take the care of responding with the natural language of the humans, this can be called by the *Natural-language Generation* (*NLG*) service.

This problem can be tackled using two different modules, the *retrieval-based models* or the *generative-based models* [8].

The **retrieval-based** technique simply rely on a large database or *dataset* of candidate responses and matches them with the information from the data message of the user to find the most appropriate message. The main advantage of this technique is the full control of the responses to the specific domain of the *conversational* system and then avoid inappropriate responses [8].

On the other hand, the **generative-based** technique rely on generative models to generate new replies without the need of an extensive database or dataset. This can be used when a large amount of data is available and the system can be trained on that data, the response is generated based on an algorithm used on the *NLP*, like the *Recurrent Neural Network* or the *LSTMs* can be used in this model. However, it's not possible yet to have purely generative models. Even the most advanced systems like *Alexa*, *Siri* or *Cortana* are semi-rule based [8].

2.6 User Experience

So, now that we already have identified the main techniques to create the conversational interfaces, we need to know if these interfaces can be a possibility to optimise the user experience. We need a better understanding of what is user experience and how it's related to the conversational interface.

In fact, defining *UX* is a difficult task as it is dynamic, context dependent and subjective, suggested in *Understanding, Scoping and Defining User Experience, A Survey Approach* [25]. Following the *ISO (International Organization for Standardization)* definition, in 2008, was defined with the following sentence, *"A person's perceptions and responses that result from the use or anticipated use of a product, system or service"*, so we can immediately assuming that is something narrated about the use of a system, product or service [25].

According to the Don Normal and Jakob Nielsen, in the *The Definition of User Experience (UX)*, the first requirement for the user experience design is the hassle-free satisfaction of customer needs, while adding simplicity and elegance that makes a product captivating to the user. True *UX* goes beyond giving what customer‘s what they want or want to be done in common tasks or processes, to deliver the best *UX* possibly it should exist a fusion on multiple areas of an organization, such as engineering, marketing, graphic design and interface design [13].

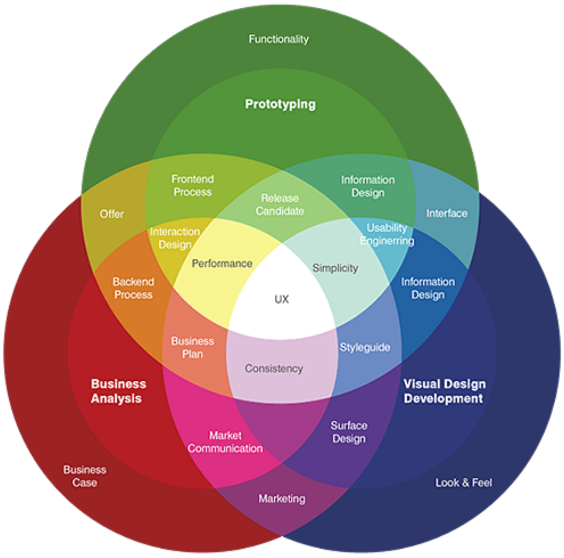

Figure 2.17: *User Experience* diagram, source: A Wide Perspective for Designing User Experience [21]

Figure 2.17 intends to idealize all the process of the optimisation of the *UX* with the fusion of the mentioned areas.

Mentioned by *Golden Krishna* in *The Best Interface is no Interface* [24], *"The User Experience has currently stagnated with graphical interfaces, instead of looking for the most creative, inventive and useful ways to solve problems, we limit them by solving them with graphic interfaces, when we see a problem we put an interface, UX is no longer about people and it's about round rectangles and parallax animations"* .

If we analyse this criticism by *Golden Krishna* to the *state-of-art* of the *UX*, it's pretty straight that a conversational interface can change how we solve the problems in other way than the using of graphic interfaces.

2.7 Measuring the *User Experience*

Since we have reviewed the most important techniques and topics to create a conversational interface, we should consider the most consistent methods to evaluate the performance on the optimisation of *UX* over the current user interface.

Nowadays, it's possible to do a wide range of common tasks "in the cloud", some of them which were only possible to do in native client applications (e.g, "Photo Editing") [15].

For the professionals of the *UX*, one of the keys of this shift is the ability to use the web server log data to track the product usage on a large scale. If we apply additional instrumentation it's also possible to run the famous *A/B* controlled tests that can compare the interface alternatives (a method of comparing two versions of a *webpage* or app against each other to determinate which one is better), but this raises a important question. What criteria should they be compared from an *user-centered* perspective [15] ?

Also, despite the large body of work done on the proper design and analysis of the controlled tests *A/B* where we can find in the article *Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO* [37], it's still challenging to use this tool effectively, the standard web analytics may be too generic to apply on a particular product goal or research question [15].

We still have the low-level and direct metrics, the *Pulse* Metrics, While businesses should still track these metrics, they should remember that they lack context for measuring *UX*, we can define that by a simple example, an average time of 5 minutes on the website might mean users are extremely engaged with the product, or they are just not finding the content they need.

2.7.1 The ***PULSE*** metrics

These metrics are mainly focused on the business or technical aspects of the product, in fact they are the mostly large-scale metrics used by the organizations to track the overall product health [15].

The *PULSE* metrics can be divided in 5 aspects.

- **P**age Views, reflects the amount of users visiting the product.
- **U**ptime, percentage of time the server is up, running and serving content.
- **L**atency, gives a proper indication of the overall performance.
- **S**even-day active users, can be for example, the number of unique users who used the product at least once in the last week.

- **E**arnings, gives a good indication if the product works or not.

These metrics are extremely important and also related to the *UX*, if we go in depth with a simple example, a product that has a lot of outages or is very slow will be hard to attract the users, in the other way, the excellent user experience on the product obviously is more likely to increase the page views and unique users. However, they're all low-level or indirect metrics of the *UX*, making them problematic when used to evaluate the impact of the user interface changes. Also, they can be ambiguous interpreted, the most basic example, a rise in the page views of the particular feature may occur because the feature is popular by itself [15].

2.7.2 The *HEART* metrics framework

Created by *Google*, and based on the problems which were introduced on the *PULSE* metrics, this metrics should work as a complementary framework.

The *HEART* framework comes from five categories, **H**appiness, **E**ngagement, **A**doption, **R**etention and **T**ask Success. These categories should be from each team must define the metrics that will use to track progress towards goals [15].

- **Happiness**, this term should describe the metrics that are attitudinal in nature, they're related to the subjective aspects of the user experience, satisfaction, visual appeal, likelihood to recommend, and perceived ease of use. In this category a general well-designed survey should be possible to track some metrics to see the progress as changes are made [15].

- **Engagement**, is the user level of involvement with a product, in the metrics context the term is normally used to refer to the behaviour intermediaries like the frequency, intensity or depth of the interaction over some time period. If we intend to use on examples, might include the number of visits per user per week or the number of photos uploaded per user per week [15].

- **Adoption** and **Retention** metrics can be used to provide a stronger insight into counts of the number of the unique users in a given time of period (e.g, seven-day active users) which will address the problem of distinguishing the new users from existing users, the adoption metrics should track how many new users start using a product during a determined time period, the retention in the other way, should track how many of the users in the determined time period are still present in some later time period, (e.g, the percentage of seven-day active users in a given week who are still seven-day active three months later) [15].

- **Task Success** which centres the several user experience traditional behaviour metrics, like the efficiency (e.g, time to complete a task) or effectiveness (e.g, percentage of the tasks successfully completed) and the error rate, to measure these on a large scale, should be via a remote usability or a *benchmarking* study, where users can be assigned to some specific tasks over the product [15].

We believe the *HEART* metrics, should be capable enough to highlight the *UX* of a conversational system in the *user-centered* perspective.

2.7.3 The System Usability Scale *(SUS)* Case

The usability is a narrower concept than *UX* since it's focused on the goal achievement. However, still very important in order to measure the overall *UX* of a system.

The requirement to evaluate the usability of a system mean that in the most of the cases is neither cost-effective nor practical to perform a full-blown context analysis and selection of suitable metrics. Often, all that is necessary is a general indication of the overall level of usability of a system compared to its competitors or its predecessors. [6]

The *System Usability Scale* is targeted to provide a "quick and dirty" reliable tool to measure the usability of a system. It's a simple ten-item in *Likert* scale with five response options for respondents, from strongly agree to strongly disagree, and then, giving a global view of subjective assessments of usability. [6]

The *SUS* has become an industry standard, with many references to articles and publications, the global benefits of using this scale refers to the following list [42].

- Is a very easy scale to administer to participants.
- Can be used on small sample sizes with reliable results.
- Is valid it can effectively differentiate between usable and unusable systems.

The participants are asked to score the following 10 items (table 2.1) with one of five responses that range from *Strongly Agree* to *Strongly disagree*. [6]

Item	Description
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

Table 2.1: *SUS* likert scale items

Interpreting the scores can be complex, the scores for each question are converted to a new number, added together and then multiplied by 2.5 to convert the original scores of 0-40 to 0-100. Though the scores are 0-100, these are not percentages and should be considered only in terms of their percentile ranking [42].

Based on the research *Determining what individual SUS scores mean: adding an adjective rating scale* [28], the *SUS* score above 68 should be considered above average and any number below 68, as below average, however the best way to interpret the results should be normalizing the results to produce a percentile ranking [42].

Chapter 3

Software Features, Architecture & Implementation

This chapter will talk about the main features and the overall architecture of the conversational interface and his implementation. More specifically, we will identify the main features of the first version of the conversational interface in the section 3.1. After, we will introduce an overall standardization that how a conversational interface architecture should be implemented in the section 3.2, and then we will purpose the practical architecture cases which we will implement in the section 3.3, in the section 3.3 will be explained each module of the final architecture. After, we will justify the use of the programming languages and libraries/frameworks at the section 3.4, after that, the section 3.5 will be introduced where we will explain how we extracted the Portuguese words, then we will explain the choices made to create the *NLG* in the section 3.6, and finally, in the section 3.7 will be explained how the communication between the conversational system and the current software interface was created.

3.1 Identified Features and Use Cases

In order to start the implementation of the conversational interface we should define a few features which the agent should be able to respond it correctly or ask some more details if needed.

A few user scenarios were defined which we provide in the following table 3.1 .

Feature	Description
Obtain the balance of a specific customer at the moment.	It should be possible to identify the customer name or number, ask if it's not correctly provided, search for the customer on the client database, and provided the correct current balance.
Get the latest task for a specific customer.	It should be possible to identify the customer name or number, ask if it‘s not correctly provided, search for the customer in the client database, and get the latest task available on the system.
Get the most debtor customers.	It should be possible to provide a short list with the most debtor customers on the system, it should also be possible to get the fixed number of the list. (E.g, "I want the top 8 of my customer debtor‘s")
Get the won values by opportunities for a specific customer in a specific year.	It should be possible to identify the customer name or number, ask if it's not correctly provided, search for the customer in the client database, and then get the opportunities value for the specified year or the current year if the year is not specified.

Table 3.1: Initial identified features for the development of the conversational interface

To reinforce or summarize the visualization of the features we also provide the practical use case's, where the actor is the user of the system, which can contextual asks for a specific feature that we already identified, each of which should interact with the *Service Provider* of the system, in this case, the *FOXAIO API*.

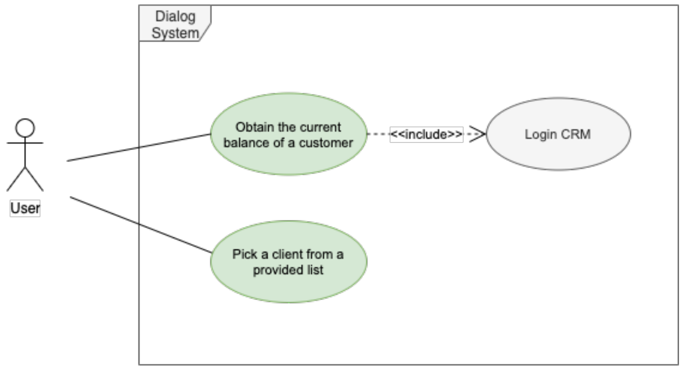

Figure 3.1: *Use Case* for the feature to obtain the client balance

In figure 3.1 we can detail the feature to obtain the current balance for a customer on the system, once the intent is correctly identified by the dialog system, it will communicate with the service provider where it will be responsible to authenticate correctly the user on the *API*, and also search for the specified entity which should be a customer, if the search on the system obtain more than one customer, it should provide the list of available customer's and let the user pick the intended one.

