TÉCNICO LISBOA

Chattuga

A meta-chatbot for the portuguese language

Mariana Rodrigues Seguro Gaspar Fernandes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Dr. Maria Luísa Torres Ribeiro Marques da Silva Coheur

Examination Committee

Chairperson: Prof. Dr. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Dr. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Member of the Committee: Prof. Dr. Francisco António Chaves Saraiva de Melo

November 2019

Acknowledgments

First and foremost I would like to thank the person who always taught me that this is the way: first to choose a dream, and then to pursue it without ever giving up. That person is my greatest hero and also my Mom, who, unfortunately, was not able to witness my journey through college but whose teachings were what got me through said journey. Thank you so much my dear and beloved Mom.

I would also like to thank my advisor, teacher and friend Luísa for being great at everything she does. For not only advising me and guiding me through this journey but also for being so gentle, encouraging and funny through the whole process. You light up every room that you walk into. Thank you for being the great person, friend, advisor and teacher that you are. This thesis was only possible because of you. I have a profound admiration, respect and care for you. Thank you a lot!

Dad, thank you for introducing me to the wonderful world of computers. All of those computers that we fixed both hardware and software wise throughout the years ever since I was a little girl really ignited my passion for this fascinating world. Thanks to you I found my vocation. I am also very grateful that you let me focus only on school in all these years.

Sofia, Filipa and Madalena, my dear sisters, you were my rocks through these hard years. I love you all beyond words. To my brothers in law, thank you for all that you have done for me and for taking such good care of my sisters. To my two nephews, thank you for making me want to be better so I can be a good role model for you.

Catarina C. thank you so much for your help through the hardships I have been through and for helping me in my growth as a person. Your support and care through all these years are invaluable to me and I am going to carry them with me and treasure them my entire life.

Prof. Alexandra Forte, thank you for being incredibly supportive in a time where I so much needed support. Thank you also for believing and making me believe in myself and thus helping me succeed in your course. Prof. Juvenal Brites thank you for being the first teacher to see in me my ability for mathematics and helping me in feeding that. If it were not for you, my college life would be much much harder. Prof. Ana Moura Santos, I will never forget the dedication you showed in teaching us and helping us pass your course, you were with us for hours (you did not even eat!!!). Thank you so much. Prof. Paula Gouveia, thank you for helping me after I had to miss classes for two weeks, you did not have

to but you did it anyway and I will be forever thankful. Prof. Ana Cardoso-Cachopo, I am sorry that I did not pass your course right away because your incredible passion and dedication and your bet in me should have been rewarded with an equal amount of effort on my part. Thank you for not giving up on me and for helping me even after you were no longer my teacher. Prof. Francisco Melo, as Prof. Ana Moura Santos you too were several hours helping us with our project for your course when you could have been just one or two hours. You stayed a whole evening and afternoon just so you could help everyone. In that day my project was failing miserably and without your help I would not have been able to make it. Thank you so much. Prof. Francisco Santos, thank you for teaching me a valuable life lesson when I was doing your course and did not dedicate enough time to the project and also for being a great teacher and person. You are one of the nicest persons I have ever met. Prof. Ricardo Chaves, thank you for failing me in your course the first time I did it. If you had not done that, I would have not learned anything, and I very much liked to learn all of it in the next year. Prof. Miguel Pardal, thank you for your kindness and extreme helpfulness not only when I was taking your course but also afterwards. Prof. António Ferraz thank you for your patience in teaching me electromagnetism and optics, a course that brought me much headaches but with your calm and support I was able to finish it with a relatively good grade. Thank you all for showing me what a great professor looks like (this also applies to my advisor).

Anisa, Dannie, Bruno, Francisco, Henrique, Luís R., Andreia, António, José, Diana , Luís G., Luís S., thank you all for being there for my journey through college and for being great through it all. I honestly could not have done this without the laughs, trips, support and all that you have provided me with. Also to Anisa, I am glad you said ”hi” enough times for me to respond back. Thank you, Anisa, for not giving up, for being great when I was not, for all the work I did alongside you, all the support you gave me and for being so funny and silly, you are a friend for life – I adore you. Dannie, I am so glad we met at the time we did, our lives were really at the same point and you have become a huge part of my life in such a short amount of time. Your kindness, love and jokes are mesmerizing, I adore you. I look forward to expanding this short amount of time to the timespan of our lives. To Luís R., Francisco and Henrique, doing this thesis alongside you was a lifeline. Each of you showed me something I admired: Henrique, you have such passion and thirst for knowledge — I hope one day to learn how to be more like you; Francisco, all those mornings and afternoons we met and talked a little bit were a huge support; Luís R. you were an inspiration of perseverance throughout this thesis – I respect and admire that. Working on my thesis with you all made it easier. To Bruno and Francisco, thank you for being such a great team in all those projects we performed together, I could not have done them without you. Thank you migs for making this journey easier and lighter.

João and Kiko, thank you for being my best friends for 14 plus years and putting up with my humours and for supporting me and growing up with me. João, it was great to share the thesis experience with you. It truly helped a lot. All those days at Amoreiras and Caleidoscópio made this thesis easier. Sofia,

thank you for joining our group and for being so kind, funny and caring.

Carla C. there are no words to describe all that you have done for me over these years. You called me into your office and I responded with little interest. After being there a few minutes you saw that I was in dire need of help and you gave me your hand and all the tools that I needed to pick myself up. I see in you a friend for life. Thank you so much!

Paula, Inês, Isabel, my oreo's, I will never forget those afternoons laughing and occasionally studying with you. You are crazy, but the same crazy as me. Thank you, you have made this journey easier.

Leonor, I could not listen to a single class when I was by your side. You made me laugh countless times and I had so much fun in boring settings with you. Thank you so much for being so goofy and crazy!

Patrícia, the talks, the rides, fatacil, the bad and good moments, they were all incredible. Thank you so much for everything!

Leonor L., João S. and Luísa, manos was one of the best teams I ever had, thank you for making it so. I had a blast with you guys.

To all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To my grandmother, aunts, uncles and cousins, one of the things I love the most in this world is having a huge family and you have staggered me with your unconditional love. Thank you especially to aunt Belinha, uncle Emílio and cousin Tiago for taking me in in the first semester when I was with crutches, your love, care, rides and food were invaluable. To my aunt Adília and Manuel, thank you for picking me up every day of my first semester to have lunch for free at your restaurant. To my aunt Olinda, you were the most supportive aunt ever during these years, I have no words to describe how incredible you were. Thank you so much.

To AMA, FCT, INESC-ID, all the crew of SLATE, Teresa, Luísa, Pedro and Catarina for the amazing experience in Coimbra when I went to present the UC-PT corpora work. It was a perfect experience that I will always treasure.

To FCT and AMA. This work contributes to the FCT, Portugal INCoDe 2030 National Digital Skills Initiative, within the scope of the demonstration project "Agente Inteligente para Atendimento no Balcão do Empreendedor" (AIA).

Last but not least, to all of IST and its community. When I entered this college, I was living the worst time of my life and here I found a home where I healed. I will forever be thankful to this wonderful school.

It takes a village. To each and every one of you – Thank you for being my village.

Abstract

Most works on question answering systems follow a similar approach: they start by gathering data, which is then pre-processed, classified and used in a system which replies queries in the domain of that data. In this work, we propose a plug and play (for the addition of corpora and agents and for the control of system modules) “collaborative” chatbot framework for the Portuguese language that has as basis the assumption that all agents can, potentially, answer all questions. We study classification approaches and similar systems and we base our architecture on that study. The system built in this thesis, receives a query from a user and redirects it to all chatbots integrated within it. These chatbots all provide their suggested answers to the user query. When the program finishes gathering all these answers it evaluates the agent’s answers as well as their expertise in the domain of the question and combines that with the classification of the user’s query intent to perform an answer ranking process. In both classification processes, both Machine Learning and rule-based approaches are combined. The evaluation of the program developed in the scope of this thesis is made with the metrics of Accuracy and Mean Reciprocal Rank, where the first is used for the query classification evaluation and the second is used for the system’s answer ranking decision evaluation. Results show that our system has a strong classification component both for questions and answers and also that it is good at identifying specialist agents.

Keywords

Question Classification; Question and Answering systems; Conversational Agents; Machine Learning; Rule Based.

Resumo

A maioria dos trabalhos sobre sistemas de *question answering* seguem uma abordagem semelhante: começam por recolher dados, que são depois pré-processados, classificados e utilizados num sistema que responde a perguntas no domínio desses dados. Neste trabalho, propomos uma framework *plug and play* (para a adição de corpora e agentes e controlo de módulos do sistema) de *chatbot* "colaborativo" para o português que tem como pressuposto que todos os agentes podem, potencialmente, responder a todas as questões. Estudamos abordagens de classificação e sistemas semelhantes e baseamos a nossa arquitectura nesse estudo. O sistema construído nesta tese, recebe uma *query* de um utilizador e redirecciona-a para todos os chatbots integrados no mesmo. Todos estes fornecem as suas respostas sugeridas para a *query* do utilizador. Quando o programa reúne todas essas respostas, ele avalia-as, bem como a sua especialidade no domínio da pergunta e combina isso com a classificação da intenção da *query* do utilizador para realizar um processo de *rank* de respostas. Em ambos os processos de classificação, tanto *machine learning* quanto as abordagens baseadas em regras são combinados. A avaliação da framework desta tese é feita com as métricas de *Accuracy* e *Mean Reciprocal Rank*, onde a primeira é usada para a avaliação da classificação da *query* e a segunda é usada para a avaliação da decisão do *rank* de respostas. Os resultados mostram que nosso sistema tem uma forte componente de classificação, tanto para perguntas quanto para respostas, e também que a framework é boa na identificação de agentes especializados.

Palavras Chave

Classificação de questões; Sistemas de pergunta-resposta; Agentes Conversacionais; Machine Learning; Sistemas Baseados em Regras.

Contents

List of Figures

List of Tables

Acronyms

NLP	Natural Language Processing
IST	Instituto Superior Técnico
FAQ	Frequently Asked Questions
NLP	Natural Language Processing
IR	Information Retrieval
SSS	Say Something Smart
AMA	Agência para a Modernização Administrativa
CNN	Convolutional Neural Networks
LSI	Latent Semantic Indexing
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
TF-IDF	Term Frequency - Inverse Document Frequency
NN	Neural Networks
NER	Named Entity Recognizer
SVM	Support Vector Machine
TRR	Total Reciprocal Rank

1

Introduction

Contents

This thesis is a work named Chattuga for the Portuguese language in the field of artificial intelligence, more specifically, in the area of Natural Language Processing (NLP) that intends to present a new plug and play architecture for a chatbot (also known as conversational agent) in which we will use several other conversational agents. Next we will present the motivation (Section 1.1), objectives (Section 1.2), contributions (Section 1.3) and project overview (Section 1.4) for this thesis.

1.1 Motivation

With the exponential increase in internet services, it has become clear that online support is very important for costumer satisfaction. One approach to this is to use conversational agents who can answer client's questions about a given service. There are chatbots that do this for a strict domain. However, for an online organization that has many different services, it would be interesting to have many chatbots: one for each service. For instance, an online retailer could have an agent that only knows about payment methods, other agent that has knowledge of technological products, another one that is a specialist in legal procedures, among others. Each have its area of expertise and they can work in parallel in providing online support for costumers. Not only is such an architecture useful for company's needs: a website or application with recreational and/or educational purposes could use such an architecture of multi-agents to answer user's questions about a multitude of subjects such as movies, personal matters, science, space, among others. One work that already integrates several conversational agents is described in [1]. However, this work does not take into account the fact that there may be some intersection between what two conversational agents can answer. For instance: an agent that retrieves answers from Wikipedia [1] can answer factoid questions, and an agent that retrieves answers from IMDB [2] can answer questions about cinema. Both can answer questions about cinema, although the agent that uses IMDB can potentially do it better since it is its domain of expertise. In this thesis we will tackle this issue by assuming that "all agents can potentially answer all questions". Then, a Coordinator will decide how to rank their answers based on their topics of expertise.

1.2 Objectives

For this thesis we have three main objectives: to have a system for the **Portuguese language** that works in a **plug and play** fashion in the way it adds agents and corpora and in the way it controls the modules of the system, and has a **Coordinator** controlling the different agents. A simplified overview of the components involved is represented in Figure 1.1. Note that the numbers in the figure represent the order in which the different components are called.

[1] https://www.wikipedia.org/ (Last accessed on: 25/11/2018).
[2] https://www.imdb.com/ (Last accessed on: 25/11/2018).

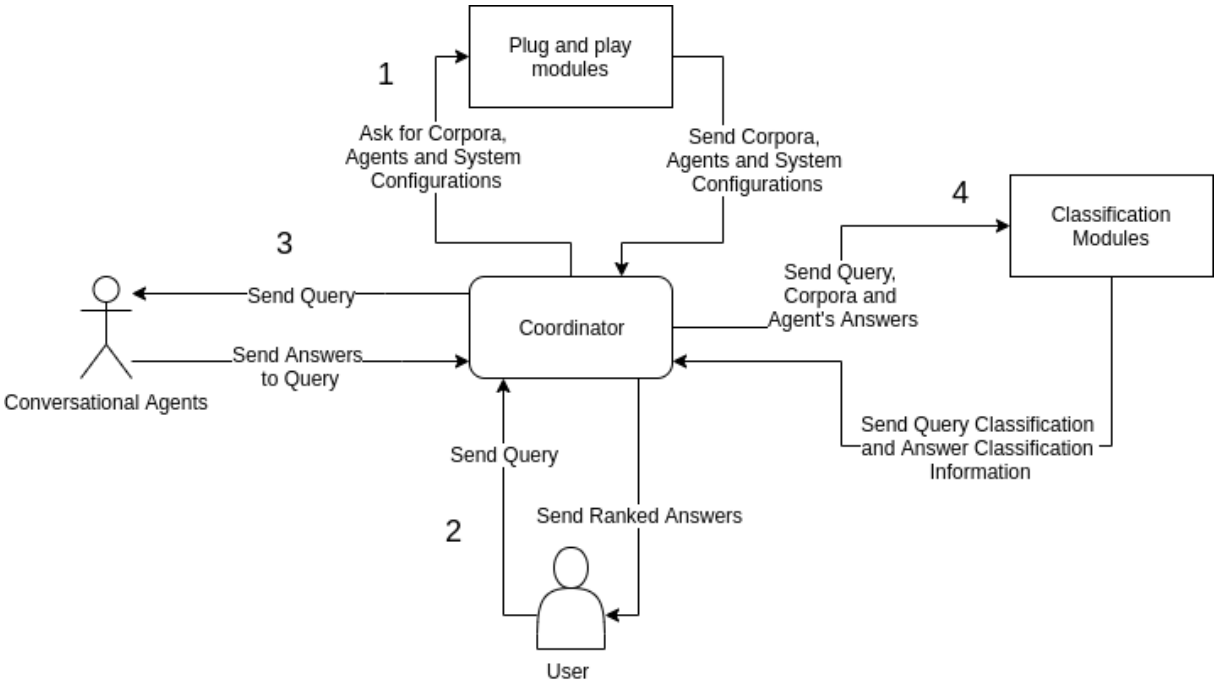


Figure 1.1: Overview of the system's components.

The focus of this work will not be on building conversational agents of different domains (those are already built and then integrated in our system), but rather on understanding a user's intention when he introduces a query in order to understand which answer among the ones returned by the various conversational agents is the most suitable. To perceive which answer is more suitable to the user's query, we are going to use machine learning techniques, similarity techniques and rule based techniques. First, we classify the query with machine learning modules trained in the corpora that is added to the system using the plug and play module. The plug and play module also provides the system with conversational agents which will have specified their areas of expertise. With the classification of the query, we can compare it with the area of expertise of the agents. Secondly, depending on whether the query is personal or impersonal (information we obtain from the classification process), we use similarity measures, rule based approaches and/or machine learning techniques to determine to which extent the answers are close to the query. Lastly, if a corpora of answers is added through the plug and play module, we use machine learning techniques to classify the agent's answers and compare the classification of the answer with the classification of the query. The plug and play module also allows a control over which modules are used to rank the answers.

To ensure a coordination of the different techniques (machine learning techniques, similarity techniques and rule based techniques), we require an agent that is capable of retrieving from the combination of all these models, the ranking of answers that, according to the techniques, is the most correct one. This agent will be the Coordinator that was previously mentioned.

1.3 Contributions

The main contributions of this work are:

- A new plug and play multi-bot architecture for chatbots for dynamic addition of corpora and agents as well as to control the system modules that are used. This includes a way to dynamically generate init files for Python which allows systems to import other Python systems that are in different folders (which is required for the agents plug and play addition);
- A corpora [4], namely a corrected corpus for the Portuguese language that allows to distinguish: (a) questions from non-questions; (b) impersonal questions from personal questions; and (c) yes/no questions from other questions;
- Proposal of techniques and pipelines of classification to employ in conversational multi-agent systems that help in the task of answer ranking, which also generated another contribution that is a proposal as to how to evaluate such systems;
- A system tested with two specialized agents.

1.4 Project Overview

Chattuga is a framework that will be composed of several modules in order to achieve its proposed functionality. The first thing that is done in this system is to retrieve informations from the **plug and play module** that will indicate which agents, corpora and system modules will be used in Chattuga. When the system has these informations, it passes them to a module called **Coordinator** which instantiates the agents and gets the corpora. After it has the corpora, it trains another module, the **query classification module**, with the informations provided by the **corpora plug and play module**. When the user enters the query, the system is already trained to classify the user's query intention. The query is redirected by the **Coordinator** to the agents and to the **query classification module**. The agents return their answers to that query and the **query classification module** returns the labels it predicted for the query. The **Coordinator** then evaluates the informations it got from the **system plug and play module** to ascertain to which **answer classification modules** it can send the agents' answers to. After it gets this information, it sends the agents answers to the active **answer classification modules**. These send their scores of each agent answer which the **Coordinator** combines to obtain the combined score per agent answer and, with that information, ranks the answers according to their predicted relevance to the user query.

