DÉBORA CRISTINA AMARAL TEIXEIRA
Graduate in Computer Science

ANICE : AN ARTIFICIAL NEURO-LINGUISTIC INTERACTIVE COMPUTER ENTITY

MASTER IN ⟨COMPUTER SCIENCE⟩

NOVA University Lisbon
December, 2022

DEPARTMENT OF
COMPUTER SCIENCE

ANICE : AN ARTIFICIAL NEURO-LINGUISTIC INTERACTIVE COMPUTER ENTITY

DÉBORA CRISTINA AMARAL TEIXEIRA

Graduate in Computer Science

Adviser: Nuno Miguel Cavalheiro Marques
Professor Doutor, NOVA University Lisbon

MASTER IN ⟨COMPUTER SCIENCE⟩

NOVA University Lisbon
December, 2022

ANICE : An Artificial Neuro-Linguistic Interactive Computer Entity

Acknowledgements

First and foremost, I want to thank professor Nuno Marques for all the guidance and constant support during the development of this thesis.

I am also grateful to my institution - Nova School of Science and Technology - specially to the Department of Informatics for the knowledge it provided me the last five years.

I want to thank my mom for being my number one supporter and always being proud of my achievements, my dad for being the best engineer and pushing me to do my best, my brother for being my role model and Francisco for believing in me even when I didn't believe in myself.

Lastly, I want to thank all my friends who supported me and helped me throughout the development of this thesis.

“Just as soon as you attain to one ambition you see another one glittering higher up still. It does make life so interesting.”
(Lucy Maud Montgomery)

Abstract

Mental health problems are hard to talk about, especially when the questions asked do not allow the individual to answer freely. That is the case for most inquiries, where questions usually request very restricted answers like yes or no. This thesis proposes a chatbot that tries to avoid the problem of restricting users to small answers. The chatbot will focus on people feeling burned out due to stress related to their studies. The chatbot tries to replicate two forms with questions about burnout that are used as guidelines. Both these forms are developed based on questions done by psychologists.

Because rule-based chatbots have a limited vocabulary, natural language understanding and neural-based techniques are tested and validated to see if the chatbot performs well using these techniques. The techniques tested are word2vec and spacy components.

The evaluation results show that it is feasible to implement a chatbot that uses rules and also techniques for natural language processing. Additionally, the tests did indicate that both spacy and word2vec are great resources for NLU. Word2vec proves to perform slightly better at specific times related to identifying intents that are domain-specific. Finally, the results from the users experience show that this is a promising work that could help students dealing with burnout.

Keywords: Chatbot, mental health, burnout, spacy, YAML, rule-based, neural-based

Resumo

Problemas de saúde mental são um tema difícil de abordar, especialmente quando as perguntas feitas não permitem ao indivíduo responder livremente. Este é o caso da maioria dos inquéritos, onde as perguntas geralmente exigem respostas muito restritas, como sim ou não. Esta tese propõe um chatbot que tenta evitar o problema de restringir os utilizadores a pequenas respostas. O chatbot concentrar-se-á em utilizadores que se sentem esgotados devido ao stress relacionado com os estudos. O chatbot tenta replicar dois formulários com perguntas sobre burnout, isto é, estes formulários são utilizados como diretrizes. Ambos os formulários são desenvolvidos com base em perguntas feitas por psicólogos.

Como os chatbots baseados em regras têm um vocabulário limitado, a compreensão da linguagem natural e as técnicas baseadas em redes neuronais são testadas e validadas para ver se o chatbot tem um bom funcionamento utilizando essas técnicas. As técnicas baseadas em redes neuronais que são testadas são o word2vec e componentes spacy.

Os resultados da avaliação mostram que é viável implementar um chatbot que utilize regras e também técnicas de processamento de linguagem natural. Além disso, os testes indicam que tanto os componentes spacy quanto o word2vec são ótimos recursos para processamento de linguagem natural. O Word2vec tem um desempenho um pouco melhor em momentos específicos relacionados à identificação de intenções do domínio de estudo. Por fim, os resultados da experiência dos utilizadores mostram que este é um trabalho promissor que pode ajudar os utilizadores a lidar com o burnout.

Palavras-chave: Chatbot, saúde mental, burnout, spacy, YAML, técnicas baseadas em regras, técnicas baseadas em redes neuronais

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Motivation

Mental disorders are considered the "diseases of the 21st century", with an estimation of 264 million people affected by depressive disorders, about 46 million people worldwide suffer from bipolar disorder and approximately 284 million people develop anxiety disorders [45]. Additionally, over 50% of the latest generations (Millenials and Gen z) are reporting burnout, according to studies [30]. These problems however are not covered by health insurance companies, meaning that a lot of people do not have access to treatment due to its high price. Also, some individuals do not feel comfortable sharing their experiences and fears with a human therapist, due to privacy concerns [37].
A recent study from Oracle [19], carried out during the COVID-19 pandemic, showed that in 2020 the workplace stress, anxiety and burnout increased significantly. Furthermore, the majority of the participants of the study said that they would feel more comfortable sharing their problems with a robot counselor rather than their managers, since they believe robots provide a judgement free zone and give quick answers to health-related questions.
It is known that chatbots are growing in popularity specially in costumer help services. A lot of conversational agents like **Siri** and **ALEXA** are used everyday by millions of costumers who mostly ask questions that facilitate their life, like making a phone call without having to search for the contact or asking the conversational agent to control the lights on a house. But what is also important to notice is that these agents are trying to break a barrier between technology and mental health. Alexa for instance has a command where you can open a guided meditation with the intent of reducing stress, anxiety and sleeping problems. Trying to break this barrier is really important due to the amount of people searching for help in the current days. Google estimated that in 2021 the question *Why do I feel sad?* was asked more than ever before, increasing 10% since the year before [24]. Furthermore, according to Google, *How to maintain mental health?* was searched in 2021 globally more than ever before [61]. In figure 1.1, there is a graphic demonstrating the growth of this question.

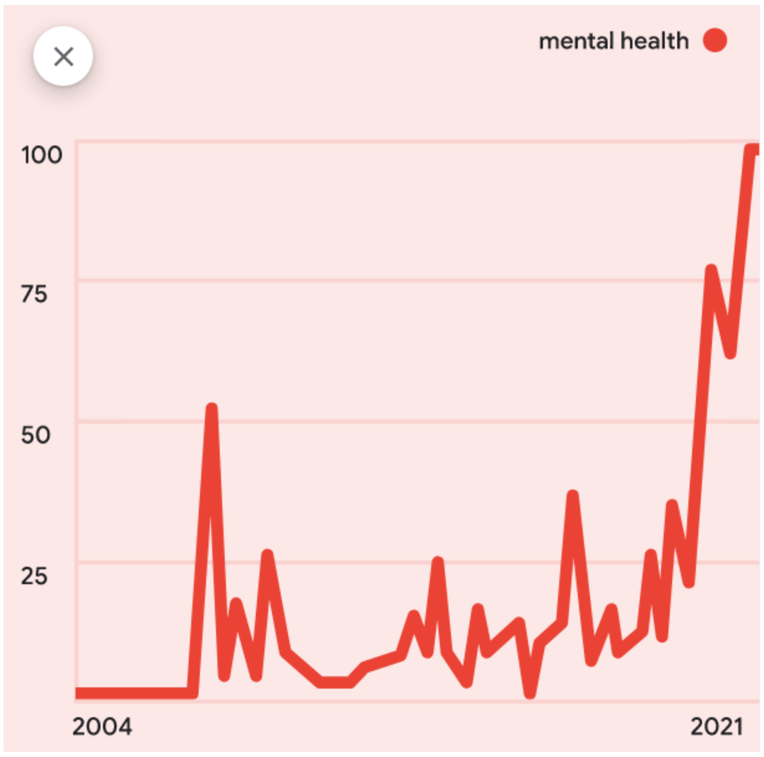


Figure 1.1: Question *How to maintain mental health?* throughout the years [61]

Despite this information there are very few solutions involving technology, which shows that the connection between mental health and engineering can still be explored thoroughly.

1.2 Goals

The goals established for this thesis are the following:

- Study an *English language* system for a chatbot and its adaption to a specific case study. The case selected is related to the topic of people feeling burned out;
- Validate if it is feasible to integrate both rule-based and neural-based techniques that will work alongside each other on the chatbot. Namely, we will study the possibility of using word embeddings:
  - Use rule-based mechanisms in order to provide a simple and fast way of introducing response patterns;
  - Use neural-based mechanisms in a chatbot that tries to do a better recognition of the user's request;

- Validate whether spacy and word2vec are good modules for complementing an English chatbot;
- Identify and validate the use of adequate interfaces for the forms and the chatbot systems in order to survey users' opinions related to the project;
- Validate the opinion of users related to the use of the chatbot through the development of inquiries applied to a population of students and young adults.

1.3 Document Structure

This document starts with a small introduction in chapter 1 where it is indicated the motivation for this thesis and what are the goals to be reached throughout this project.
After the introduction there is chapter 2, where important concepts related to this thesis' topic are explained, the state of the art on Chatbots is presented, a connection between chatbots and mental health issues is shown and a few important aspects are explained in more detail - YML, vector representations, Spacy, Word2vec, BERT and RASA.
Chapter 3 is dedicated to the comparative analysis between several tools in order to decide what are the best tools to approach this thesis' goal.
Following the comparative analysis, chapter 4 presents an analysis of two forms that were applied to two groups of people with questions related to Burnout. In this chapter, a feedback given by the participants who answered the forms is provided.
In chapter 5 the focus is on describing what is the architecture used for the whole project and in chapter 6 there is a description of the implementation of the architecture described in the previous chapter.
After, in chapter 7 we present the results of applying some tests to the components of our architecture and there is also an evaluation of the user experience when trying the chatbot. This evaluation is done by analysing the results of a form applied to a population chosen to test the chatbot. This form contains questions related to the users' experiences.
Finally, chapter 8 is where conclusions are taken from all the tests and experiences made throughout the development of this project.

2 Literature Review

The theme of this thesis falls within a vast area with a lot of work. Therefore, it begins with some basic important concepts that are mentioned throughout the development of this thesis. After, relevant examples related to this thesis' topic are talked about - for example, ELIZA and PARRY. It is also mentioned the importance of this thesis in the psychological field. Finally, we will talk about tools that we consider very important for this thesis' topic: *YML*, *Vector representations*, *Spacy* and *RASA*.

2.1 Chatbots' Fundamental Concepts

Nowadays the market related to AI has been growing, making AI one of the most interesting areas to study in technology. *Chatbots* are a perfect example of a system ruled by AI. To better understand it, an overview of concepts is given in this section. These concepts are important because they are used throughout this thesis.

- **Conversational agent** - A *Conversational agent* is a dialogue system that through the use of natural language understanding can respond to a person using human language. It is usually employed as a chatbot or as portable device assistants, like ALEXA. These agents can be conducted not only via text but also with speech [12].
- **Chatbot** - A *chatbot* is a type of conversational agent. One goal of this system is to try to demonstrate intelligence [3], whether by using rules or NLU techniques. There are several criteria to classify chatbots but the broad classification according to [27] is done using the following criteria:
  - **Interaction mode** - it focuses on how the interaction with the chatbot is made. It can either be text based or voice/speech based [27];
  - **Chatbot Application** - it focuses on how the chatbot is used i.e. if it focuses on completing certain tasks (**Task-Oriented**) or not (**Non-Task-Oriented**) [27];
  - **Domain-specific or Open-Domain** - It defines if the chatbot is dedicated to a particular problem or not [27];

  - **Rule-based or AI** - defines what techniques the chatbot implements. These techniques are explained below [27].

  Chatbots can implement two strategies:

  1. **Rule-based techniques** - the systems that implement *Rule-based techniques* focus on rules as their knowledge representation. These rules are programmed as *if-then-else* statements and therefore the system does not have a learning capacity, but rather pretends to know about a certain topic [3];
  2. **Neural-based techniques** - the systems that implement *Neural-based techniques* have the ability to learn using AI methods. Due to its learning feature, these systems can provide longer and more detailed answers and also do a better understanding of the intentions of the users, which helps keeping the user engaged in the conversation [3].

- **Turing Test** - The *Turing Test* is a method used to evaluate if a machine is able to demonstrate intelligent behaviour similar to a human. Its creator, Alan Turing, proposed that a machine can only possess artificial intelligence if it can give the same or similar responses as humans under particular conditions [18].

- **Sentiment Analysis** - *Sentiment analysis*, also known as *Opinion mining* is the computational study that identifies and extracts subjective information in source material. It helps businesses understanding people's opinions, sentiments, emotions and appraisals towards their brands [23].

- **Word Vectors** - *Word Vectors*, or *Word Embeddings*, are rows of numbers where each point captures a dimension of the word's meaning and where semantically similar words have similar vectors [2]. Related to word vectors, we have the following concepts:

  - **Transformers** - A *Transformer* is a deep learning technique that computes dense, context-sensitive representations for tokens. It uses the self-attention mechanism which means that it gives different weights to each part of the input sentence depending on its importance in the context of the sentence [50];
  - **Sequence-to-sequence** - *Sequence-to-sequence* or *Seq2seq* is a model that receives a sequence of elements (words, time series,etc.) for input and has an *encoder* that captures that input as a *hidden state vector* with the input item and its context and a *decoder* that transforms the vector into an output item [15].
  - **Pretrained word embeddings** - *Pretrained word embeddings* are embeddings that carry some kind of linguistic knowledge since they are learned in one task and then are used to solve another task. Because they store knowledge, they are considered a form of Transfer Learning [39].

- **Transfer Learning** - *Transfer learning* is a method used to perform similar tasks in different models by sharing generalised knowledge between the models [52]. It takes a model trained on a large dataset and transfers its knowledge to a smaller dataset. It can be used, for instance, in pretrained Word Embeddings, as mentioned before.

- **Intent** - In a chatbot, an *intent* is what the user is looking for when typing a sentence or question. Therefore, the intent is the action that the user wants to take everytime it speaks to a conversational agent [46];

- **Entity** - In a chatbot, *entities* extract data from what the user typed in order to help the chatbot give the best answer or recommendation. It can be any type of field and describe just about anything [46];

- **Featurizer** - a featurizer is a component that contains code for transforming the input data into processed data that is useful for machine learning algorithms to use [16];

- **Tokenizer** - Tokenizers are components that are responsible for turning the input data into structured data useful for natural language understanding. They take the input stream and divide it into tokens [16];

- **Bag-of-Words** - Bag-of-words (BoW) is a model used for natural language processing. This method is essentially used for extracting features from the input. It keeps track of the number of occurrences of a word in the input, disregarding the grammatical details or the order to which the word appeared [21];

- **Conditional Random Fields** - Conditional Random Fields (CRFs) are models that are discriminative and are used for natural language processing. These models take into consideration the contextual information of the neighbours to try to make a good prediction for the current field [43].

Following these concepts, the next section will explore some works related to chatbots.

2.2 State of the art on Chatbots

Chatbots have been evolving and growing in popularity, especially with the growth of the AI field [10].

2.2.1 Rule-based chatbots

The first chatbots developed were focused solely on simple rules, using rule-based techniques, and had the main goal of passing the Turing test. This is the case of **ELIZA** and **PARRY**, both developed using these types of techniques [40].

ELIZA is the most well-known pre-Internet era system [48] and was one of the first programs passing the Turing test. It simulated a simple conversation, based on text, between a user and a machine that pretended to be a psychiatrist. In short, ELIZA examined the text that the user wrote as input and searched for keywords. When found, ELIZA set values to the keywords and applied its rules of transformation to the input [40]. This system was great at fooling users in small conversations and some users even said to prefer the interaction with a machine rather than a human when talking about feelings and struggles. Additionally, a few psychiatrists said that "Eliza's potential computer-based therapy" could be used as "a form of psychological treatment" [48].