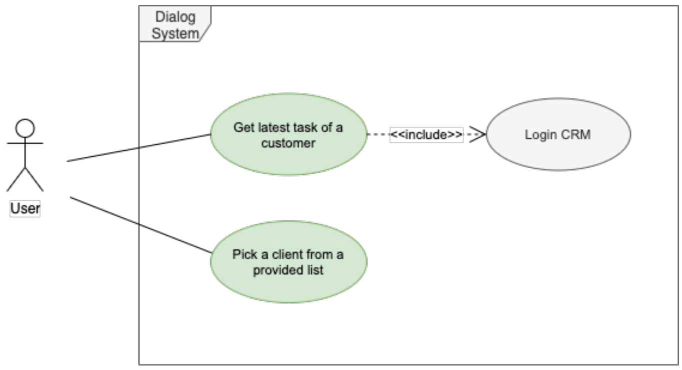

Figure 3.2: *Use Case* for the feature to obtain the latest task

Similar to the feature to get the balance, in figure 3.2 the flow is exactly the same, the only difference in the practical case, it's the communication to the *API*, where it should request for the latest task to the client as the name elucidates.

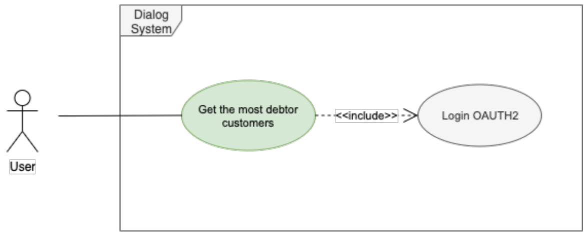

Figure 3.3: *Use Case* for the feature to obtain the most debtor's clients

The figure 3.3 explain the flow of the feature to fetch the most debtor's customers on the system, once the intent is correctly identified, it will communicate with the *Service Provider*, this will get the most debtor's customers based on the provided ranked number (e.g, top 10 most debtors clients), if the ranked number is empty it will fetch by a default number defined on the conversational interface system.

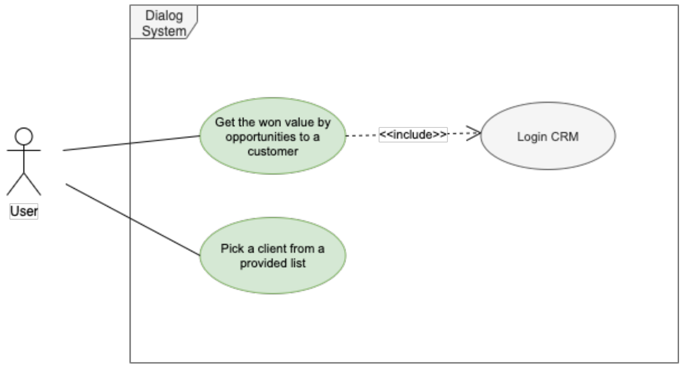

Figure 3.4: *Use Case* for the feature to obtain the the values won by opportunities

Once more the flow at the figure 3.4 is identical to the flow of the figure 3.1, the differences here remains to the request to the *API*, and also to the custom entities with the intent that served to filter correctly the query, for example, we could fetch the won value at opportunities only at the year of 2018 for a specified customer, it should also, let the possibility of the client to pick the customer from a provided list if it's necessary.

Summarizing, the conversational agent must be born with simple features, but with a good and stable implementation structure in order to easily maintain the system, and also enrich his skills without the need to create new modules.

3.2 Standardized Architecture

In order to identify how an architecture of a conversational interface should be in depth, it should be better to study and introduce a standardized architecture proposed by *Roshan Kan* [22] for the typical *chatbots*.

In figure 3.5, we introduce how *Roshan Kan* thinks how an architectural view of a typical conversational agent solution and its processing ecosystem should look like.

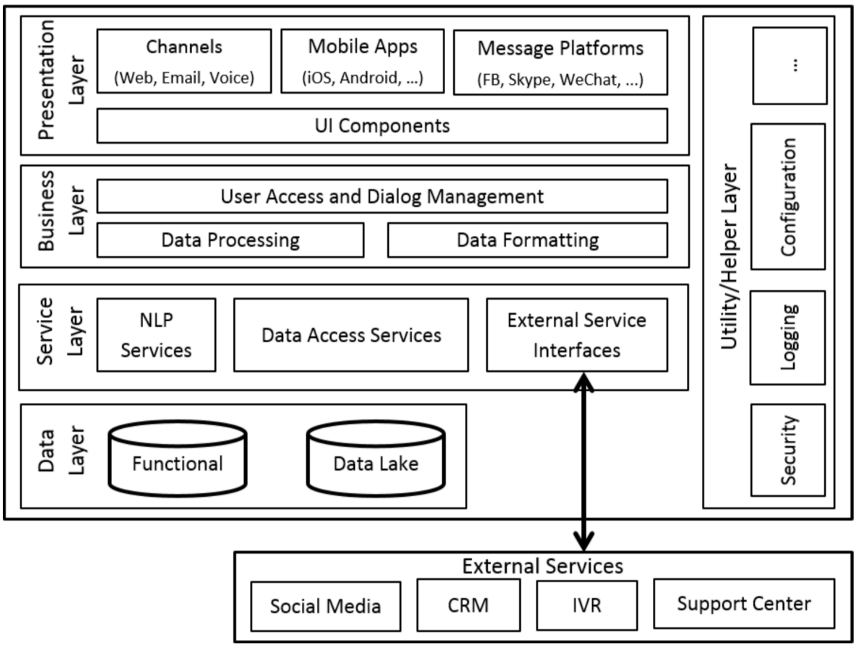

Figure 3.5: *Conversation Interface* typical architecture, source: Standardized Architecture for Conversational Agents a.k.a. ChatBots [22]

In the *Presentation Layer* it should contains all the components that implement and display the user interface and manage the user's interaction [22].

Also, it should correspond to the following points:

- Multi-Channel Support
- Multi-Platform Support
- User Interface Components

The *Business Layer* should take care of the *Data Processing* which should be the transformation of the data from the *Service Layer* to the real-word business entities, such a product or orders and not as database entities. Also the *Business Layer* should take care of the *Data Formatting* which is the component who has the responsibility to convert the data into the required format to the specific channel. And finally, The *Dialog Management* which was being already explained, which is the component responsible to manage the dialog with the final user. [22]

The *Service Layer* provides access to internal and external services, business functionality, *middleware* connectivity and other services. The *NLP* service should be a part of the *Service Layer* where it should be the most efficient as possible using the most advanced *state-of-art* artificial intelligence algorithms. Also should have the *Data Access Services*, when the messages are being passed between a service and a consumer, most of the times, the message needs to be transformed into a format that the consumer can understand. As the agent would need to integrate with different set of services, we need to implement adapters to provide access to these services which converts the data from services in a format that the other components also understands. Finally, we should also have the *External Service Interfaces*, which depends on the context of the *chatbot*, it might need to integrate with different set of external services. [22]

Then we have the *Data Layer (Storage)* module, it is very critical to have an efficient and secure data access and that is why it is of utmost importance to have a well-defined approach in designing the data layer. Since the *chatbot* need a lot of operations with data, it should be important to have a fast data access. There are lots of services and components which rely on this data storage and access at all times, e.g., storage of all communication with the users, analysis of the data collected, performing machine learning techniques on the data. [22]

And finally, we have the *Utility Layer*, which is not considered a functional part of the system, but it's also important. Since we can have a solution that can be exposed to a multitude of systems, itself, its highly vulnerable, and we should need to monitor all the inherit risks. It should also have tools to scale and optimise the deployment of the solution. [22]

Of course, all of these components cannot be present in a specific architecture of a conversational agent, it always depends on the context of the agent and also on the choice of the *Dialog Management System*. It's an overall view of how the flow should work on a typical *chatbot*.

3.3 Architecture Proposals

In order to implement the *conversational* system, it was required to design the best approach possible following the research done in the chapter 2. We divided that in two proposals, the first one is the architecture for the *rule-based DM* system, and the second one, is nothing more than the product core provided by the framework *Rasa*. It's a hybrid approach where we will explain, since it is important to explain how the conversational system works in the depth, in the end, we explain the choice we made justifying to the specific domain of the conversational system we wanted to implement.

3.3.1 *Rule-based DM* system architecture

In figure 3.6, we introduce the design of the architecture for the conversational system using the *tool-kit Botkit* linked to the *NLU* component.

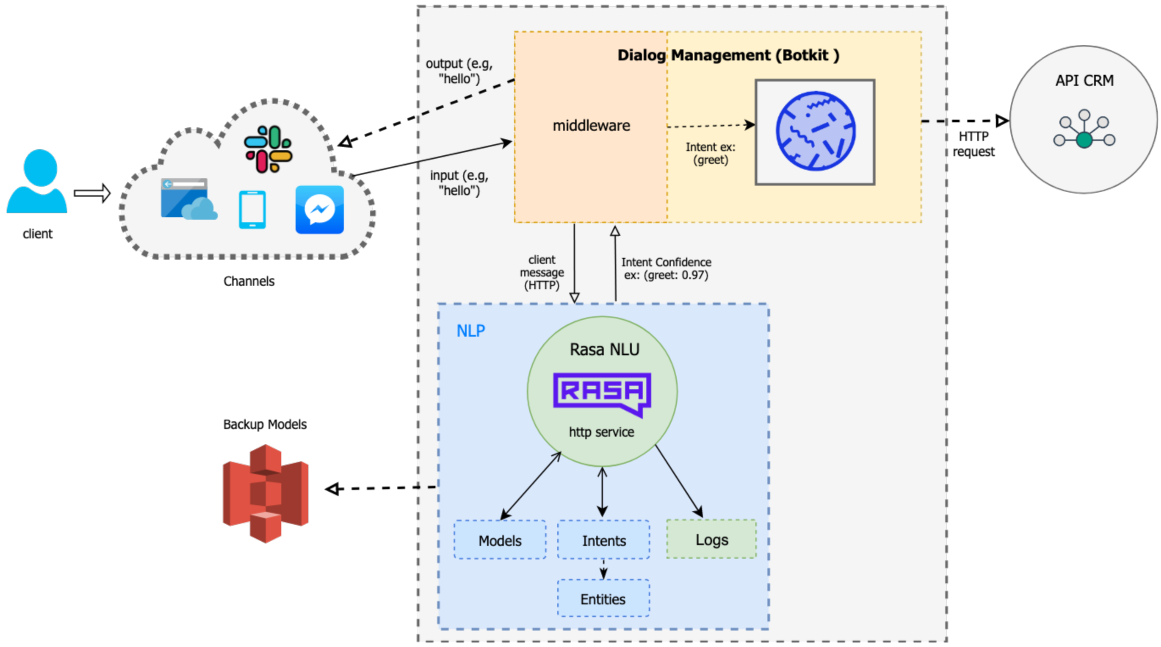

Figure 3.6: *Rule-based* system architecture

This was the first architecture which we implemented in order to obtain the first practical results of the conversational system.

As required, we could have multiple channels from where the user has been logged in, from the *CRM* itself or from another message platform available. Once the authenticated user sends a message, the *DM* implemented on the top of the *Botkit framework* would listen to it, specifically the *middleware* should redirect the received message text to the intended *NLP* component implemented with the *Rasa NLU*, this component would be responsible to convert the raw text into meaningful information back to the *Botkit*, with due intent and entities. Once we have the intent or entities in the *dialog management* system, we would need to code every possible flow of the conversation based on every intent.