This work is divided as follows: Chapter 2 covers related works. These are inserted in four main categories that follow the same structure as Chattuga, that is, each Section supports a component

of the framework. Those are:: **similar architectures** to the one we intend to propose (Section 2.2); **question classification** which will be of the essence to help us in the choice of an appropriate answer (Section 2.3); **answer ranking**, using different techniques (Section 2.4, Section 2.5, Section 2.6); finally, we will describe some **agents** that are integrated in our system (Section 2.7). Chapter 3 describes the corpora that we have used in this work to perform the training of the classification models. Chapter 4 provides a guide through the implementation of this project. Chapter 5 explains what we have done to evaluate our work and the results of that evaluation. Lastly, Chapter 6 presents our conclusions and contains some suggestions of future work.

2

Related Work

Contents

2.1 Introduction

In this Chapter we will look at works that will be important for our project, namely works with similar architectures (Section 2.2), works on question classification (Section 2.3), to a module of answer search (Section 2.4), a work on query/answer match (Section 2.5), some similarity measures (Section 2.6) and some agents that we can integrate in our system (Section 2.7).

2.2 Similar Architectures

In this Section we present similar architectures to the one we intend to propose.

2.2.1 TalKit

TalKit [1] is a program developed by a masters student at Instituto Superior Técnico (IST). TalKit receives as input a user question and, through several NLP techniques, it redirects that question to a conversational agent that it considers to have some expertise in the topic of that query and thus, possibly, provides a good answer. First, it pre-processes the line of dialog, thus facilitating the next steps in the pipeline; second, classifies the inputted question according to a taxonomy (this classification is detailed later in this Chapter); third, according to the classification attributed in the previous step, it redirects the question to the appropriate agent. TalKit's architecture can be seen in Figure 2.1.

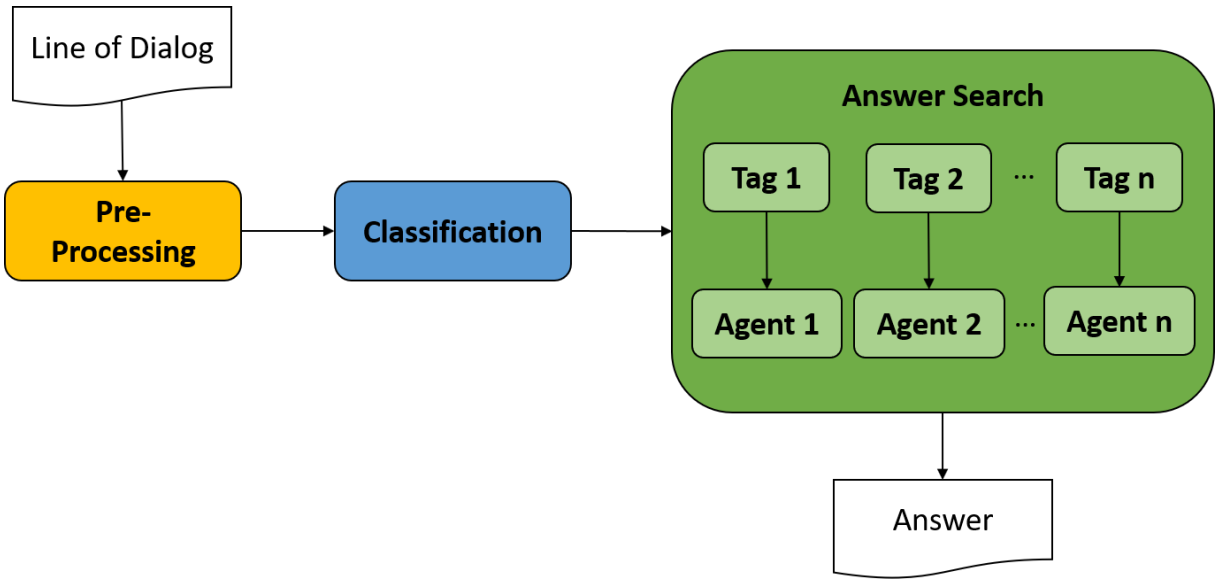


Figure 2.1: TalKit Architecture. This diagram was based on images extracted from [1] to showcase relevant components.

We intend to use multiple conversational agents to answer a user input, which is somewhat similar to

what is presented in TalKit. A query is received and the system *knows* some chatbots and redirects the query to them. Our approach will be different in the sense that we will redirect the query to all the agents that our program *knows*, that is all the conversational agents that were added to our system and they each provide an answer for the user interaction. The final answer that will be sent to the user is chosen among those answers. It is a different approach to TalKit because TalKit only redirected a question to one chatbot (the one seen as the most adequate to answer the received user query).

2.2.2 Microsoft QnA Maker

QnA Maker [1] is a system by Microsoft that receives as input a Frequently Asked Questions (FAQ) or other semi-structured source of information containing multiple question/answer pairs that are converted into a conversational agent to which a user can make various queries. It will return the most likely answer from the semi-structured information source. Furthermore, this system also returns several possible answers to the same question. The company and clients using it can choose the best answer (if the one that is returned first is not the most correct one) thus coaching the system as to what is the best solution to a given question. It is a plug and play service that, in the end, can provide a chatbot for any subject that it is fed data about. For example: if we give as input a FAQ about health-care, then a chatbot that knows how to answer health-care questions that are already answered in the FAQ is built. The same applies to other subjects. QnA Maker also has the ability to train itself by receiving user input which in turn makes the system recalculate the order to which it gives the answers the next time it is asked that particular question. It is not detailed how this training is done. It is only mentioned that machine learning is used without further details. The architecture here described can be seen in Figure 2.2.

Our work, similarly to the one presented in this section, intends to be plug and play, that is: a user provides a chatbot and the training data and our system should be able to integrate the new elements provided.

[1] https://docs.microsoft.com/en-us/azure/cognitive-services/qnamaker/overview/overview (Last Accessed on: 06/01/2019).

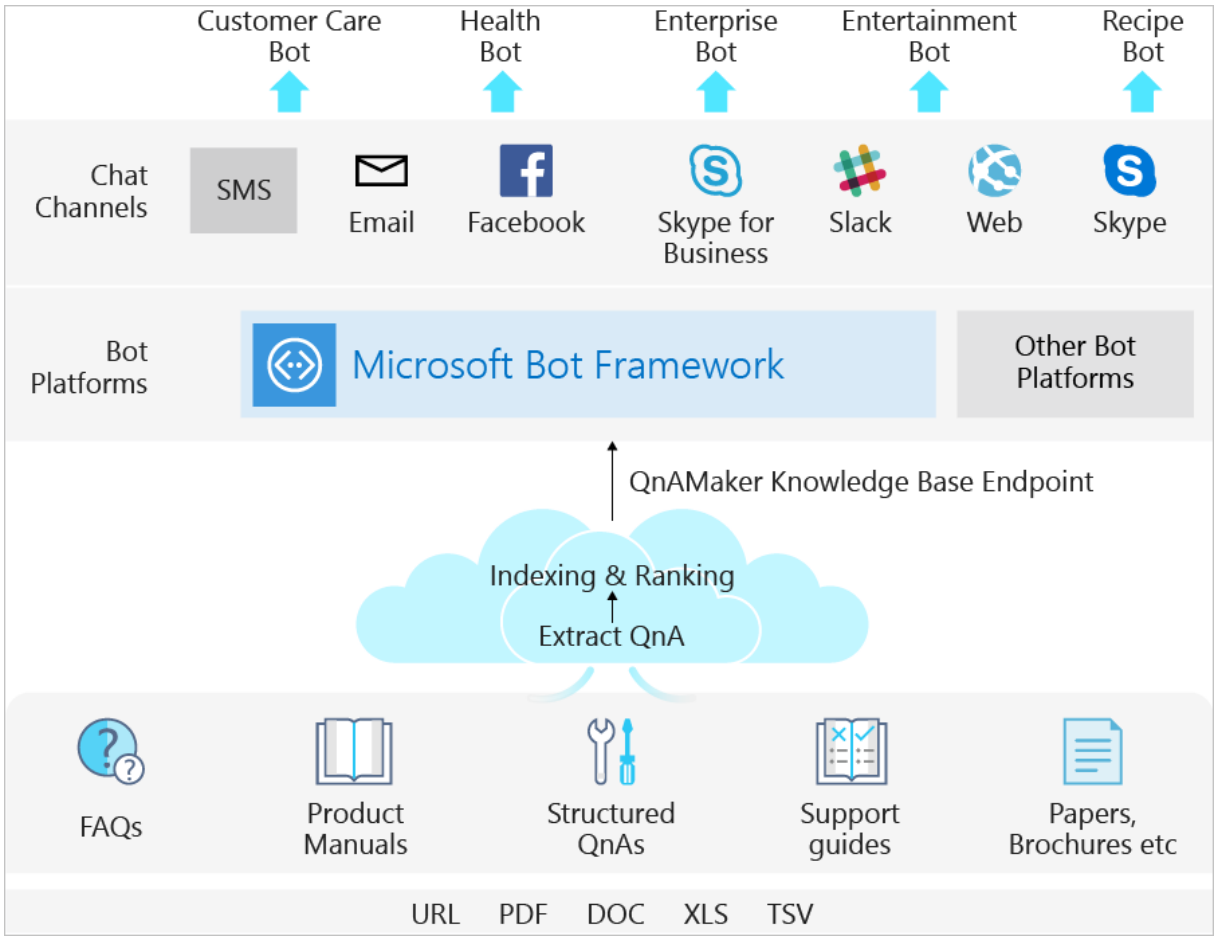


Figure 2.2: QnA Maker Architecture. Image extracted from Microsoft's website (footnote 1).

2.3 Classification

For our work, question classification is an important module considering that we can only evaluate if an answer is good if we know the question and its intention (works [5] and [6] discuss the importance of question classification in question answering). Some systems that provide meaningful results in this topic will be presented next.

The state of the art in question classification up until the year of 2011 is summarized in the survey made by Loni [7] which details works over several datasets and several different approaches to question classification. Since 2011 there has been some more works on question classification, namely the one by Pota et al. [8] in 2015, which provides a study of the comparison between different sets of features that can be used in machine learning approaches, with the aim of finding smaller and less complex sets of features that still provide good results. Still by Pota et al. [9] a work where a procedure of forward-selection of features is used with support vector machines to classify questions with a smaller set of

features with the goal of reducing the noise in the feature space. In 2015, Feng et al. [10] presents a system that uses semantic similarity between questions to do question classification but also to find a similar question that has an answer in the dataset. Kim, detailed in [11], uses Convolutional Neural Network (CNN) with word vectors to perform sentence classification and sentiment analysis. Kiros et al. [12] uses what he called Skip-Thought Vectors (a model that uses a sentence to predict the sentences around it). The author then tested this in tasks of classification, one of them being question-type classification. The works that will be used for this thesis are detailed next.

2.3.1 Taxonomies

Li & Roth [3] proposed a taxonomy where a factoid question can be classified with a broader (coarse grained) category or a more specific (fine grained) category. For instance, the question "What is the proper name for a female walrus?" is classified as "ENTITY:animal" because that is what is expected in the proper answer for this question. The full set of coarse and fine grained categories can be found in Table 2.1. The dataset that was made in conjunction with this taxonomy has been translated to the Portuguese language by Costa et al. [13].

Table 2.1: Li & Roth taxonomy. Information extracted from [3].

Coarse	**Fine**
ABBREVIATION	abb, exp
ENTITY	animal, body, color, creative, currency, dis.med., event, food, instrument, lang, letter, other, plant, product, religion, sport, substance, symbol, technique, term, vehicle, word
DESCRIPTION	definition, description, manner, reason
HUMAN	group, ind, title, description
LOCATION	city, country, mountain, other, state
NUMERIC	code, count, date, distance, money, order, other, period, percent, speed, temp, size, weight

In TalKit's [1] taxonomy a sentence is classified as QUESTION or NON-QUESTION. If it is a question (receives the tag "QUESTION") it can also have a question sub-category such as:

- **"yes/no question"**: tag "YN_QUESTION", that is a question that is meant to be answered with just "sim" ("yes") or "não" ("no") (example: "Usas óculos?" ("Do you wear glasses?"));
- **"or question"**: tag "OR_QUESTION", a question that is a choice between two alternatives (example: "Preferes carne ou peixe?" ("Do you prefer meat or fish?"));

- **“rhetorical question”**: tag “RHETORICAL QUESTION”, a question that is, usually, not meant to be answered, it is only rhetorical (example: “És maluco?” (“Are you crazy?”));
- **“open-ended question”**: tag “OPEN_ENDED_QUESTION”, these are questions that have no specific answer, but an answer is expected (example: “Mais alguma coisa?” (“Anything else?”));
- **“list-question”**: tag “LIST_QUESTION”, questions that ask for indications of something (example: “Diz o nome de três presidentes portugueses.” (“Say the name of three portuguese presidents.”));
- or **“what/when/where/who/whom/which/whose/why/how question”**: tag “WH_QUESTION”, a question that starts with one of the words enumerated at the beginning of this sentence (examples: “Qual é a capital de Portugal?” (“What is the capital of Portugal?”), “Como se faz o chocolate?” (“How is chocolate made?”)).

However, having just a sub-category of a QUESTION is not enough to know how to redirect it through the system, given that a question can be of a certain sub-category and depending of whether or not it is a personal question it can require a different kind of agent to produce a response to the same sub-category: tags “PERSONAL” or “IMPERSONAL”. An agent who is trained to respond to small talk can be better suited to a personal question whereas an agent that can answer to facts can be better at answering impersonal questions. An impersonal line of dialog can be further classified by the Li & Roth Taxonomy to extract a relevant feature concerning the topic of the line of dialog.

If the user interaction is tagged as “NON_QUESTION”, it can be further classified as:

- **Social obligation**: tag “SOCIAL_OBLIGATIONS”, these are the sentences that are uttered as part of the socially required obligations (example: “Bom dia!” (“Good morning!”));
- **Acknowledgements**: tag “ACKNOWLEDGEMENTS”, such as acknowledging an answer (example: replying “ok” to a sentence sent by the agent);
- **Declarative sentence**: tag “DECLARATIVE SENTENCE”, that, as the name indicates, is a sentence that is declarative (example: “O Porto fica no norte de Portugal.” (“Porto is located at the north of Portugal.”)).

There is an additional classification (tag “NO_UNDERSTANDING”) in this taxonomy, given after we determine if the sentence is a question or not, whose sole purpose is to determine if it is a sentence that indicates a lack of understanding on the part of the user, that is, indicates that the sentence received is the user telling the system that he did not understand the reply that the agent gave to him (example: the agent returns some sentence and the user replies “I’m sorry, I did not understand.”, this reply of the user is classified with the tag “NO_UNDERSTANDING”). This classification with the tag

"NO_UNDERSTANDING" is given regardless of the line of dialog being classified as a QUESTION or as a NON-QUESTION, contrarily to the others mentioned before. A part of this taxonomy will be used in this work. If needed, some minor adaptations will be performed.

2.3.2 Rule Based Approaches

The work by Silva et al. [2], is a rule based one combined with a machine learning one. The system developed in the previously mentioned work [2] uses several techniques combined in order to obtain a semantically relevant tag for a question. To identify the categories of questions (according to the previously described Li & Roth's taxonomy), the system uses manually built patterns but, as that is not enough, it also has several steps to identify a headword and map it into a question category. Then, this is combined with the questions unigrams in a machine learning approach. The steps taken by the system are described next.

Manually Built Patterns

A great deal of questions within a category follow the same pattern regarding its construction (example: when asking for the meaning of an acronym, we usually ask "What does" + the acronym + "means?", like the following **different** questions "What does DNA mean?" and "What does ONU mean?"). These patterns were identified and used in the *question classification system* presented in this section. Whenever a line of text is received by the system it is compared against these patterns and if a match is found, a question category is also found. A list of the patterns used is presented in Table 2.2.

Table 2.2: List of question patterns for the identification of a semantic tag without using headwords. Extracted from [2].

Category	Question pattern description	Example
ABBREVIATION.:exp	Begins with *What do(es)* and ends with an acronym – i.e., a sequence of capital letters possibly intervened by dots –, followed by *stands for/mean*;	*What does AIDS mean?*
	Begins with *What is/are* and ends with an acronym	*What is F.B.I.?*
DESCRIPTION:definition	Begins with *What is/are* and is followed by an optional determiner and a sequence of nouns	*What is ethology?*
ENTITY:term	Begins with *What do you call*	*What do you call. . .*
ENTITY:substance	Begins with *What is/are* and ends with *composed/made of*	*What is glass made of?*
DESCRIPTION:reason	Begins with *What causes*	*What causes asthma?*
HUMAN:description	Begins with *Who is/was* and is followed by a proper noun	*Who was Mozart?*

Headwords

In order to identify a category of a question, it is often useful to find the most relevant word within the question: the one that gives the best clue to uncover the question's intention. This relevant word can also be called *headword*. To extract a *headword*, the investigators based their approach on the morpho-syntactic categories of words. Some sequences of morpho-syntactic categories can tell us which word is the best to our purposes. The subset of head-rules used to the work here presented can be found in Table 2.3. What the algorithm developed by João Silva does is to go through the parse tree and match it to manually built patterns. A subset of those patterns can be seen in Table 2.3. A representation of the parse tree can be seen in Figure 2.3. So, if it comes from a node "S", it then searches in the priority list for the "child" nodes starting at the left of the priority list, if it finds the node in the priority list it repeats the process with the node found now being the parent, until it reaches a terminal symbol which will be the headword.