PARRY was a more advanced program than ELIZA, and it attempted to simulate a patient with paranoid schizophrenia. Psychiatrists analysed a combination of real patients and a computer running PARRY and were only able to make the correct identification 48 percent of the time. Another interesting fact is that PARRY was “counseled” a few times by ELIZA [40]. PARRY was said to be a great asset for researchers in computer science and psycopathology and also to mental health educators [48].

Another chatbot extremely relevant is **ALICE** [1]. ALICE stands for Artificial Liguistic Internet Computer Entity. This chatbot is an open source natural language processing chatbot that uses a very large number of rules matching input patterns in order to provide the output. To store knowledge about conversation patterns ALICE uses files in **AIML**, a XML dialect. ALICE has three types of categories, all manually "hand-coded":

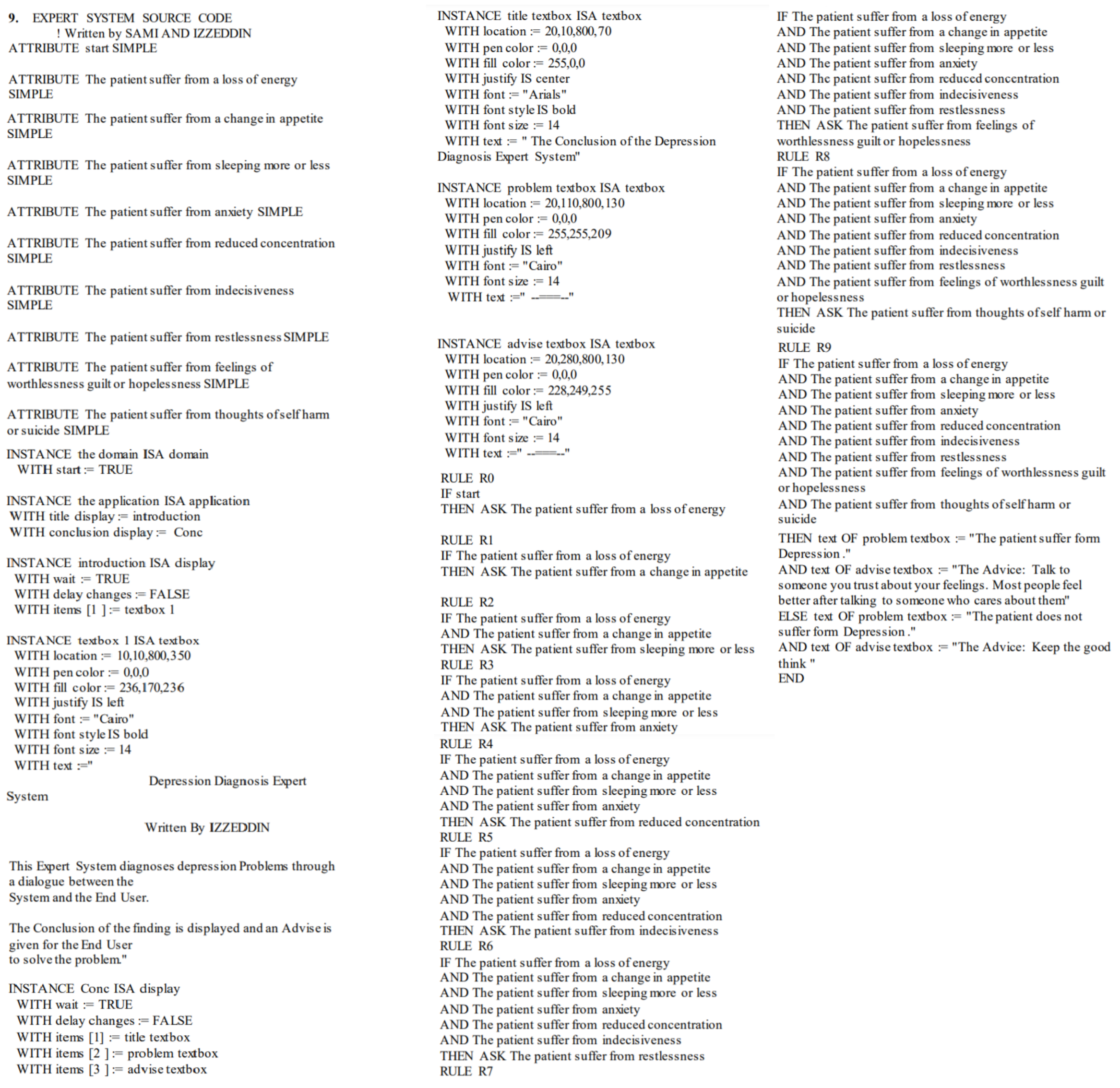
- **Atomic Categories** - these categories do not have wildcard symbols _ and * [1];
- **Default categories** - these contain patterns with wildcard symbols * or _ [1];
- **Recursive categories** - these contain templates that have <srai> and <sr> tags, that refer to recursive artificial intelligence and symbol reduction [1].

The technique used by the AIML interpreter in ALICE is to try to match word by word in order to obtain the longest pattern [1]. ALICE is considered one of the "most human" chatbots and won the Loebner Prize three times (2000, 2001 and 2004) even though it didn't pass the Turing Test [31].

Another relevant example that was created was an expert system developed using rule-based techniques with the goal of diagnosing *major depressive disorder*, commonly known as depression, and to give the correct treatment based on tips about the disease [4]. It diagnoses depression by asking the user to answer yes or no questions and giving a diagnosis and recommendations at the end. For the diagnose the system uses nine symptoms: loss of energy, change in appetite, sleeping more or less, anxiety, reduced concentration, feeling of worthlessness, guilt or hopelessness and thoughts of self-harm or suicide. This is a limitation since the system can only provide correct diagnoses based on the symptoms it specializes in [4]. Even though it is a great starting point, the fact that it only focuses on nine symptoms is a limitation.

This demonstrates that it is possible to create an illusion of intelligence of the computer using only rules, and that systems using rule-based techniques can deceive people into believing that they are talking to a human [13]. Furthermore, it also demonstrates that expert systems using rule-based techniques can be great for helping people with mental problems, such as depression [4].

In the figure 2.1, we can find the rules used in the expert system for diagnosing depression. The language used for the development of these rules was SL5 Object Language [4].



```
9.   EXPERT  SYSTEM  SOURCE  CODE
          ! Written by SAMI AND IZZEDDIN
ATTRIBUTE  start SIMPLE

ATTRIBUTE  The patient suffer from a loss of energy
SIMPLE
ATTRIBUTE  The patient suffer from a change in appetite
SIMPLE

ATTRIBUTE  The patient suffer from sleeping more or less
SIMPLE

ATTRIBUTE  The patient suffer from anxiety SIMPLE

ATTRIBUTE  The patient suffer from reduced concentration
SIMPLE

ATTRIBUTE  The patient suffer from indecisiveness
SIMPLE

ATTRIBUTE  The patient suffer from restlessness SIMPLE

ATTRIBUTE  The patient suffer from feelings of
worthlessness guilt or hopelessness SIMPLE

ATTRIBUTE  The patient suffer from thoughts of self harm
or suicide SIMPLE
INSTANCE  the domain ISA domain
  WITH start := TRUE

INSTANCE  the application ISA application
 WITH title display := introduction
 WITH conclusion display :=  Conc

INSTANCE  introduction ISA display
  WITH wait := TRUE
  WITH delay changes := FALSE
  WITH items [1 ] := textbox 1

INSTANCE  textbox 1 ISA textbox
  WITH location := 10,10,800,350
  WITH pen color := 0,0,0
  WITH fill  color := 236,170,236
  WITH justify IS left
  WITH font := "Cairo"
  WITH font style IS bold
  WITH font size := 14
  WITH text :="
                              Depression Diagnosis Expert
System

                              Written By IZZEDDIN

 This Expert  System diagnoses depression Problems through
 a dialogue between the
 System and the End User.

 The Conclusion of the finding is displayed and an Advise is
 given for the End User
 to solve the problem."

 INSTANCE  Conc ISA display
   WITH wait := TRUE
   WITH delay changes := FALSE
   WITH items [1] := title textbox
   WITH items [2 ] := problem textbox
   WITH items [3 ] := advise textbox

INSTANCE  title textbox ISA textbox
  WITH location := 20,10,800,70
  WITH pen color := 0,0,0
  WITH fill  color := 255,0,0
  WITH justify IS center
  WITH font := "Arials"
  WITH font style IS bold
  WITH font size := 14
  WITH text := " The Conclusion of the Depression
Diagnosis Expert  System"

INSTANCE  problem textbox ISA textbox
  WITH location := 20,110,800,130
  WITH pen color := 0,0,0
  WITH fill  color := 255,255,209
  WITH justify IS left
  WITH font := "Cairo"
  WITH font size := 14
   WITH text :="  --===--"


INSTANCE  advise textbox ISA textbox
  WITH location := 20,280,800,130
  WITH pen color := 0,0,0
  WITH fill  color := 228,249,255
  WITH justify IS left
  WITH font := "Cairo"
  WITH font size := 14
  WITH text :=" --===--"

 RULE  R0
 IF start
 THEN  ASK The patient suffer from a loss of energy

 RULE  R1
 IF The patient suffer from a loss of energy
 THEN  ASK The patient suffer from a change in appetite

 RULE  R2
 IF The patient suffer from a loss of energy
 AND The patient suffer from a change in appetite
 THEN  ASK The patient suffer from sleeping more or less
 RULE  R3
 IF The patient suffer from a loss of energy
 AND The patient suffer from a change in appetite
 AND The patient suffer from sleeping more or less
 THEN  ASK The patient suffer from anxiety
 RULE  R4
 IF The patient suffer from a loss of energy
 AND The patient suffer from a change in appetite
 AND The patient suffer from sleeping more or less
 AND The patient suffer from anxiety
 THEN  ASK The patient suffer from reduced concentration
 RULE  R5
 IF The patient suffer from a loss of energy
 AND The patient suffer from a change in appetite
 AND The patient suffer from sleeping more or less
 AND The patient suffer from anxiety
 AND The patient suffer from reduced concentration
 THEN  ASK The patient suffer from indecisiveness
 RULE  R6
 IF The patient suffer from a loss of energy
 AND The patient suffer from a change in appetite
 AND The patient suffer from sleeping more or less
 AND The patient suffer from anxiety
 AND The patient suffer from reduced concentration
 AND The patient suffer from indecisiveness
 THEN  ASK The patient suffer from restlessness
 RULE  R7
IF The patient suffer from a loss of energy
AND The patient suffer from a change in appetite
AND The patient suffer from sleeping more or less
AND The patient suffer from anxiety
AND The patient suffer from reduced concentration
AND The patient suffer from indecisiveness
AND The patient suffer from restlessness
THEN  ASK The patient suffer from feelings of
worthlessness guilt or hopelessness
RULE  R8
IF The patient suffer from a loss of energy
AND The patient suffer from a change in appetite
AND The patient suffer from sleeping more or less
AND The patient suffer from anxiety
AND The patient suffer from reduced concentration
AND The patient suffer from indecisiveness
AND The patient suffer from restlessness
AND The patient suffer from feelings of worthlessness guilt
or hopelessness
THEN  ASK The patient suffer from thoughts of self harm or
suicide
RULE  R9
IF The patient suffer from a loss of energy
AND The patient suffer from a change in appetite
AND The patient suffer from sleeping more or less
AND The patient suffer from anxiety
AND The patient suffer from reduced concentration
AND The patient suffer from indecisiveness
AND The patient suffer from restlessness
AND The patient suffer from feelings of worthlessness guilt
or hopelessness
AND The patient suffer from thoughts of self harm or
suicide
THEN  text OF problem textbox := "The patient suffer form
Depression ."
AND text OF advise textbox := "The Advice:  Talk to
someone you trust about your feelings. Most people feel
better after talking to someone who cares about them"
ELSE  text OF problem textbox := "The patient does not
suffer form Depression ."
AND text OF advise textbox := "The Advice:  Keep the good
think "
END
```

Figure 2.1: Rules used for diagnosing depression

Even though conversational agents using rule-based techniques are a revolutionary discovery in the computer science field, the answers they give to the users are often repeated and the chatbot cannot adjust its answers based on the input of the user.

2.2.2 Neural-based chatbots

Neural-based chatbots try to deal with the problem of creating a chatbot that restricts the conversation to small and predefined answers. A few techniques have been used throughout the years in different chatbot systems.

Huyen Nguyen, David Morales and Tessera Chin proposed in [35] a chatbot that tried to have personalities. This chatbot implements **sequence-to-sequence** learning (*seq2seq*), which is a model with an **encoder-decoder** architecture and is explained in detail in the section 2.1.Using this model, the authors were able to build a chatbot that could imitate the characters in popular TV shows - Barney from *How I met your Mother*, Sheldon from *The Big Bang Theory*, Michael from *The Office* and Joey from *Friends*. The results were promising, as they showed that more than 50% of people were likely to believe that a response was actually given by the real character.

There were other important innovations in the AI field that were useful in building chatbots, like **word2vec** explained in detail in section 2.5.1. In [60], the authors propose a system that automatically generates responses for users requests on social media. This system is trained with about 1M Twitter conversations between users and agents from over 60 brands. To generate word-embeddings the authors used the tweets as training data for word2vec models. This study's results were interesting, since they concluded that most requests from users on social media are emotional rather than formal requests. This shows that it is important for a system nowadays to have a more meaningful conversation other than quick questions and answers.

Mandy was a chatbot proposed in [36]. This system was proposed with the intent of assisting healthcare staff by automating the patient intake process. To generate word embeddings for this project, the authors also used the word2vec model, trained on a large dataset of medical documents. By training with word2vec, the system was able to find symptoms in the dataset that best aligned with the input description of a patient.

One chatbot proposed that is also interesting is "Touch Your Heart" [26]. In this work, the authors developed a chatbot that was tone-aware and tried to generate toned responses to users requests. They were able to identify eight different tones in costumer service: *anxious*, *frustrated*, *impolite*, *passionate*, *polite*, *sad*, *satisfied* and *empathetic*. Due to the fact that the normal seq2seq model is not capable of dealing with controlling tones, the authors of this chatbot proposed a novel **seq2seq** model that was capable of controlling the tones in a conversation. The results of this study showed that the system was able to generate toned responses as good as responses generated by human agents.

MILA was a deep reinforcement learning chatbot proposed in [47] by authors from the Montreal Institute for Learning Algorithms. This project aimed to converse with users on popular topics through both speech and text. The system created used a combination of natural language generation and retrieval models, including **bag-of-words, seq2seq** and latent variable neural network models. One important feature used in this system was the reinforcement learning, which helped the system with the selection of appropriate responses. This project was created for the Amazon Alexa Prize competition and proved to be one of the most engaging systems in the competition.

Neural-based chatbots are a field of study in technology that can still be thoroughly explored in different aspects. One of these aspects is the identification of emotions.

2.2.3 Emotions

Affective Computing is an interdisciplinary field that addresses concepts of computer science, cognitive science and psychology. It is defined as the "study and development of systems and devices that can recognize, interpret, process and simulate human affects" [7]. What started the development of this area in computer science was the paper of Rosalind Picard on affective computing [42]. The author defends that a system should be able to interpret the emotional state of humans and adapt its behavior to them. That means the system should try to demonstrate empathy and give answers appropriated to the emotion the user is feeling. We can understand affective computing as human-computer interaction where a device should detect and respond correctly to a user's emotion or stimuli. To extract meaningful patterns from the gathered data (that can be gathered using several techniques like speech or gesture recognition) affective computing uses machine learning techniques that deal with different aspects, like speech recognition, natural language processing, or facial expression detection [7].