The difficulties at this system version came when, for example, the user identifies correctly the first intent he wants, but at the middle of the conversation for the specified intent we could say some other intent which is out-of-scope in the scripted conversation by the developer, but in fact with a semantic correct context for the domain of the agent, it would originate an error or a wrong response. Now if we imagine the specific domain of the conversational system has a lot of intents scripted, how hard it should be to maintain and scale a correct script flow from one intent to another. Ideally, this *rule-based* approach would be the best solution to a *Q&A* agent or a very simple *chatbot* who would only resolve a few tasks.

3.3.2 *Hybrid DM* system architecture

We learned from the first architecture view at figure 3.6, that modelling a conversation system can become a difficult task, an interesting explanation about this case can be found in the research [27].

In fact, following the requirements of the project and the domain of the conversational system, we found easier to use a different approach than the *rule-based* system. We used the *Rasa Core* component as the *DM* system, which works on the top of a neural network, this means, controlling the conversation flow can be learned by the conversational system by itself.

For a better explanation of the *DM* architecture, we must observe the figure 3.7, which is the public available architecture of the *Rasa Core*.

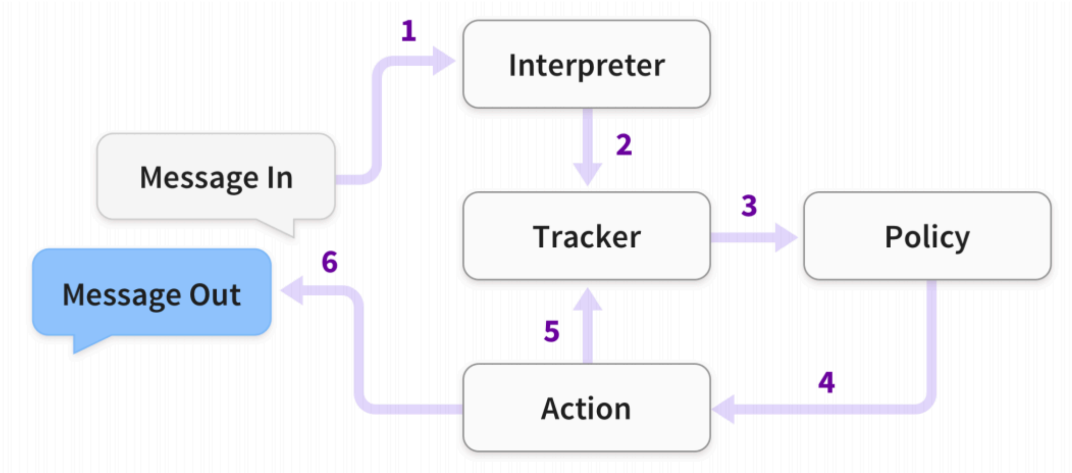

Figure 3.7: *Rasa Core DM* architecture, source: Rasa: Open Source Language Understanding and Dialogue Management [30]

In this architecture (figure 3.7), when the messages come in should be passed to the *NLU component* on the step 1, which is a aside component of the *DM*, in this case the *Rasa Core*, this component will extract the intent and entities or any other structured information [30].

After the step of the interpreter has been done, the Tracker comes in at the step 2, it will maintain the conversation state, and it will also receive a notification when a new message comes in [30].

Then we have the policy on the step 3, it will receives the current state at the moment of the tracker, also in the step 4, the policy is responsible to choose which action to take based on the current state of the tracker.

Finally, the action comes in, it should firstly be logged by the tracker and updates the current state as the figure 3.7 shows at the step 5, and finally in the step 6, it should execute the predicted action by the policy, this may include sending a response back to the user, or call an external service to fill specific slots in the state and then send the response with these slots filled [30].

If the predicted action fails for some reason (*api* error for example), the flow will back to the step 3 [30].

We think that hybrid *DM* option would be the ideal structure to implement in our case, since the domain of the conversation, it's almost completely about the final customer, so it should be important to transport the context from one message to the other, this kind of implementation become's hard to implement and especially to maintain in the architecture presented in the section 3.3.1.

3.4 Programming Languages and Libraries

In order to understand the context of the related programming languages and libraries choices, we will discuss briefly the available languages and libraries to develop the conversational system and the inherit structure, of course, these, were chosen based on the goals and the constraints already discussed in the section 1.5.

3.4.1 Technology Choices

Considering the context of the project and the company's software environment, initially has been chosen the framework *RASA NLU* which is an open source library for build a conversational software using the language *Python* [30], should not have any interference with the company tools and languages, since the language itself can be deployed in any server, indeed the mainly languages of the company is *PHP*, *.NET* and *Javascript*, however, since there's an *API* interface provided over *HTTP* protocol, this should not be a problem. The *RASA NLU* component should be able to handle the *Natural Language Understanding* of the conversation interface, in fact, this component uses *other's Python* libraries to provide a consistent *API*.

For the *Dialog Management*, the first tests were made by a simple rule-based *tool-kit* based in *Node.js* called *Botkit* [19], were this *DM* would interact with the *NLP (NlU)* component, however this rule-based system has some limitations which we will explain further ahead. The second option would be using the *Rasa Core* (DM) which is a *hybrid approach* already explained in section 2.4 that run's on the top of a *LSTM* neural network.

We ended up using the second option since it's best suited for the present domain of the conversational system, and also, it should easily to maintain and scale a *DM* which learns from the intent and current state of the conversations, instead of scripting all the flow of the conversation in an option like the *Botkit*, especially when the conversation can complicates on the context of the conversation agent domain, Ideally the *Botkit* rule-based approach would fit well in a simple Q&A conversation system which is not the case. The *Rasa Core DM* system is also implemented in *Python* programming language so it should be required to extend or script some of the actions at this same programming language.

3.4.2 The Open Source Machine Learning Framework, *Rasa*

Conversational systems are becoming pervasive as a basis for human computer interaction as we seek more natural ways to integrate automation into everyday life [30].

The modern open source libraries are held to a high standard of professionalism, and this extends to implementations of machine learning algorithms. There is a large amount of non-research work involved in maintaining a widely used project, also generally the code produced by research groups often falls short of expectations. The *Rasa NLU* and *Core* aim to bridge the gap between the research and the final application [30] .

The *Rasa's API* uses third-party libraries already with some recognition in the study area. The *scikit-learn* [46] focused on the consistent *API's* and the *Keras* [20] the high-level neural networks API (specially used in the *DM*). The *Rasa NLU* component provides some out of the box techniques used in the deep learning on the *NLP* processing explained in the section 2.2, like the *bag-of-words* technique. *Rasa's* language understanding and dialog management are fully decoupled, this means that each one of them can be used independently of the other [30].

For a better understanding of the *Rasa* ecosystem, we will detail a little bit of each component on the framework.

Each conversation session on *Rasa* has a tracker object, this *tracker object* is responsible to handle the *Dialog state* which is detailed explained in 2.4, this tracker stores the *slots* (that can be the entities of the *NLU* service) and the log of all events that led to the determined state of the conversation [30].

Then we have the actions, each iteration predicted by *Rasa Core*, it should also predict which action should take off from a predefined list of actions. Those actions can be a simple utterance, message, or an arbitrary function to execute. All of those actions executed are passed to the tracker instance, so it can update the relevant state of the conversation [30].

The *Natural Language Understanding* module, which is decoupled and can work independently of the *Rasa Core DM*, this module works as the interpreter of the conversation system, it should take care of the raw text provided by the user and convert it into meaningful information to the *DM*.

And finally the *Policies*, the *policy* is an element of the *DM* from the *Rasa Core* where the job is to select the next action to execute given the tracker object [30]. The default policy defined in the *Rasa Core* is the *KerasPolicy* which works at top of the *Keras* [20], and if we go depth, this policy should work at top of a *RNN* network, more precisely an *LSTM* network, where we already explained in section 2.2.4.

3.5 Extracting the Portuguese Words to the *NLU* Model

The starting point to get a good model for the *NLU* data will be the collecting data where we will train with the explained deep learning algorithms. In order to train some generic intents for the *NLU* model we started to write and fetch some text from the Portuguese words on the domain for our main intents. the main intents for our use case tests would be something like the following numeration.

3.5.1 Training Intents

Since we already had the scenarios of the main features for the first version of the conversational system described in section 3.1, we will now discuss the main identified intents at the conversational system, as well as the casual intents, these should take care of the conversational context to the main intents. The main intents are the following items.

- `check_balance_client`, the intent which would be responsible to identify the goal to get the current balance of a specified client.
- `check_last_task_client`, the intent that would classify the goal to fetch the latest task details for a specified client.
- `who_owes_more_money`, the intent that would classify the goal to get a top listing of client's debtors on the system.
- `won_by_opportunities`, the intent which would classify the goal to get the final value of the won opportunities for a specified client in a specified year.

Of course, we would need more generic intents for the simple casual conversation before reaching these goals, like the greeting intent.

So we created a list of casual intents which are related to the *Domain* of the conversational agent. This list will contain intents like (the intent name label should specify the contextual mean by itself), `greet`, `how_are_you`, `iam_fine`, `iam_not_fine`, `goodbye`, `who_are_you`, `none_of_these`, `what_you_can_do`, `thanks`, `affirm`. These intents (which we didn't enumerate all of them) should be able to handle the contextual conversation before and after reaching the specified goals of the user cases identified in section 3.1

To generate the data for the main intent goals and the casual one's we tried to write some of them, and then to be more efficiently and generally, we fetched some of the utterances for external people of the conversational system. Figure 3.8 summarizes the counts of extracting utterances we had for the model of our *NLU* component.

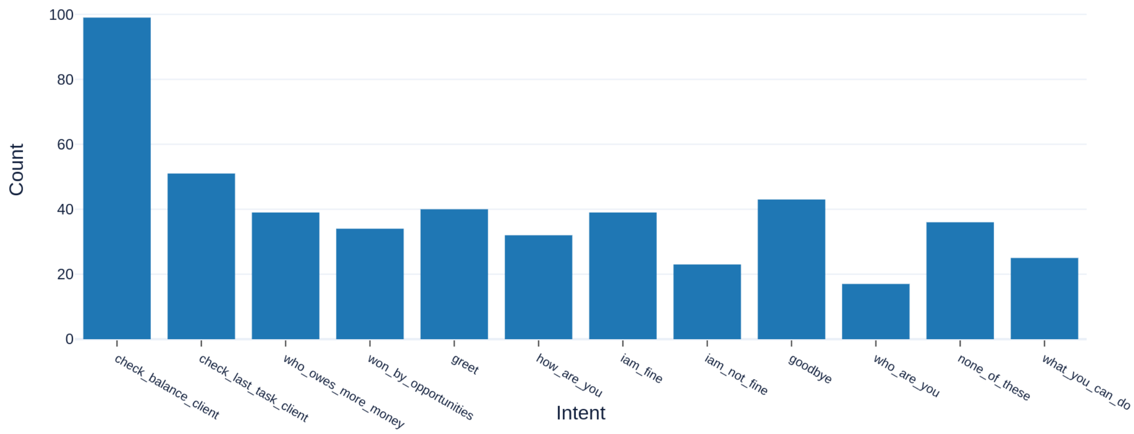

Figure 3.8: *Dataset intent* distribution

As can be clearly seen, the *dataset* does not have a lot of utterances for each intent identified in the list, however this was the start point to get the decent results for the model, in order to make it more accurate and robust logically we need to log the failed intent prediction results and re-train a new model with these failed intents in the appropriate context.

3.5.2 Pre-trained word vectors

It would be important to create a comparison from the *dataset* where has been generated from the raw text to the specific context of a more generic *dataset*, for that, we used a *pre-trained* word vector model from the *Spacy* [41] open-source software, this *pre-trained* model would have the advantage of the predicting for example some synonymous words in the utterances which are not trained in the first model, since the model already has a lot of corpus text from the *Wikipedia* on the Portuguese language, however, we suspect that can also have the disadvantage of being in trouble to clearly identify some specific intents for the *CRM* domain, since there's a vector of millions of words that can be in trouble to identify the context. The details of the full model can be found in [40].