In addition to those patterns presented in Table 2.3, there are some extraordinary cases that would be incorrectly identified by this method. For those cases, some non-trivial rules were established. The complete rules have not been described here, but can be seen in [2].

Table 2.3: Manually built patterns list to help with the identification of headwords. Extracted from [2].

Parent	Direction	Priority List
S	Left	VP S FRAG SBAR ADJP
SBARQ	Left	SQ S SINV SBARQ FRAG
SQ	Left	NP VP SQ
NP	Right *by position*	NP NN NNP NNPS NNS NX
PP	Left	WHNP NP WHADVP SBAR
WHNP	Left	NP
WHPP	Right	WHNP WHADVP NP SBAR

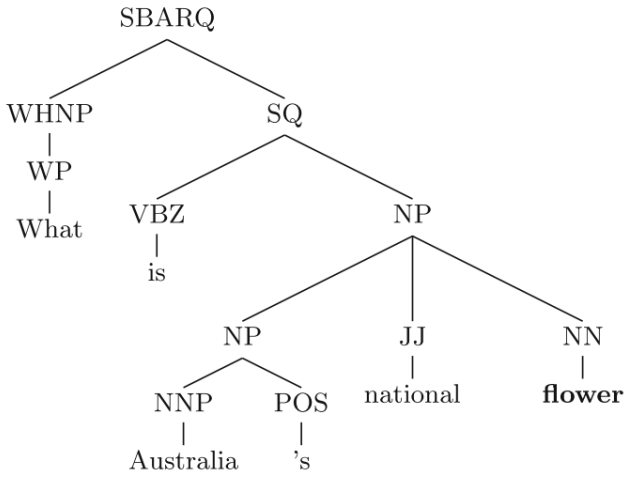


Figure 2.3: Parse tree of a question. The headword is represented in bold. Image extracted from [2].

Mapping Headwords to categories

Once a *headword* is identified by the aforementioned algorithm, a mapping to a category must be done in order to finish the classification of the question. To perform this action, the word identified was mapped into WordNet[2] [14] to find a general category for the question. For instance: the word "actor" is mapped into "HUMAN:individual".

This process of classification is particularly useful for our work since by having a category of a question we can know what to expect in an answer. Having that knowledge can provide us the insight we need to sort through the multiple answers for the same question that we are going to have and give them weights accordingly.

In TalKit, the first phase of classification is to go through a set of rules manually crafted to detect some of the tags (detailed in Section 2.3.1). The tags that are not identified by this approach are, afterwards, identified by the machine learning approach.

1. For the tag "YN_QUESTION", the morpho-syntactic tags identified in the pre-processing phase are used, so the sentence is identified as a yes/no question if it starts with: a verb; a pronoun followed by a verb; a proverb; an adverb; a determinant followed by a noun; or a noun.

[2] A large lexical database that contains over 155.000 words that are organized in groups by semantic meaning.

2. For the tag “OR_QUESTION”, if we find an “ou” (“or”) in the sentence, it is identified with this tag.
3. For the tag “WH_QUESTION”, if a pronoun or an interrogative adverb is found in the beginning of the sentence, the aforementioned tag is attributed.
4. For the tag “LIST_QUESTION”, if a keyword that usually appears in this type of questions is present such as “Indique” (“Indicate”), “Diga” (“Say”), etc. then the sentence is identified as a list question.
5. Finally, for the tags “RHETORICAL_QUESTION”, “OPEN_ENDED_QUESTION”, “SOCIAL_OBLIGATIONS”, “ACKNOWLEDGEMENTS” and “NO_UNDERSTANDING”, if a list of sentences/words that are coherent with the type of interaction are found, the appropriate tag is attributed.

A work that was able to improve on the results for fine grained categories of Silva et al. [2] using a rule based approach is described in [15]. This work is purely rule based using Syntactic Maps, Types [16] and Concepts. The approach is simple: the investigators first start by checking if the question is in a format known by the system. For example, the question “In 139 the papal court was forced to move to Rome from where?” is rewritten as “To where was the papal court forced to move from Rome in 139” – a more standard form of writing a question. If it is not in this format, then the question is rewritten in a format that the system can *understand*. When the question is in the known format it is mapped to a syntactic map which is a general template that they created for questions that, with the help of parse-trees, captures the question’s “Wh-word”, the noun phrase contained in a WHNP and finally the auxiliary verb, noun phrase and first main verb in the SQ sub-tree. If any of those elements does not exist in the phrase, its value will be set to None. If the question cannot be mapped into a syntactic map, it cannot be classified using this work. After they do that, they proceed to match the words of the question to Types which *tell* the system some sense of the meaning of a word with the help of the WordNet [14] Synsets, of its type. After they have all of what they need (the question’s wh-word – explained in Section 2.3.1 – and the head noun, head noun adjectives, verb and auxiliary verb and the respective types) they match the question to its question class (fine-grained category).

2.3.3 Machine Learning Approaches

Silva et al. [2] combines a machine learning approach (Support Vector Machines) with the aforementioned rule based classifier. The question’s unigrams are combined with the features provided by the rule based classifier (those features are question category and question headword) and they are used in a machine learning module which yields an accuracy of 95% for coarse-grained categories of questions (the broadest category of a question) and an accuracy of 90.8% for fine-grained classification (the more specific category of a question).

The state of the art in coarse grained categories for the UIUC dataset was obtained by Van-Tu and Anh-Cuong [17]. What this work has done was to experiment with different sets of features to use as input to a linear kernel functions Support Vector Machine to increase the accuracy of the classification, ultimately finding a solution that obtained the aforementioned state of the art results. The features used in this work were:

- **Unigrams** which consists on using each individual word of a question as a feature;
- **Bigrams** which is using two consecutive words on a question as a feature;
- **Word-Shapes** that is to use the shape of the words as features (example: if they are lowercase, if they are or contain digits, etc.);
- **Headword** is to use the "key" word of a question, that is, the one that provides better information about a question as a feature (these are extracted using a syntactic parse tree, not detailed in the paper how);
- **Related Words** which are words that represent a category of a word that is in a question, for instance if the question contains the word "month" then "date" will be added as a feature;
- **Query Expansion** consists on collecting the hypernyms of the headword using WordNet [14] and add them as a feature as well as their distance to the headword in the WordNet *tree*.
- **Question Category** is comparing the headword with all question categories available and finding the most similar one using WordNet. When that word is found it is added to the features;
- **Question Pattern** are a set of patterns that the investigators built having in mind "Wh" questions (explained before, in Section 2.3.1) and their most common structures. These were also used as a feature.

After considering these features, the investigators wrote an algorithm that calculated which were the features that increased the similarity for each type of question **for the development set**. From that algorithm they have concluded that, for questions with "How", "Who", "Why", "When", "Where" and "Which", the set of features that reached the highest accuracy were Unigrams, Bigrams, Word-Shapes and Question Pattern. For questions with "What" the set of features found were Unigrams, Bigrams, Headword, Word-Shape, Related Words, Question Pattern, Query Expansion and Question Category. Lastly, for other questions, the set of features is Unigrams, Bigrams, Word-Shape and Related Words. This proved to be a good approach since it reached state of the art results for coarse-grained categories, thus, having better results than Silva et al. [2] on the UIUC dataset.

In TalKit [1] a wide variety of training corpora was fed to the algorithm (Support Vector Machine) such as:

- two corpus built in the context of the TalKit thesis – one contained yes/no questions and the other contained rhetorical questions, open ended questions, greetings and farewell expressions, confirmation expressions and questions/sentences that indicate that the user did not understand the system's answer;
- one corpus containing Factoid Questions – corpus retrieved from the work by Costa et al. [13] that translated the corpus created by Li & Roth [3] to the portuguese language;
- one corpus containing Personal and Impersonal questions – the personal questions corpus was built with basis on the corpus used for the aforementioned agent "Just.Chat" [18] which was translated for the Portuguese language to be used in TalKit. The impersonal questions corpus was made with questions from the Costa et al. [13] corpus which is the Li & Roth corpus translated to the Portuguese language, as mentioned before;
- one corpus containing Questions and Non-Questions – corpus built with questions from the previously mentioned yes/no corpus and the personal and impersonal questions corpus. The Non-Questions corpus was built with answers to the question's corpus retrieved from the Say Something Smart (SSS) work, described in [19] and mentioned in Section 2.7.

The features used in the machine learning algortihm were *n-grams* (sequence of n consecutive words: were used Unigrams, Bigrams and Trigrams), *Binary N-grams*, *length* and *Word-Shape*.

2.4 Answer Search in TalKit

With TalKit's taxonomy and the two classification approaches previously described, the system is able to redirect the lines of dialog according to the respective tag as described below.

When a Tag is identified, it is time to produce an answer. If the line of dialog is found to be of type "WH_QUESTION" or "LIST_QUESTION" and of type "IMPERSONAL" it will be considered a factoid question and it is, thus, redirected to a chatbot that retrieves facts from Wikipedia and replies them to the user according to what was asked. If the sentence introduced by the user is tagged as "PERSONAL" then it is redirected to a system (chatbot) which knows how to answer personal interactions by means of pairs of question/answer retrieved from movie subtitles. For all other tags, there is a set of pre-defined answers from which an answer is selected according to the type of question or non-question and then returned to the user. For instance, for the tag "SOCIAL_OBLIGATIONS", if the user introduces the query "obrigado" ("thank you"), the system replies "De nada!" ("You are welcome!") or "Sempre às ordens!"

("Always at your disposal!"). This work assumes that each agent has a specialty and it only redirects a question for the specialist in it. Contrarily, in our work, we assume that all agents can answer all questions.

2.5 Query Answer Matching

One approach to perform a match between query and answer is having a large corpora containing pairs of query/answer. Such a corpora is, for example, the ubuntu chat corpus [20] which is a collection of chat interactions from Ubuntu's IRC support channels (more specifically, the eleven more frequently used) made publicly available containing multi-participant chat logs. This corpora was then modified twice ([21], [22]) to transform it into a format where instead of having a large dataset with multiple dialogues with multiple users, we have a dataset with multiple dyadic dialogues (dialogues between two users). This dataset has over seven million utterances and 100 million words – which makes it an excellent candidate to use with Neural Networks (NN) considering they need huge amounts of data to be trained. Upon having this, the sequences of utterances were transformed in sets of tuples of context, answer and flag in which context corresponds to an utterance in the dialogue and the answer, if it is correct, to the next utterance from another user in the same dialogue. The incorrect answers were randomly taken from the various dialogues in the corpus. Flag indicates whether it is an answer to the context (flag 1) or not (flag 0). The tuples were organized in two groups, one group had an utterance, a correct answer and an incorrect answer (1 in 2); the second group contained one correct answer and nine incorrect answers (1 in 10). Having this configuration, they performed some experiences on two natural language tasks: response classification and automatic generation of next utterance. Of interest to this work is the response classification task, which aims to distinguish whether a response is valid or not to a given utterance. The algorithms they performed experiences with to evaluate this corpus for this task were: Term Frequency - Inverse Document Frequency (TF-IDF), Recurrent Neural Network (RNN) Dual Encoder and Long Short-Term Memory (LSTM) Dual Encoder. To which they chose the Recall@k metric to evaluate their performances. Both for the group of one in two and the group of one in ten, the algorithm that rendered the best results was the LSTM Dual Encoder, obtaining the following results:

- Group of one in two with Recall@1: 86.9%
- Group of one in ten with Recall@1: 55.2%
- Group of one in ten with Recall@2: 72.1%
- Group of one in ten with Recall@5: 92.4%

2.6 Similarity Measures

On the task of text classification, one approach commonly used in Information Retrieval (IR) and NLP is the use of similarity techniques. Similarity is divided in three main groups: string-based similarity, knowledge-based similarity and corpus-based similarity. Where the first corresponds to lexical similarity and the last two correspond to semantic similarity. Lexical similarity focuses on the characters of a string, whilst semantic similarity focuses on its meaning. The first step to apply these techniques usually includes defining a logical framework for the representation of the textual data. These can be sets, vectors, probability distributions, among others [23]. As this is not the scope of this thesis, in this Section we only provide a brief overview of some similarity methods.

A survey made in 2013 [24] sums the main methods used in this context. From these, we detach cosine similarity, n-grams and Latent Semantic Analysis (LSA) (also know as Latent Semantic Indexing (LSI)) which are the most widely known and used. The cosine similarity consists on performing the dot product between two vectors, these vectors are some representation of the texts we are trying to compare. This similarity approach is commonly used with the bag of words model which is a form of representation of text where the text is represented as a set of the counts of its words. One advanced form of the bag of words model is the TF-IDF model which is a calculation of the term frequency in a document over the total number of terms in the same document multiplied by the logarithm of the number of documents (or texts) where a certain term appears over the total number of documents. N-grams are usually sets of n words (or stems) in a given text which are then used to compare the similarity of two texts by calculating the number of similar n-grams. LSA is an approach which uses singular value decomposition to build a matrix of word counts per paragraph. This matrix is then compared by column by calculating the cosine between the angle of any two columns (each column represents a paragraph).

More recently, NN and word embeddings[3] have been used to compute the similarity between sentences and/or texts. One work that calculates similarity using word embeddings is [25] which computes the cost of moving one word from one sentence to another sentence. As for NN, some of the works that use them to compute similarities are [26], [27], [28] which use CNN and Siamese LSTM Long Short-Term Memory NN.

2.7 Agents

The purpose of our work is not to build agents that answer questions, but rather to select an answer from those agents. There are already some agents developed in IST that we can use in our work. Some of these agents were also used in TalKit. They are Talkpedia [29], SSS [19], and Edgar [30].

[3] Representation of words as vectors of numbers.

- **Talkpedia** uses Wikipedia to retrieve answers to factoid questions adding a template to it, so it provides a complete answer instead of just pieces of information.
- **Say Something Smart** is a system that uses movie subtitles to to reply to an user interaction. In this work, the version of SSS used is the one built in a simultaneous master thesis to this one [31].
- **Say Something Smart AMA** is a system that uses the SSS agent framework with a corpus built with the Agência para a Modernização Administrativa (AMA)[4][5];
- **Edgar** is a virtual museum assistant in Palácio de Monserrate (Monserrate Palace) that has expertise about the palace. Edgar was at Palácio de Monserrate answering questions that users had about the palace.
- **Cheat** is an agent built in the context of this work and with some similarities to TalKit's rule based approach, that uses rules to answer queries. It answers salutations with the equivalent salutation (good morning with good morning, hello with hi or hello, etc.); answers or questions by replying the word after the or in the query; answers yes/no questions with yes, no or maybe. If it is none of the above it says that it is not its area of expertise.

[4] https://www.ama.gov.pt/ (Last accessed on 02/01/2019).
[5] A public Portuguese institution that intends to bring some modernization to the administrative processes.

3

Corpora

Contents

3.1 Introduction

In this Chapter, we describe some of the corpora that we used to perform the classification required for our work. We start by addressing the TalKit Corpora (Section 3.2), we proceed to talk about the Li & Roth corpus (Section 3.3), next we describe the Multieight corpus (Section 3.4) and we finish by talking about the SubTle corpus (Section 3.5).

3.2 TalKit Corpora

The TalKit corpora is composed of: the Question vs. Non-Question corpus which aims to classify an utterance as "question" if it is a question and as "NON_QUESTION" otherwise (Section 3.2.1); the Personal vs. Impersonal corpus which aims to distinguish personal questions from impersonal questions using the labels "PERSONAL" and "IMPERSONAL" (Section 3.2.2); finally, the Yes/No vs. Other corpus that uses the label "YN_QUESTION" when a question only requires a yes/no/maybe answer and the label "OTHER" otherwise (Section 3.2.3). The sentences in all the three corpora came from different sources, namely:

- from the translation into Portuguese [13] of the widely used Li & Roth corpus [3];
- from a manual Portuguese translation of parts of the corpora of a chatbot called Just.Chat [18];
- from the B-Subtle corpus, a corpus built from movies Subtitles, as described in [19].

In addition, some sentences were gathered from the web, created by the author of TalKit or suggested by Técnico Students at Taguspark, in a Natural Language course.

The proposed use of this corpora, is to use the Personal vs. Impersonal and the Yes/No vs. other corpus only if the utterance is first classified with the label "QUESTION" using the Question vs. Non-Question corpus.

In the context of this thesis, some corrections were made. The main corrections that were made were on the level of structure of sentences, grammar, ortographical mistakes, fixing incorrect labels, etc. To evaluate the magnitude of the changes, a calculation of the edit distance[1] with the Levenshtein distance formula[2]. After performing the calculation of the Levenshtein distance [32] of the whole corpora (train and test) before and after modifications, the results were:

- 2306 for the question vs. non question corpus (Section 3.2.1);
- 4455 for the personal vs. impersonal corpus (Section 3.2.2);

[1] Distance between two strings according to their differences

[2] Computes the number of insertions, deletions and substitutions that are needed to convert one string into another. Each operation costs 1. The bigger the sum of the costs of these operations, the bigger the edit distance between two strings.

- 78 for the yes/no vs. other corpus (Section 3.2.3).

These corrections and analysis of the corpora resulted in a paper [4] that was published on July 2019.

In the following we describe each one of the three corpora.

3.2.1 Question vs. Non-Question corpus

In this Section we give a brief description of the Question vs. Non-question corpus, namely, the different formulations of questions that were gathered, as well as some examples of the non-questions.

3.2.1.A Questions

Several types of questions were taken into consideration. Besides the usual *direct questions*, we also gave some room to *imperative sentences*[3] that constitute a request for information.