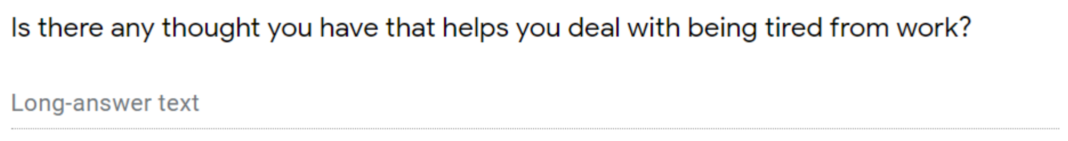
Another important and revolutionary aspect of conversational agents developed is the **sentiment analysis**, explained in section 2.1. According to studies [28], this technique is efficient for detecting positive emotions, but it still has a low performance when trying to identify negative utterances. In the paper "*Sentiment Analysis for AIML-Based E-Health Conversational Agents*" [28], the authors present five sentiment prediction models for conversational agents and conclude that all of them are good for predicting positive utterances, but do not score well when predicting negative utterances. One big problem that arises when trying to use sentiment analysis is the fact that this technique needs large data to train the model used, which can be quite hard to find depending on the topic being studied.

2.3 Chatbots & Mental Health Issues

As mentioned in the previous section, some works in the chatbots' field are focused on mental health problems [4] [40]. There are also a few other conversational agents in the field of psychiatry. In [56], the authors give a review of chatbots that are related to the matter of mental health since they believe it is important to have chatbots that can help giving access to mental health treatment specially to those reluctant to speak with a therapist. Among other benefits, the authors highlight that having chatbots for mental health problems allows for self-psychoeducation and adherence, and most importantly they report that "the effectiveness of chatbots with individuals with major depressive disorder further suggests that chatbots would be feasible to use in clinical populations" [56].

This connection between chatbots and mental health is a vast area with little free and accessible data available. To have data of people feeling burned out two forms were created - one of them was meant for people with jobs and the other one was meant for college students -, and the goal was for people to answer these forms and provide us with information about their mental health when it comes to feeling burnout. These forms are explained with great detail in chapter 3.

To decide what questions we wanted to put on the forms we focused on the questions of the *Copenhagen Burnout Syndrome Inventory*, which is an inquiry that aims to measure burnout on people and it focuses mainly on fatigue and exhaustion [5]. These questions are important because they were made by psychologists who understand the pathology and are more suitable to develop these types of questionnaires than engineers. Additionally, there were open-answer questions added (unlike the questions on the inquiry) so that the people questioned could openly write about what they felt regarded to each question.
In figure 2.2, we have an example of question presented on the form made for people with jobs: *Is there any thought you have that helps you deal with being tired from work?*.


Is there any thought you have that helps you deal with being tired from work?

Long-answer text

Figure 2.2: Question *Is there any thought you have that helps you deal with being tired from work?* presented in the form *Burnout - General*

In appendix A, you can see the form *"Burnout - General"* with all of the questions. Chapter 3 contains some explanations and results of the beta version of these forms and opinions related to the questions given.

Because the forms developed provide a small amount of data, it was important to discover more data for the development of this project. The problem with trying to find data for this thesis falls within the topic of this thesis: there are not many information available for free that is related to mental health problems, namely burnout and stress. Despite this, two other datasets were found and used as they proved to be helpful for our project:

- **Dreaddit** [53] - Dreaddit is a dataset that contains three thousand sentences from Reddit that are related to stress. To obtain this dataset, the authors of [54] took examples of posts from Reddit that include stressful and non-stressful text, as well as different ways of expressing stress. The goal of the authors was to provide a corpus that could help develop models that deal with solving health problems related to stress;

- **Stress Annotated Dataset (SAD)** [33] - SAD is a dataset that contains about 6850 sentences that are then classified into stress enablers for society. The authors of [33] developed the dataset with sentences that can be classified into 9 stressor categories. Their goal was to provide a dataset that could help build models and methods for providing the correct advice during a conversation with a chatbot about mental health.

2.4 YAML

YAML stands for "YAML Ain't Markup Language" and it is a data serialization programming language that is human-readable and easy to implement [8]. RASA, which is explained in 2.7, uses YAML to manage all the training data. One of the goals of YAML is to simplify its use for people working with data. Some of the structural characters used by YAML are:

- **Indentation** - commonly used for structure;
- **Colons** - colons are used to separate pairs of key/value;
- **dashes** - dashes are used to create "bulleted"lists.

2.5 Vector Representations of Words

This section is dedicated to *Vector Representations*, focusing on *word2vec* that is a way of representing words as numerical vectors [6]. It also mentions a few details about a more recent approach named *BERT* [14].

2.5.1 Word2Vec

Word2vec is a word embedding technology that represents words through numerical vectors in a high-dimensional space, maintaining the semantic and syntactic relationships of the words [6]. In short, word2vec builds word vectors that map words to points in space. These points encode effectively semantic and syntactic meanings. For example, the words *"sorry"*, *"apologize"* and *"glad"* are equally distant from each other in a discrete space, but by using word2vec we can represent these words in a continuous space and thus the distance between *“sorry”* and *“apologize”* is shorter than the distance between *“sorry”* and *“glad”* [60] Additionally, word vectors have certain algebraic relations [51]. For instance, "v[man] - v[king] + v[queen] $\approx$ v[woman]". In figure 2.3 there is a graphic designed by Nikhil Birajdar that demonstrates this relation.

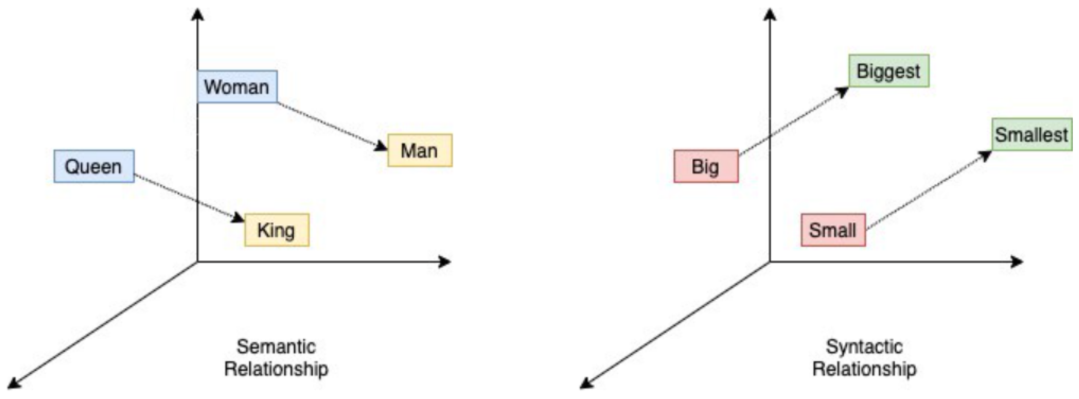


Image by author (Trained Word2Vec Vectors with Semantic and Syntactic relationship).

Figure 2.3: Word2vec vectors

Mikolov et al. used two neural networks present in the word2vec library - Continuous-Bag-Of-Words (CBOW) that predicts the current word based on the context and Skip-gram that predicts surrounding words given the current word [34]. Word2vec models are used to discover how related two words are and therefore are useful for synonym detection, concept categorization, semantic relatedness and a few other tasks that need to understand the relation between words.
A great advantage of word2vec is that it can provide pre-trained models which help in Natural Language Processing tasks.

In [34] Mikolov et al. present the results of word2vec in its very beginning. Firstly, they conclude that similar words tend to be close to each other in vector representation and can have multiple degrees of similarity. Therefore, it is possible to make the algebraic operations represented in figure 2.3. Additionally, they concluded that there can be different types of similarities between words. That being said, it is possible to pair words like "*big-biggest*" and "*small-smallest*" and ask questions like: "What is the word that is similar to *small* in the same sense as *biggest* is similar to *big*?" To answer this question we would have to compute the vector "X = *vector("biggest") - vector("big") + vector("small")*", and the answer would be the word closest to X in vector space measured by cosine distance. Finally, the last conclusion is that training "high dimensional word vectors on a large amount of data" generates vectors that can answer semantic relationship questions like a city and the country it belongs to e.g. Portugal is to Lisbon as Germany is to Berlin [34]. Table 2.1 is taken from [34] and demonstrates word pair relationships using word vectors from data trained in Skip-gram model.

The results in table 2.1 score about 60% of accuracy when assuming the exact match.

Relationship	**Example1**	**Example 2**	**Example 3**
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: IPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Table 2.1: Examples of the word pair relationships, using the best word vectors from Skip-gram model trained on 783M words with 300 dimensionality.

2.5.2 BERT

BERT stands for Bidirectional Encoder Representations from Transformers. Presented by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanovat at Google, BERT was a major innovation in Natural Language Processing because it applies the bidirectional training to language modelling, which was proven to be better at understanding the context of a sentence [14].

BERT uses the encoder mechanism of a Transformer to generate a language model. What it does is it reads the text input all at once, which allows the model to learn the context of a word having in consideration all of the surroundings of the word instead of only one side. This is an innovation in this field because up until BERT all systems read the words left-to-right or right-to-left.

This technique applies *pre-training* which consists of training the data on unlabeled data over different tasks and is computationally expensive, and *fine-tuning* where the system is initialized with the pre-trained parameters and then fine-tunes the parameters using labeled data. Fine-tuning is relatively inexpensive when compared to pre-training. BERT was trained on two tasks:

- **Mask Language Model** - For this task, 15% of the tokens were masked and BERT had to predict them from context [44];

- **Next sentence prediction** - BERT predicts wether one sentence is probable or not considering the first sentence [44];

2.6 Spacy

This section is dedicated to *Spacy*.
Spacy is a library in Python and Cython used for text processing [25]. It helps you build useful features because it gives you easy and efficient access to NLP technologies and therefore it can be used to build information extraction or natural language understanding systems.
Spacy is useful for this thesis due to the following reasons:

1. **Linguistic Features** - Spacy evaluates raw data and returns a Doc with annotations from that data. After, it can predict which tag or label applies to the context. If we provide enough examples to the system, it can generalize to the whole language - for instance, after the word "the" in English it most likely comes a noun. In figure 2.4, we can see an illustration of the dependencies and tags spacy gives to a sentence.

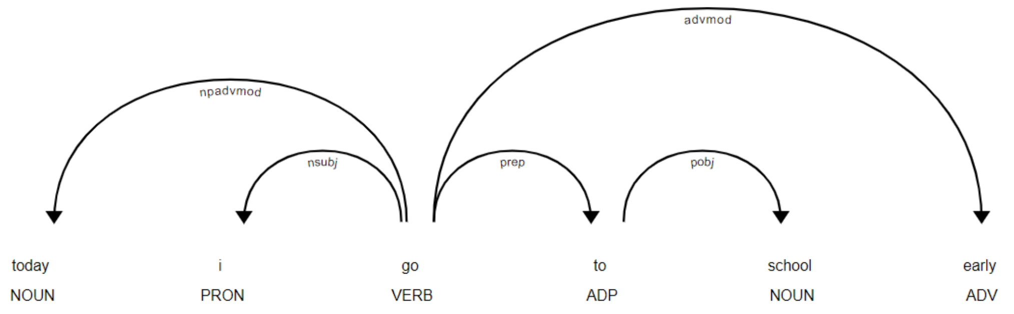


Figure 2.4: Word dependencies and tags using Spacy

   Another important linguistic feature in Spacy is that it allows to create and modify rules for tokenization. This way, some specific words that we wish to transform in tokens in a certain way can have special rules applied to them. Additionally, it also provides four different ways for sentence segmentation [49].

2. **Word Vectors** - To find similarity, Spacy uses word vectors. This is very helpful when we want to suggest something based on what the user is saying or looking for. Spacy implements **sense2vec** which allows the system to learn more interesting, detailed and context-sensitive word vectors. It also has a method called *similarity()* that gives you the similarity between two word vectors (or parts of it) [49].

3. **Transfer Learning** - Spacy supports transfer learning in order to improve efficiency and accuracy by reusing knowledge already obtained [49].

As we can see, Spacy has a lot of good features, namely being a fast *Natural Language Processing* software and including tokenization, sense2vec and liguistic features.

2.6.1 Tokenization

Spacy performs tokenization from left to right and does it in the following way [49]:

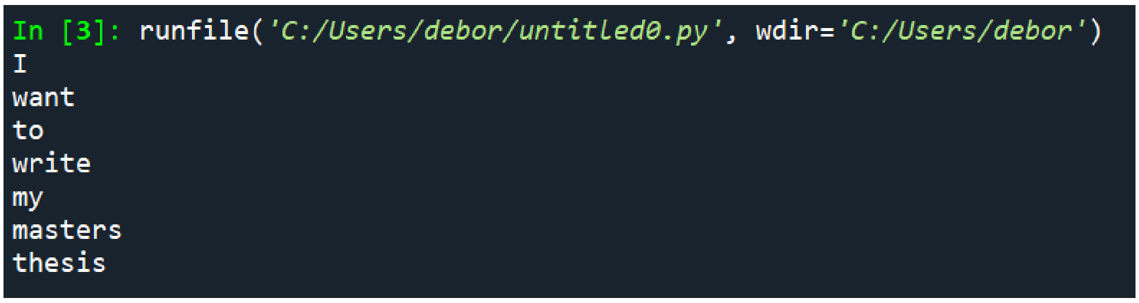
1. Firstly, it splits the tokens on whitespace;
2. Then it checks if it matches any tokenizer exception rule;
3. Finally, it checks for a prefix, suffix, or infix in a substring (it can be commas, periods, hyphens, or quotes). If it matches, the substring is split into two tokens.

Using tokenization in python while importing Spacy is really simple. Listing 2.1 illustrates the code used to turn into tokens the sentence "*I want to write my masters thesis*".

Listing 2.1: Convert sentence into tokens

```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp('I want to write my masters thesis')

for token in doc:
    print(token.text)
```

As we can see in listing 2.1, to perform tokenization we import Spacy, load *"en_core_web_sm"*, which is the english pipeline optimized for CPU and then we perform natural language processing to the sentence chosen. After this, we can print all the tokens presented in the class. The result of the code for tokenization is presented in figure 2.5.



```
In [3]: runfile('C:/Users/debor/untitled0.py', wdir='C:/Users/debor')
I
want
to
write
my
masters
thesis
```

Figure 2.5: Result of applying tokenization to the sentence "*I want to write my masters thesis*"

2.6.2 Parts of Speech

With Spacy, we can also identify parts of speech on a sentence since it contains a trained pipeline and statistical models that enable Spacy to make a classification of which tag a token belongs to [49].

Listing 2.2 contains the code used for tagging parts of speech of the sentence *"I want to write my masters thesis"*.