3.6 Generating the Responses *(NLG)*

Due to the constraints outlined in section 1.5, the technique taken to generate responses is the retrieval-based model (already explained in section 2.5). Indeed, doing that allows the developers or maintainers of the conversational system, control properly his behaviour in answering to the messages. Also, since we are using the *Rasa Core DM* system, the responses can be generated in an instance of an action, more precisely for example the action can retrieve the data information from the external database to ensure that the response is meaningful to the user. These responses were implemented in the *MD* (Markdown) format, which is a lightweight markup language with plain text formatting syntax, would be the ideal case to main the possible responses.

For example, in order to respond to a casual greeting from the user, the domain had a few examples to pick once the action is called by the tracker of the *DM*, the following items are the generic Portuguese responses template for the greeting intent action.

- Hello, {greetPrefix} {username}!, all good?
- {greetPrefix} **{username}** , how are you?
- Hi **{username}**, it's everything ok?
- Hello, {greetPrefix}, how have you been?

We can identify in these possible responses the inclusion of variables which in the response action would be filled by the respective slots of the conversational system, these slots should be the entities filled with the conversations messages or the actions of the *DM*, the *DM* would pick one of these responses randomly.

3.7 Conversational Widget Implementation

Following the requirements of the section 1.5, would also be necessary to create a small widget to interact with the conversational system from the current *UI* of the software.

This small conversational widget would require an interface to interact with the conversational system which can also be called by a channel so we have implemented a `socket_channel` class in the language *Python* on the top of the conversational system to interact with the core system by a socket protocol.

3.7.1 Socket Channel Constraints

This class would be responsible to accept the connections from the *UI* widget of the interface from socket protocol, transporting the required details for a correct accessing to the *API* of the *CRM*, however, it should not be a good practice on transporting for example the *Authentication Token* from a socket protocol, respecting that, this class would be responsible only to send a user *id* of the connected user, and then, the conversational system would be responsible to do the appropriate authentication to use the *API* by itself.

Having this class implemented, this provides the possibility of interacting in other ways to the connected users on the system, like logging the connected users per day to estimate the usage of the system, and so on.

3.7.2 Widget User Interface

Since we implemented the socket channel on the conversational system, we would need to design the (*UI*) of the widget to write and receive the messages on the system, also, it would have to be necessarily volatile in order to be able to serve as many systems as needed.

Due to the demanding of the conversational system by itself, the ideal would be to use an active project with some of the implementation already done, and that's what we did, we used the project from *GitHub* called *rasa-webchat* [1], where it already implements the connection to the specific endpoint by a socket protocol.

The final *UI* widget interface can be visualized at figure 3.9, it should only be visible after opening from a small button on the bottom of the system, otherwise it should never intrude on the daily user actions over the system.

Figure 3.9: *Widget UI* to connect with conversational system

Deep down, we had to make some changes in the design of the channel to match the current layout of the *software FOXAIO*, but also, we had to implement some changes in the core code to implement the security architecture of this specific system, and also to interact with the current user interface, for example having the ability to open a modal box or navigate to a specific page as a response for the user's intent.

In order to connect to the correct way into to socket channel created previously explained in the section 3.7.1, this widget would be responsible to observe the user id of the authenticated user in the software *FOXAIO*, and then transporting it to the socket channel in the server side, so this class would be responsible to properly authenticate the user in the *API*. Also, we have created a custom component in the library *React JS* (a *Javascript* library to create user interfaces), if in some cases would be necessary to open something from the traditional user interface, this

[1] https://github.com/mrbot-ai/rasa-webchat

component would be responsible to transport the action in the final user interface, and then, deliver the most appropriate action/information for the user request (can be a simple text, or for example opening a modal box to show something particular for the case).

In figure 3.10, we can observe the conversational interface, responding with a custom component at the end, with the ability to open the latest task (in this case) from the traditional user interface.

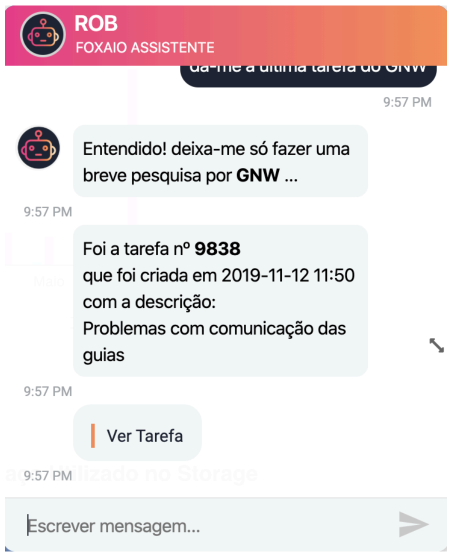

Figure 3.10: *Widget UI* button to interact with traditional *UI*

Chapter 4

Experiments and Results of the Conversational Interface

This chapter will focus on the experiments, results and performance performed on the conversational system modules as well as the results obtained from using two different techniques and methods explained in the previous chapters.

More precisely, we will experiment two different methods for the intent classification, a pre-trained embedding approach for the Portuguese words, and the *rasa tensorflow* embedding which trains the words from the scratch, wherein a high-view it shows more acceptable results in this particular domain of the conversational system.

4.1 Intent Classification

The first problem to solve is the intent classification, since it should be from *Portuguese* text, it would be necessary to tuning the best as possible, it's necessary to test with the different methods or techniques.

4.1.1 *Preprocessing*

Before the actual training of the data and due to their design, generally the textual data cannot be fed directly into a neural network, they require an extra step to *preprocessing.* In depth they need to be transformed into sequences of integers with the same length. The *preprocessing* steps are listed below.

- Turn all characters to lowercase
- Filter the insignificant punctuation characters. (e.g, "?!,;..)
- Tokenize the utterances by using words as list of tokens.
- Convert the tokens into word vectors, at the end of this step we will have a list of numbers only.

At the end of these steps the result obtained will be suitable to train into the *machine learning* algorithm.

4.1.2 Performance Metrics

To achieve or identify the main problems/errors of the *NLU* component in the conversational interface, logically would be necessary to quantify the quality of the training data. Also, in order to evaluate the model performance would be necessary to obtain more utterances for each intent from real people who are in the context with the domain of the conversational system, so we made some online surveys to trying to fetch the test *dataset*, we achieved something like 25/35 utterances for each intent, of course, would be perfect to test with a bigger number of utterances in each intent on the *dataset*, but since the domain is specific we think these utterances would be enough to identify the problems and compare the solutions.

4.1.2.1 Pre-trained embedding approach with *Spacy Portuguese Model*

Since we wanted to compare more than one approach to identify the best classifier pipeline to our *NLU* component, we tested the test *datatest* with the *pre-trained* vector approach from the model which can be found in [40].

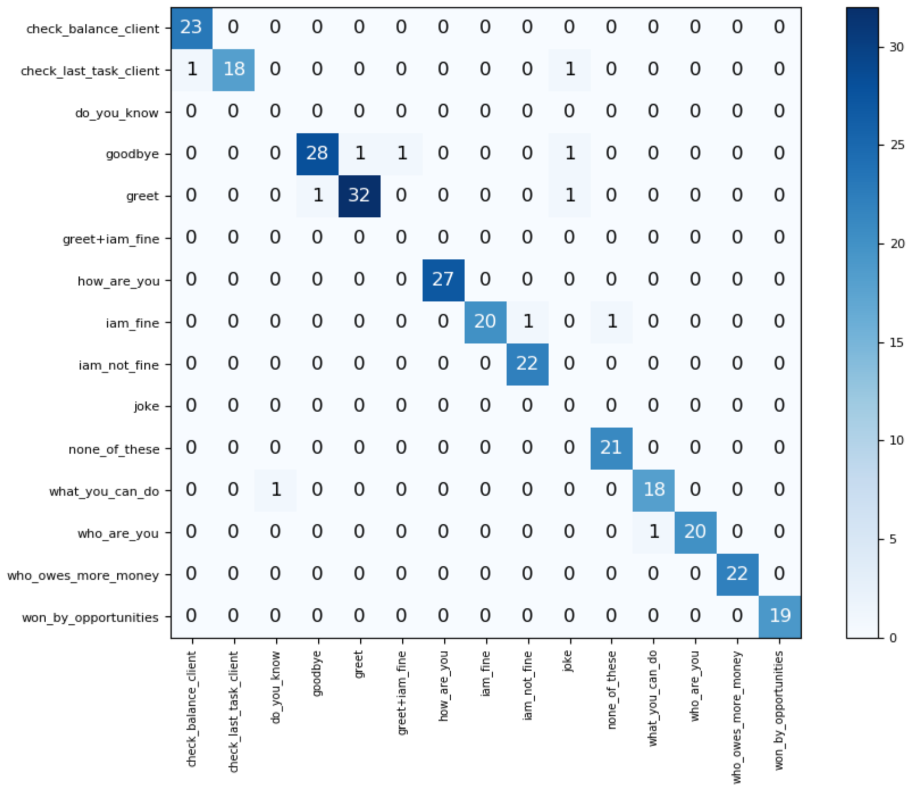

Figure 4.1: Confusion Matrix to the pre-trained embedding approach

If we observe the figure 4.1, the classifier seems to correctly identify the intents for the test *dataset*, however, if we observe it more precisely, we can figure out some misses on some intents, also some of trained intents on the model (`do_you_know`, `greet+iam_fine`, `joke`) appears without an utterance from the test *dataset* which it defines some other intent was incorrectly predicted. Overall, this approach is not perfect, but looks viable and robust enough to work with the values obtained from the confusion matrix, it would be interesting in fact, in fetching more results and test it out.

Table 4.1, shows the values of *precision*, *recall* and *f1-score* obtained for each intent used on the test *dataset* using this current approach.

intent	**precision**	**recall**	**f1-score**	**support**
`goodbye`	0.965	0.933	0.949	30
`greet`	0.969	0.969	0.969	33
`none_of_these`	0.954	1.0	0.976	22
`what_you_can_do`	0.947	0.947	0.947	19
`iam_fine`	1.0	0.909	0.952	22
`check_balance_client`	0.958	1.0	0.978	23
`won_by_opportunities`	1.0	1.0	1.0	19
`who_are_you`	1.0	0.952	0.975	21
`who_owes_more_money`	1.0	1.0	1.0	22
`check_last_task_client`	1.0	0.9	0.947	20
`how_are_you`	1.0	1.0	1.0	27
`iam_not_fine`	0.956	1.0	0.977	22
micro average	0.978	0.967	0.972	279
macro average	0.979	0.967	0.972	279

Table 4.1: *Metrics* values to the pre-trained embedding approach

If we observe in detail the table 4.1, these values are in fact very accurate, however, to make the solution totally viable, we should now observe the confidence levels of the intents, which will be introduced in figure 4.2.

However, despite the good results on the intent confusion matrix for the *pre-trained embedding* approach, if we evaluate the confidence on the distribution intents at the test *dataset*, as we can observe at the figure 4.2, it‘s visually noticeable that the confidence is not so good as we already expected at the section 3.5.2, this is because there‘s a lot of words already pre-trained from the *Spacy Model* where will be confused with some words that we expect to be meaningful on this specific context.