Regarding *direct questions*, the corpus contains several examples of the so called "Wh-questions", that is, questions that contain the keywords "quem" ("who"), "onde" ("where"), "porquê, porque" ("why"), "o quê, qual" ("what"), "o quê, qual" ("which"), "quando" ("when") and "como" ("how"). Examples of such questions are:

- "Quem é Alan Turing?" – "Who is Alan Turing?";
- "Qual é o nome abreviado do Mississippi?" – "What is the nickname for the state of Mississippi?";
- "O que é a viscosidade?" – "What is viscosity?";
- "Quando foi travada a batalha de Somme?" – "When was the battle of Somme fought?";
- "Por que motivo foi inventado o fecho de correr?" – "Why was the zipper invented?".

It was also included in the corpus wh-questions that present a possibility, like an imagined scenario, and then inquire something with that scenario in mind (e.g., "Se o mundo inteiro estivesse ouvindo, que dirias?" – "If the whole world was listening, what would you say?"). In addition, questions whose answer can be a simple "Sim" ("Yes") or "Não" ("No") (e.g., "Gostas de ler?" – "Do you like to read?" or "Andas na escola?" – "Do you go to school?"), including questions that are only one word (e.g., "Jantar?" – "Dinner?") were also addressed. Moreover, questions that contain two possibilities of answer separated by the connector "or" (choice questions), were also added to the corpus (e.g., "Do que gostas mais: factos ou ficção?" – "What do you like more: facts or fiction?").

In what concerns *imperative sentences* that constitute a request for information, or ask for a description or definition of something, several cases were included in the corpus. Some examples are:

[3] Sentences that are an order, an instruction or a request to do something [33].

- “Mencione um cetáceo.” – “Mention a cetacean.”;
- “Diga o nome da organização que é presidida por um Conselho de Segurança.” – “Say the name of the organization that is presided by a security counsel.”;
- “Descreva a aparência do músico Finlandês Salonen.” – “Describe the Finnish music personality Salonen’s appearance.”;
- “Defina cosmologia.” – “Define cosmology.”.

Finally, some cases where several questions are formulated in the same entry were also added. The reason for this is that sometimes people ask several questions related to each other in a row (e.g., “Se tivesses de escolher, qual animal de uma quinta gostarias ser? Porquê? Podes fazer o som?” – “If you had to pick, which farm animal would you like to be? Why? Can you do its sound?”).

In summary, the utterances labeled as questions encompass: choice questions, wh-questions, yes/no questions and imperative sentences.

3.2.1.B Non-questions

The non-questions part of the corpus is constituted of sentences such as:

- “A ideia é os dez formarem um círculo de protecção em torno do possuído.” – “The idea is that the ten form a circle around the possessed.”;
- “O David precisa de ir ao lançamento de um filme.” – “David needs to go to a movie launch.”;
- “Deixou a faculdade de direito, não tem emprego.” – “Left law school, has no job.”;
- “Não é motivo para renegar a família.” – “It is no motive to renegate the family.”.

3.2.1.C Some Statistics

The corpus was randomly splitted in two, one part for training and one part for testing. The training corpus contains 4526 entries, from which 2280 are labeled as “question” and the remaining 2246 as “non-question”. The testing corpus contains 508 entries from which 264 are “non-questions” and 244 are “questions”. Extra details can be found on Table 3.1[4].

3.2.2 Personal vs. Impersonal Questions

In this section we explain what can be found in the Personal vs. Impersonal corpus.

[4] In this and in the remaining corpora, the number of tokens and the number of characters take into consideration punctuation.

Table 3.1: Statistics about the Question vs. Non-question corpus.

	Training Set	**Testing Set**	**Training + Testing Set**
Number of Tokens	42614	4552	47166
Number of Unique words	7741	1509	8253
Average Word Length	4.26	4.31	4.26
Number of Characters	162610	17464	180074
Number of StopWords	13651	1442	15093
Number of Words	36812	3901	40713

3.2.2.A Personal Questions

In European Portuguese, the way personal questions are formulated depend on who we are talking to. Hierarchy and age difference, among others, will lead to more formal/informal conversations. When two people engage in an informal conversation, the second person of the singular is usually used; otherwise the third person of the singular is employed. For instance, if we ask a friend if he likes to read, we would ask “Gostas de ler?” or “Tu gostas de ler?”, but if we asked a person we do not know or has one of the aforementioned differences, we would ask “Gosta de ler?” or “Você gosta de ler?” (being the latter in a more Brazilian Portuguese style). In the English language all these questions translate to “Do you like to read?”. In the corpus for personal and impersonal questions these cases were taken into account. Examples are (the first one is an example of formal speech, and the second of informal speech):

- “Diga algo que fez em criança que os seus pais não sabem.” – “Say something that you did as a child that your parents do not know of.”;
- “Diz 1 coisa que desejavas mudar em ti.” – “Say 1 thing that you wish to change in yourself.”.

Other examples of personal questions that can be found in the corpora are related with: a) situations in which the user presents a scenario and then asks what the other person would do considering it (e.g., “Se tivesses que comer um guaxinim ... como irias cozinhá-lo?” – “If you had to eat a raccoon... how would you cook it?”); b) personal preferences (this can be regarding to movies, food, among other personal tastes) (e.g., “Qual é o teu filme favorito?” – “What is your favourite movie?”); c) family, friends, romantic relationships, among others (e.g., “O que me podes dizer sobre um dos teus avós?” – “What can you tell me about one of your grandparents?”); d) feelings, opinions, beliefs and visions in life: (e.g., “Achas que é correto namoriscar se tens namorado/namorada?” – “Do you think it is ok to flirt if you have a boyfriend/girlfriend”); e) past and/or a person’s experience (e.g., “Indica 1 coisa que te

faz falta das férias quando eras criança." – "State 1 thing that you miss of the vacations you had when you were a child."); f) what a person wears and his/her appearance, habits, skills, personal info/data, personal options, facts about personal life, etc. (e.g., "És bom a escrever na tua língua materna?" – "Are you any good at writing in your mother tongue?"). In conclusion, personal questions are questions about the interlocutor's personal matters, such as his opinions, feelings, memories, home city, friends, among others. If the questions are about the personal life of a person that is not an acquaintance of the interlocutor and if that question is not asking for an opinion, then it is not personal.

3.2.2.B Impersonal Questions

As for the impersonal questions, they are mostly factoid questions extracted from the aforementioned translation of Li & Roth corpus for Portuguese. Some examples include:

- "O que faz com que um tornado gire?" – "What makes a tornado turn?";
- "Quais são os dois países cuja costa faz fronteira com a Baía de Biscaia?" – "What two countries' coastlines border the Bay of Biscay?";
- "Que actor casou com a irmã de John F. Kennedy?" – "What actor married John F. Kennedy's sister?".

3.2.2.C Some Statistics

The personal/impersonal training corpus has 3329 queries, from which 1746 are labelled as "impersonal" and the other 1583 are labelled as "personal". The testing corpus has 369 entries from which 205 are tagged as "impersonal" and the other 164 are tagged as "personal". More detailed statistics about this corpus can be found on Table 3.2.

Table 3.2: Statistics about the personal and impersonal corpus.

	Training Set	**Testing Set**	**Training + Testing Set**
Number of Tokens	33407	3733	37140
Number of Unique words	5714	1173	6099
Average Word Length	4.38	4.29	4.37
Number of Characters	132257	14413	146670
Number of StopWords	10197	1117	11314
Number of Words	29272	3248	32520

3.2.3 Yes/No Questions vs. Other

In this Section we explain what are Yes/No questions and we provide some examples of the questions of this kind that can be found in this corpus. We also present some examples of the questions that cannot be answered with a simple “yes”, “no” or “maybe”.

3.2.3.A Yes/No Questions

Examples of Yes/No questions are:

- “Lês muito?” – “Do you read a lot?”;
- “Gostas de dançar?” – “Do you like to dance?”;
- “Tens dinheiro?” – “Do you have money?”;
- “Ontem choveu?” – “Did it rain yesterday?”.

Notice that, in the set of Yes/No questions, one can find questions constituted of one single word (e.g., “Pizza?”).

3.2.3.B Other

As to the questions labeled as other, they are similar to the ones presented in Section 3.2.1, excluding the Yes/No ones. Under the label “other” we can find questions such as ”Wh-questions”, imperative sentences, among others. Here are some examples (extracted from the corpus):

- “Indique um pesticida.” – “State a pesticide.”;
- “Em que cidade se encontra a Basílica de São Pedro?” – “In what city is Saint Peter’s basilica located?”;
- “És de que clube?” – “Of what club are you?”;
- “Quanto custou o Túnel da Mancha?” – “How much did the channel tunnel cost?”.

3.2.3.C Some Statistics

The training corpus has 320 entries, from which 157 are labeled as “yes/no-question” and the other 163 as “other”. As for the testing corpus it contains 40 entries from which 19 are labelled as “other” and the other 21 are labelled as “yes/no-question”. More detailed information can be found in Table 3.3.

Table 3.3: Statistics about the Yes/No Question and Other corpus.

	Training Set	Testing Set	Training + Testing Set
Number of Tokens	2058	261	2319
Number of Unique words	723	147	787
Average Word Length	4.59	4.69	4.60
Number of Characters	8199	1057	9256
Number of StopWords	530	65	595
Number of Words	1711	216	1927

3.2.4 Inter-annotator Agreement

A random sample of 100 queries was selected from each of the above corpus, rendering for each corpus 50 queries for each label. This sample was given to three different annotators (one external annotator for each corpus) which, in turn, gave their annotation for each query. Upon doing this, the results were compared with the original labelling, made by a single annotator, using the Cohen's kappa coefficient metric (using the implementation provided by Scikit-learn [34]). The results obtained can be found in Table 3.4, and show that, for the Question vs. Non-question corpus, there is a perfect agreement between the annotators. As for the other two corpus there is a near-perfect agreement.

Table 3.4: Inter annotator agreement results.

Corpus	Cohen Kappa Score
Question and Non-Question	1.00
Personal and Impersonal	0.88
Yes/No and Other	0.98

Some examples of sentences in which the annotators did not agree in the Personal vs. Impersonal corpus are:

- **impersonal**: "Porque estamos na Terra?" – "Why are we on Earth?";
- **personal**: "Quando saem os objectos de Halloween nas lojas no teu país?" – "When do the Halloween objects come out in your country's stores?".

As these questions could be answered with both opinions and facts, it is understandable that the ambiguity causes a non-agreement between the two annotators.

The only sentence in which the annotators did not agree in the Yes/No question vs. Other corpus was:

- **yes/no-question**: "Do Stephen King? Um filme de terror?" – "From Stephen King? An horror movie?".

This question could be answered with a simple yes or no, and with a movie, which explains why the annotators did not agree on the label.

3.3 Li & Roth Corpus

The Li & Roth corpus [3] named UIUC dataset is a widely known corpus of factoid questions that contains 5500 questions manually labeled according to the taxonomy discussed in Section 2.3.1. It contains English questions collected from the USC [6], 500 manually built questions, and 1403 from the conferences TREC 8,9 and 10. The distribution of questions per coarse categories can be found in Table A.2 (Appendix A). The distribution of questions per fine categories can be found in Table A.1 (Appendix A). The information in these two tables was gathered by the author of this thesis. The author of this thesis also compiled a table that contains some statistics about this corpus (Table 3.5).

Table 3.5: Statistics about the Li & Roth corpus.

	Training Set	**Testing Set**	**Training + Testing Set**
Number of Tokens	60676	3901	64577
Number of Unique words	9732	1070	10057
Average Word Length	4.36	4.69	4.34
Number of Characters	240667	14236	254903
Number of StopWords	17552	1156	18708
Number of Words	53648	3384	57032

Some examples of this corpus are:

- **DESCRIPTION and manner** – How did serfdom develop in and then leave Russia?
- **ENTITY and creative** – What films featured the character Popeye Doyle?
- **ENTITY and animal** – What fowl grabs the spotlight after the Chinese Year of the Monkey?
- **ABBREVIATION and expansion** – What is the full form of .com?

- **HUMAN and individual** – What contemptible scoundrel stole the cork from my lunch?
- **HUMAN and group** – What team did baseball 's St. Louis Browns become?
- **DESCRIPTION and definition** – What are liver enzymes?
- **NUMERIC and date** – When was Ozzy Osbourne born?
- **LOCATION and state** – What sprawling U.S. state boasts the most airports?
- **NUMERIC and count** – How many Jews were executed in concentration camps during WWII?

Note that in boldface are the coarse and fine categories of each example of question presented. We do not present examples for each fine category as it would be too extensive, but these can be consulted in the full corpus at the website[5] that contains it.

This corpus is of special interest to our work due to the facts that it is publicly available, it has a considerable amount of data, it has a benchmark on classification and it was translated to the Portuguese language (as described in [13]). It constitutes a powerful tool for question classification.

3.4 Multieight-04 Corpus

The Multieight-04 corpus [35] was a corpus built in the context of CLEF 2004 QA track. The focus of this track was multilingual information access. One example of an application of this track, which is also mentioned in the paper cited above, is, for instance, automatic translation of texts. Having this purpose, there was a need for a multilingual corpus to support this task. From this need, the multieight-04 corpus was built having 700 questions translated to 8 languages (one of them being Portuguese) in a XML format. Each of these questions also has an answer (a very objective answer, containing only what was asked without proper sentence formation, for example: the question "Com quem se casou Michael Jackson?" ("Who is Michael Jackson married to?"), the answer is just "Lisa Marie Presley") and an answer type which is what can be expected to be found on the answer out of 8 categories: person, location, organization, other, measure, time, object and manner.

For this thesis, slight adaptations were made to this corpus. First, we parsed the XML file to obtain only the Portuguese questions and answers as this work is focused solely on the Portuguese language. Then, using a model trained in the Portuguese Li & Roth corpus, we predicted the labels that the questions has within that taxonomy. Upon having the predicted labels, these were manually corrected when the author of this work considered that they were wrong. This produced a corpus that had both questions and answers and also labels. The interest of adding Li & Roth's labels to this corpus stems from the fact that the questions encompassed within this corpus and its answers are factoid and thus, can be

[5] https://cogcomp.seas.upenn.edu/Data/QA/QC/ (Last Accessed on: 30/10/2019).

labeled with this taxonomy. One example of a question answer pair that was labelled in the context of this thesis is “Em que ano foi atribuído o prémio Nobel a Thomas Mann? ” (“What year was Thomas Mann awarded the Nobel Prize? ”) which had the label “TIME” in the original multieight corpus and was changed for “NUMERIC” and “date” for this work. As we already have a corpus with 5000 questions to train a model using this taxonomy, we can have a much more accurate label for a question. With the multieight corpus, we can now have labels for the answers and compare them with the labels for the questions obtained from the previously trained model with the Li & Roth corpus. Statistics about the multieight corpus used in this work are presented in Table 3.6 (at the end of the Chapter).

Table 3.6: Statistics about the multieight corpus.

	Whole Corpus
Number of Tokens	7896
Number of Unique words	2448
Average Word Length	4.89
Number of Characters	35172
Number of StopWords	2068
Number of Words	7051

3.5 SubTle Corpus

The SubTle corpus [36] was a corpus built from movie subtitles, where the movies were selected with basis on IMDB[6] lists, and the subtitles for those movies were requested to OpenSubtitles[7]. The criteria for the selection of subtitles was their format and their average user rating. Upon the collection and processing of subtitles, they were organized in the following format:

subtlesample.txt

```
1 SubId - 101330
2 DialogId - 2
3 Diff - 1505
4 I - Preparada para começar a cavar ?
5 R - Penso que sim
```

[6]https://www.imdb.com/ (Last accessed on: 06/06/2019)
[7]https://www.opensubtitles.org/ (Last accessed on: 06/06/2019)

For our work, we are especially interested in question/answer pairs due to our answer ranking component. Upon some analysis of this corpus, we decided that it would be interesting to save the pairs I/R (Interaction/Response) that were a question and its answer. With this in mind, we used a model trained in the Question vs. Non-Question mentioned above, in Section 3.2, to predict which sentences were questions. We also used a rule that checked whether a sentence ended with a question mark or not. We only considered questions that were in the "Interaction" part of the subtitle, so that we could retrieve the "Response" part as being the answer to such question. Upon identifying all of the pairs I/R that were questions and answers, we wrote them in a new file forming a new corpus that contains pairs question/answer. We then made a subset of this corpus that contains only questions that were deemed personal by the model trained with the corpus mentioned above, in Section 3.2. Statistics about the personal SubTle corpus can be consulted in Table 3.7 (at the end of this Chapter).

Table 3.7: Statistics about the personal subtle corpus.

	Whole Corpus
Number of Tokens	6091509
Number of Unique words	109680
Average Word Length	4.24
Number of Characters	21524868
Number of StopWords	1437921
Number of Words	4759209

4

Chattuga

Contents

4.1 Introduction

In this chapter, we describe how this project is built. In Section 4.2 is provided an overview of the whole system, in Section 4.3 we describe the baseline of our work, in Section 4.4 we talk about the plug and play module of Chattuga, in Section 4.5.1 we refer the techniques we used to perform the task of classification, in Section 4.5.2 we detail the module of answer classification and lastly, in Section 4.6, we explain how the Coordinator decides on the answer ranking.

4.2 Chattuga Overview

Chattuga has three main components which are all managed by the Coordinator (as shown in Figure 4.1, at the end of this chapter). These components are the plug and play module (Section 4.4), the query classification module (Section 4.5.1), and the answer classification module (Section 4.5.2). The plug and play module is set up before the Coordinator starts (under number 1 at Figure 4.1). It gets the information from the configuration files that a technical user prepared (this information is represented at Figure 4.1 as the element that says “Corpora, Agents and Answer Classification Methods”). When the Coordinator starts, it sets up the agents (under the number 2 at the figure that is in the end of the chapter). The query classification models are already trained when the Coordinator starts. After the Coordinator sets up the agents, it prompts the user to insert a query. The user introduces a query, for example, the query “Como te chamas?”. This query is routed through the query classification module which provides its predictions for the set of classification labels that this query should have according to the corpora at its disposal. With the base corpora that were detailed in the previous chapter (Chapter 3), these labels would be: “QUESTION”, “PERSONAL”, among others. The Coordinator then proceeds to send the query to the agents and receive their answers. The answers and the query classification information are sent to the answer classification module which has several approaches to score the answers (detailed later, in Section 4.5.2) – this process can be seen at Figure 4.1 where it shows the Coordinator sending the query, answers, and corpora to the classification module and then receiving all the classification information it needs to rank the answers. The Coordinator uses this classification information, which includes the scores attributted by each module of answer classification to the agents answers and it combines them to perform an answer ranking of the answers. When all the information is gathered, the Coordinator provides the answers ordered according to their ranking to the user.