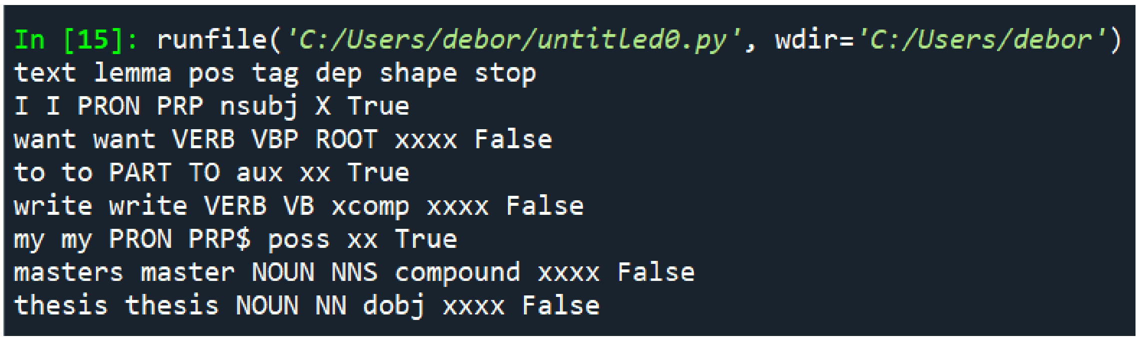
Listing 2.2: Parts of Speech

```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp("I want to write my masters thesis")
print("text lemma pos tag dep shape stop")

for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_,
    token.dep_, token.shape_, token.is_stop)
```

The code for POS is also simple. We begin the same way as tokenization, by importing Spacy and loading the pipeline. After, we save the sentence we want to tag and print all the important information we want:

- **Text** - the word we are evaluating
- **Lemma** - The base form of the word
- **POS** - The simple part of speech tag
- **Tag** - The detailed part of speech tag
- **Dep** - The syntactic dependency, that is, the relation between tokens
- **Shape** - The word shape – capitalization, punctuation and digits
- **Stop** - If the word belongs to the most common words of the language

Figure 2.6 illustrates the result of the code for tagging parts of speech of the sentence *"I want to write my masters thesis"*.



```
In [15]: runfile('C:/Users/debor/untitled0.py', wdir='C:/Users/debor')
text lemma pos tag dep shape stop
I I PRON PRP nsubj X True
want want VERB VBP ROOT xxxx False
to to PART TO aux xx True
write write VERB VB xcomp xxxx False
my my PRON PRP$ poss xx True
masters master NOUN NNS compound xxxx False
thesis thesis NOUN NN dobj xxxx False
```

Figure 2.6: Result of tagging POS of the sentence "*I want to write my masters thesis*"

2.6.3 Word Vectors & sense2vec

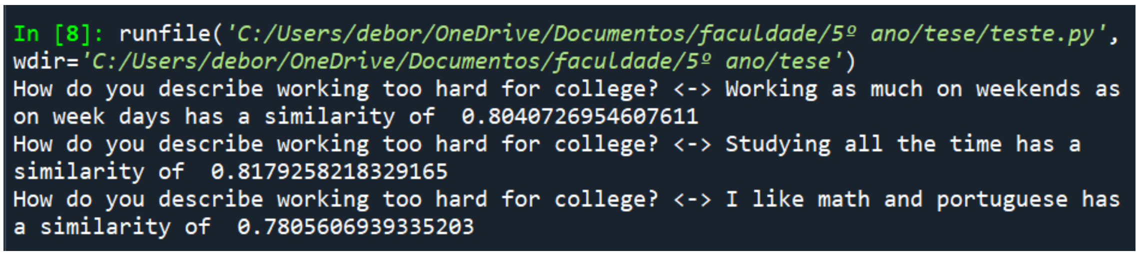
Spacy provides a few resources to work with word vectors. The first one is the method *similarity()* [49]. When given two word vectors, this method returns the similarity between the two. However, it might not be the best solution because it measures the distances between the word but it does not take into account the context of the sentences. For example, in listing 2.3 we have a code comparing three different sentences to the question *"How do you describe working too hard for college?"* - the first two sentences are answers to the question and the last sentence is a sentence written randomly.

Listing 2.3: Similarity between sentences

```
import spacy

nlp = spacy.load('en_core_web_lg')

doc1 = nlp("How do you describe working too hard for college?")
doc2 = nlp("Working as much on weekends as on week days")
doc3 = nlp("Studying all the time")
doc4 = nlp("I like math and Portuguese")

#Similarity of two documents
print(doc1, "<->", doc2, "has a similarity of ",
doc1.similarity(doc2))
print(doc1, "<->", doc3, "has a similarity of ",
doc1.similarity(doc3))
print(doc1, "<->", doc4, "has a similarity of ",
doc1.similarity(doc4))
```

In figure 2.7 we can see the results after running the code for comparing phrases. It is true that the first two sentences have a higher similarity to the question, but the difference between their scores and the score of the random sentence is not that significant. The similarity between the question and the first sentence is 0.80, between the question and the second sentence is 0.82 and between the question and the last sentence is 0.78. Therefore it is difficult to highlight the value that defines if a sentence is related to a question or not.

Sense2vec is a twist on *word2vec* that allows the system to learn more interesting, detailed and context-sensitive word vectors. The authors of [51] explain that sense2vec was developed as a method for sense disambiguation in neural word embeddings. Until sense2vec most word embedding techniques used a single vector to encode the potential meanings of a word, which created for words that had multiple meanings a "superposition in vector space where a vector takes on a mixture of its individual meanings" [51]. Sense2vec generates a vector for each sense of a word.



```
In [8]: runfile('C:/Users/debor/OneDrive/Documentos/faculdade/5º ano/tese/teste.py',
wdir='C:/Users/debor/OneDrive/Documentos/faculdade/5º ano/tese')
How do you describe working too hard for college? <-> Working as much on weekends as
on week days has a similarity of  0.8040726954607611
How do you describe working too hard for college? <-> Studying all the time has a
similarity of  0.8179258218329165
How do you describe working too hard for college? <-> I like math and portuguese has
a similarity of  0.7805606939335203
```

Figure 2.7: Code for similarity between phrases

Sense2vec's main goal is to approach the problem that word2vec cannot solve - take into consideration the context of a word. To do that, Sense2vec generates context-keyed word vectors for each context of a word - for instance, Apple has two vectors because it can either be a name of a company or a fruit [29]. It differs from **word2vec** because instead of having just one vector for each word that combines all senses of a word, it creates one vector for each sense of a word. Therefore, words with various meanings depending on the context will have more than one position in space, with each position corresponding to one vector of the word.
To work with Sense2vec we first need to assign one or more labels to a word, either manually or automatically. Then, we train a model to learn the embeddings of each sense i.e. each pair of word and label. Sense2vec can be added as a component to a Spacy pipeline and we can either initialize it with random values and train it or we can update a pre-trained model considering our goal. The Spacy pipeline contains *pos tagger* and *named entity recognizer* components that are used before sense2vec so that the sense2vec component can use the result of the previous components to create word senses.

One important factor of sense2vec is that it uses supervised natural language processing labels instead of unsupervised clusters in order to discover a word instance's sense, which eliminates the need to train embeddings multiple times. [51]

2.7 RASA

This section is dedicated to *RASA*.
RASA is a framework for developing AI powered chatbots. The goal is to give non-expert users who want to implement conversational AI systems access to recent advances [9]. This framework is split into two services, fully decoupled:

- **Rasa NLU** - Rasa NLU is used for natural language understanding. This part of Rasa has the advantage of allowing users to customize their NLU system. According to [9] language understanding is easy to configure in order to suit the needs of a project due to the fact that it is performed by a number of components implementing the same API. Rasa NLU can be treated as the "ear" of the system since it takes the input from the user and extracts the intents and entities that it recognizes;

- **Rasa Core** - Rasa Core is used for dialogue management. Here, the dialogue policy only receives entities and intents. Dialogue management is seen by Rasa as a classification problem, where Rasa Core predicts the next action to be taken in each iteration [9].

The fact that Rasa NLU and Rasa Core are decoupled allows the components to be used independently of one another. In figure 2.9 we can see what happens when a user sends a message to the chatbot. First, the message is sent by the user and Rasa NLU (the interpreter) extracts its intent and entities. After that, Rasa Core sends the extractions to the tracker, that maintains the state of the conversation. The next step is to send the current state of the tracker to the policy and this last one mentioned will choose what action to take. After choosing the next action, the action will be logged by the tracker and executed.

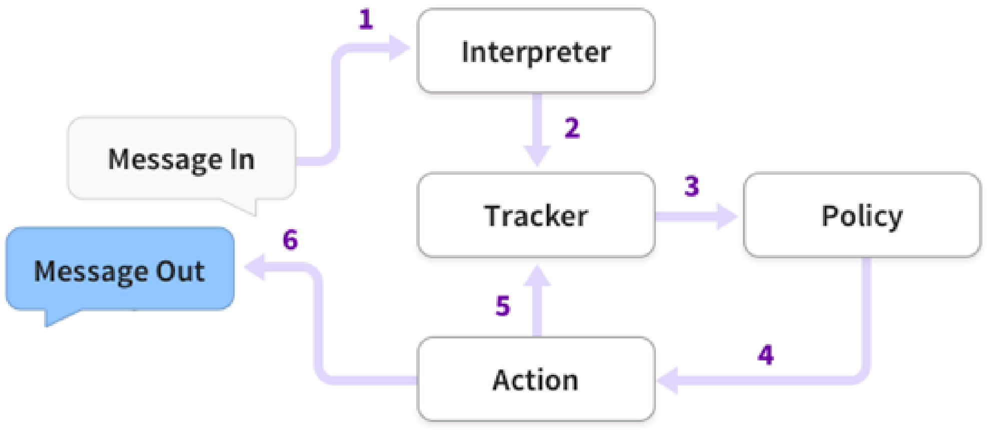


Figure 2.8: Rasa's procedure when receiving a message

For **language generation**, the creators of Rasa advise the developers to generate responses by authoring multiple templates for each response - they defend that this is not only easier but also more reliable than a neural network generating grammatically and semantically correct responses.

Rasa's **architecture** is modular, which makes it easy to integrate with other systems. Additionally, both Rasa Core and Rasa NLU can expose HTTP APIs, which makes it possible to use them in other projects using other programming languages other than Python. One feature of RASA's architecture that is important to mention is that the state of the dialogue is saved in a **tracker object**, the only stateful component in the system. There is only one tracker object per conversation session. The tracker object also saves the events that led to the current state of the system, which makes it possible to reconstruct the conversation.

The formats for **training data** (YAML) are human-readable for both Rasa Core and Rasa NLU, and the latter requires a list of utterances annotated with intents and entities. As for Rasa Core, it needs the specification of training dialogues (also known as stories). The body of a story is a sequence of events [9].

Regarding **pipelines**, it is important to understand how they work. There are two pipelins in RASA - one for NLU and the other one for policies.

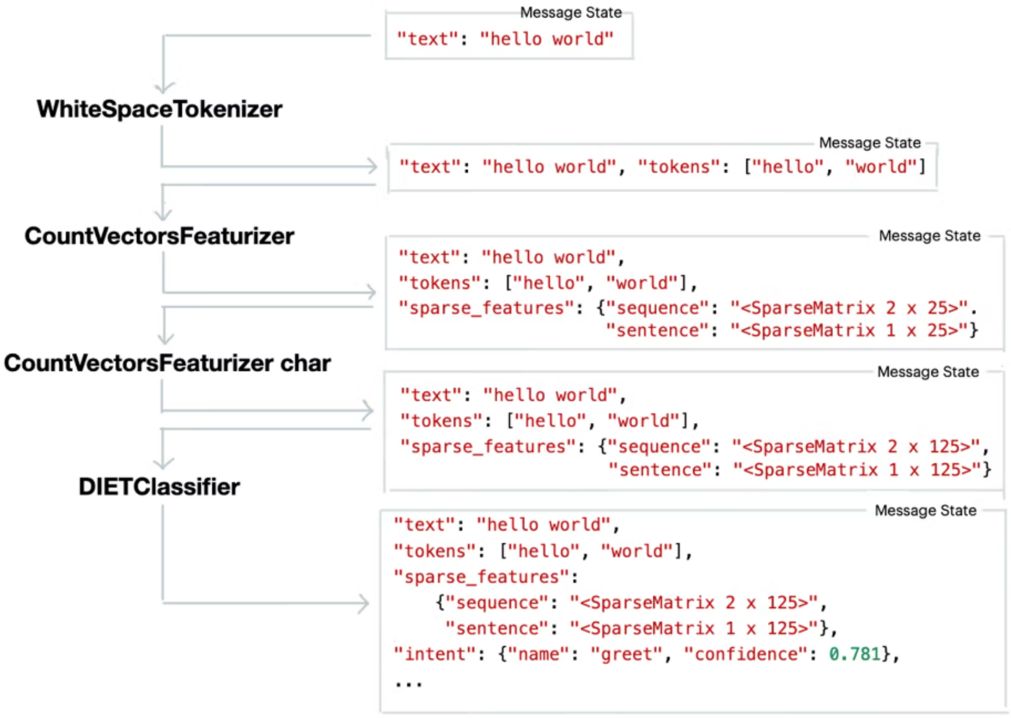
RASA NLU pipeline is a sequence of components that occur with the order presented in the pipeline and that can depend on the result of another component [58]. In RASA NLU pipeline there are 4 types of components that need to be defined:

- **Tokenizers** - These components are responsible for dividing the data into tokens. Some tokenizers add extra information to tokens which can be useful information for the next components [58];
- **Featurizers** - These components generate numerical features for machine learning models. They generate two types of features [58]:
  - **Sparse Features** - Sparse features only save non-zero values and their position in the feature vectors, which helps with computational savings;
  - **Dense Features** - Dense Features contain mostly non-zero values, which requires more memory.
- **Intent Classifiers** - These components are useful for identifying entities. They can also identify intentions [58];
- **Entity Extraction** - These components extract entities. It is very common to have more than one in a pipeline since a lot of these components are domain-specific - for example, extract telephone numbers [58].

The components, as mentioned before, can depend on each other - for instance, one component can calculate feature vectors, save it and later on another component can classify intentions based on those feature vectors. For each message sent by the user, RASA keeps that utterance in an object called "Message" - that object is processed in each step of the pipeline and its state is updated. Figure shows an example of the message state being updated on each step of the pipeline. The names of the components are examples of common components chosen for each step.

One advantage of RASA's pipeline is that it allows the developer to customize the pipeline with the components he wants to, even allowing for new components created by the developer to be used [58].

Adding to the NLU pipeline, there is also the **Policies Pipeline**. This pipeline predicts what is the next action to be taken at each step in a conversation, and uses the NLU predictions and also the state of the conversation up to that point to do that prediction [59]. There are three main policies in RASA:



```
"text": "hello world"
```

```
"text": "hello world", "tokens": ["hello", "world"]
```

```
"text": "hello world",
"tokens": ["hello", "world"],
"sparse_features": {"sequence": "<SparseMatrix 2 x 25>".
                    "sentence": "<SparseMatrix 1 x 25>"}
```

```
"text": "hello world",
"tokens": ["hello", "world"],
"sparse_features": {"sequence": "<SparseMatrix 2 x 125>",
                    "sentence": "<SparseMatrix 1 x 125>"}
```

```
"text": "hello world",
"tokens": ["hello", "world"],
"sparse_features":
    {"sequence": "<SparseMatrix 2 x 125>",
     "sentence": "<SparseMatrix 1 x 125>"},
"intent": {"name": "greet", "confidence": 0.781},
...
```

Figure 2.9: Message State in each component of the NLU pipeline

- **RulePolicy** - This policy decides the next step of the conversation based on rules written by the developer in the *'rules.yml'* file. Basically, it operates the rule with the best match;

- **MemoizationPolicy** - this policy checks whether there is a story that matches the ongoing conversation - if there is the prediction of the next action will depend on the story;

- **TEDPolicy** - This policy uses machine learning. Like the name suggests, the Transformer Embedding Dialogue (TED) uses a transformer to predict the next action and do entity recognition.

The three policies work in a *priority based hierarchy*, where RulePolicy operates before MemoizationPolicy and the latter operates before TEDPolicy. It is important to have stories defined by the developer since they are memorized by the MemoizationPolicy and also used as training data by the TEDPolicy. One important feature that RASA allows is that it is possible to configure the RulePolicy in a way that the developer can write a specific action to show the user if none of the policies makes a confident prediction of what action to take next. For instance, the bot can answer *"I'm sorry, could you rephrase?"* if it doesn't understand what was said by the user. This works by setting to true the RulePolicy's field **enable_fallback_prediction**, defining the threshold value in the field **core_fallback_threshold** and then setting the value of the field **core_fallback_action_name** to the name of the action to be taken.