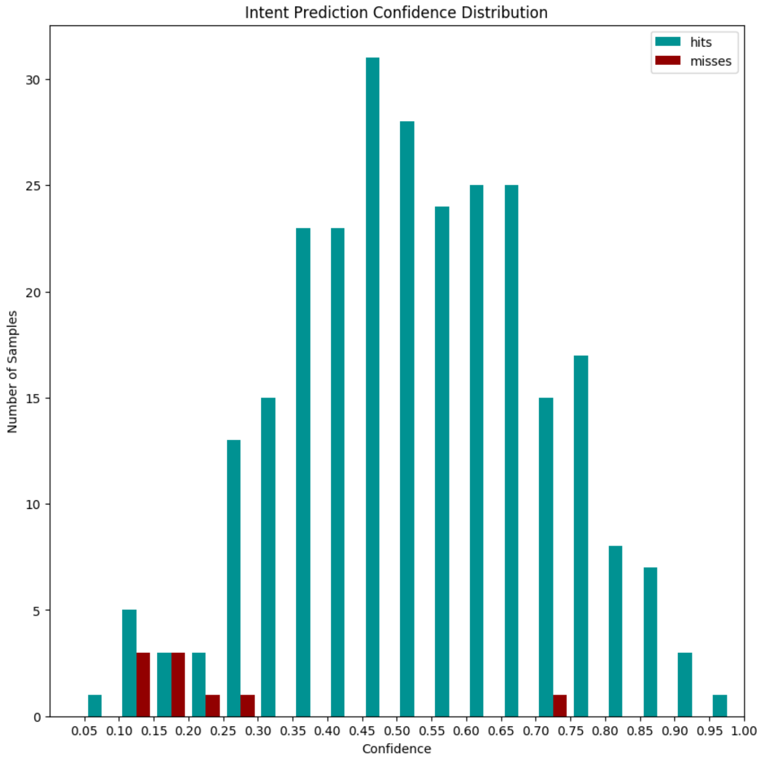

Figure 4.2: *Histogram Confidence Distribution* to the *pre-trained embedding* approach

In a real word case would be very dangerous to identify an intent with the level of confidence around the 0.5 (50%), this could generate an out of the context conversation, and also there are a lot of misses on predicting the intent correctly. So if we define the threshold for a fallback action over the 0.7/0.8 (should be the action to take when the *NLU* component is not able to identify correctly the intent defined by the threshold on the system), this approach should not be so good for our conversational system domain.

4.1.2.2 *Rasa tensorflow* embedding

One of the main advantages on using this technique, the fact that is inherently language independent and it's not reliant on good word embedding for a certain language [45], also it's adoptable on the specific domain since it trains the words from the scratch, we can expect good results from the current context of the conversational system, however we can also expect some gaps once the user says some new word that is not trained on the current *dataset*.

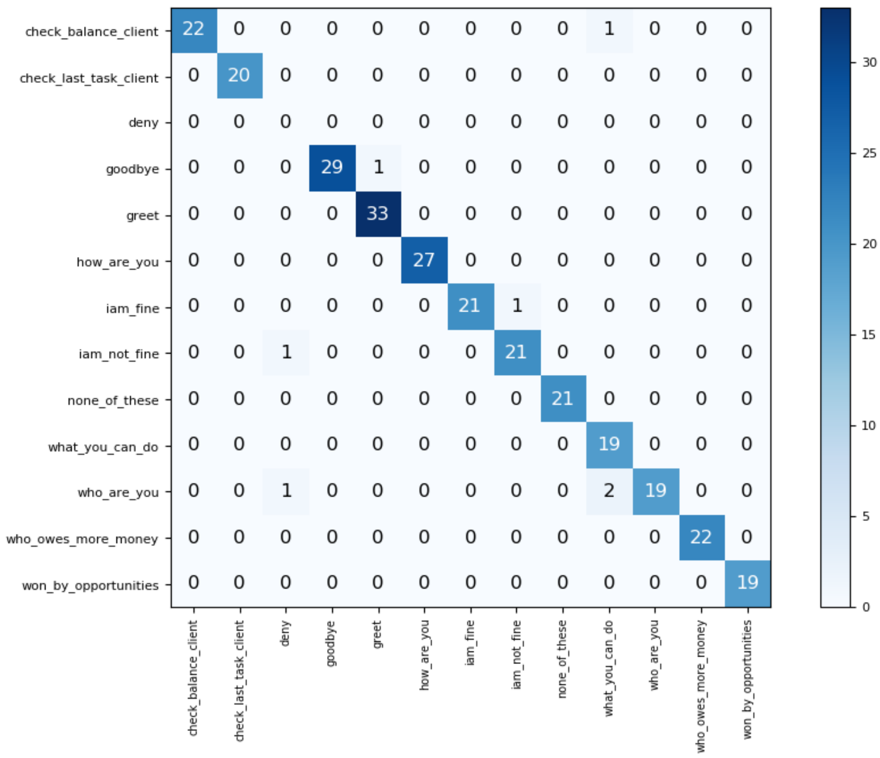

Figure 4.3: *Confusion Matrix* to the *rasa tensorflow* embedding approach

Observing the figure 4.3, again the classifier presents good results in identifying the correct intents from the test *dataset*, in fact, it also looks more accurate than the *pre-trained model* approach from the figure 4.1 where only one intent who are not on the test *dataset* appears (`deny`). Overall we expect to perform well on the real use case for this specific domain of the conversational agent.

The presented table in 4.2 shows the values of *precision*, *recall* and *f1-score* obtained for each intent used on the test *dataset*.

intent	**precision**	**recall**	**f1-score**	**support**
`goodbye`	1.0	0.966	0.983	30
`greet`	0.970	1.0	0.985	33
`none_of_these`	1.0	1.0	1.0	22
`what_you_can_do`	0.863	1.0	0.926	19
`iam_fine`	1.0	0.954	0.976	22
`check_balance_client`	1.0	0.956	0.977	23
`won_by_opportunities`	1.0	1.0	1.0	19
`who_are_you`	1.0	0.904	0.950	21
`who_owes_more_money`	1.0	1.0	1.0	22
`check_last_task_client`	1.0	1.0	1.0	20
`how_are_you`	1.0	1.0	1.0	27
`iam_not_fine`	0.954	0.954	0.954	22
micro average	0.982	0.978	0.980	279
macro average	0.982	0.978	0.979	279

Table 4.2: *Metrics* values to the *rasa tensorflow* approach

Observing in depth the table 4.2 the values are also pretty decent from these metrics, in fact, if we compare to the table of the pre-trained model approach in 4.1, we can notice a slight improvement.

Observing the figure 4.4, we can detail a noticeable improvement over the results obtained from the *pre-trained model* approach in the 4.2, almost all of the intents from the test *dataset* were predicted over 0.85 (85%) of confidence, which is a satisfactory accurate value for a real use case.

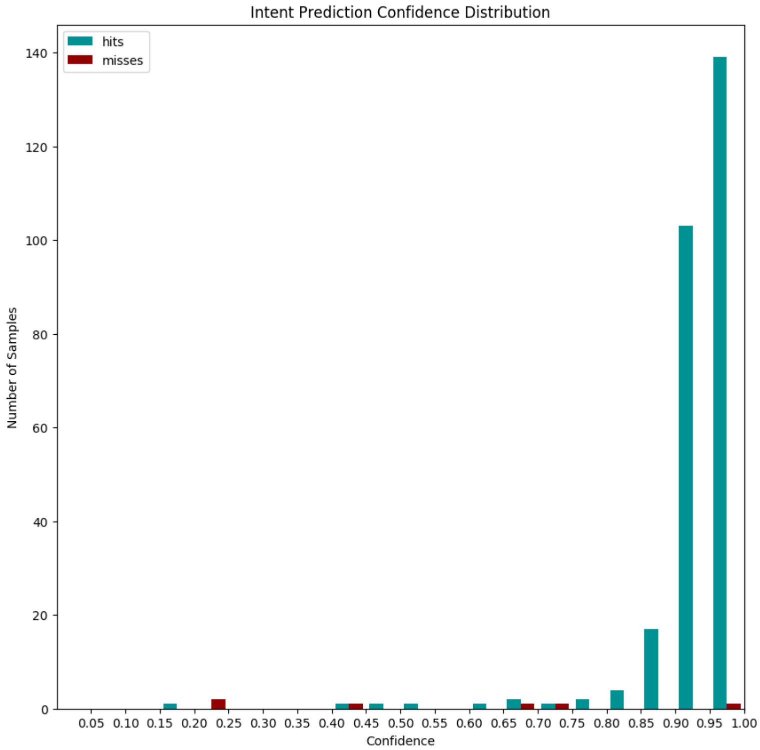

Figure 4.4: *Histogram Confidence Distribution* to the *rasa tensorflow embedding* approach

It's important to refer that all of the utterances obtained from the surveys to origin the test *dataset* were people that are in the context of the conversation system domain (*CRM*), the disadvantage in using this approach would be the possibility of failing some specific words which are not present on the training *dataset* yet, this means this model can contain in some cases a gap of *overfitting* where it would be necessary to train with more data, in fact, we can observe some fails with intents in the histogram, falling below 50%, we expect these failed intents increasing if we manage to get some out of the context people of a *CRM* software, interacting to this particular case.

However, in this particular case, it's suggested this approach would be the most robust and appropriate classifier model to be used on the real conversational interface.

4.1.2.3 Comparing the *pre-trained* embedding and the *rasa tensorflow* embedding approachs

In figure 4.5, we observe the *F1-Score* graph for the intents classification (explained in the section 2.3), the process to make this graph was the following, we create/split a new test *dataset* from the original training *dataset* five times (0%, 25%, 50%, 75%, 90%) and then we train multiple times each pipeline with the percentage of data excluded from the training *dataset*. Finally we get the *F1-Score* of each exclusion percentage recorded.

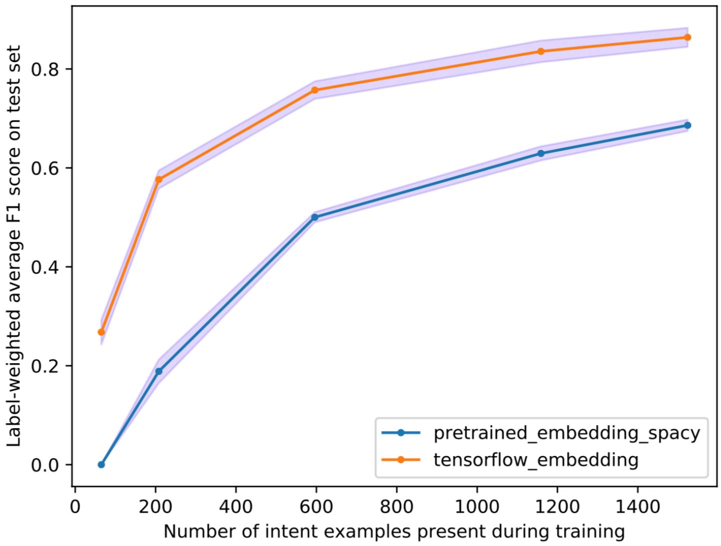

Figure 4.5: Graph with *F1-Score* of the *pre-trained spacy model vs rasa tensorflow* embedding

If we observe the detail in figure 4.5, obviously increasing the percentage of the data training from the split data the *F1-Score* will also increase gradually, however, it's clearly visible that the *rasa tensorflow* embedding gets better results with this same growth than the pre-trained *bag-of-words* approach. Once again this affirms that the approach of the embedding words from the scratch is more exciting for this specific context domain and the Portuguese language of the conversational system.

4.2 Manual Testing

Despite all the presented metrics in the previous section, the article *How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation* [34] as stated previously, mention the performance metrics do not tell the effective story when it comes to the conversational agents.

Thus, In order to make sure the conversational system exhibits the expected behaviour from the designed use cases at section 3.1 and using the widget created which is detailed explained in section 3.7.2 as the *mockup* environment testing it would be possible to test the actual conversational system, to simulate the authenticated user on the *CRM* software, we created a new test user on the system with some dummy data to simulate those use cases.

In the following sections we expose some of the conversations that we had with the conversational system implemented.