4.3 Baseline

The baseline of our work is the TalKit project (mentioned in Chapter 2). This work had a multi-agent architecture for query answering that redirected a query to an agent that was an expert in that type of query (for example: a query that was classified as “PERSONAL” was redirected to an agent expert in “PERSONAL” queries). It did not take into consideration the agent’s answer. In Chattuga, it is assumed that all agents can, potentially, answer all questions. With that assumption, it was taken as baseline the work developed in TalKit and iterations were made to it. Chattuga started by having what TalKit had: a process of choosing an answer based solely on the agents expertise. This process, however, was already slightly different, because whilst in TalKit a query was only given to a single agent, in Chattuga the query goes to all the agents and each agent gets a score for their expertise (or lack thereof) in the labels that the query has from the classification process. As Chattuga evolved, it started to have three more modules to help the Coordinator with the answer ranking. The three modules have in consideration the answer of the agent instead of their expertise which is not always the best metric to choose an answer. Beyond the answer ranking modules added, the system developed in the context of this thesis, also has a plug and play module which allows for further addition of agents and corpora and also for inner system configurations. Another addition to the baseline, is the correction of the corpora that was developed for the TalKit thesis, as described in Chapter 3. The previous system used some answer templates to reply to Yes/No questions, or questions and social talk (examples of social talk are: “Olá” (“Hello”), “Bom dia” (“Good Morning”)). In Chattuga, these templates were used to create the Cheat Agent, mentioned previously, in Chapter 2. Last but not least, it was used some of the code written for TalKit to embed the agents in a multi-agent system and, for Chattuga, a Python file that uses this code was created to enable the integration in its framework.

4.4 Plug and Play Module

A technical user that wishes to utilize Chattuga in his own tasks can do both external additions to the system, by using the plug and play module to add new agents and corpora, but also control its inner components by turning on and off the answer ranking modules available. In this Section, it is detailed how this module (represented in Figure 4.2) works.

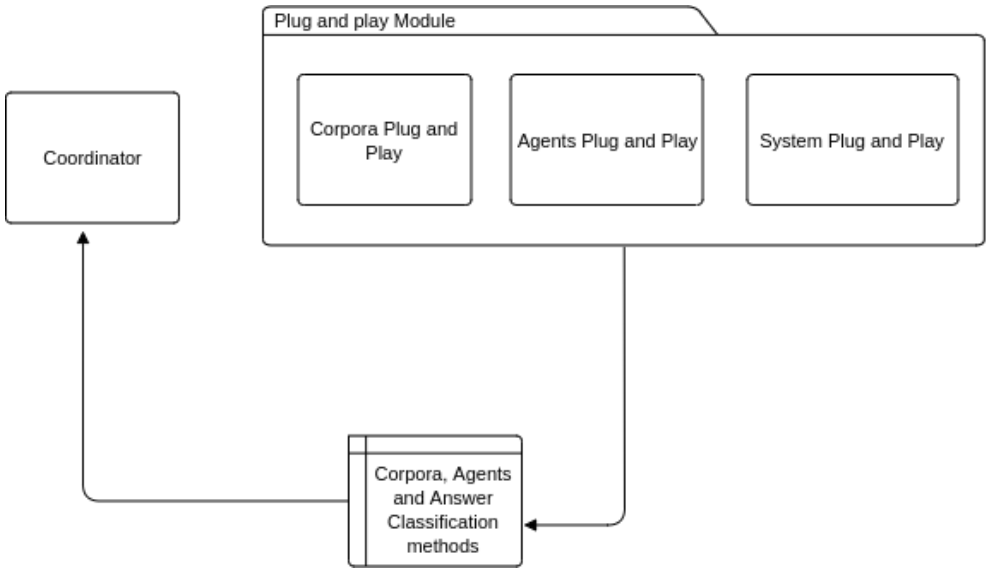


Figure 4.2: Simplified plug and play module representation.

4.4.1 Agent Plug and Play Module

To add a new agent, the user simply needs to have a Python file (representing the new agent) that has a method which receives as argument a query and returns an answer to that query. This file must be encompassed in a folder that contains everything that is needed for the execution of the agent. This folder must be located in the "agents" folder of the Chattuga project. Upon performing all of these actions, the technical user must then proceed to add, in the "agents_config.xml" file, the required configurations. For example, to add the agent SSS, these were the configurations inserted in the "agents_config.xml":

tables_and_code/agents_config.xml

```
1   <agent>
2     <classname>SaySomethingSmart</classname>
3     <dialogMethod>dialogue</dialogMethod>
4     <folderName>sss</folderName>
5     <labels>
6       <label score='0.7'>QUESTION</label>
7       <label score='0.7'>PERSONAL</label>
8     </labels>
9     <escapeSentences>
10      <escapeSentence>Não sei responder a isso</escapeSentence>
11    </escapeSentences>
12  </agent>
```

Where the configurations in that example file mean:

- **classname** – the correct name of the file where the method that receives a query and returns an answer to that query is. In this case, the classname is “SaySomethingSmart” which is the name of the Python file where the dialog method is.
- **dialogMethod** – The name of the dialog method that receives a query and returns an answer to that query. In this case the name of that method is “dialogue”.
- **folderName** – The name of the folder where the aforementioned file is and all the others that the new agent needs has to be written. This folder has to be located in the agents folder. In this case, the folder name is “SSS” and it contains the file “SaySomethingSmart.py”.
- **labels** – Contains the sub-fields “label” where in each sub-field it should be a label where the agent is an expert. This label should also be in the corpora. The element “label” has to contain an attribute named “score” which corresponds to a score of “how good” that agent is in the said label. For this example of SSS, the labels in which it is an expert are “QUESTION” and “PERSONAL” and it is an expert in them with a score of 0.7.
- **escapeSentences** – Contains the sub-fields “escapeSentence” where in each of these sub-fields is a sentence that the agent uses to “escape” the query that the user inserted because it was unable to find an answer. In the case of SSS, the agent only has one escape sentence which is “Não sei responder a isso”.

These configurations are then used to perform the import of the agents and the call to the corresponding method through reflection [1]. To be able to perform these imports, an “__init__.py” file had to be created in the agents directory that dynamically imports all agents and its Python files. The inner representation of this configuration file within Chattuga is a dictionary containing the informations in that file.

4.4.2 Corpora Plug and Play Module

For the dynamic addition of corpora, the technical user needs to have in consideration some factors which are:

- The type of hierarchy that the corpora being added has. In Chattuga, two types are accepted. The first one follows the same logic as the Li & Roth corpus where there are two labels for the same query and a classifier needs to first classify a query with the first label, and only then classify it

[1] The ability of a program to reason with itself in runtime.

with the second label using the classifier trained with the utterances labelled with the first label (for instance, a question can only be classified as “animal” if it first was classified as “ENTY”). The second follows the approach of the TalKit corpora where a label from one corpus indicates whether a second corpus is used (e.g. if a query is not labeled as “QUESTION” by a first corpus that distinguishes utterances that are questions from utterances that are not, it does not make sense to try to label it with a second corpus that classifies types of questions).

- The type of corpus being added to the system. For instance, the Monserrate corpus contains almost exclusively questions about Monserrate’s palace and labels about Monserrate and it does not make sense to have a question labelled with this corpus if the query is about a generic domain.

In the corpora configuration file, which is showcased below, the fields “corpusType”, “hierarchyType”, “sublabel” and “parentLabel” were added to tackle the factors discussed in the above itemization. Another important factor is that, as there can be corpora of questions as well as corpora of answers, the fields that are detailed next should be encompassed within the tags “question” and “answer” depending on whether it is a corpus of questions or answers, respectively. Examples of a corpora configuration file are:

tables_and_code/corpora_config.xml

```
<corpus>
  <name>PI</name>
  <path>/novo corpora/query/Treino/personalImpersonal.csv</path>
  <corpusType>generic</corpusType>
  <parentLabel>QUESTION</parentLabel>
  <vectorizer>
    <name>TfidfVectorizer</name>
    <param>encoding='utf-8'</param>
    <param>ngram_range=(1,2)</param>
    <param>tokenizer=TweetTokenizer().tokenize</param>
  </vectorizer>
  <classifier>
    <name>SVC</name>
    <param>kernel="linear"</param>
  </classifier>
</corpus>
```

tables_and_code/corpora_config.xml

```
<corpus>
  <name>LR</name>
  <path>/novo corpora/query/Treino/liRoth.csv</path>
  <corpusType>generic</corpusType>
  <hierarchyType>sublabel</hierarchyType>
  <vectorizer>
    <name>TfidfVectorizer</name>
    <param>encoding='utf-8'</param>
    <param>ngram_range=(1,2)</param>
    <param>tokenizer=TweetTokenizer().tokenize</param>
  </vectorizer>
  <classifier>
      <name>LinearSVC</name>
  </classifier>
  <sublabel>
    <vectorizer>
      <name>CountVectorizer</name>
      <param>encoding='utf-8'</param>
      <param>ngram_range=(1,1)</param>
      <param>tokenizer=TweetTokenizer().tokenize</param>
    </vectorizer>
    <classifier>
      <name>LinearSVC</name>
    </classifier>
  </sublabel>
</corpus>
```

The complete list of fields required are presented next:

- **name** – Where it should be written the name of the corpus to be added.
- **path** – The path to the new corpus.
- **classifier**/**vectorizer** (*optional*) – If the technical user wishes to apply a specific vectorizer or classifier to classify the new corpus, he just needs to insert the tag classifier or vectorizer and inside those, the name and/or parameters of the classifier or vectorizer, as shown in the examples above. All Scikit learn vectorizers and classifiers are accepted. If this field is not used, the default classifier and vectorizer (LinearSVC and CountVectorizer) will be used for that corpus.

- **corpusType** – In this parameter it has to be specified if the queries in the new corpus have a generic nature or a very specific one. This field mitigates the problem discussed above, about classifying queries with specific corpus such as the Monserrate corpus.

- **hierarchyType** (*optional*) – This can only have two values, that the technical user should add. Those are “sublabel” and “subcorpus”. This indicates the program whether you want to have a hierarchy of classification of corpus or of labels. As explained above, Chattuga accepts two types of hierarchy. If the hierarchy is like the Li & Roth hierarchy then it is “sublabel”; if it is as the TalKit corpus it is “subcorpus”.

- **parentLabel** – This element should only be used when the field “hierarchyType” is set to “subcorpus” and is used to indicate the system which label should prompt a further classification with the corpus being added. For instance, if we want to further identify a question to learn if it is a question that can be answered with a yes/no or if it is another kind of question, we should, within the element that identifies the Question vs. Non_Question corpus, have another element for the Yes/No vs. Other corpus which in its turn should have the element “parentLabel” that should encompass the word “QUESTION”.

- **sublabel** (*optional*) – This field should only be used if the hierarchyType is “sublabel” and its purpose is to contain the fields classifier and vectorizer if a specific vectorizer or classifier is to be used for the sublabels.

It should be noted that, for different hierarchy types, a different approach to classification is applied. If the hierarchy type is “subcorpus”, then a classifier is trained for each corpus and then, in the prediction phase, the model for the subcorpus is used only if its parentLabel has been already identified to be the query’s label. For the hierarchy type “sublabel”, a classifier is trained on the labels of the first level of the hierarchy, and then, for a sublabel, a classifier is trained for each parent label. For instance, in the case of the Li & Roth corpus there are six coarse categories and fifty fine categories. In this case, a classifier is trained to identify which coarse category is, and then a classifier for each coarse category is trained for the fine categories. This will render seven models: one for each coarse category and one for all the coarse categories. To better understand, lets suppose the query “Quem é o arqui-inimigo do Snoopy?” enters the system. The system then proceeds to identify that query with a coarse category using the model trained in all coarse categories. It then uncovers a coarse label for that given query which is “HUMAN”. Upon having that coarse label, it calls the model that was trained with the fine categories associated with that coarse label, that is, the model trained with the queries labeled as “HUMAN”, and uncovers a fine category (“individual”) for the query based on its coarse category. Also worthy of note is that Chattuga creates a model for each “specific” corpus that is trained with utterances from the “specific” corpus and utterances from a generic corpus and it learns to distinguish if a query is specific or generic

before further classifying it with the “specific” corpus labels.

4.4.3 System Plug and Play Module

The third plug and play’s module purpose is to provide the possibility to the technical user to choose which methods it uses in the answer ranking process (explained in the next Section). If the technical user considers that one of the modules is not useful or it even is prejudicial to his purposes he has this way to turn them off. As there are four available answer classification modules ((1) – query/agent label match module; (2) – personal answers module; (3) – impersonal answers module; (4) – query/answer label match module), the configuration file for the system has four elements, which correspond to the answer ranking methods available. To use one of them, the user just needs to write “True” inside the corresponding element. By default the system uses all three methods. They are all set to “True”. Additionally, for the personal answers module (explained in the next section), as it has four available methods, the technical user needs to input in the attribute of the corresponding element, a string containing the methods or method which it intends to use separated by “|”, in case it uses more than one. The default configuration file for the system has the following appearance:

tables_and_code/system_config.xml

```
<system>
  <query-agent-label-match>True</query-agent-label-match>
  <answer-classification-impersonal>True</answer-classification-impersonal>
  <answer-classification-personal
    sim_measure='tfidf_normal'>True
  </answer-classification-personal>
  <query-answer-label-match>True</query-answer-label-match>
</system>
```

As it can be seen in the example file, there is a xml field for each of the enumerated answer classification modules and they are all set to True, which means that the Coordinator will use all available methods to rank the answers of the agents. As to the similarity measure(s) that the Coordinator will ask the personal answers module to use, it will be the TF-IDF normal as that is the only one specified in the “sim-measure” property in the field of this module in the system configuration file.

4.5 Classification Module

In Chattuga, there are two main classification modules, which are the query classification module (detailed in Section 4.5.1) and the answer classification module (explained in Section 4.5.2). A representation of this module can be seen in Figure 4.3.

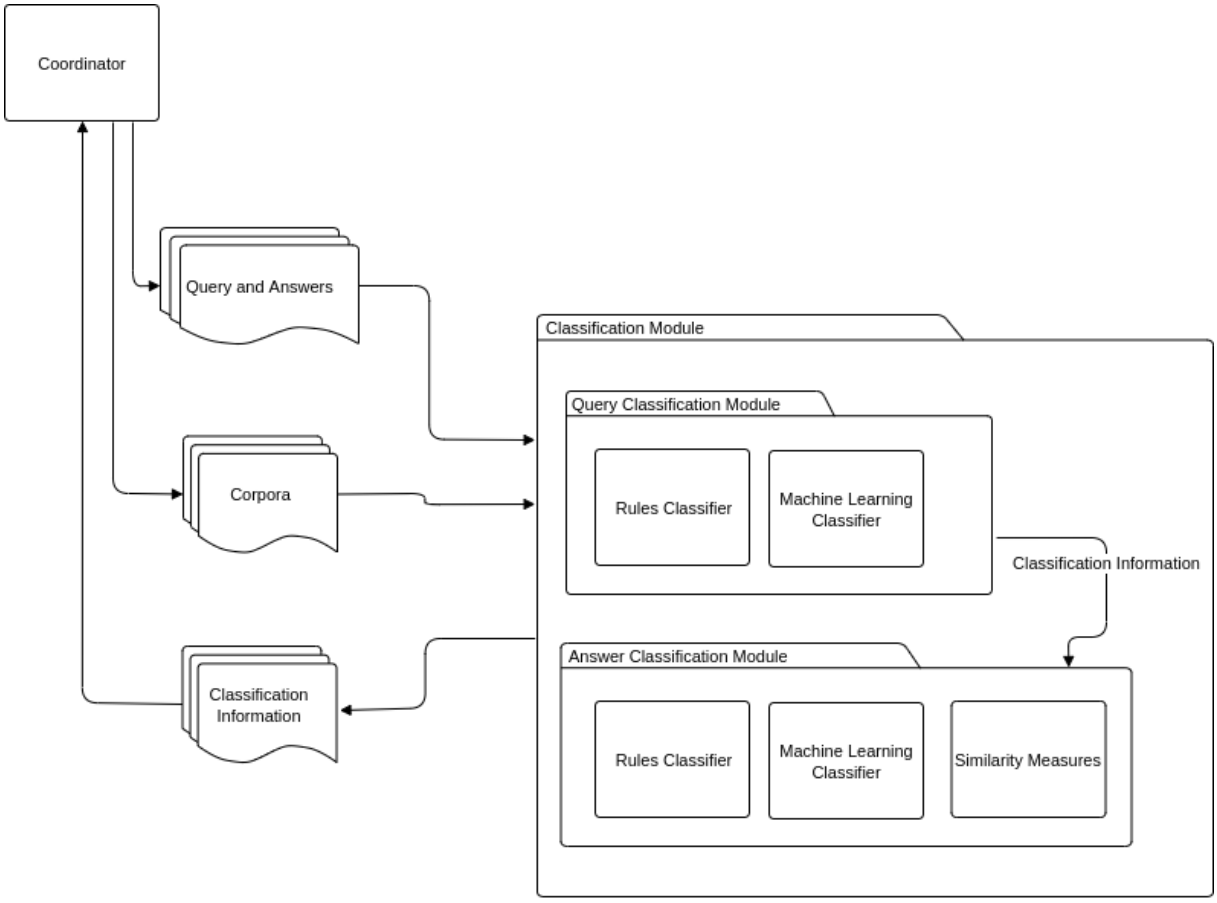


Figure 4.3: Simplified classification module representation.