Finally, it is important to highlight the fact that RASA can communicate easily with other APIs. This connection between the two tools allows users to integrate their chatbot to a website.

3

Theoretical Comparative Analysis

Following Chapter 2, it can be concluded that there are numerous ways of implementing a chatbot nowadays, and the resources used depend on what the developer intents. This chapter presents a detailed comparative analysis of the resources available in order to conclude what is the best solution for our specific problem.

3.1 Word2vec vs BERT

As mentioned before, both **word2vec** and **BERT** are techniques used for natural language processing. Despite this, they differ in a few aspects, represented in table 3.1.

	BERT	**Word2vec**
Context	Dependent	Independent
Word Ordering	Takes into account	Does not take into account
Embeddings	Needs the model	No need for the model, just the embeddings
Ouf-of-Vocabulary	Supports	Does not support
Computational Cost	High	Low
Easy to adapt to new domains	No	Yes

Table 3.1: Comparison between BERT and Word2vec.

Table 3.1 presents the main differences between the two models. BERT depends on the context of the sentence whereas word2vec generates embeddings that are independent of the context of the sentence. Therefore, BERT needs to receive a whole sentence as input in order to evaluate the surroundings of a word. Word2vec on the other hand only needs the word as input since it only produces one vector representation for each word - this means that even if a word has more than one sense they are all combined into one single vector. For instance, the word "bank" only has one vector in word2vec but BERT creates two different vectors for the same word, one for each meaning [22].

Another difference is that we do not need the model of word2vec in order to generate embeddings because all it is needed is the embeddings the model generated beforehand and there are pre-trained word embeddings available. When working with BERT we need to have the model since the embeddings will depend on the context of the input and therefore need to be created after receiving the input [22].
BERT has the advantage of supporting out-of-vocabulary words (by slicing the unseen word into subwords), whereas word2vec does not allow that feature. In terms of accuracy, BERT is a better model. However, it has a really high computational cost both in speed and space it occupies. Due to the fact that BERT occupies a lot of space, it becomes difficult to deploy in devices with limited resources. BERT is one of the best models for natural language processing. The fact that is has a high accuracy, it is trained on a big corpus and it is available and pre-trained in more than 100 languages makes it a strong candidate for the development of this project. However, it has a big inference time, which is not something desirable for a chatbot that is talking with a person feeling burned out. In conclusion, for this specific problem with limited time for development and the necessity for quick answers it was decided that BERT was not a good option for the stage of development we are in. Therefore, this project uses word2vec as a model for natural language processing.

3.2 SpaCy vs NLTK

SpaCy and NLTK are two of the most well known Natural Language Processing tools available. Both these approaches offer great strategies to deal with the processing of text so the decision was based on small details.

First, even though both SpaCy and NLTK process text as input, the tools return different outputs. NLTK works with strings, so the results are given in strings - this becomes complicated because the developer is required to analyse the documentation in order to learn how to deal with string handling. SpaCy returns objects, which is really helpful because words and sentences are objects with attributes and methods, thus not needing the string-handling system. The important information about each word is stored in an object and there are methods that return what the developer want to analyse [32].

Second, one key factor for our choice was performance. In figure 3.1 there are the results of running tests on the text of Wikipedia's article on NLP, which contains about 10kB of text, using both NLTK and SpaCy. As it is shown, Spacy outperforms NLTK in both word tokenization and Parts-Of-Speech tagging. NLTK performed better than SpaCy in sentence segmentation but that is due to the fact that NLTK simply splits text into sentences whereas SpaCy builds a syntactic tree for each sentence that contains way more information, as it is presented in picture 2.4 [32].

Third, unlike NLTK, SpaCy supports word vectors, which is really useful because word vectors can express relations between words and therefore calculate which words are better in certain sentences. Also, Rasa already has Spacy components (as optional

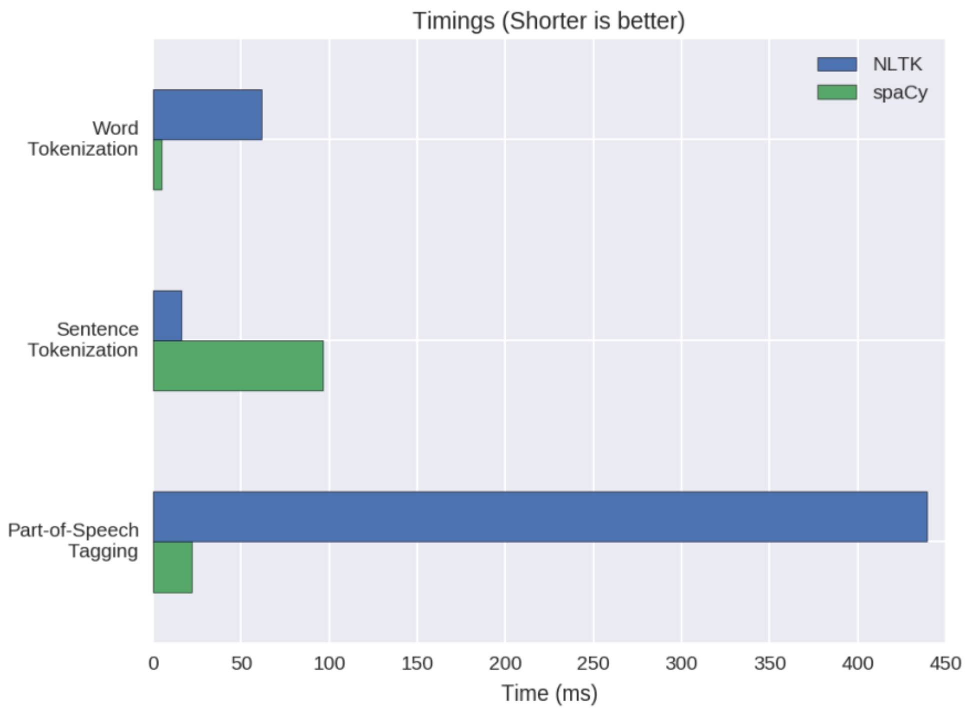


Figure 3.1: Differences in timings between NLTK and SpaCy

features) that return these word vectors and these components can be added to Rasa's NLU pipeline in order to do natural language processing.

Finally, when using NLTK developers can choose which algorithm they want to use since NLTK provides a lot of different algorithms. However, deciding which algorithm to choose can be complicated and take a lot of time. SpaCy, on the other hand, saves the best algorithm for a certain problem in its toolkit and when a better algorithm comes up it updates automatically, so developers don't have to worry about keeping up with what is the best algorithm at the moment. This can save a lot of time and makes sure that results always go as expected [32].

A major downside of Spacy is that it is not available in many languages at the moment [32]. However, for this particular project that is not a problem since we plan to develop a chatbot that uses the English language. Therefore, this project uses Spacy.

3.3 RASA vs other Frameworks

Currently, there are a lot of Chatbot Frameworks available to develop chatbots. Some are focused on certain tasks while others offer a simple way for companies to develop a chatbot quickly. In figure 3.2it is shown the 9 best Chatbot Development Frameworks.

All frameworks are great options for developing a chatbot so rather than comparing all of them we will simply mention the reasons why RASA was chosen.

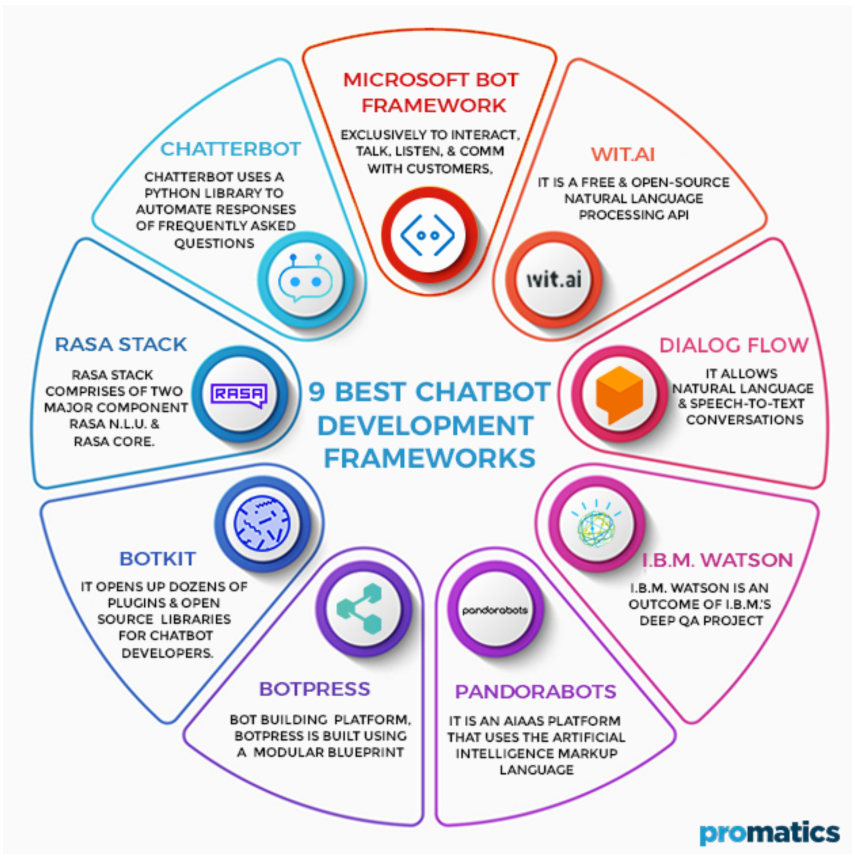

MICROSOFT BOT FRAMEWORK
EXCLUSIVELY TO INTERACT, TALK, LISTEN, & COMM WITH CUSTOMERS.
WIT.AI
IT IS A FREE & OPEN-SOURCE NATURAL LANGUAGE PROCESSING API
wit.ai
DIALOG FLOW
IT ALLOWS NATURAL LANGUAGE & SPEECH-TO-TEXT CONVERSATIONS
I.B.M. WATSON
I.B.M. WATSON IS AN OUTCOME OF I.B.M.'S DEEP QA PROJECT
PANDORABOTS
IT IS AN AIAAS PLATFORM THAT USES THE ARTIFICIAL INTELLIGENCE MARKUP LANGUAGE
pandorabots
BOTPRESS
BOT BUILDING PLATFORM, BOTPRESS IS BUILT USING A MODULAR BLUEPRINT
BOTKIT
IT OPENS UP DOZENS OF PLUGINS & OPEN SOURCE LIBRARIES FOR CHATBOT DEVELOPERS.
RASA STACK
RASA STACK COMPRISES OF TWO MAJOR COMPONENT RASA N.L.U. & RASA CORE.
RASA
CHATTERBOT
CHATTERBOT USES A PYTHON LIBRARY TO AUTOMATE RESPONSES OF FREQUENTLY ASKED QUESTIONS
9 BEST CHATBOT DEVELOPMENT FRAMEWORKS
promatics

Figure 3.2: Nine Best Chatbot Development Frameworks

One advantage of RASA is the fact that it offers a great amount of features for free. Additionally, it supports NLP and it is written in Python, a language that is common and has a variety of algorithms stored in packages, like word2vec. It also supports Javascript and allows the developer to integrate RASA with interfaces that are designed to improve the user's experience.

RASA is divided in two main components: **RASA Core** and **RASA NLU** - this is helpful because it decouples a bit the NLU from the development of a chatbot using rules while also allowing the developer to use both options if he finds more appropriate.

Adding to this, RASA can use different pipelines like one designed with Spacy components, and the developer can customize its own model and use it in RASA. This allows the developer to have more control on how the project performs and to adjust the project based on what are the main goals [57].

RASA is easy to use and provides a lot of documentation and videos that explain how to implement certain features, which is helpful when we want to start a new project from scratch and also if someone else continues this project later on [38].

One advantage of RASA is that it provides machine learning components created by RASA's team that work well for their specific tasks, which is great if the developer does not have a specific machine learning algorithm in mind that it wants to use in the project.

However, if the developer does in fact have an algorithm developed by himself that it wants to use for NLP, it might be difficult to do so since, in order to integrate its model in the project, the developer needs to download a project from github (Github project) and needs to make sure that both RASA and Python are using versions that are compatible, namely with the RASA framework and the github project.

4 Forms Developed Based on Psychological Questionnaire

This chapter presents a simple form based survey for stress complaints. An overview of the reasons for burnout in college students and workers is provided, as well as opinions provided by the participants about a future chatbot for burnout management.

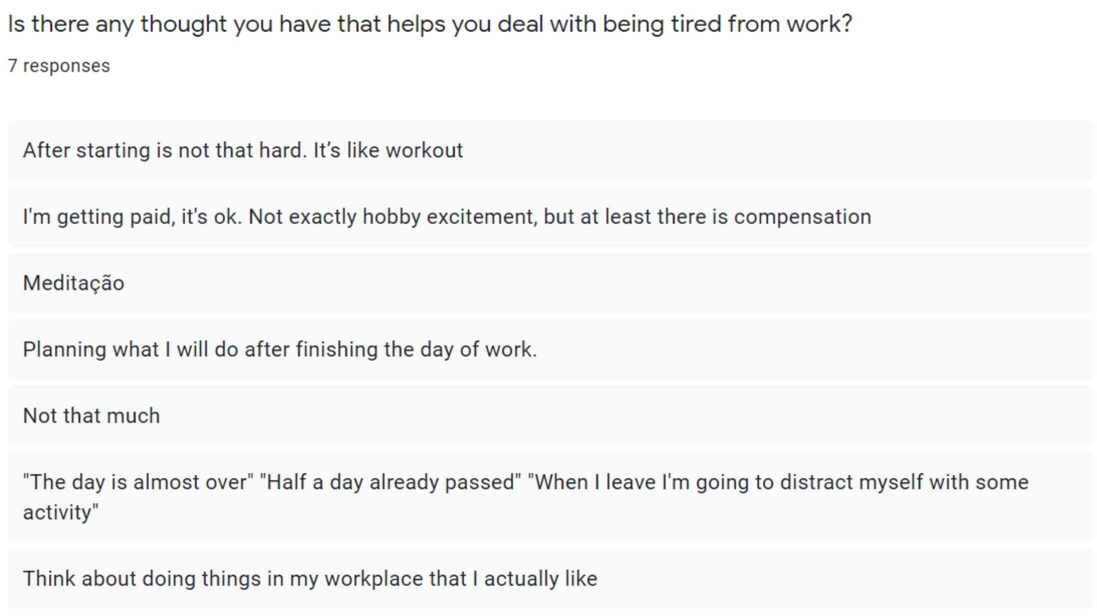
The results shown in this chapter also serve as a baseline for the development of the chatbot since they provide important data. Therefore, this data is studied and evaluated thoroughly in this chapter.

4.1 Results

In order to apply the forms developed,the participants had to be chosen. Our population sample consists of two groups of people:

- **College Students** - 87 college students were chosen to answer the questions of the form **Burnout in Students**. The sample was chosen randomly, by sharing the link of the form through different social media platforms and asking students to answer;
- **Workers** - 16 persons with jobs were chosen to answer the questions of the form **Burnout - General**. Like the students, workers were chosen randomly through social media. However, a few participants were hand picked to answer the form due to a less amount of people in this group answering the form through social media. These participants were relatives and close friends that gave us their contacts in order to send the form - their answers are anonymous just like all answers.