4.2.1 Test getting a specific client balance conversation flow

The following conversations are transcripts exchanges between the user and the conversational agent in the widget *mockup* environment trying to get a balance from a specific entity in the context.

```
USER: Olá

CHATBOT: Olá Miguel Costa !, Bom dia, está tudo bem contigo?

USER: está sim tudo bem, e contigo?

CHATBOT: Ainda bem que estás bom, comigo também está tudo ótimo!

CHATBOT: Então, em que posso ser útil?

USER: poderias me dar a conta corrente do cliente Mapex ?

CHATBOT: Estou só a fazer uma breve pesquisa na tua base de dados por Mapex ...
```

CHATBOT: Encontrei várias empresas parecidas.. |MAPEX S.A| |MAPEX ANGOLA, LDA|

USER: MAPEX S.A

CHATBOT: A atual conta corrente da entidade MAPEX, S.A. é de 6542.99 euros

USER: Obrigado

CHATBOT: Sem problema, estou cá para isto..

In this scenario the user starts the conversation with a `greet` intent, then the conversational system automatically knows who he is, since the user id has been shared from the socket protocol, so he can be authenticated upon the *CRM* software and obtain more details about him, important to highlight when we correctly asks for a balance of the entity *MAPEX*, the bot will search for this keyword on the system, then gives an answer with buttons to choose the correct wanted entity, however, the user isn't bound to pick from the given buttons, we can write the client name more precisely and then the system with pick the best option, or in anoother way, can just stop the conversation by giving the intent `none_of_these` in raw text.

Finally and after the system identifying the correctly entity that the client asked for, will get the client balance from the *CRM* and give it at the next response.

4.2.2 Test getting the won value by opportunities to a specific client from a specific year conversation flow

The next conversation was made with the goal to fetch the won value based on the opportunities to a specific entity on the *CRM* system.

USER: olá rob, tudo bem?

CHATBOT: Olá, Boa tarde, eu estou ótimo, e tu como tens andado?

USER: está tudo otimo comigo, obrigado :)

CHATBOT: Ainda bem que estás bem! Fico satisfeito por saber..

USER: da-me o valor ganho em oportunidades de vendas para a entidade ARP Advogados em 2017

CHATBOT: Ok, deixa-me fazer uma pesquisa rápida por ARP Advogados ...

CHATBOT: Ao cliente ARP Advogados - Lda foi vendido cerca de 3,867.55 euros em oportunidades de negócio calculado no ano de 2017

USER: muito obrigado rob :)

CHATBOT: Não tens de quê Miguel Costa, é um prazer ajudar

In this case the user start once again with the `greet` intent and then has a normal conversation until the user reach's the main goal, asking for the values won by opportunities to a specific entity in the year of 2017, the conversational system search's for the provided keyword by the user and identities, there's only one entity with this keyword, so he fetches automatically the won value by the software *API* and generate the appropriate response with the data obtained from the service provider.

Once again we had a decent conversation flow with a beginning, middle and an end.

4.2.3 Test getting the the most debtors entities conversation flow

Here the user tried to get the most debtors entities on the system available, also, he made some social conversation before that.

USER: boa tarde, tudo bem?

CHATBOT: Boa tarde Miguel Costa , está tudo bem, obrigado, e contigo?

USER: comigo também está tudo..

CHATBOT: Fico feliz por saber que estás bem!

CHATBOT: Poderei ser útil em alguma coisa? em que posso ajudar?

```
USER: porque te chamas ROB?

CHATBOT: Boa questão, eu na verdade não gosto muito do meu nome, mas foi o nome
que me deram.

USER: diz-me os 3 clientes que mais me devem dinheiro..

CHATBOT: Ora aqui tens a lista com as top 3 entidades mais devedoras do momento

CHATBOT: 1. MAPEX, S.A. - 6542.99
         2. MAGICPRINT - IMPRESSORAS, LDA - 5595.81
         3. Publicidades João - LDA - 5535.00
```

As expected, the *chatbot* correctly identifies every intent at the flow of this conversation and responds conveniently to the user identify the intent `who_owes_more_money` where the conversational system will fetch the most debtors on the system for this specific user based on the top rank number provided if exists, otherwise he will generally fetch the top 5 entities by default.

Overall, it shows a decent conversation between the user and the *chatbot*, of course the problems will come when the user starts to write something randomly where the system cannot understand, but we don't get it as a big problem, since we only want to provide some complementary method to interact with the *CRM* software, in short the goal of this specific conversational system isn't driven to talk socially, but in a business context.

However, this does not invalidate the possibility of having a social conversation with the *chatbot* without a main purpose over the *FOXAIO* system, as we can observe in the next section.

4.2.4 Test social conversation flow

In this conversation, the user talks to the *chatbot* without any specific purpose in getting more details from the *CRM* system.

```
USER: Olá, boa tarde, tudo bem?
```

```
CHATBOT: Olá, Boa tarde Miguel Costa!, Eu estou bem e tu como estás?

USER: também estou bem obrigado :)

CHATBOT: Ainda bem que estás bem! Fico satisfeito por saber..

CHATBOT: Então, em que posso ser útil?

USER: qual o teu nome?

CHATBOT: Eu sou o Rob, prazer Miguel Costa

USER: és capaz de me fazer rir?

CHATBOT: Qual a diferença entre um sapo e um príncipe? As cinco cervejas que a
princesa tomou.

USER: meh, não teve grande piada ..

CHATBOT: Bem.. se achas isso, eu posso tentar-te dizer outra piada..

USER: diz lá outra piada então

CHATBOT: O que não pode faltar no meio da festa? A letra s.
```

As we see, we still had a decent conversation based on some social intents trained on the conversational system, however the focus still on the business goals over the social *chit-chat* conversations.

Overall, within these experiments, we can see the *chatbot* performing well in identifying the user intents defined on the use cases in section 3.1. It even displays a somewhat robustness when he is not clear enough about what he should do, as we detailed on the conversation 4.2.1, in this particular case the *chatbot* didn't had enough certain in which entity he should've picked, so he asks the user with a list of possible entities to pick one.

Chapter 5

Measuring the Optimisation of the User Experience

This chapter will analyze the possible optimal measure of the user experience over the software described by the conversational interface, we will discuss the obtained results based on the available tools to quantify the optimisation of *UX*.

5.1 Considerations to measure the *UX*

In order to analyze if the conversational interface implemented had potential to effectively optimise the overall user experience of the *FOXAIO* software in that specific case we had to gather some user details and feedback to feed some metrics, and also logging some actions on the conversational agent, important to mention, we managed to log these actions for approximately a month and a half. Since the software *FOXAIO*, does not have a large base of active users yet, currently we have like 50 active users daily, exposing that, the main objective is in the first instance, expose how to apply the use of the *HEART* framework discussed in the section 2.7.2, and then present the obtained results of this user-centered *framework*, however, these results shouldn't be considered meaningfully until the *software* reach a larger base of active users. In this specific case would be better to use another tool to measure the *user experience* with a small sample of data, applying that we had implemented the *System Usability Scale (SUS)* already discussed at section 2.7.3 which is focused on the usability and the goal achievement. Since we had a few use cases implemented, it should be reliable to analyze the feedback of the users, achieving these goals in the conversational system, in comparison to the regular *UI* interface of the *software*.

5.2 *HEART* Framework Case

The *HEART* framework is a set of user-centered metrics, itťs accurate to identify the quality of the user experience and help the product manager or the teams to measure the impact of the *UX* changes [4].

5.2.1 Setting Goals, Signals and Metrics

To understand correctly the impact of the changes that conversational interface brought to the current stage of the software, it's important to identify clearly the goals of that feature on that case, since the most important aspect of this particular situation is the ease of the use comparing to the traditional *UI* process, we will focus on the task success of the identified features at the section 3.1.

The next step was defining signals, every goal has related user actions, mapping the goals for the intended actions can help to understand if the respective feature is doing well. Was also a good moment to think about data collection (surveys). Those signals should closely correspond to their respective goals.

Finally, the last stage of implementing the *HEART* framework over this specific feature, was the definition of the metrics, it should transform the signals into measurement scales, where we can observe for some time, in this case, it was necessary to log all the user actions over the conversational system.

Corresponding to this process, the following table presented in 5.1 is intended to plot the *HEART* categories against the respective Goals-Signal-Metrics.

	Goals	**Signals**	**Metrics**
Happiness	Users find the *chatbot* helpful and easy to use	• The level of user satisfaction	• Ratings submitted by surveys
Engagement	The *chatbot* must facilitate the execution of the identified features	• Spending more time on the *chatbot*	• Calculate the average session length
Adoption	New users see the the value in the *chatbot*	• Increasing number of users launching the conversational interface	• How many new users decided to check a customer balance within 7 days compared to the number of all users
Retention	Users keep talking to the *chatbot*	• The number of users who talks to the *chatbot*	• How many users that checked a customer balance within 7 days decided to check it again on the next 7 days
Task Success	Users complete the identified features goals easily	• The number of errors on achieving the features	• The number of displayed error messages compared to the success messages

Table 5.1: *HEART framework* categories plot against goals-signal-metrics

5.2.2 Applying Metrics and Analyzing Results

After defining the methods and tools to obtain the results over the *HEART* metrics, we will now present how we gathered the data for the metrics and analyze the results that was possible to obtain.

5.2.2.1 Results

In this section, we will present the meaningfully visual results (charts) for all the metrics we explained in the section 5.2.1, it should work as an introduction for the explanation we will have further ahead.

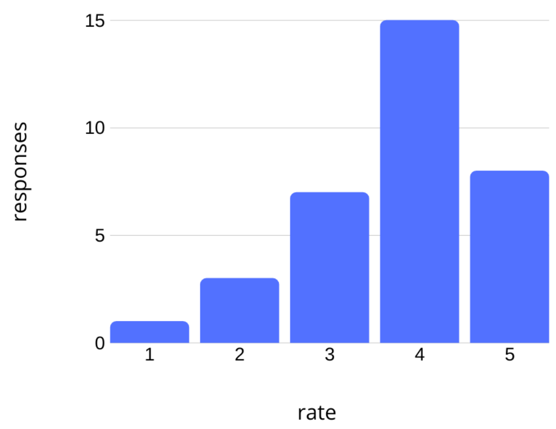

Figure 5.1: Happiness Metric survey question 1 (Did you find the *chatbot* easy to use?)

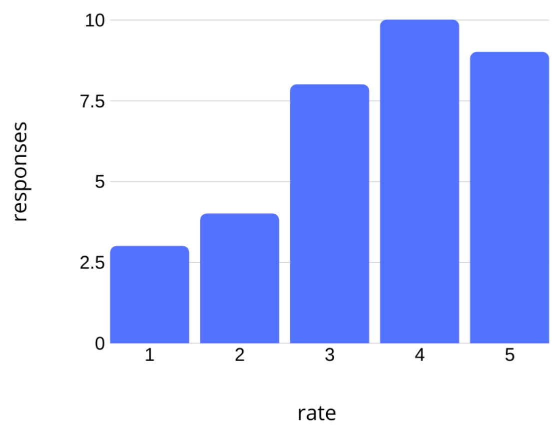

Figure 5.2: Happiness Metric survey question 2 (Did the *chatbot* help you to perform the requested tasks?)

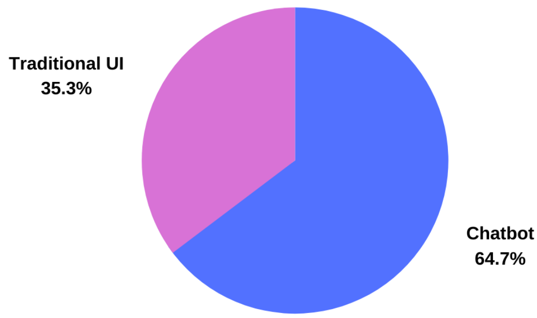

Figure 5.3: Happiness Metric survey question 3 (At performing the requested tasks, would you prefer to use the *chatbot* or the traditional interface ?)