4.5.1 Query Classification

Chattuga, without any modifications to the configuration files, uses Support Vector Machine (SVM) with a linear kernel, using unigrams or unigrams+bigrams as features (experiments conducted in this thesis have shown that these are the pipelines that scored higher in accuracy. These experiments are detailed later, in Chapter 5), trained with the Question vs. Non-Question, the Yes/No vs. other corpus, the Personal vs. Impersonal corpus, the Li & Roth corpus and the Monserrate corpus[2] to classify queries. The system also has handcrafted rules, to classify questions as "WH_QUESTION", "OR_QUESTION"

[2] Edgar's corpus containing informations about Monserrate.

and “LIST_QUESTION”. This rule based classifier is very simple: it only looks for words that may indicate that we are in the presence of one of these questions:

- For the “WH_QUESTION” label, it searches for “como” (“how”), “quando” (“when”), “quem” (“who”), etc.
- For the “LIST_QUESTION” we look for “Indique” (“Indicate”), “Mencione” (“Mention”), “Diga” (“Say”), among others.
- Finally, for the “OR_QUESTION” we simply search for the word “or”. This rule based architecture is very similar to the one of TalKit.

4.5.2 Answer Classification

To understand, among a set of answers from different agents, which one is the most adequate to a given query, there are several approaches that can be followed. One is related to knowing the agent’s area of expertise (for example, the agent Edgar is an expert in all the labels that concern Monserrate’s palace). In this case, if the query is classified with a label that is related with that expertise, we could assume that an answer from that agent will be more valuable than the other agents’ answers. Another approach would be to have corpora with queries and answers labelled with the same set of labels. Having these corpora and using machine learning algorithms, one could obtain a classification on the query and on the set of agent’s answers and, thus, perform a direct matching. If the label of the query and the label of the answer matched, that answer could be considered more valuable. Without having this kind of data, considering that most labelled corpora available contain only queries, another kind of corpora could be used: a corpora that contains pairs query/answer. This kind of corpora could be used in two manners:

- If the corpora is very large, it could be used with neural networks to obtain a classification of whether that answer is adequate to that query or not (this approach was explained in Section 2.5).
- If we knew that the corpora had a specific characteristic, for instance, that it has only “PERSONAL” utterances, we could compare the agents answers when the user inserts a query that is considered “PERSONAL”, with the whole corpora using similarity measures. The most similar answer to that corpora would then be considered the most adequate.

In Chattuga, we use all the aforementioned strategies:

- We compare the labels of a query with the labels of expertise of an agent (Section 4.5.2.A): for instance, we attribute a bigger score to SSS if the query is classified as “PERSONAL” because personal queries are its area of expertise;

- We use rules and machine learning approaches to determine labels of an answer and compare it to a query (Section 4.5.2.B and Section 4.5.2.C): for example, an agent's answer that has two labels in common with the user query gets two points;
- Finally, we use similarity measures to compare agents' answers to a corpus of answers (Section 4.5.2.D): e.g. we compare agents' answers to a query classified with the label "PERSONAL" with a corpus of personal answers. The score each answer receives is its degree of similarity with the corpus.

The way in which we use them is detailed next.

4.5.2.A Query Labels/Agent Labels Matching Module

The first method of ranking the agent's answers is heavily drawn from TalKit, as mentioned before. This method uses the information it has on the agent's expertise. From the plug and play module explained before, the labels that each agent has an expertise on are drawn. These have a score that pertains to a measure of "how good" an agent is in answering utterances classified with a certain label (for instance, Edgar has a score of 1.0 in all Monserrate's corpus labels). The query is also classified with a machine learning module to get its labels. Once we have both the query's labels and the agent's labels, we sum the scores of each agent's label that matches the query's label. This will render a final score that will be used to rank the answers. For instance, in the previous section (Section 4.4), where the agent's plug and play module was discussed, it was mentioned that the agent SSS has an expertise score of 0.7 both for queries with the label "QUESTION" and the label "PERSONAL". This means that if a user inserts a query that has these two labels, SSS receives 1.4 points for being an expert in the domain of that query. However, if the query is classified with only one of those labels, SSS only receives 0.7 and so on and so forth. The Coordinator receives these scores and uses them to make a decision on the answer ranking.

4.5.2.B Answer Labels/Query Labels Matching Module

The module of answer labels/query labels matching is quite similar to the query classification module. The only thing that is done in this module is to use the answer corpora that are inserted in the corpora configuration file to train a model and perform a classification of the agent's answers with that corpora. As the purpose of adding a corpora of answers is for it to have answers that are annotated with the same labels as one of the corpora of questions, it only gives a score above 0 to an answer if it is found to have labels in common with the ones found for the query in the query classification phase. If there are labels in common, the answer being analysed receives 1 point for each label it has in common with the query. For example, the query "Quantas janelas tem o palácio?" ("How many windows does the palace have?") is classified with, among others, the label "MONSERRATE_INTERIOR" from Edgar's corpus, and the

answer “O palácio tem vinte e cinco janelas ogivais nos seus dois pisos principais e sete janelas mais pequenas ao nível do piso superior dos torreões.” (“The palace has twenty-five arched windows on its two main floors and seven smaller windows on the towers’ top floor.”) is classified with the same label. In this module, this answer would receive 1 point due to the label in common between query and answer.

4.5.2.C Impersonal Answers Module

This module uses the multieight corpus described earlier, in Section 3.4. However, as this corpus is very small, its accuracy cannot be relied to obtain a classification. For this reason, it was only used when a classification with rules could not be found. In this module, agent’s answers are iterated over to find clues to ascertain whether they have answered the question or not. To perform this operation, the query’s labels are used, more specifically, the ones that come from the classification with the Li & Roth taxonomy.

The first approach that this module tries is to use the Wikipedia api for Python[3] to send the ngrams of the answers to wikipedia and get a summary of the articles it finds. The impersonal answers module then searches for the fine category obtained in the classification of the query within that summary. The reason for this is that if, for example, a search is performed in Wikipedia about “Girassol” (“Sunflower”) (a plant), usually the summary returned contains the word “plant” which is one of the fine categories of the Li & Roth taxonomy. For fine categories that are more broad such as “cremat” that encompasses books, movies, plays, among others, a list of words to search was compiled (in the case of “cremat”, the aforementioned examples were included). If a user inputted the query “Que planta contém a palavra sol no seu nome?” (“Which plant contains the word sun in its name?”) and it was classified correctly as “ENTY:plant”, if an agent answered “É o girassol!” (“It is the sunflower!”), this module would be able to retrieve the word plant from the wikipedia summary and thus attribute the correct fine category to the answer. If this wikipedia method takes too long or is unable to find the category, the impersonal answer module proceeds to use another module based on the work in [37], which, depending on the coarse category, uses different methods:

- **Coarse category Entity**: The module uses nltk’s WordNet [14] [38] for the Portuguese language to find all of the hyponyms of the hypernym corresponding to the fine category. For instance, for the fine category “animal” it will look in the synset tree for all the hyponyms of animal, then it will search in all the words of the agent’s answer for a word that matches an hyponym of animal. If it finds one it returns the category “ENTY:animal”. This has five exceptions: creative, other, letter, term and word to which wordnet cannot be used, these are classified using the model trained on the multieight corpus.

[3] https://pypi.org/project/wikipedia/ (Last accessed on: 27/10/2019)

- **Coarse category Location**: For the location coarse category, we use Spacy's Named Entity Recognizer (NER) to identify named entities in the answer, due to the fact that usually this method is able to identify locations. Then, if the fine category is country, city, mountain or state, we use the same approach as described above to check whether one of the hyponyms of the fine categories matches a word in the answer. For example, if a user inserts the query "Qual é a montanha de maior altitude no Mundo?" ("What is the highest mountain in the world?") and an agent answered "A montanha de maior altitude é o monte Evereste." ("The highest mountain is mount Everes.t") the NER would extract "Evereste" which is then used as the hyponym in wordnet and the fine category "montanha" ("mountain") as the hypernym, and then a search in the hyponyms of the hypernym is done to check if the hyponym "Evereste" is there. If a classification is not obtained using these steps, the named entity extracted from the answer (in the previous example, this named entity would be "Evereste") is classified using a SVM trained in the multieight corpus. Only the named entity is classified with the multieight corpus trained model due to the fact that the answers of the multieight corpus are just names or dates or small answers, as mentioned in Section 3.4, which means that if just the named entity is classified with this model, there is a bigger chance of getting a more accurate classification.

- **Coarse category Human**: Similarly to the approach followed in the location coarse category, in the human category we also perform an extraction of named entities and then classify that named entity with the SVM trained on the multieight corpus. We only classify the named entity due to the fact that the multieight corpus has very small and direct answers and thus, using only the named entity there is a higher chance of a correct classification. For example, for the answer "O primeiro rei de Portugal foi D. Afonso Henriques." ("The first king of Portugal was D. Afonso Henriques.") to the query "Quem foi o primeiro rei de Portugal? ("Who was the first king of Portugal?"), the name "D. Afonso Henriques" would be extracted by the NER and classified with the multieight module. The reason for only classifying the named entity with the multieight trained classifier is the same as the one described in the previous topic about the category Location.

- **Coarse category Abbreviation**: In the fine category "abbreviation" we iterate over all words of the answer to determine whether one of them matches a regular expression that checks whether the word is a set of two or more capitalized letters separated by dots or not. As for the fine category "expansion", we extract from the query the abbreviation using the aforementioned regular expression and then, we look in the answer for a set of words whose first letter correspond to the abbreviation. For instance, in the query "O que é que a sigla NASA significa?" ("What does the abbreviation NASA mean?") we extract NASA. Then for the answer "A sigla NASA quer dizer National Aeronautics and Space Administration." ("The abbreviation NASA means National Aeronautics and Space Administration.") we iterate over the words of the answer and look for the set of

words that matches the letters “NASA”, if we find those, the classification with this coarse and fine category is returned.

- **Coarse category Numeric**: In this category we used regular expressions, spacy’s PoS tagging and some rules to verify whether a number was present in the answer or not. For the fine category “date” we used regular expressions to verify numbers separated by “/”, “-” or “.”. Additionally, in this fine category, we also check if the answer has keywords such as the months of the year, the days of the week or simply the words “today”, “yesterday”, among others. For the fine category “code” we simply check if it matches a Portuguese zip code. Finally, for the categories “distance”, “temperature”, “size” and “weight”, we check if the answer contains any unit that could point to one of these categories using wordnet.

As we could not cover all methods with rules, when a question category is not found with this rule based classification, this module uses a SVM trained on the multieight corpus to perform the classification of the answer.

4.5.2.D Personal Answers Module

Working with the resources that we possess for the Portuguese language, that is, the corpora presented in the previous Chapter, and due to the lack of corpora to distinguish personal answers from impersonal answers, i.e., corpora with answers labeled with “PERSONAL” and “IMPERSONAL”, as we have for questions, which would allow us to compare the answers’ labels with the question’s labels, we, in this work, tried two methods to be able to use all the resources available. The first method we tried to employ to determine which answer was better for an user query was the one followed in [22] (detailed in Section 2.5). However, this method, although very adequate for our work, takes a long time to train and to produce good results. We ran the method on our personal answers corpus. It took over a week just to run one epoch which rendered recall@5 in a group of 10 answers (1 right answer and 9 wrong answers) of a little over 50%, which is not very good. The best results should come after epoch 2, unfortunately due to the process of the neural network taking too much resources for far too long it had to be terminated before it could finish this epoch. For these reasons, we could not use this method on this work. Instead, as we only contain a corpus with one example (“PERSONAL” answers) of answers, we decided to use an approach that simply compares each agent’s answer with a corpus that contains personal answers (the subset of the SubTle corpus described in Section 3.5). If a query is classified as “PERSONAL” and among the set of answers we can find one that is “PERSONAL” as well, we can give a higher score to that answer. We do this by using some similarity approaches to compare an answer to the corpus of personal answers, as if an answer is similar to a the aforementioned corpus of personal answers, it can be argued that the more similar it is, the most likely it is that that answer is personal. These similarity

approaches are described next.

Word embeddings + Mean Embedding Vectorizer + Cosine similarity

For this method and the following one, we used an implementation (with slight modifications) available online[4] to use word embeddings with Scikit learn functions. For this, the developer made two custom vectorizers, the **Mean Embedding Vectorizer** which computes the mean weights of word embeddings vectors and the **TF-IDF Embedding Vectorizer** which computes the mean values of the TF-IDF weights of the word embeddings vectors. In this section we will focus on the mean embedding vectorizer. This vectorizer simply computes the mean values of the word vectors of the word embeddings model (trained on the personal answers corpus). Upon transforming both the answers and the personal answers corpus, we compare the similarity between them using the cosine similarity.

Word Embeddings + TF-IDF Embedding Vectorizer for Embeddings + Cosine similarity

Similarly to the previous approach, we use the aforementioned implementation which computes the mean values of the TF-IDF weights as well as the word vectors weights. With the vectors corresponding to the personal answers and the answers transformed with this vectorizer we perform the cosine similarity to determine the closest answer to the personal corpus.

LSI similarity

Another approach that can be used to test the similarity between an answer and the corpus of answers is **Latent Semantic Indexing**, explained before, in Section 2.6. Using gensim's (Section 5.2.4) implementation of this similarity approach, we first train the model with the corpus of answers, then we create a matrix containing all the similarities that came from the lsi model. Using this matrix, we then compare each agent's answer to this matrix and obtain a similarity for each one of them.

Regular TF-IDF Vectorizer + Cosine similarity

The last and the default approach of our system is to use the regular TF-IDF vectorizer from Scikit learn, to get the vectors of weights to each word and then perform the cosine similarity. This method is the default because of its speed. It takes three seconds to compute four agent's answers as opposed to the more than ten seconds that the other methods take (they take a little over ten seconds for four

[4] http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/ (Last accessed on: 09/09/2019)

agents using parallel computing, otherwise they would take more than that). As this method is fairly good at performing the similarity, we considered that it could be a default method considering the trade off time/effectiveness.

4.6 Coordinator's answer ranking decision

To return a ranked list of answers to the user, the Coordinator checks the system's configuration file to check which answer classification modules it can use and, in case the personal answers module is active, the similarity measure or measures it can use. With this information, the Coordinator calls these modules and sums the different scores it receives from each one of them. The answers are showed to the user ordered by their score, from highest to lowest. In case there is a draw, this information is given to the user as well.

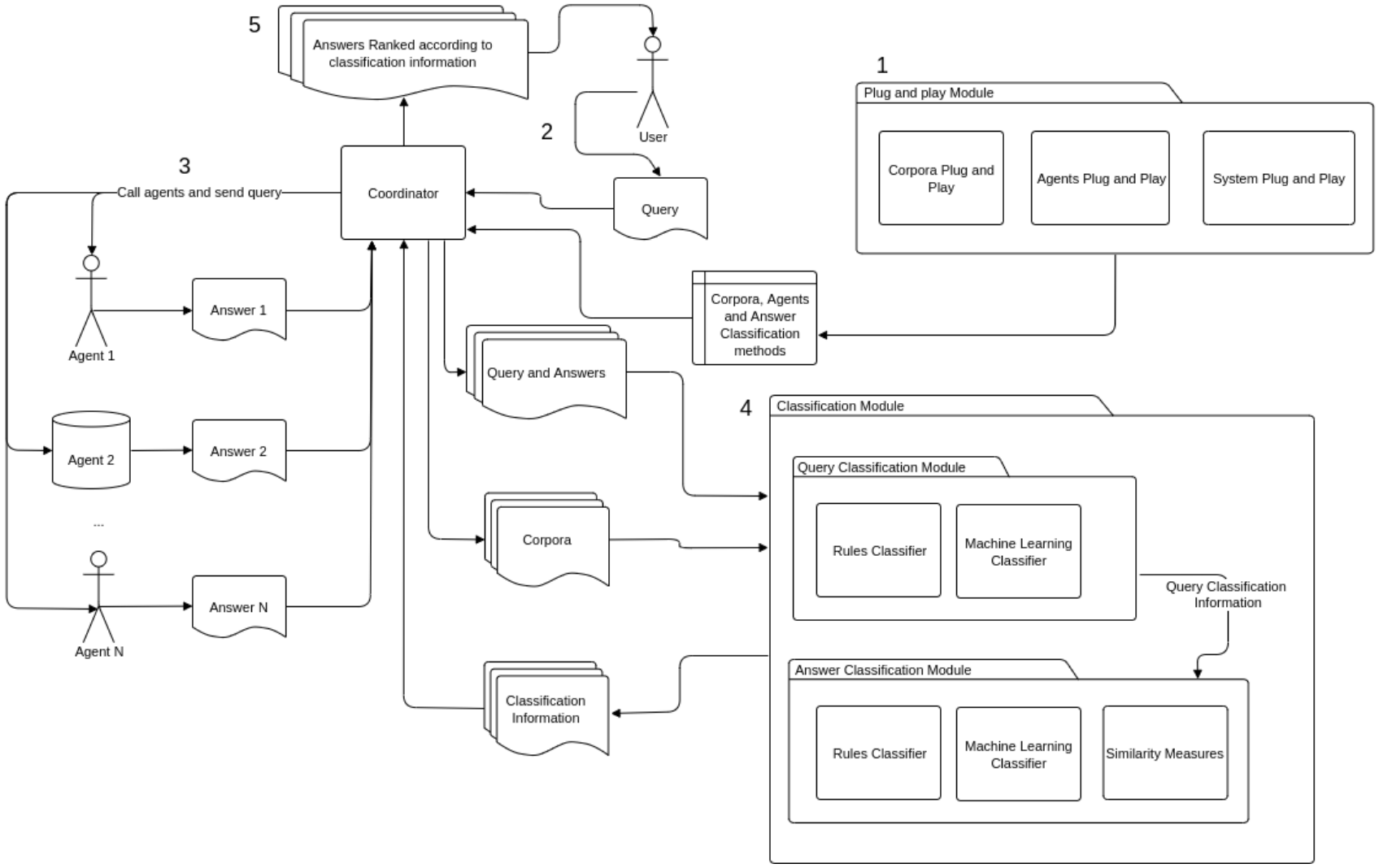


Figure 4.1: Simplified Chattuga's general architecture.