After applying the forms to our chosen population, we can conclude that the important part of the results for this thesis is the written answers, rather than the multiple choice ones. The multiple choice questions would be quite useful to develop a study about burnout, but for the goal of this thesis the written questions are preferable since they provide new information that the conversational agent can learn. For instance, in figure 4.1. it is shown seven answers for the question "Is there any thought you have that helps you deal with being tired from work?". These answers are very important because they let the agent know how different people deal with burnout i.e. what strategies they use and what type of thoughts help them relax. This is very useful considering that the main objective of the conversational agent is to help people dealing with burnout.


Is there any thought you have that helps you deal with being tired from work?

7 responses

After starting is not that hard. It's like workout

I'm getting paid, it's ok. Not exactly hobby excitement, but at least there is compensation

Meditação

Planning what I will do after finishing the day of work.

Not that much

"The day is almost over" "Half a day already passed" "When I leave I'm going to distract myself with some activity"

Think about doing things in my workplace that I actually like

Figure 4.1: Answers to question *Is there any thought you have that helps you deal with being tired from work?*

Figure 4.2 shows the answers for the question "Do you feel emotionally drained from your job?". These answers are multiple choice answers which means that this question is not very relevant for the chatbot because it does not provide any new information for it to learn. However, this type of questions is still very important for this survey due to the fact that the next questions asked are based on the answer of the user. For example, in the question present in figure 4.2, if the user answers *Never* or *Rarely*, it will lead to the question "Does you job make you frustrated?" but if you answer *All the time*, *Frequently* or *Sometimes*, it leads to the question "Why do you feel emotionally drained from your work?". There are also two options that the user can choose if he does not feel like the question is relevant or is written in a way that is not the most correct, which leads to questions about what would be a better thing to ask.

In conclusion, both questions are very important for this thesis: the open-text answers give new and relevant information about the topic asked and the multiple-choice answers lead to other questions that also provide new information.

Another important aspect of applying these forms to the target population is to understand if people agree with the topic of this thesis and are willing to share personal information with a conversational agent. To understand that, a few questions were asked after getting answers for the forms:

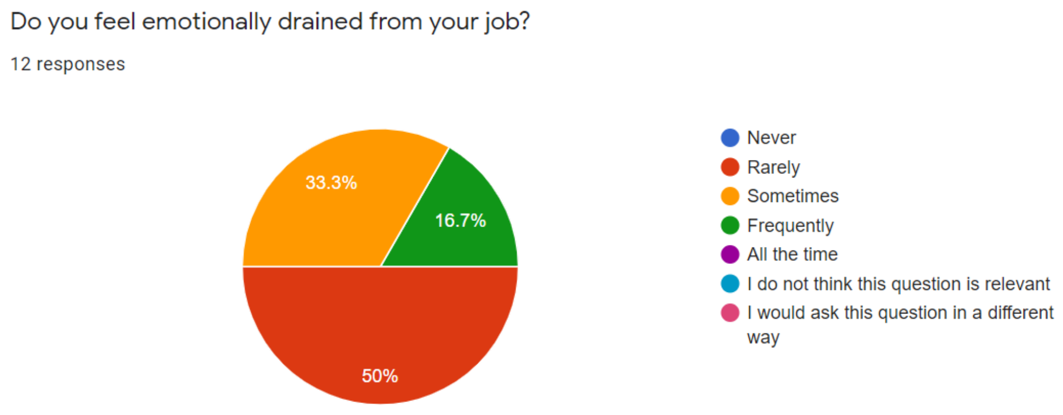


Figure 4.2: Answers to question *Do you feel emotionally drained from your job?*

- **Did you find the questions asked pertinent for the theme Burnout?** - When asked this question, the participants answered that they found the questions pertinent;
- **Did you feel like the questions were too intrusive?** - The participants did not find the questions in the form intrusive. They said that they did not feel uncomfortable sharing the information they shared while answering the questions;
- **Do you think it would be important to have a chatbot that could help people feeling burned out?** - All of the participants said that they find it important to have a chatbot that could help people feeling burned out;
- **Do you think companies should invest in ways to help their workers who are feeling burned out (e.g. using a chatbot)?** - In this question, there were different opinions among the participants. Half of them said that they believe companies should invest in ways to help their workers who are feeling burned out. However, the other half believes that companies should provide some type of help but feel like some ways of helping could be too intrusive or workers could feel uncomfortable;
- **Would you feel uncomfortable sharing personal experiences with a chatbot that is funded by your company?** - When answering this question, participants said that they would feel suspicious about sharing personal information with the chatbot because it could be a company's strategy to control their workers and therefore the participants believe that they would feel uncomfortable sharing personal experiences with the chatbot;
- **Would you feel comfortable sharing your personal details with a chatbot that does not belong to the company you work in (e.g. an app)?** - In this question, the participants had two different opinions - half would feel comfortable sharing their personal information, but the other half said they would not use this method as a way to deal with burnout;

- **Do you feel like a chatbot could be a good addition to therapy sessions?** This question was asked directly to a psychologist who believes that a chatbot could be a good complement to therapy but only as a way to alert users when they are in need of psychotherapy instead of a way to replace therapy.

After analysing the feedback given, some conclusions were made. Firstly, it is feasible to conclude that people would be receptive to talk to a chatbot about burnout, which is a positive conclusion taking into consideration the theme of this thesis. It is also possible to conclude that a chatbot for the burnout theme would be better implemented as an individual system (for example, an app) rather than a chatbot specifically for a company due to the lack of trust shown by the participants when asked about sharing personal details with a company's chatbot. Therefore, it is important to develop an interactive system that presents the questions and tries to engage in a conversation with the user without connecting this system to the users' company.

4.2 Results Explained

As mentioned before, two forms were developed in order to receive some input from participants. Due to the fact that we received more data from students, we decided to focus this work on how college students are feeling in the current days.
To do this, we did a detailed analysis of the questions on the form *"Burnout in Students"* and the answers given by the students to each question. Some of the information received is really important, not only for the development of this work, but also to understand the importance of this work and related topics.

4.2.1 What do you think your stress is related to?

What do you think your stress is related to? was one of the first questions on the form and it is a very important question for this thesis since it highlights the importance of works such as this project and provides information about the main sources of stress in college students.
Out of 87 people, 73 said that one of the reasons for being stressed was **college and/or work**, as we can see in figure 4.3. Additionally, students also mentioned other three reasons for their burnout. Due to this reason we believe it is important to focus our chatbot in the four topics provided by students for their main sources of stress: **covid-19, pressure, college** and **personal problems**. These answers also justify the aim of our project that is to develop a system that can provide a safe environment for students to vent and to provide a few solutions to reduce their stress, focusing the system on the main sources of burnout for students.

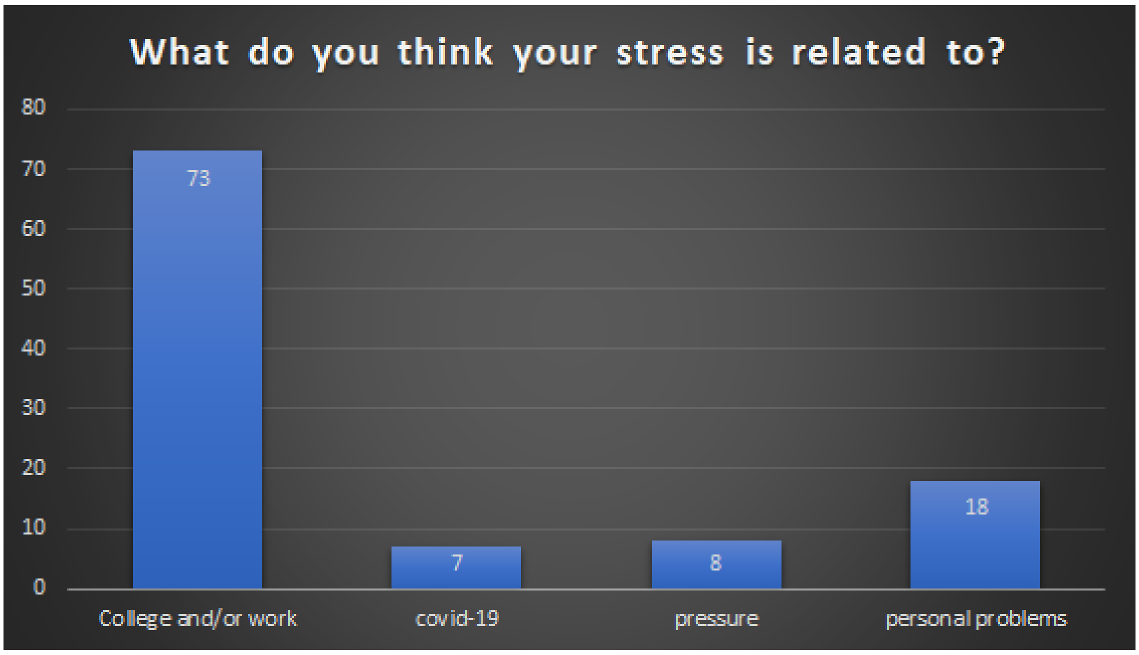


Figure 4.3: Answers to question *"What do you think your stress is related to?"*

4.2.2 How do you describe a good supporting system for burned out students?

The question *How do you describe a good supporting system for burned out students?* gave us a great insight on what students believe would help them overcome burnout. A great deal of students said that the main action that should be taken was to go to the root of the problem and act there - this means that people should focus on changing the students' stress enablers (for instance, teachers could reduce the amount of projects to be delivered).
However, the most said answer was that students should be provided with free counseling. **50 out of 72** students believe that some sort of counseling should be given to students, either by a psychologist, a platform where they could vent or both, as can be seen in figure 4.4.
Altough we are aware that a platform is not a substitute for a psychologist, we believe that a system well developed could help reduce stress levels and therefore work alongside psychologists to help students overcome burnout. This system aims to tackle students' burnout symptoms and understand if the student might be in need of psychological counseling.
We also analyzed the 50 **free counseling** answers and noticed that 13 out of the 50 students who answered that they wanted free counseling clearly stated that they would like to have a system that does regular check-ups on them and gives them moral support such as advice to keep the stress levels low.

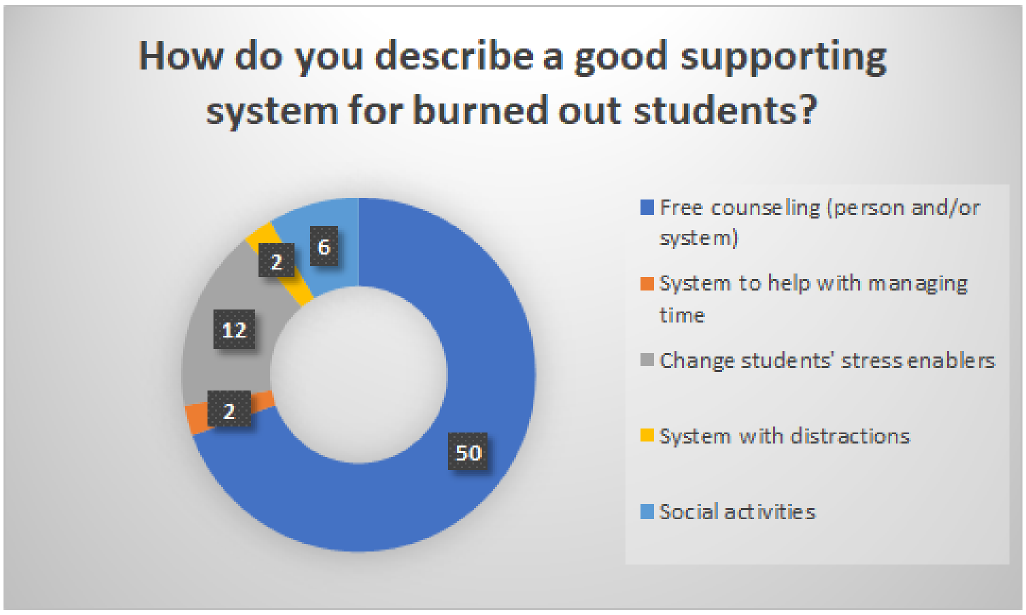


Figure 4.4: Answers to question *"How do you describe a good supporting system for burned out students?*

4.2.3 Other questions

There were other questions in the form developed, mainly focused on obtaining data related to burnout. These questions give us an insight on small details that the system might focus on when talking to a user.
The analysis of the question **Why do you feel emotionally drained from your studies?** shows that the answers given can be grouped into 4 different categories: *No rest/Workload*, *Pressure*, *No accomplishments/Not feeling good enough* and *No motivation*. These results are present in figure 4.5. This way, we can develop a system that is aware of these categories and ask some questions related to them. This helps the user to become more engaged in the conversation with the system since it will cover topics that users consider important.

One question also important to analyse and discuss results is **Is there any thought you have that helps you deal with being tired from school?**. These results are not that important for defining topics of conversation but rather to help the system. This question provides a few suggestions that people feeling burned out can follow in order to try to relax and therefore it teaches what the system can advise a user to do. For instance, as we can see in figure 4.6, a few students like to think about opportunities in the future after finishing their studies. Knowing this, we can develop a system that tries to influence the user to have more positive thoughts about what there is to come and therefore become more excited about the future and more motivated to finish their studies.

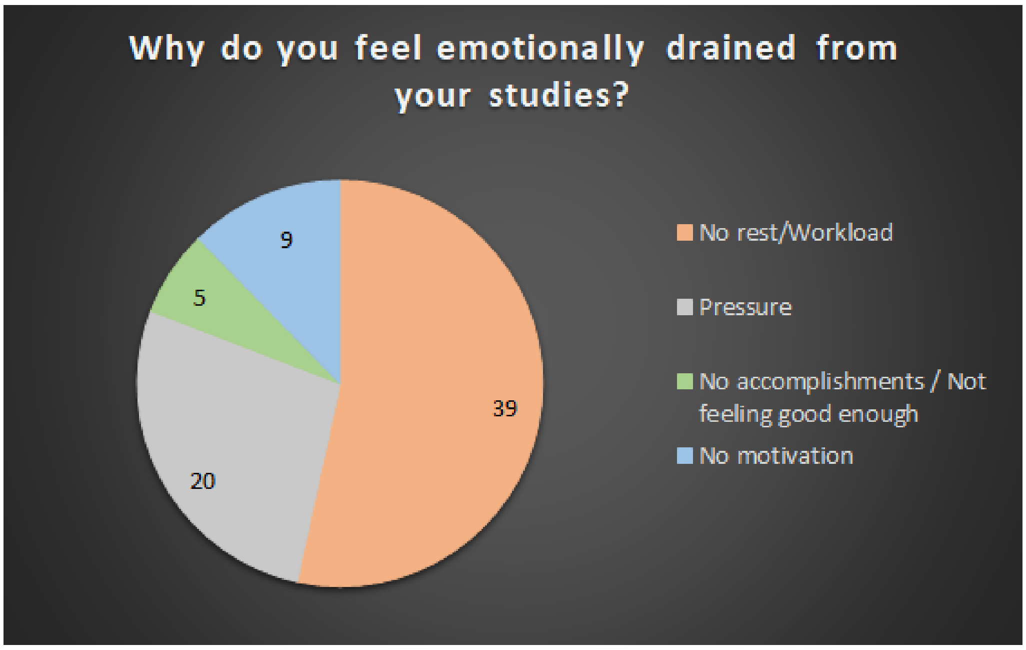


Figure 4.5: Answers to question *"Why do you feel emotionally drained from your studies?"*