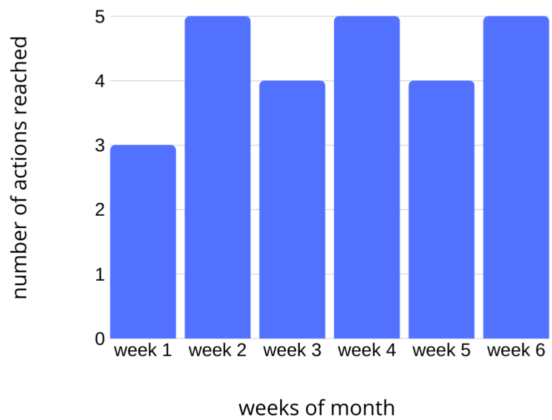

Figure 5.4: Adoption Metric Results Graph

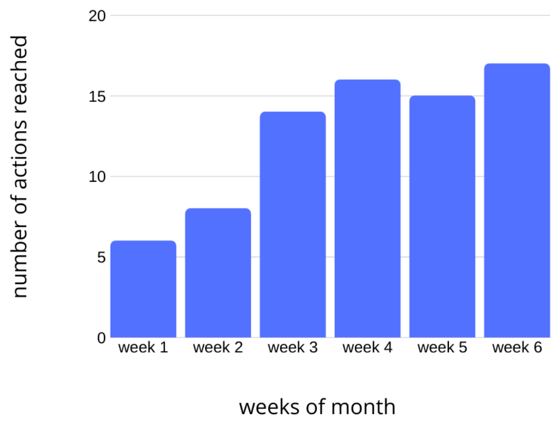

Figure 5.5: Retention Metric Results Graph

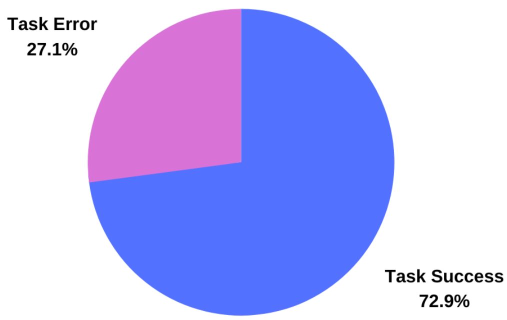

Figure 5.6: Task Success Metric Chart

5.2.2.2 Applying the *HEART* metrics

We will now explain how we managed to get the results from the section 5.2.2.1, applying all the metrics set in the section 5.2.1, to analyze properly the results, we had **34 persons** to test out the conversational system, in order to obtain the necessary feedback to fill these metrics.

In order to analyze the **happiness** metric, was created a small survey with main objective to identify the satisfaction level, some of them in a *Likert* scale and others with a short text or multiple choice, with the following questions:

1. Did you find the *chatbot* easy to use?
2. Did the *chatbot* help you to perform the requested tasks?
3. At performing the requested tasks, would you prefer to use the *chatbot* or the traditional interface ?
4. Did you find it hard to perform the requested tasks?, if yes, why?
5. What other skills would you like the *chatbot* to do?

On the first question "Did you find the *chatbot* easy to use?" the responses were generated by a *Likert* scale from 1 to 5.

Observing the results in figure 5.1, it's clearly visible the persons in the study are pleased with the ease of the use of the conversational interface as expected.

For the question "Did the *chatbot* help you to perform the requested tasks", the results of the survey are represented in figure 5.2.

The results elucidate the people who used and answered the survey the conversational interface were satisfied with their overall performance on that first version deployed.

The results for the question "At performing the requested tasks, would you prefer to use the *chatbot* or the traditional interface ?" can be analyzed in the plot graph at the figure 5.3, and can be visually detected the persons prefer to execute the tasks in the conversational interface rather than the traditional user interface.

For the question "Did you find it hard to perform the requested tasks?, if yes, why?" we obtained **45.6** % of positive responses, where the problems fall mostly on responses like, "*he didn't recognize the company name, and had to ask me*", however, we had some responses detecting some hypothetical problems on intent classifier "*He didn't understand the way i asked my customer balance*". We still think the 45 % is not so bad for the first version of the *chatbot*.

Finally, in an optical way of perceiving what kind of abilities users want for the conversational interface and for the question "What other skills would you like the *chatbot* to do?" we can enumerate the most wanted skills for the users.

- I want the *chatbot* to send an email with the customer balance if i ask for it after obtaining it.

- I want to open a client sheep over the *bot*.
- I want to create a task or a business opportunity.

This can be a good point to continue developing the abilities of the conversational interface, detecting what people would like to see on a *chatbot* and corresponding with that, improving the overall experience of itself.

To evaluate the **engagement** metric, we considered for logging the session length on the conversational interface that meet the following conditions.

- The user sent at least 5 messages to the conversational system .
- The user strictly sent messages to the system with the goal to reach the provided abilities identified at 3.1.

Exposing that, to quantify the session length we created these conditions in the logging system, and if these same conditions were validated, we saved on an external database the user details and the start and end timestamp of the conversation, so we could calculate the overall session length.

We gathered around 100 entries in the database, we obtained an average session length of **236** seconds, which represent a session length around 4 minutes, we identify this number as a good start point, since the first version of the conversational system only provides a few abilities, however we expect this number will grow as the skills of the conversational interface grow. In the future, it should be important to compare and evaluate this number over the time.

To interpret the **adoption** metric on the conversation interface, we had to implement the logging system to save for an external database the actions of users with the current date and time.

The total users of the system who logged in within 30 days are around 70/80, the figure (5.4) shows how many new sessions on the system archived the goal of "get client balance" within 7 days that ran for a month and half.

Based on the number of users, we can interpret the users are keeping trying the conversational system which is good, it did not enter the trend of having a higher number at the beginning and then decreasing over the time.

Following the same method to retrieve the adoption results, we have implemented the that for the **retention** metric too, but for this one, we removed the condition of being a "new user" talking to the system, and added the condition of the user who reached the goal of getting a client balance, repeated the same goal on the following 7 days.

The following graph in 5.5 was the data stored by the logging class of the conversational interface in order to measure the same quantification, and has been stored for approximately a month and a half.

It's visible that the number of users using the conversational interface with the purpose of getting the information by talking is increasing easily over the time, it represents the users in some way actually find the abilities useful on their daily tasks, even when there are only a few abilities available.

Finally, to achieve the **task success** metric, it was pretty simple, we logged every successfully goal reached on the database, as well as the failed actions, we intended by a failed action when a user tried to reach the goal of the use cases available and the conversational interface was not able to identify that correctly, once the entity extraction failed, or even worse, the intent prediction failed, on the second scenario we had manually to detect these failings and re-train the model, the graph in figure 5.6 is identifying the task success ratio versus the error ratio stored by a month and a half.

It's noticeable the higher success ratio over the error ratio, however, we didn't measure the error ratio with the number of failed intents predicted by the social/casual conversations or some other messages with the goal to achieving other functionalities which are not available on that version of the conversational interface, still the number of errors are still high at our perspective, since half of the failed intents was because of incorrectly entity extraction, where the users for example, are spelling the name of a company with a lowercase letter, and we have like 90 % of the company names in our data training set for the entity extraction of a company name with a start uppercase letter.

5.3 System Usability Scale Case

For a different view and perspective of the user experience optimisation it would be good to analyze the usability of the system in the usertś perspective, thattś the main goal of the *SUS*, in a high view and from a single number, understand what the users feel about the usability.

Important to mention, in order to compare the conversational interface against the current user interface of the software, it was necessary to create the 10-item *Likert* scale for each version, obviously it was mentioned to the users, the questionnaire was targeted to the specific features created and identified in the section 3.1, and not for the all features at whole.

5.3.1 Interpreting the Results

To obtain the result of the *System Usability Scale (SUS)* tool, and remembering it should yield a single number representing the overall usability of the conversational interface, the following operations were made with the individual items on the scale, it was necessary to sum the score contribution of each item, noting each item's score contribution was ranged from 0 to 4, for the item's 1,3,5,7 and 9 the score contribution was the scale position minus 1. For the item's 2,4,6,8 and 10 the score was 5 minus the scale position, and finally multiplying the sum of scores by 2.5 it generated the overall value of the *SUS* [6].

5.3.2 Analyzing the Results

The following results were generated by the proposed *Likert* scale which has been filled by 29 persons, the current users who are familiar with the software, the questionnaire in cause can be found in the section 2.7.3.

The following table 5.2 represents the scores obtained against each question on the questionnaire implementing the proper interpretation which can be found in the article [6], higher is better in any question following these guidelines, to obtain the final score, every number of each row of the table was divided by the number of responses referring to the number within parentheses (obtaining the number between 0 and 4), to obtain the final result, every score was been normalized and summed, and finally multiplied by 2.5.

Question	*Chatbot* Score	*UI* Score
I think that I would like to use this system frequently	86 (2.97)	72 (2.48)
I found the system unnecessarily complex	73 (2.52)	71 (2.45)
I thought the system was easy to use	79 (2.72)	50 (1.72)
I think that I would need the support of a technical person to be able to use this system	84 (2.90)	61 (2.10)
I found the various functions in this system were well integrated	58 (2.00)	62 (2.13)
I thought there was too much inconsistency in this system	70 (2.41)	58 (2.00)
I would imagine that most people would learn to use this system very quickly	83 (2.86)	61 (2.10)
I found the system very cumbersome to use	74 (2.55)	71 (2.45)
I felt very confident using the system	77 (2.66)	66 (2.28)
I needed to learn a lot of things before I could get going with this system	89 (3.07)	65 (2.24)
Final Score	**70**	**58**

Table 5.2: *SUS* results table

In the *SUS* scale perspective, the *Chatbot* is clearly the winner, which has a result of 70, according to the study at [28] this number is above average, in the other case, the traditional user interface of the software scored 58 which is below the average, however, these scores should be only considered as a sample of the experience, since the results were only considered using the use cases on the study, expanding this to a complex use case scenario, it should probably be quite different.

5.4 Review of Results

In a review of the obtained results, we can detect some hypothetical improvement on the user experience, however, it should be highlighted the small number of active users on the software for this experiment, so it should not be considered by an effective affirmation, still the feedback obtained by the *HEART* framework and the *SUS* scale was good and promising in an overall perspective.

In fact, we believe these results and numbers obtained at the end could have several changes, if the conversational interface in the case had more capabilities and if the number of the active users in the experience were higher.

Still, we consider all of these methods applied in order to measure the user experience of the conversational system should be considered as an effective way to obtain the real user-centered feedback in chatting to a *chatbot*, and they should work for a higher sample of feedback data.

Chapter 6

Conclusion

This chapter will summarize the content of all previous chapters in order to recap what has been accomplished. It will highlight the most important steps of the project we had in order to have an operational conversational system, we will briefly detail the pointers for the future development of the subject.

6.1 Summary of Accomplished Work

After the study and review of the main techniques used on the development for the conversational systems, more precisely in the area of the *Natural Language Processing*, the subjective definition of the *chatbot* goal's has come, a set of the objectives and constraints from the company we worked with were laid down on the paper in the chapter 1.5. After that in the chapter 3 a scalable software solution was designed and presented as well with other possible architecture based on the *Dialog Management* system architecture, also the implementation of the widget on the current software *UI* was created and developed, that required the implementation of a new specific channel on the conversational system to receive the messages based on a socket protocol. Then in the chapter 4 we discussed the most robust techniques based on the chosen libraries to implement the intent classifier of the system, all decisions were made based on the results of the metrics that we have shown, however, and since the way we analyze the performance of a *chatbot* can be very subjective, we made some manual tests on the current *chatbot* system. Finally, in the chapter 5, we analyzed in a detailed view the results of the possible optimisation of the current *CRM* system with the conversational interface based on the *Google* framework *HEART*.

Looking back at the accomplished work, it can be said the actual developed solution is satisfying and also allows the easy continuous development if changes need to be performed, still, the solution is easy to deploy in any available server on the present year, since we only need a *Python* environment on the top of that which is available to the all *OS*. Furthermore, and also because of the *framework* used, *Rasa*, it has allowed possibly to extend to the other programming languages, for example a specific action on the conversational system can be redirected to any other external service (coded in *Java* or *PHP* for example) as long as it returns the required object for that. However, there might be a few cases where the *chatbot* might not behave in the best manner possible, so it's important for the future maintainers or developers to be aware of that.