5

Evaluation

Contents

5.1 Introduction

The Chattuga framework consists of several functionalities which connected make a meta-conversation system. As the focus of this work is not the agents that integrate it, in this evaluation we focus on its true functionality: how good is Chattuga in the **ordering of answers according to their relevance** and how good is it in the **integration of new agents and corpora**. Thus, we have three main characteristics on which we base this evaluation:

- **The query classification module** – In this module, the user's intent is classified and it is further used to discern between the agent's answers. This helps the system in the **ordering of answers according to their relevance**.
- **The plug and play module** – This module has three parts: the agents plug and play part to add agents; the corpora part which is to add corpora; finally, the system part that controls which answer ranking methods are used. The first two parts of this module (agents and corpora addition) will allow the system in the **integration of new agents and corpora** and the third part (system) will help the system in the **ordering of answers according to their relevance**.
- **The answer ranking process** – This process is consisted of four modules of answer classification ((1) – query/agent label match module; (2) – personal answers module; (3) – impersonal answers module; (4) – query/answer label match module) that each produce a score for each agent's answer which are then summed by the Coordinator. The system uses this information to perform the **ordering of answers according to their relevance**.

In Section 5.2 we provide a brief explanation of the tools that were used in this work. In Sections 5.3, 5.4 and 5.5 we describe how the evaluation was performed in each of these modules.

5.2 Experimental Setup

To perform the implementation of the system and its evaluation, several tools for Python were used. These are detailed in this section.

5.2.1 Scikit-learn

Scikit-learn [34] is a package for Python that contains several implementations of state-of-the-art algorithms for machine learning that are both supervised and unsupervised. It contains not only a classification component, but also models such as bag-of-words that can be fed to classification algorithms. It also provides implementations for evaluation metrics such as accuracy, recall, precision, among others. This

tool is very important for the whole functionality of our system as we use its classification algorithms, its vectorizers, similarity tool, among others.

5.2.2 Pandas

Pandas [39] is a library for Python that provides a set of tools to structure and analyze data. The most interesting feature of this tool for our work is its functionality to import files and store them in data frames that are structured and also its integration with scikit-learn which allows us to easily have the data in the format required by scikit in its functions. We use this tool for our corpora.

5.2.3 NLTK

Nltk [38] is a toolkit for Python that provides lexical resources to process natural language. These include tokenization, stemming, parsing, tagging, among several others. This toolkit includes resources for the Portuguese language such as Floresta Sintática [40]. We use its tokenization tool, WordNet, and set of stopwords.

5.2.4 Gensim

Gensim [41] is a Natural Language Processing software framework which is focused in scalability and ease of use. This tool provides implementations of several natural language processing tools such as the vector space model, latent semantic analysis, word embeddings among others. In our work, we use the tools related to word embeddings, Latent Semantic Indexing similarity and Vector Space Model. We did made some experiments with the word movers distance similarity. However it was proven to be very slow and not very scalable so we have decided not to use it.

5.2.5 Spacy

Spacy [42] is an open source library for natural language processing in Python that supports more than 52 languages, including Portuguese. For our work, we use its tokenizer and the named entity recognizer.

5.2.6 Numpy

Numpy [43] is a scientific computing tool that performs array operations very efficiently. In Chattuga, this tool is used mainly implicitly, within the other tools. It was used an implementation[1] from another developer to perform word embedding operations that uses numpy to perform the word vectors operations. We also used the mean function to obtain the mean similarities.

[1] http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/ (Last accessed on: 09/09/2019).

5.3 Query Classification

To evaluate the performance of the different corpora (Question vs. Non-Question, Yes/No vs. Other, Personal vs. Impersonal and Li & Roth) we conducted experiences in the suggested partition of train and test corpus with several scikit-learn algorithms of classification (DecisionTreeClassifier, KNeighborsClassifier, MultiLayerPerceptron, all Naive Bayes' algorithms and Support Vector Machines with different kernels), all the vectorizers (CountVectorizer, TfidfVectorizer and HashingVectorizer) and with these together with the TweetTokenizer of NLTK. We experimented with Unigrams, Bigrams, Trigrams and combinations of them. When possible we maintained the default parameters of the classifiers and of the vectorizers. In some cases this was not possible as it would raise a warning or an exception (for instance, in some combinations of the HashingVectorizer with a classifier, an exception would be raised and we had to set the parameter "non-negative" to True). The algorithm that achieved the best results in all the corpora was the SVM with linear kernel. The results are presented next, in Table 5.1:

Table 5.1: Results of the best pipeline (combination of features, classifier and vectorizer) of classification for each corpus.

	Accuracy	**Features**	**Vectorizer**	**Classifier**
Question vs. Non-Question	100.0%	Unigrams	CountVectorizer	LinearSVC
Personal vs. Impersonal	98.1%	Unigrams + Bigrams	TfidfVectorizer	SVC + linear kernel
Yes/No vs. Other	100.0%	Unigrams	CountVectorizer	LinearSVC
Li & Roth coarse	90.5%	Unigrams + Bigrams	TfidfVectorizer	LinearSVC
Li & Roth fine	82.4%	Unigrams	CountVectorizer	LinearSVC

Additionally, using the algorithms, features and vectorizers found to be the best according to these experiences, we performed a cross validation with 10 folds on these corpora (for the cross validation, we used the whole corpus, that is, the test and train corpus together for each corpus). The results are presented in Table 5.2.

The accuracies obtained show that the system has a strong module of classification for the labels in these corpora. Some of these corpora are quite small and the results obtained can be high due to that fact. The values obtained for the accuracy are quite similar to the one's obtained in TalKit [1]. However, a direct comparison cannot be done due to the fact that the cross validation in that work was done solely on the train corpus whilst the values presented in this work were obtained for the combination of the train and test corpus.

Table 5.2: Results of the cross-validation with 10 folds executed with the classification pipeline found to have the best results.

	Mean Accuracy	**Standard Deviation**
Question vs. Non-Question	100.0%	1.0%
Personal vs. Impersonal	97.0%	1.0%
Yes/No vs. Other	97.0%	6.0%
Li & Roth coarse	86.0%	4.0%
Li & Roth fine	78.0%	3.0%

5.4 Evaluating the Plug and Play Module

To evaluate the first two parts of this module, we have considered a case study which is Edgar. We built the entire system which started by containing only the chatbots Talkpedia and SSS. We then decided to get a highly specific chatbot as is Edgar and add it to the system to see how it adapts. In section 5.4.1 we describe the experience of adding the agent Edgar, in section 5.4.2 we detail how it was to add corpora and finally, in section 5.4.3 we detail some experiences made with the parameters of the system configuration file.

5.4.1 Evaluating the Agents Plug and Play Module

The first thing that had to be done to integrate Edgar in Chattuga was to run it as a standalone agent. The agent was ran and, in doing that, the required steps to invoke the agent and what libraries were needed to run it were identified. When the agent was ran successfully and a conversation with it was held, an investigation had to be done to see how the agent could be adapted so that it would receive a single query instead of being always running and asking for more queries. This is because the only agent that should be asking the user for a query is Chattuga. After performing the required adaptations to Edgar's code, we had to build a wrapper in Python for Edgar's code as it was made in Java. The hardest part was performing all these adaptations to Edgar's code and understanding its architecture and its requirements. The construction of the Python wrapper, after having an example from the other two agents did not prove to be difficult. The actual part of the plug and play, which is to add the agent to the config file, was straightforward: all the required arguments were written in the config file and when the program was ran, the agent immediately started answering questions alongside the other agents.

5.4.2 Evaluating the Corpora Plug and Play Module

The original corpora of the Edgar project was built in a XML format where each file corresponded to a question about a different part of Monserrate's palace or about the butler itself (a different category of question). Chattuga requires a csv file or a txt file in a different format. The first step to integrate this corpus in the system was to transform it into this format. To do this operation, a parsing of the xml files was done. The questions and the category were retrieved and all the files of Edgar were combined in a single txt and a single csv file. The path for these files and the remaining informations required by the plug and play module were added to the corpora config file, including that this corpora is of type "specific" due to its restrict domain. Chattuga was ran after the configurations were added to the corpora config file and the labels of this corpora were promptly returned for the queries that were introduced as part of the experiment.

5.4.3 Evaluating the System Plug and Play Module

To evaluate this module, we introduced a query that we knew the agent Edgar could answer, and then, with the same query we tried different combinations of settings in the system config file. In this way, we can assert how well this module works. The query that was first inserted in the system was "Quantas janelas tem o palácio?" ("How many windows does the palace have?") with the following configurations:

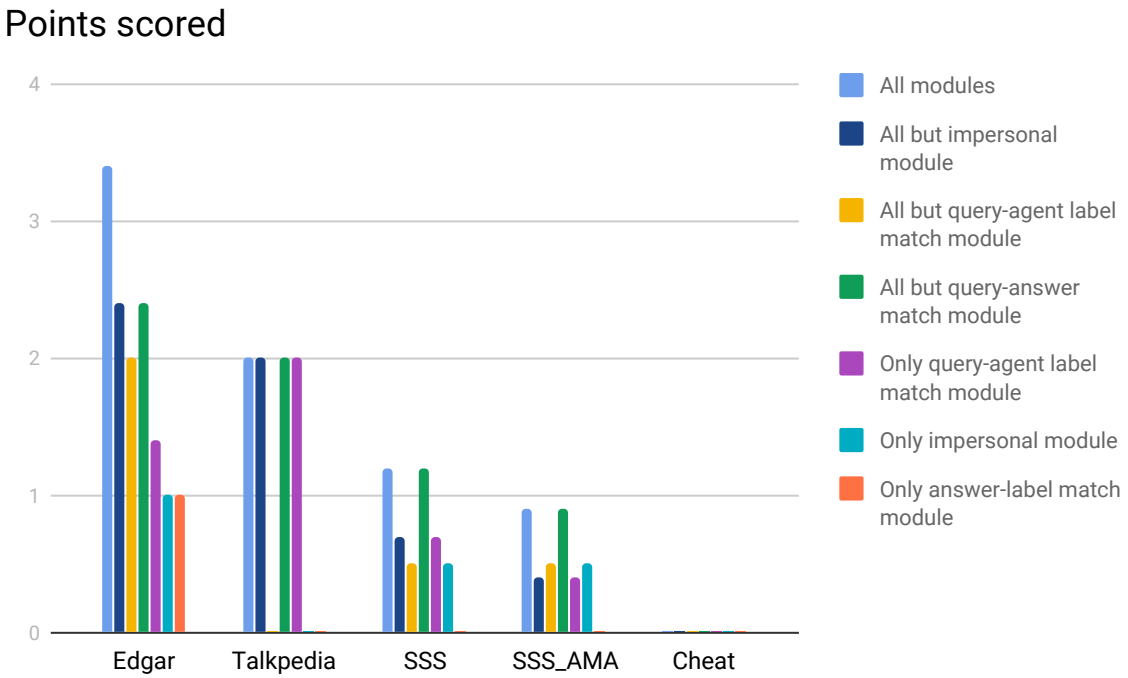
system_config.xml

```
1 <system>
2   <query-agent-label-match>True</query-agent-label-match>
3   <answer-classification-impersonal>True</answer-classification-impersonal>
4   <answer-classification-personal
        sim_measure='tfidf_normal'>True</answer-classification-personal>
5   <query-answer-label-match>True</query-answer-label-match>
6 </system>
```

This led the system to choose Edgar's answer as the best one with a score of 3.4 (2 points from the query-agent label match module; 0.5 points from the impersonal answers module; 1 point from the query-answer label match module). Disabling the modules one by one (this is done by setting them to False in the config file), the scores obtained for the same agent differ drastically, as can be seen in Figure 5.1.

This chart illustrates not only that this plug-and-play module works as supposed but also it shows its utility. If a technical user deems one of the modules less adequate to its agents, that module can be turned off.

Figure 5.1: Variation of scores by varying the parameters of the system config file.



Another setting that can be changed in this config file is the similarity measure or measures that are to be used to obtain a score to personal answers. The first experiment that was conducted was to use all similarity measures (to perform this experiment and the following experiments, we only activated the personal module as that was the focus of these experiences) and then run the system with the query "Como te chamas?" ("What is your name?"). With that configuration, the system config file has the following appearance:

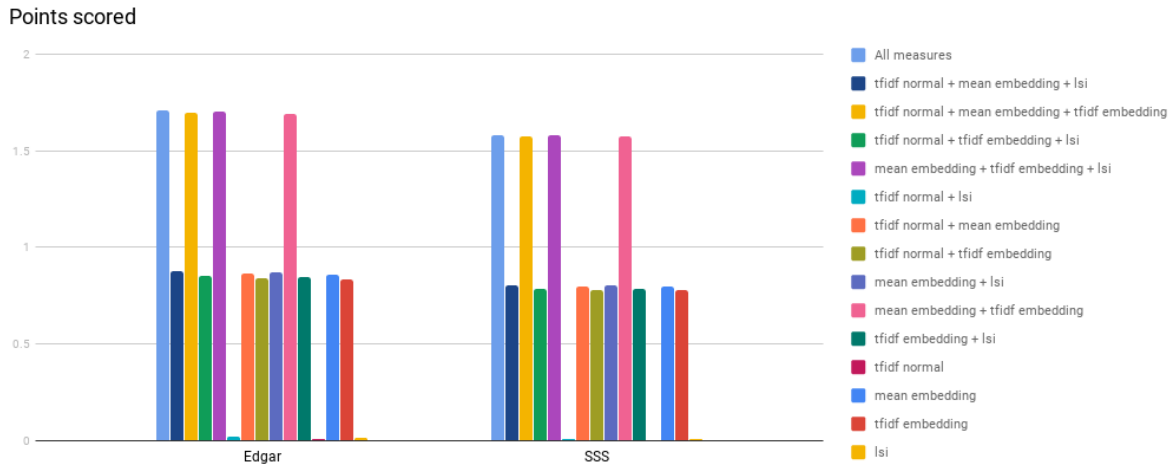
system_config2.xml

```
<system>
  <query-agent-label-match>False</query-agent-label-match>
  <answer-classification-impersonal>False</answer-classification-impersonal>
  <answer-classification-personal
    sim_measure='tfidf_normal|tfidf_embedding|mean_embedding|lsi'>True
  </answer-classification-personal>
  <query-answer-label-match>False</query-answer-label-match>
</system>
```

All combinations of similarity measures were then tried and are represented in Figure 5.2. Note

that in this chart the agents “Cheat”, “Talkpedia” and “SSS_AMA” are not included as they returned an utterance that indicated that they did not know the answer and thus were not scored by this module. This chart shows that the system allows for different configurations of measures quite easily; it also shows that TF-IDF + cosine returns a score that is much lower than all the other four measures. However, as it is the fastest and it is consistent in the result that it deems as the most similar with the other measures, it is used as the default.

Figure 5.2: Variation of scores by varying the similarity measure of the personal answers module on the system config file.



5.5 Evaluating the Answer Ranking Process

One of the most important jobs of Chattuga is its ability to detect, among a set of agent’s answers, the one or ones that are more relevant to a query. In information retrieval, it is studied the retrieval of relevant documents to a given query. These techniques are mostly used in search engines to order the results found according to their relevance. As the system discussed in this thesis has an analogous function in the sense that it intends to order its results (answers) according to their relevance, to evaluate the Chattuga’s answer ranking capabilities, we have decided to compute the Mean Reciprocal Rank [44] which is one of the metrics used in the field of information retrieval to evaluate a system’s effectiveness in ordering the retrieved documents according to their relevance. This metric will help us establish the system’s performance benchmark. As we do not have a corpus that establishes the relevance of each answer to each query inserted in the system, we have asked three users to rank the answers given to the system according to a scale of 1 to 5. We have randomly selected 100 queries from various sources. The queries that we used to perform the evaluation of Chattuga’s answer ranking process were randomly sampled queries of the following types:

- 25 impersonal corpus questions;
- 25 personal corpus questions;
- 10 Edgar's corpus questions;
- 10 AMA's corpus questions;
- 10 queries from the non-questions corpus;
- 5 queries such as "bom dia" ("good morning"), "boa tarde" ("good afternoon"), etc.
- 5 yes/no corpus questions;
- 5 or questions;
- 5 list questions.

We then took these ratings given by the annotators as the correct ones (to calculate the Mean Reciprocal Rank) and calculated the Mean Reciprocal Rank between the rating of each annotator and the rating the system gave to each answer[2]. If there is more than one answer graded with a one, we just compute the reciprocal rank for the first one to be rated with a one. The results for each annotator are presented in Table 5.3. Note that the labels of expertise of:

- **Talkpedia** are "QUESTION" and "IMPERSONAL";
- **Edgar** are "QUESTION", "PERSONAL, IMPERSONAL", and all Monserrate's labels;
- **SSS** are "QUESTION" and "PERSONAL";
- **SSS_AMA** are "QUESTION", "IMPERSONAL" and all AMA's labels;
- **Cheat** are "OR_QUESTION" and "YN_QUESTION".