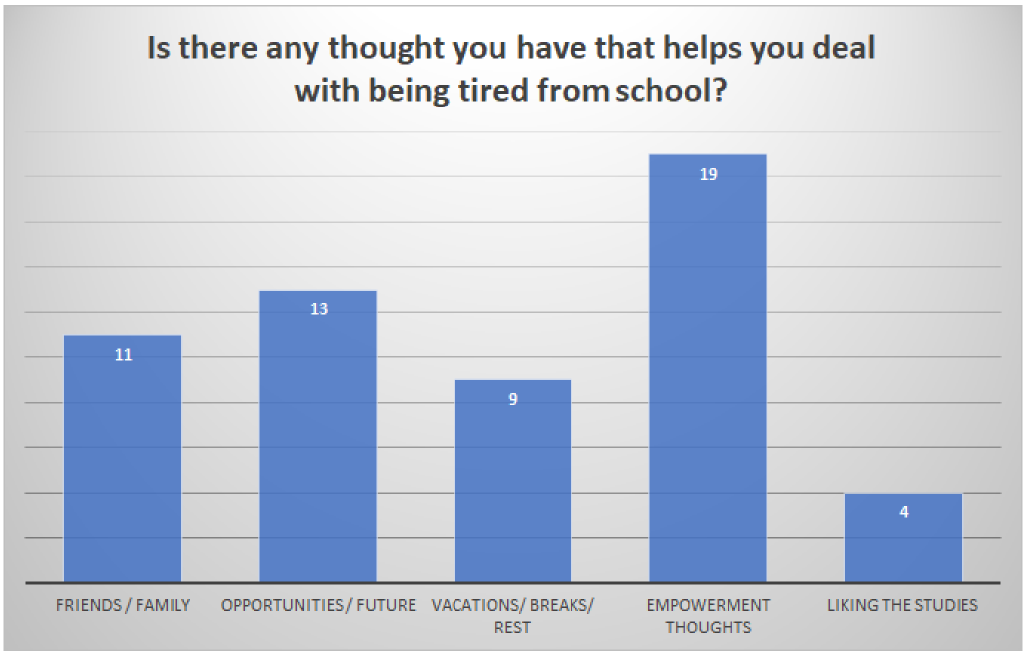


Figure 4.6: Answers to question *"Is there any thought you have that helps you deal with being tired from school?"*

5

Chatbot's architecture

5.1 Diagrams

Following the step of analysing the questions and answers in the forms, it was important to develop diagrams to represent the structure of this project. In order to do that, the topics for the chatbot had to be well structured and the main goals had to be set (the connection between neural and rule-based techniques, for example). In the following subsections we demonstrate the diagrams that were considered important for the development of this thesis.

5.1.1 Components Diagram

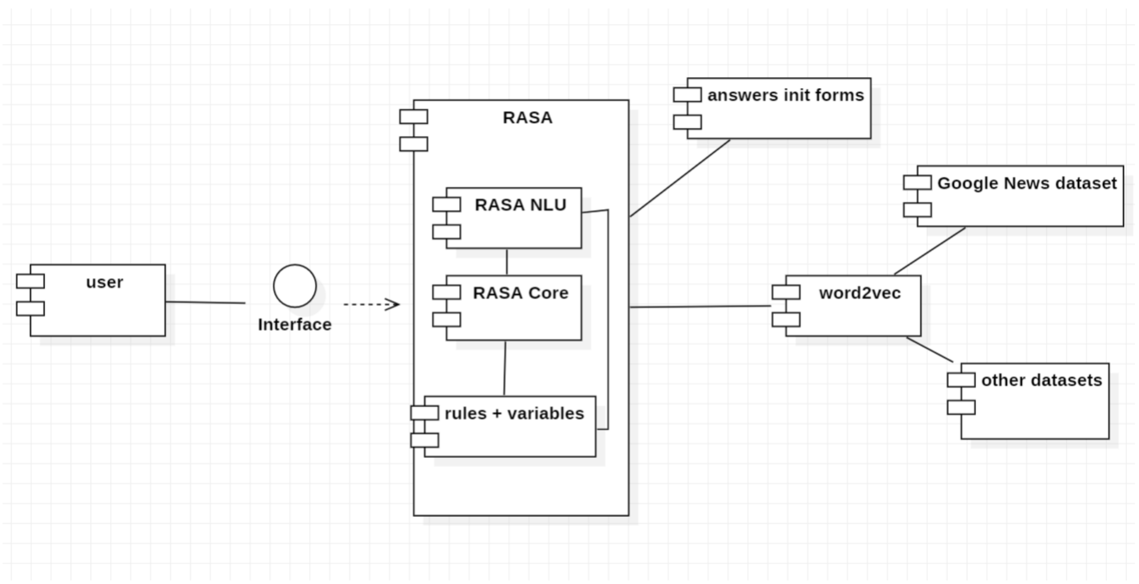


Figure 5.1: Components Diagram

Figure 5.1 represents the components diagram for this project. This diagram uses a simplified notation based on the UML (Unified Modelling Language) Component Diagram [11].

The user and the chatbot communicate through an interface. This interface is explained in more detail in section 6.4. The first action is taken by the user when it writes to the chatbot in the interface whatever it wants to say. After that, the message is sent to Rasa NLU, which is one component of Rasa. This component receives the message and tries to understand what is the intent of the user and to extract entities from the user's message. To do this, Rasa NLU performs natural language understanding to turn the user's input into structured data. Then, Rasa Core tries to decide what is the action that the chatbot must take based on the user's input and the data that contains examples of conversations and rules for the conversation to flow. Both components take advantage of the examples provided by the results from the initial forms - these results are previously studied and divided based on the intent that they represent, and are also converted into real life conversation examples. Additionally, RASA NLU and RASA Core also take advantage of another component from RASA, the "tracker" component. This component saves important information throughout a conversation session and RASA NLU and RASA Core might access this information whenever they need.

For Rasa to work, it contains pipelines that process the information and decide what actions the chatbot must take. These pipelines are explained in more detail in section 2.7.

One of the goals of this project is to integrate Word2vec in Rasa's pipeline to help mapping the words and vectors that are similar. Therefore, we have Word2vec as a component since we want to create word embeddings using word2vec that map words related to stress, burnout and words that are important in the context of this thesis. First we will only use the Google News dataset [20], which contains the mappings of 3 million words and phrases present in news. After, we will develop a dataset that contains information more related to the topic we want to approach - burnout. This dataset includes the words and phrases present in the answers of the initial forms. Additionally, the dataset contains words and sentences from two other datasets, explained in section 2.3.

The final approach will be to try to add our dataset to the Google News dataset, in order to add to the Google News dataset more examples related to our topic. Therefore, the word2vec component will be used in three different ways:

- as the Google News word2vec model provided by [61];
- as the model developed by us using the combination of three different datasets, all mentioned above;
- as a combination of the word2vec model developed by us and the Google News model.

The word2vec model that contains the combination of the two models used will be added to Rasa's NLU pipeline. RASA NLU will use this pipeline to process the input and predict the right intent of the user based on its input.

The final step is for the chatbot to return a message to the user, responding to its input. This will be done by RASA Core. Rasa Core will first analyse what was the intent determined by RASA NLU. Then it will use the policies pipeline to decide what is the utterance that the chatbot needs to give the user based on the prediction of the intent. Both pipelines are explained with greater detail in section 2.7.

5.1.2 Scripts Diagram

The scripts diagram uses the notation of the UML (Unified Modelling Language) Class Diagram [55]. The adaptation of this model was done in the following way:

- each **class** of the class diagram represents one script that the system might follow;
- the **attributes** of each class correspond to each variable that the script might have - for instance, in college script, the user might mention what motivates him/her to stay in school. If this happens, the system will save this reason for motivation in a variable, in case the chatbot wants to mention it later in the conversation;
- the **operations** (or methods) correspond to what is possible for the user to do in each script (i.e. the **intents** of the user).

The scripts diagram is present in figure 5.2.

Since each script corresponds to a topic of conversation for the chatbot to have with a user, the scripts were only defined after analysing the results of the forms mentioned in chapter 4. These results provided us not only with questions we might ask the users, but also with topics of conversation. After doing a thorough analysis of the data, we came to conclusion that the best topics to have on our chatbot were **pressure, college, covid-19** and **personal problems**, that were the reasons present in figure 4.3 when the users were asked *"What do you think your stress is related to?"*.

Six scripts were defined for our chatbot - four correspond to the reasons for stress in our users, one corresponds to the initial script and the other corresponds to the final script. Each conversation starts with the **initial script**, where the user greets the chatbot and makes small conversation with the system for it to learn what is the reason for the user's burnout. After, depending on the reason given by the user for its burnout, the conversation flows to the script that contains information related to the reason for burnout that the user mentioned (for instance, if the user chose "*college*" as its burnout reason, the conversation would flow to the College Script). The conversation in this step revolves around the reason that the user chose for burnout, therefore each one of the four scripts has different operations - these operations are explained in section 5.2. At the end, each conversation finishes with the same script, the **final script**. Here, the user asks for final advice and says goodbye to the chatbot.

There is also the possibility of the user not feeling burned out. If this happens, the conversation flows directly from the **initial script** to the **final script**, without passing a script that talks about a burnout reason.

Each script will be implemented as a story in RASA, and this implementation is described in section 6.1.1.

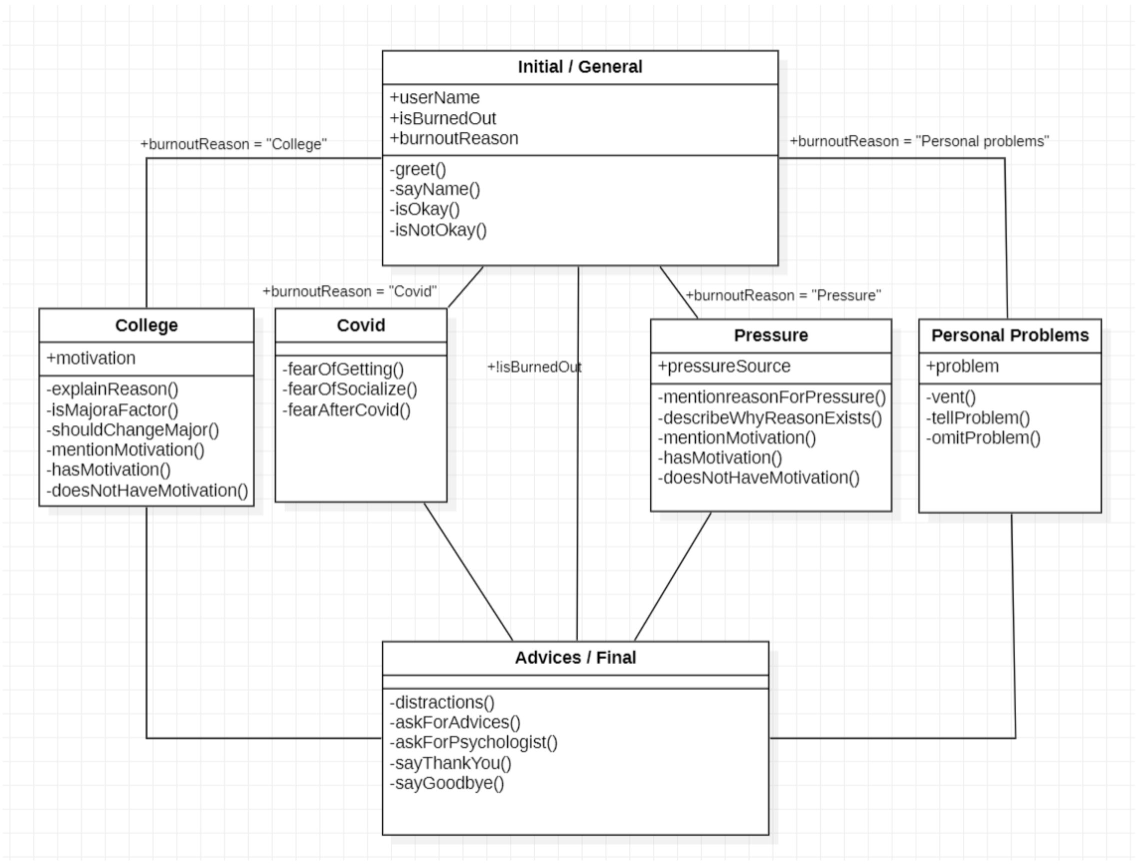


Figure 5.2: Scripts Diagram

5.2 Scripts

As mentioned in the previous section, the scripts that were defined for this project take into consideration the results of the forms that were applied in the early stages of the development of this project. Based on these results, there were 6 scripts established - the initial script, the final script and the four scripts that are specified taking into consideration the four main reasons for burnout present in figure 4.3. In each following subsection, there is an explanation of each script.

5.2.1 Initial Script

Every conversation starts with the **initial script**. The operations present in this class describe what can happen in the side of the user when a conversation with the chatbot begins. Additionally, in this script, the system can learn the user's name, if the user is feeling burned out and in the case where the user is feeling burned out the system can learn the reason for its burnout. This can be saved in the following variables - **username, state** and **reason** for burnout.

Variables

- username;
- state (is it burned out);
- reason for burnout.

Intents

1. **Greet** - the user starts by greeting the chatbot;
2. **Say Name** - the user mentions its name;
3. **Say it is okay** - the user says it is okay and therefore not feeling burned out;
4. **Say it is sad** - the user says that it is feeling sad;
5. **Say whether it is burned out or not** - the user decides whether it is feeling burned out;
6. **Say reason for burnout** - if it is feeling burned out, the user will mention what is leading to this feeling of distress.

5.2.2 Pressure Script

The pressure script appears when the user describes its burnout reason as "*pressure*". In this script, the system can save two important details - what motivates the user and who or what is putting pressure on the user. The motivation for the user is a really important information since the chatbot can use that to remind the user of the positive side of things and therefore make him/her feel less distressed.

Variables

- motivation;
- who/what puts pressure.

Intents

1. **Say reason for feeling pressured** - the user mentions to the system why it is feeling pressured;
2. **Describe why the reason for feeling pressured exists** - the user describes why there is a reason for feeling pressured;
3. **Mention motivation** - the user mentions what motivates him/her in life that might help with dealing with the pressure;

4. **Describe thoughts** - the user describes the positive thought he/she has, in order to feel less distressed;

5.2.3 College Script

The college script appears when the user describes its burnout reason as "*college*". In this script, the system can learn what is the motivation that the user has to keep studying. This motivation is really important since the chatbot can revolve the conversation around this motivation and persuade the user to feel less distressed.

Variables

- motivation for study

Intents

1. **Say why it chose college as reason** - the user mentions why college is making him/her feel burned out;
2. **Decide whether the college major influences its burnout** - the user says whether its major is making him/her feel more tired/stressed;
3. **Decide if it should change majors** - the user debates with the system if it should change majors in college;
4. **Mention motivation** - the user says whether there is a motivation that keeps him/her studying for college;
5. **Describe thoughts** - the user describes the positive thoughts it has related to college;

5.2.4 Covid Script

The covid script appears when a user is feeling very distressed due to the covid-19 pandemic (which is a common reason for distress nowadays). There are no variables in this script that the system might save, but the conversation is important nevertheless, to calm the user.

Intents

1. **fear of catching the disease** - the user describe its worst fears related to the possibility of catching the disease (namely, passing the disease to a parent or grandparent);
2. **fear of repercussions** - the user mentions whether it is afraid that some side effect might last;
3. **fear of socializing** - the user talks about the fear of socializing with other people when there is a pandemic.

5.2.5 Personal Problems Script

The personal problems script appears when the user is having personal problems that are making him/her feel very stressed. Here, the system can learn what is the problem that is leading to this feeling of stress and try to focus the conversation on that problem in order to try to find solutions for it.