We can also relate with a high view of the results obtained by applying the tools to measure the optimising of the user experience the conversational interface actually was received well by the active user's on the software used in this case, they actually found some improvements on their daily tasks using the *chatbot*, however, the current solution is not a panacea, several improvements can be made to obtain the most precise feedback as possible, and therefore, improving the *UX* in a better way.

6.2 Future Work

Since the presented solution will not resolve all the problems, several improvements could be made in order to boost the robustness of that solution. The following list, which will act more like a brainstorming than an actual *roadmap* (since there are no extensive research's to confirm they're indeed viable in that specific case), will highlight some of the possible improvements.

- Restructure of the software architecture in order to identify the user language authenticated to the system and then redirect it to the matched *NLP* covered language.
- Possibility to extend the current components in the conversational system, for example, a sentiment analyzer can be implemented to identify the positive or negative responses from the authenticated user.

- Extend the actual channel implemented to other Message Platforms available, like *Slack*, *Facebook Messenger*, however, in that case, it would require somehow an authentication method on the *CRM* system, to fetch the meaningful data from the system.
- If the user is unable to clarify itself, instead of going to a fallback action, before that, the *chatbot* could suggest potential problems to the user in the form of a selection list with common problems.
- Request a human agent in the case that the user gets frustrated from the conversation.
- Exploring other methods and tools to evaluate the user experience improvement, like applying the famous *A/B* tests.

6.3 Final Thoughts

Some of the enthusiasts in this particular area, call the *chatbot* system as the *New Apps*, this address a lot of responsibility on their shoulders, In the 2016 the *chatbots* was considerate a new trend, but a little time after this trend has faded and still living in the background. In all of the cases, there is still a long way to go for conversational systems before they can be used completely untouched by the human hands.

At this specific context domain, the *CRM* systems, the use of a *chatbot* to complement the way a user interact to the software can be very exciting, the example of that is the attention of the most recognized *CRM* software *Salesforce* is given on that [5].

The future holds promising results in the field of the *Natural Language Processing*, new methods are born (like the *rasa tensorflow embedding* created in the past year and used in the implementation) and older methods become even better, in fact the future of the success or failure upon the *chatbot* it's depended on that field. However, the growth of the *chatbot's* left a mark on the History, and also in the mind of the consumers, it might be entirely possible in a near future to see the roles reversed between the *bots* and humans, we might see one day a bot interaction with a human using the human language in order to take care of the daily tasks of human's life, an excited example of that can be found at [43].

In any case, after the difficulties I have found in working with such technology, I am still optimistic of these types of implementations and developments and how they would shape the future of our societies. I think we will see one day this kind of technology indispensable for the human's life, and obviously, substantially improving the overall user experience.

References

[1] Vincent Van Asch. Macro-and micro-averaged evaluation measures [[basic draft]], 2013. URL: `https://pdfs.semanticscholar.org/1d10/6a2730801b6210a67f7622e4d192bb309303.pdf`.

[2] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. On the difficulty of training recurrent neural networks, 2012. URL: `https://arxiv.org/abs/1211.5063`, `arXiv:1211.5063`.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994. `doi:10.1109/72.279181`.

[4] Emily Bonnie. How to use the google heart framework to measure and improve your apps ux, 2018. [Online accessed in 21 of August, 2019]. URL: `https://clevertap.com/blog/google-heart-framework/`.

[5] James Boutel. Einstein chatbots build your own automated salesforce robot!, 2018. [Online; accessed in 15 of August, 2019]. URL: `https://www.salesforceben.com/einstein-chatbots-build-your-own-automated-salesforce-robot/`.

[6] John Brooke. Sus - a quick and dirty usability scale, 1986. URL: `https://hell.meiert.org/core/pdf/sus.pdf`.

[7] Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cambria. Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709, 2017. URL: `http://arxiv.org/abs/1708.02709`, `arXiv:1708.02709`.

[8] Ketakee Nimavat, Tushar Champaneria. Chatbots: An overview types, architecture, tools and future possibilities. *IJSRD - International Journal for Scientific Research Development*, 5, 2017. URL: https://www.researchgate.net/publication/320307269_Chatbots_An_overview_Types_Architecture_Tools_and_Future_Possibilities.

[9] Chatbots.org. Consumers say no to chatbot silos in us and uk survey. [Online; accessed in 20 of January, 2019]. URL: https://www.chatbots.org/images/news/chatbot_survey_2018.pdf.

[10] Katie Costello. Gartner forecasts worldwide public cloud revenue to grow 17.5 percent in 2019. [Online; accessed in 13 of January, 2019]. URL: https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL: https://arxiv.org/abs/1301.3781.

[12] Erik Devaney. The 2018 state of chatbots report: How chatbots are reshaping online experiences. [Online; accessed in 10 of January, 2019]. URL: https://www.drift.com/blog/chatbots-report/.

[13] Jakob Nielsen Don Normal. The definition of user experience (ux). [Online; accessed in 15 of December, 2018]. URL: https://www.nngroup.com/articles/definition-user-experience/.

[14] freeCodeCamp. Oauth2 protocol. [Online; accessed in 6 of June, 2019]. URL: https://guide.freecodecamp.org/security/oauth2/.

[15] Kerry Rodden, Hilary Hutchinson, Xin Fu. Measuring the user experience on a large scale: User-centered metrics for web applications. In *Proceedings of CHI 2010*, 2010. URL: http://www.rodden.org/kerry/heart-metrics-chi2010.pdf.

[16] Mateusz Hapon. Ai chatbot: How artificial intelligence and machine learning can improve your chatbot, 2018. [Online; ac-

cessed in 13 June, 2019]. URL: https://www.netguru.com/blog/ai-chatbot-how-artificial-intelligence-and-machine-learning-can-improve-your-chatbot.

[17] J. Harms, P. Kucherbaev, A. Bozzon, G. Houben. Approaches for dialog management in conversational agents. *IEEE Internet Computing*, 23(2):13–22, March 2019. doi:10.1109/MIC.2018.2881519.

[18] Lian Meng, Minlie Huang. Dialogue intent classification with long short-term memory networks, 2018. URL: http://tcci.ccf.org.cn/conference/2017/papers/1158.pdf.

[19] XOXCO Inc. Botkit: Building blocks for building bots. [Online; accessed in 1 of May, 2019]. URL: https://botkit.ai/.

[20] Keras. Keras: The python deep learning library. [Online; accessed in 3 May, 2019]. URL: https://keras.io/.

[21] Ecem Keskin. A wide perspective for designing user experience. [Online; accessed in 5 of January, 2019]. URL: https://stories.jotform.com/a-wide-perspective-for-designing-user-experience-1fac19643c5.

[22] Roshan Khan. Standardized architecture for conversational agents a.k.a. chatbots. *International Journal of Computer Trends and Technology (IJCTT)*, 50(2), 2017. URL: https://www.researchgate.net/publication/321637578_Standardized_Architecture_for_Conversational_Agents_aka_ChatBots.

[23] Ravindra Kompella. Conversational ai chat-bot architecture overview, 2018. [Online; accessed in 15 June, 2019]. URL: https://towardsdatascience.com/architecture-overview-of-a-conversational-ai-chat-bot-4ef3dfefd52e.

[24] Golden Krishna. *The Best Interface Is No Interface.* New Riders, 2015.

[25] Law, Lai-Chong, Roto, Virpi, Hassenzahl, Marc, Vermeeren, Arnold, and Kort, Joke. Understanding, scoping and defining user experience: A survey approach. pages 719–728, 04 2009. doi:10.1145/1518701.1518813.

[26] Lorenz Cuno Klopfenstein, Saverio Delpriori, Alessandro Bogliolo, Silvia Malatini. Rise of bots: A survey of conversational interfaces, patterns, and paradigms, 2017.

[27] Michael McTear. Conversation modelling for chatbots: current approaches and future directions. In André Berton, Udo Haiber, and Wolfgang Minker, editors, *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2018*, pages 175–185. TUDpress, Dresden, 2018.

[28] Aaron Bangor, Philip T. Kortum, James T. Miller. Determining what individual sus scores mean: adding an adjective rating scale. *Journal of Usability Studies*, 4(3):114–123, 2009. URL: https://pdfs.semanticscholar.org/3399/f83ff6149dc65b52600f52ed372be5a6aa86.pdf.

[29] Alan Nichol. Supervised word vectors from scratch in rasa nlu, 2018. [Online; accessed in 10 of May, 2019]. URL: https://blog.rasa.com/supervised-word-vectors-from-scratch-in-rasa-nlu/.

[30] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, Alan Nichol. Rasa: Open source language understanding and dialogue management. *CoRR*, abs/1712.05181, 2017. URL: http://arxiv.org/abs/1712.05181, arXiv:1712.05181.

[31] Christopher Olah. Understanding lstm networks. [Online; accessed em 15 of May, 2019]. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[32] Tatiana Parshina. Understanding rasa tensorflow intent classifier, 2019. [Online; accessed in 11 may, 2019]. URL: https://medium.com/@tatiana.parshina/understanding-rasa-tensorflow-intent-classifier-e9d4ef019c6.

[33] PAUL PINARD. Classification metrics (part 1) how to boost your bot performance through data, 2018. [Online; accessed in 10 June, 2019]. URL: https://cai.tools.sap/blog/classification-metrics/.

[34] Chia-Wei Liu, Ryan Lowe, Iulian Vlad Serban, Michael Noseworthy, Laurent Charlin, Joelle Pineau. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *CoRR*, abs/1603.08023, 2016. URL: http://arxiv.org/abs/1603.08023, arXiv:1603.08023.

[35] Niranjana Ragurama. What is the best example for false negative, false positive, true negative and true positive in machine learning?, 2018. [Online; accessed in 20 of February, 2019]. URL: https://qr.ae/TWKF3c.

[36] Sebastian Raschka. Confusion matrix, 2019. [Online; accessed in 10 June, 2019]. URL: http://rasbt.github.io/mlxtend/user_guide/plotting/plot_confusion_matrix/.

[37] Kohavi, Ron, Henne, Randal, and Sommerfiel, Dan. Practical guide to controlled experiments on the web: Listen to your customers not to the hippo. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 959–967, New York, NY, USA, 2007. ACM. URL: http://doi.acm.org/10.1145/1281192.1281295, doi:10.1145/1281192.1281295.

[38] Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735, doi:10.1162/neco.1997.9.8.1735.

[39] Amit Mandelbaum, Adi Shalev. Word embeddings and their use in sentence classification tasks. *CoRR*, abs/1610.08229, 2016. URL: http://arxiv.org/abs/1610.08229, arXiv:1610.08229.

[40] Spacy. Available pretrained statistical models for portuguese. [Online; accessed in 10 of May, 2019]. URL: https://spacy.io/models/pt.

[41] Spacy. spacy - industrial-strength natural language processing in python. [Online; accessed in 5 of May, 2019]. URL: https://spacy.io/.

[42] usability.gov. System usability scale (sus). [Online; accessed in 10 of September, 2019]. URL: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

[43] Chris Welch. Google just gave a stunning demo of assistant making an actual phone call, 2018. [Online; accessed in 20 of August, 2019]. URL: https://www.theverge.com/2018/5/8/17332070/google-assistant-makes-phone-call-demo-duplex-io-2018.

[44] Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, Jason Weston. Starspace: Embed all the things! *CoRR*, abs/1709.03856, 2017. URL: http://arxiv.org/abs/1709.03856, arXiv:1709.03856.

[45] Tobias Wochinger. Rasa nlu in depth: Part 1 intent classification, 2019. [Online; accessed in 10 may, 2019]. URL: https://blog.rasa.com/rasa-nlu-in-depth-part-1-intent-classification/.

[46] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, pages 2825–2830, 2012. URL: https://arxiv.org/abs/1201.0490.