From the answers selected as the most relevant for each query out of the 100, some were answers stating that the chatbot was unable to find an answer to that particular query. Examples of such answers are "Não sei responder a isso" ("I do not know how to answer that"), "Essa não é a minha área de especialidade" ("That is not my area of expertise"), among others. In Table 5.4 we present the number of queries to which such answers were deemed more relevant per annotator.

[2]We had to keep in mind some details when calculating the mean reciprocal rank. As both the system and the annotators could consider two or more answers as the best (thus, giving the rate 1) we had to make a decision as to what to do in those cases. One possibility was to use the Total Reciprocal Rank (TRR) formula which is mentioned in [45]. This formula is proposed to be used in these cases where multiple results are relevant and is computed by summing all of the reciprocal ranks of all the answers relevant (which could result in having the sum of the maximum of 500 reciprocal ranks which would be divided by 100 all the same). Another possibility we considered was to compute the average per query on these cases (for example: if three answers are deemed relevant, then we compute the reciprocal rank for each one and then we sum the three results and divide that sum by three). In the end, we have decided to compute the Mean Reciprocal Rank in the regular way.

Table 5.3: Mean Reciprocal Rank results (approximated) for each annotator.

	MRR
Annotator 1	68.2%
Annotator 2	63.1%
Annotator 3	51.0%

Table 5.4: Number of queries to which no satisfactory answer was found among the set of answers per annotator and Chattuga.

	Number of queries to which the most relevant answer was a discard answer
Annotator 1	46
Annotator 2	57
Annotator 3	62
Chattuga	65

These results show that for half (or more) the queries selected, both the annotators and the system were unable to find a satisfactory answer among the ones returned by the different conversational agents inserted in the system. The answers in which there was more disagreement between annotators and system were the ones provided by Talkpedia and by the Cheat agent. This is due to the fact that in the agents config file, Talkpedia was given a big score in the expertise of the labels “IMPERSONAL” and “QUESTION”. This agent, however, was not able to deliver any viable answer to any of the queries. Regarding the Cheat agent, due to the fact that it is based in very simple rules without “artificial intelligence”, it was given a low score in its expertise on the labels “OR_QUESTION” and “YN_QUESTION” causing it to have a low score against the more intelligent agents. From the queries used to perform this evaluation, we extracted some examples to illustrate this inadequacy in the trade-off of scores/agent. These are presented next, in Table 5.5.

Table 5.5: Example of a query used in the evaluation with the answers and respective rank and score.

Query: Acreditas em astrologia? Segues o teu horóscopo do zodíaco?

Agent	Answers	A1 rank	A2 rank	A3 rank	Chattuga rank	Chattuga score
SSS	Não sei responder a isso	3	3	3	1	1.40
Talkpedia	Posso dizer que no results found for: Acreditas em astrologia? Segues o teu horóscopo do zodíaco	4	4	5	2	1.00
Cheat	Não.	1	1	1	3	0.88
Edgar	Podia repetir, por favor?	2	2	4	4	0.42
SSS_AMA	Não sei responder a isso	3	3	3	5	0.20

In Table 5.6 we can see the scores that generated the Chattuga ranks for each question. These scores stem from two modules of answer classification. The agent-query label match module and the personal answers module. As can be seen, in each column there are the scores for each module. In the column of the query-agent label match there is a score for each query label, which is 0 if the agent is not an expert in that label. Both SSS and Talkpedia only received the scores of the first module due to the fact that they both did not find any answer (thus, they were only awarded for their expertise). In Table 5.6, it is presented a breakdown of the scores shown in Table 5.5, in the column of the scores (the values in that column are a sum of the scores in the following table). The modules that did not contribute to the final score were omitted from this Table (meaning they returned a score of 0.0).

Table 5.6: This Table contains the scores breakdown of the scores awarded to each agent's answer.

	Query-Agent Label Match Module				Personal Answers Module	Total
	QUESTION	PERSONAL	YN_QUESTION	ENTY:other		
SSS	**0.70**	**0.70**	0.00	0.00	0.00	1.40
Talkpedia	**1.00**	0.00	0.00	0.00	0.00	1.00
Cheat	0.00	0.00	**0.80**	0.00	**0.08**	0.88
Edgar	0.20	**0.20**	0.00	0.00	**0.02**	0.42
SSS_AMA	**0.20**	0.00	0.00	0.00	0.00	0.20

As can be seen in the Table, both SSS and Talkpedia appear before the Cheat agent which had the best answer according to all annotators, based on their expertise score. However, the module of

personal answers, deemed the answer given by the Cheat agent the best one.

5.6 Discussion

With this evaluation, we have concluded that Chattuga has a strong query classification module due to the high accuracy achieved in all the main corpora. Some of these accuracies are quite high which can be caused by two factors: either the amount of data is insufficient and with low diversity causing the algorithm to easily learn the patterns of the data or it is not very hard to distinguish the utterances of the different labels for a computer algorithm. In what concerns the plug and play module, there are some difficulties inherent to adding a new agent and corpora but these cannot be surpassed as they are related to the fact that systems and corpora are built in different ways and, to be integrated in another, some adaptations have to be made. However, it did not took a great implementation effort to make such modifications and after these were made, the only thing that remained to be done was to add some configurations in the corpora and agent's configuration files. Lastly, the answer ranking module evaluation shows that, when the agents can provide an answer, the ranking process behaves as expected by giving the right scores to the right answer. Namely for Edgar and SSS_AMA, only a portion of their score came from the module that was based on the baseline, the query-agent label match module, the rest of the score came from the other modules which took as input the agents answers. A negative point that this system has is the time it takes to run. It takes approximately 20 seconds per query (with the default settings) which is not very fluid when having a conversation. This happens because every time a query is inserted, the agents have to be initialized which takes a long time to run even with parallel computing which was used. This time problem is related to the fact that the agents are written in another programming language and they have to be called as if it were in the command line – which means that at each query a new instance of an agent is created.

6

Conclusions and Future Work

Contents

6.1 Summary of Dissertation

In this work, we proposed a different approach to multi-agent query answering where it is assumed that all agents can, potentially, answer all queries. To build such an architecture, the system is composed of a set of modules that contribute to a final choice on the answer ranking process. Additionally, this architecture is plug and play in regard to (a) external components: more agents and corpora can be smoothly integrated in the system; and (b) internal components: specific system modules to rank answers can be disabled. The modules proposed and used in this work are (a) the query classification module which provides the labels for the user inserted query; (b) the answer ranking modules: one that takes into account the agent's expertise; one that does a classification of answers to queries labelled as "IMPERSONAL"; one that uses similarity measures to compare answers to queries labelled as "PERSONAL" to a corpus of personal answers; lastly, one that uses machine learning to classify answers with labels and matches it to the queries labels; and (c) the plug and play module which has three sub-modules: agents, corpora and system. These modules all provide a score which is the basis for the answer ranking decision.

6.2 Contributions

The contributions that arose from this dissertation were: a multi-agent question-answering system for the Portuguese language, which can be used with new agents and corpora in the future without it taking much implementation effort by the use of the plug and play module; a proposal of classification and evaluation for such systems; a proposal of answer ranking without much data; a system tested with two specialized agents; lastly, a paper submitted, accepted, presented and published in the context of a Portuguese conference that took place in June in Coimbra (SLATE'19), in which context the modified TalKit corpora was presented.

6.3 Limitations

The biggest limitation that was found in the process of implementing this system was the lack of good agents. Most queries were answered unsuccessfully. Especially queries that were more generic. Queries of the domain of AMA or Monserrate were more successful. Another problem is the lack of data available for the Portuguese language. There is especially a lack of corpora that contains pairs of query/answer which would enrich this work.

6.4 Future Work

One thing that could be done in the future that can benefit a lot this work is to incorporate the Ubuntu corpus strategy (Section 2.5) as a module in this project. For this, in the context of this work, a corpus has already been built that can be used for that purpose, containing one right answer and ten wrong answers. What remains to be done is to apply the neural networks to that corpus so that a match between query and answer can be done. Another interesting addition that could be done to Chattuga is to integrate a reinforcement learning[1] module that adjusts the agents expertise scores dynamically since one of the limitations that was found on the course of this work were the agents inability to answer queries to which they had expertise on, whilst others that were supposed to have less expertise performed better. With this module, this problem can be mitigated and agent's score can be more informative for the system's answer ranking process. Another important future work is to wrap the agents that are built in other programming languages in a more time efficient fashion, since the one used in the context of this work implied the instantiation of the agent every time a query was inserted which, ultimately, weighed in the running time of the program.

[1] A technique where a conversational agent learns from user input.

Bibliography

[1] C. Dias, "TalKit - Desenvolvimento de um Sistema de Diálogo para Português," Master's thesis, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, 2015.

[2] J. Silva, L. Coheur, A. C. Mendes, and A. Wichert, "From symbolic to sub-symbolic information in question classification," *Artificial Intelligence Review*, vol. 35, no. 2, pp. 137–154, 2011.

[3] X. Li and D. Roth, "Learning question classifiers," in *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, ser. COLING '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 1–7. [Online]. Available: https://doi.org/10.3115/1072228.1072378

[4] M. G. Fernandes, C. Dias, and L. Coheur, "Distinguishing Different Classes of Utterances - the UC-PT Corpus," in *8th Symposium on Languages, Applications and Technologies (SLATE 2019)*, ser. OpenAccess Series in Informatics (OASIcs), R. Rodrigues, J. Janousek, L. Ferreira, L. Coheur, F. Batista, and H. G. Oliveira, Eds., vol. 74. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 14:1–14:8. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2019/10881

[5] D. Moldovan, M. Paşca, S. Harabagiu, and M. Surdeanu, "Performance issues and error analysis in an open-domain question answering system," *ACM Trans. Inf. Syst.*, vol. 21, no. 2, pp. 133–154, Apr. 2003. [Online]. Available: http://doi.acm.org/10.1145/763693.763694

[6] E. Hovy, L. Gerber, U. Hermjakob, C.-Y. Lin, and D. Ravichandran, "Toward semantics-based answer pinpointing," in *Proceedings of the First International Conference on Human Language Technology Research*, 2001. [Online]. Available: http://aclweb.org/anthology/H01-1069

[7] B. Loni, "A survey of state-of-the-art methods on question classification," Delft University of Technology, Tech. Rep., 2011.

[8] M. Pota, A. Fuggi, M. Esposito, and G. D. Pietro, "Extracting compact sets of features for question classification in cognitive systems: A comparative study," in *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Nov 2015, pp. 551–556.

[9] M. Pota, M. Esposito, and G. De Pietro, *A Forward-Selection Algorithm for SVM-Based Question Classification in Cognitive Systems*, 06 2016, pp. 587–598.

[10] G. Feng, K. Xiong, Y. Tang, A. Cui, J. Bai, H. Li, Q. Yang, and M. Li, "Question classification by approximating semantics," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15 Companion. New York, NY, USA: ACM, 2015, pp. 407–417. [Online]. Available: http://doi.acm.org/10.1145/2740908.2745403

[11] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, pp. 1746–1751. [Online]. Available: http://aclweb.org/anthology/D14-1181

[12] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3294–3302. [Online]. Available: http://papers.nips.cc/paper/5950-skip-thought-vectors.pdf

[13] Â. Costa, T. Luís, J. Ribeiro, A. C. Mendes, and L. Coheur, "An english-portuguese parallel corpus of questions: translation guidelines and application in smt," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*. European Language Resources Association (ELRA), 2012. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2012/pdf/356_Paper.pdf

[14] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995. [Online]. Available: http://doi.acm.org/10.1145/219717.219748

[15] H. T. Madabushi and M. Lee, "High Accuracy Rule-based Question Classification using Question Syntax and Semantics," no. 2002, pp. 1220–1230, 2016.

[16] H. T. Madabushi, M. Buhagiar, and M. Lee, "UoB-UK at SemEval-2016 Task 1 : A Flexible and Extendable System for Semantic Text Similarity using Types , Surprise and Phrase Linking," vol. 0, pp. 680–685, 2016.

[17] N. Van-Tu and L. Anh-Cuong, "Improving question classification by feature extraction and selection," *Indian Journal of Science and Technology*, vol. 9, no. 17, 2016.

[18] M. Pereira, " Just.chat - dos sistemas de pergunta/resposta para os chatbots," Master's thesis, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, 2015.

[19] D. Ameixa, "Say Something Smart - ensinando um chatbot a responder com base em legendas de filmes," Master's thesis, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, 2015.

[20] D. C. Uthus and D. W. Aha, “The ubuntu chat corpus for multiparticipant chat analysis,” in *2013 AAAI Spring Symposium Series*, 2013.

[21] R. Lowe, N. Pow, I. Serban, and J. Pineau, “The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems,” *CoRR*, vol. abs/1506.08909, 2015. [Online]. Available: http://arxiv.org/abs/1506.08909

[22] R. T. Lowe, N. Pow, I. V. Serban, L. Charlin, C.-W. Liu, and J. Pineau, “Training end-to-end dialogue systems with the ubuntu dialogue corpus,” *Dialogue & Discourse*, vol. 8, no. 1, pp. 31–65, 2017.

[23] R. Baeza-Yates, B. d. A. N. Ribeiro *et al.*, *Modern information retrieval*. New York: ACM Press; Harlow, England: Addison-Wesley,, 2011.

[24] W. H. Gomaa and A. A. Fahmy, “A survey of text similarity approaches,” *International Journal of Computer Applications*, vol. 68, no. 13, pp. 13–18, 2013.

[25] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, “From word embeddings to document distances,” in *International conference on machine learning*, 2015, pp. 957–966.

[26] J. Mueller and A. Thyagarajan, “Siamese recurrent architectures for learning sentence similarity,” in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[27] Z. Wang, H. Mi, and A. Ittycheriah, “Sentence similarity learning by lexical decomposition and composition,” *arXiv preprint arXiv:1602.07019*, 2016.

[28] H. He, K. Gimpel, and J. Lin, “Multi-perspective sentence similarity modeling with convolutional neural networks,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1576–1586.

[29] P. Mota, “LUP: A Language Understanding Platform,” Master’s thesis, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, 2015.

[30] P. Fialho, L. Coheur, S. Curto, P. Cláudio, Â. Costa, A. Abad, H. Meinedo, and I. Trancoso, “Meet edgar, a tutoring agent at monserrate,” in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, 2013, pp. 61–66. [Online]. Available: http://aclweb.org/anthology/P13-4011

[31] J. Santos, “Say Something Smart 3.0:A Multi-Agent Chatbot in Open Domain (Under Preparation),” Master’s thesis, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, 2019.

[32] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[33] B. Aarts, *Oxford Modern English Grammar*. Oxford University Press, 2011.

[34] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[35] B. Magnini, A. Vallin, C. Ayache, G. Erbach, A. Peñas, M. de Rijke, P. Rocha, K. Simov, and R. Sutcliffe, "Overview of the clef 2004 multilingual question answering track," in *Multilingual Information Access for Text, Speech and Images*, C. Peters, P. Clough, J. Gonzalo, G. J. F. Jones, M. Kluck, and B. Magnini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 371–391.

[36] D. Ameixa, L. Coheur, and R. A. Redol, "From subtitles to human interactions: introducing the subtle corpus," Tech. rep., INESC-ID (November 2014), Tech. Rep., 2013.

[37] J. Silva, "QA+MLWikipedia&Google," Master's thesis, Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, 2009.

[38] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, Inc., 2009.

[39] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51 – 56.

[40] C. Freitas, P. Rocha, and E. Bick, "A new world in floresta sinta(c)tica - the portuguese treebank," *Calidoscópio*, vol. 6, pp. 142–148, 09 2008.

[41] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[42] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[43] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[44] E. M. Voorhees, "The trec-8 question answering track report," in *In Proceedings of TREC-8*, 1999, pp. 77–82.

[45] D. R. Radev, H. Qi, H. Wu, and W. Fan, "Evaluating web-based question answering systems." in *LREC*, 2002.

A

Li & Roth Additional Statistics

Table A.1: Number of questions per fine category in the Li & Roth corpus.

Coarse Category	Fine Category	Training Set	Testing Set	Training + Testing Set
ABBREVIATION	**abbreviation**	16	1	17
	expansion	70	8	78
ENTITY	**animal**	112	16	128
	body	16	2	18
	color	40	10	50
	creative	207	0	207
	currency	4	6	10
	diseases and medicine	103	2	105
	event	56	2	58
	food	103	4	107
	instrument	10	1	11
	language	16	2	18
	letter	9	0	9
	other	217	12	229
	plant	13	5	18
	product	42	4	46
	religion	4	0	4
	sport	62	1	63
	substance	41	15	56
	symbol	11	0	11
	technique	38	1	39
	term	93	7	100
	vehicle	27	4	31
	word	26	0	26
DESCRIPTION	**definition**	421	123	544
	description	274	7	281
	manner	276	2	278
	reason	191	6	197
HUMAN	**group**	189	6	195
	individual	962	55	1017
	title	25	1	26
	description	47	3	50
LOCATION	**city**	129	18	147
	country	155	3	158
	mountain	21	3	24
	other	464	50	514
	state	66	7	73
NUMERIC	**code**	9	0	9
	count	363	9	372
	date	218	47	265
	distance	34	16	50
	money	71	3	74
	order	6	0	6
	other	52	12	64
	period	75	8	83
	percent	27	3	30
	speed	9	6	15
	temperature	8	5	13
	size	13	0	13
	weight	11	4	15

Table A.2: Number of questions per coarse category in the Li & Roth corpus.

Coarse Category	Training Set	Testing Set	Training + Testing Set
ABBREVIATION	86	9	95
ENTITY	1250	94	1344
DESCRIPTION	1162	138	1300
HUMAN	1223	65	1288
LOCATION	835	81	916
NUMERIC	896	113	1009