Variables

- problem

Intents

1. **Decide whether to tell or omit problem** - the user might not feel comfortable sharing its problem, so it can choose if it wants to share the problem or not;
2. **Vent** - if it decides to share the problem, the user can vent to the system about it.

5.2.6 Final Script

The final script always appears when a conversation is ending. It is an important script since it can provide final advice to calm the user.

Intents

1. **Request advice** - the user can decide to request some advice from the system;
2. **Talk about distractions** - the user can talk to the chatbot about something that will distract him/her of the feeling of burnout;
3. **Thank the bot** - the user might have found the information given by the system useful and therefore want to thank the chatbot;
4. **Ask about psychology** - the user might need help finding psychological help;
5. **Say goodbye** - the user might want to say goodbye to the system.

6 Implementation

[1] This chapter covers the implementation of our system. More details can be seen in Thesis' Folder.

6.1 RASA

The first step in this chatbot's implementation was the development of the chatbot itself. The tool chosen for the development of the chatbot was RASA - this tool is explained in greater detail in section 2.7.

RASA has 4 main files that need to be changed in order to develop the chatbot:

- **nlu.yml** - this file contains the intents of the user and examples of what the user might say in each intent;
- **domain.yml** - this file contains what the bot will answer to the user. Here, we only write the bot's utterances; the logic between the user's intent and the bot's utterances is in another file;
- **stories.yml** - this file contains almost all the logic that needs to exist for the chatbot to work correctly;
- **rules.yml** - this file contains more optional logic to help the chatbot when some things are mandatory to happen. For instance, anytime the user asks if it is talking to a bot or a person, it should be mandatory to have the bot answer that it is in fact a bot. This type of logic is present in a rule.

6.1.1 Stories

To implement each script, we decided to use the **stories.yml** file. Therefore, each story present in this file corresponds to a script present in section 5.2. The following code illustrates the code necessary for the pressure script to work:

[1] https://drive.google.com/drive/folders/1t26Lohbs2BT1gs2MeOHaCTS-q8ZddqmD?usp=sharing

Listing 6.1: Pressure Story

```
- story: pressure_story
  steps:
  - intent: greet
  - action: utter_greet
  - intent: say_name
  - action: action_save_name
  - slot_was_set:
    - name: debora
  - intent: mood_sad
  - action: utter_is_burnedout
  - intent: user_confirm
  - action: utter_sad
  - intent: mood_pressured
  - action: utter_pressure_begin
  - intent: reason_for_pressure
  - action: utter_why_reason
  - intent: explain_pressure_reason
  - action: utter_sorry_for_pressure
  - intent: user_oppose
  - action: utter_advice_pressure
  - intent: user_confirm
  - action: utter_first_advice_pressure
  - action: utter_second_advice_pressure
  - action: utter_third_advice_pressure
  - action: utter_advice_therapy
```

As seen in the code above, a story is composed by **intents** and **actions**. Each *intent* represents what the user wants to do in a given moment and each *action* represents what the bot needs to answer based on the intent of the user. By having stories established, the system already knows how to proceed when the user has a certain intent thanks to the **MemoizationPolicy** present in the policies pipeline of RASA. This policy and pipeline are explained in section 2.7.

Additionally, a story needs a name, as it can be seen in listing 6.1. The creation of the name does not have a mandatory rule that needs to follow, but it helps to choose something that makes it easier for the developer to identify the story, since it can become confusing when there are a lot of stories in the file.

In the code present in listing 6.1, there is also one important factor worth mentioning. When the action *"action_save_name"* takes place, the slot *name* is set with a value (in this case, "debora", which was the name used when creating this story). This happens because RASA allows the developer to build its own functions in **Python**. These functions are useful for many reasons - for instance, the one used in this story saves the name of the user, which is useful if the bot wants to learn it and say it later on in the conversation. The following code represents this action:

Listing 6.2: Python Function: action_save_name

```
from typing import Text, Dict, Any, List

from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher
from rasa_sdk.events import SlotSet


class ActionSaveName(Action):
    def name(self) -> Text:
        return "action_save_name"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
    domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        text = tracker.latest_message['text']
        dispatcher.utter_message(text=" Nice to meet you {text}!
       How are you feeling?")
        return [SlotSet("name", text)]
```

What happens in this function is that the system will read the input of the user - **tracker.latest_message['text]** - which will correspond to the name of the user. After this, it will send a respond to the user, mentioning the user's name in its response - **dispatcher.utter_message(text= "Nice to meet you {text}! How are you feeling?"**. Finally, it will save the text that the user sent (its name) to the slot named *"name"* - **SlotSet("name", text)**.

Functions are, therefore, important to store information that the user might give and might be trivial to keep.

In order to build a story, it is needed both the input of the user and the utterance that the bot will say. The input of the user is taken care of in the *nlu.yml* file.

6.1.2 Intents

The intents that the user might have in a conversation with the chatbot are described in the **nlu.yml** file.

In this file, the developer has to define what different intents the user might have during each conversation with the chatbot - therefore all possible intents have to be represented in this file. For each intent defined, the developer provides examples of inputs that the user might give to the chatbot. By doing this, the developer is providing the system with information for it to learn and therefore be able to predict an intent through the input of the user. The next code represents the intent **greet**:

Listing 6.3: Greet Intent

```
- intent: greet
  examples: |
    - hey
    - hello
    - hi
    - hello there
    - good morning
    - good evening
    - morning
    - hey there
    - let's go
    - hey dude
    - goodmorning
    - goodevening
    - good afternoon
```

In the intent **greet** there are examples of ways a user can greet the chatbot. Of course these examples are not the only ways the user can greet the chatbot, and this is where RASA uses *natural language processing*. When the input is not present in the examples of the correct intent, the system uses natural language processing in the input given and tries to decode the input in order to associate it with an intent. Therefore, it is important to have a lot of examples for each input, because the bigger the training data the more information the system knows about the intent. This is explained with more detail in section 6.2.1.

After deciding what are the intents the user might have, the developer needs to decide what the bot will answer based on the intent said. Therefore, the developer needs to edit the **domain.yml** file.

6.1.3 Utterances

The domain file is responsible for a few things, but most importantly for storing the **utterances** that the chatbot might say to the user.

Just like in intents, the developer needs to provide examples for each utterance that the bot might have. In each utterance, the developer provides different texts that correspond to the answers the bot might say if the utterance is the next action to be taken. The developer can provide as many texts as it wants and Rasa will pick one randomly to show the user. In the code below there is the **utter_is_burnedout** utterance:

Listing 6.4: Is Burnedout Utterance

```
utter_is_burnedout:
- text: Are you experiencing burnout?
- text: I'm sorry to hear that. Do you feel burnedout?
```

The utterance shown in listing 6.4 comes after the user mentions it is feeling sad, frustrated, stressed, etc.. The developer can add as many phrases as it wants for the chatbot to answer, RASA will simply choose randomly which one to show the user. In this particular example, there are two texts RASA can choose from to answer the user:

- Are you experiencing burnout?
- I'm sorry to hear that. Do you feel burnedout?

However more texts could be added if we wanted. The domain file is not only responsible for storing the utterances of the chatbot. All the slots that exist are created in this file - slots are used to store information throughout the whole session. For instance, the listing below shows the creation of the slot "name", that stores the name of the user throughout the whole conversation session:

Listing 6.5: Is Burnedout Utterance

```
slots:
  name:
    type: text
    mappings:
    - type: custom
      action: action_save_name
```

To define a slot, first we need to define the name of the slot and the type - in this case, the slot is named **name** and its type is **text** because it is filled with a string that represents the name of the user. After, we need to provide its mapping, that is, the way we will obtain this slot. In this case, we will obtain the slot through a custom action developed in Python, that is called *action_save_name*. Therefore, the type of mapping will be **custom** and the action that contains the slot is **action_save_name**.

Entities can also be defined and can be extracted through the NLU components in the RASA nlu pipeline, called entity extractors. In this project, we have defined one entity called **reason** and it represents the reason why the user is feeling burned out.

Slots and entities are very similar things, however they differ in a small aspect worth mentioning. Entities are values that can be extracted from the user's input, and are extracted through NLU components. Slots can be extracted in multiple ways - for example, our slot "name" is extracted using a Python function, therefore the value is "calculated" instead of extracted from the input through NLU components. Entities can also fill slot values, if the type of mapping for the slot is *entity*.

Another thing that can be defined in the domain file are **buttons**. Buttons are used in utterances where the user needs to select one option. In our project, a button was used in the utterance that asks users what is the reason for their burnout so that they could select the reason instead of explaining it by text. The buttons' options correspond to the topics of conversation that the chatbot provides and these topics follow the scripts provided in 5.2. Below, we have the code for the creation of the buttons:

Listing 6.6: Burnout Reason Button

```
utter_sad:
- text: Why are you feeling like this?
  buttons:
  - title: Pressure
    payload: '/mood_pressured{{"reason": "pressure"}}'
  - title: Covid
    payload: '/mood_covid{{"reason": "covid"}}'
  - title: Personal Problems
    payload: '/mood_p_problems {{"reason": "personal problems"}}'
  - title: College
    payload: '/mood_college{{"reason": "college"}}'
```

In listing 6.6, the first thing done is describing what the bot will ask the user when the utterance occurs:

- Why are you feeling like this?

After, we define the text present in each button (the **title** argument):

- **button 1** - Pressure
- **button 2** - Covid
- **button 3** - Personal Problems
- **button 4** - College

Finally, we map the choice of the user to the correct intent - this mapping is done in the **payload** of each button (for instance, if the choice of the user is **pressure**, it maps to the intent **mood_pressured**). The payload can also have extra information - for example, on these buttons we also pass the value of the entity **reason** taking into consideration what button is being clicked.

Last but not least, all the intents that are being used in the system (therefore all the possible intents that a user might have during a conversation) have to be present in the domain file, like it is shown below:

Listing 6.7: Intents

```
intents:
- bot_challenge
- explain_college_reason
- explain_pressure_reason
- goodbye
- greet
- mood_college
- mood_covid
- mood_great
- mood_p_problems
- mood_pressured
- mood_sad
- reason_for_pressure
- say_name
- user_confirm
- user_oppose
```

After defining all of the intents and utterances of the chatbot, it can have a conversation following one of the stories created in the *stories.yml* file. However, there is one more file that is responsible for logic.

6.1.4 Rules

The **rules.yml** file contains important information for a conversation to work correctly. Rules define what is **mandatory** to happen in the conversation after the user says certain things. Rules can break the course of a story. For instance, we have the following rule:

Listing 6.8: Rule for Goodbye

```
- rule: Say goodbye anytime the user says goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye
```

This rule means that everytime the user wants to say goodbye, the chatbot will say goodbye as well. So, if a user ends the conversation with a goodbye in the middle of a story, the story won't continue but rather the bot will answer goodbye since it is what the rule tells the system to do. This happens because, as explained in section 2.7, the policies present in the policy pipeline have priorities and the **RulePolicy** has the higher priority than the others. By having the highest priority, the first thing the system does is evaluate whether there are any rules that it needs to follow to give an answer. If there is, it returns what the rule demands it to return.

It is important to be careful with rules, because they can break all the logic created in a story and even stop stories from happening. For example, the *user_confirm* intent, where the user agrees with something, can happen in a lot of parts of a story. If there is a rule saying that after the user_confirm intent happens there has to be a certain utterance happening, each time the user confirms something that utterance will happen. If this utterance is not the action to be taken in a story, the story is broken because of a rule.

6.2 RASA's pipelines

As mentioned before, Rasa uses pipelines to do its interpretation of messages and to decide the path the chatbot has to follow in a conversation. These pipelines contain a lot of components that can be chosen by the developer depending on the developer's needs.

For this project, we will work on the **NLU pipeline**. This pipeline is necessary because it uses natural language processing to process the user's input and turn it into a structured output. There were two approaches for this pipeline that were tested and evaluated in this project. The first one we called it the **Spacy Pipeline** and it is explained in section 6.2.1. The second one is called **Word2vec Pipeline** and it is explained in section 6.2.2.

6.2.1 Spacy Pipeline

This pipeline, like the name indicates, essentially uses Spacy components to do natural language processing, namely entity extraction and intent recognition. Spacy is explained in section 2.6. The pipeline is present in the following listing:

Listing 6.9: Spacy NLU pipeline

```
- name: SpacyNLP
  model: en_core_web_md
  case_sensitive: False
- name: SpacyTokenizer
- name: SpacyFeaturizer
  pooling: mean
- name: RegexFeaturizer
  case_sensitive: False
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
- name: ResponseSelector
```

```
  epochs: 100
```

The goal of this thesis is not to understand deeply the architecture of the components used in our pipelines, but a brief explanation for each component is given.

The first component we use in this pipeline is called **SpacyNLP** and its use is to initialize all Spacy structures, meaning that any pipeline that uses Spacy components needs to have this initializer. This component has two attributes:

- **model** - it indicates which Spacy model the pipeline will load. In this case, we chose the *en_core_web_md* [**Spacy-2016**];
- **case_sensitive** - this field decides if the casing of a word is relevant for the mapping of its word vector - for instance, hello and Hello would return two different word vectors if this field was set to true. In our case, it was set to false since it was not relevant for this project to have it set to true.

The second component is called **SpacyTokenizer**. This component is used to turn words into tokens, using the Spacy tokenizer. An example of tokenization in Spacy can be seen in section 2.6.

The third component of the pipeline is **SpacyFeaturizer**. This component creates a vector representation of the whole sentence given by the user. This vector can be calculated in two different ways, specified in the **pooling** attribute - the component can either use max pooling or mean pooling. By observing results, it was possible to conclude that the results for this project came out better when using the "mean" option.

RegexFeaturizer is our fourth component. This component is also a featurizer, but it is useful in the pipeline since it creates a list of regular expressions in our training data. Then, Rasa will mark whether one of the regular expressions was found or not in the user message. Later on, these features ("markers") will be fed into an intent classifier and entity extractor in order to make the classification simpler (because it is assumed that the classifier learned in the training phase that these features represent certain entities or intents). In this component we also decided to not make it case sensitive.

The fifth element is **LexicalSyntacticFeaturizer**. This featurizer uses a sliding window that goes through every token and creates features for each token - these features contain information regarding the lexical and syntactic information of each word. A feature can indicate if the word is upper or lower case, if it begins with an upper letter and the rest is lower, if it is a digit or not and also if it is the beginning or the end of a sentence.

The last featurizer we have, which shows up two times in our pipeline, is the **CountVectorsFeaturizer**. This featurizer creates a bag-of-words representation of a user message, intent or response. What this means is that the featurizer will count how many times each token of a sentence exists in our data and after doing this for the whole sentence, it saves it in a matrix representation. The tokens that consist of only digits are assigned the same feature. Because the resulting matrix would contain a lot of zeros, it is used a sparse representation of the matrix (to save memory and computational time). The first time this featurizer appears, it simply creates a bag-of-words representations for each word in a sentence. However, the second time it appears, we check for characters instead of words. What it does is it creates characters n-grams only from text inside word boundaries. The boundaries we set were the minimum of one character and the maximum of four characters.

After, we have the **DIETClassifier**. DIET stands for Dual Intent and Entity Transformer, which means that this component is used to extract both entities and intents. This component has a transformer as the central piece of the algorithm, and this transformer outputs the following features:

- features that are then combined in the CRF (Conditional Random Field). This field is very important since it receives the entities from the sentence and the outputs of the transformer and by combining them, it calculates the *entity loss*;
- a representation for the class token. This token is a summary of the entire sentence and it is important because, when combined with the intent of the sentence, it gives us the *intent loss*;
- if we have a masked token, the transformer also outputs its representation and by combining this representation with the representation of the missing token, we get the *masking loss*.

With this, we have an algorithm that tries to average the 3 losses and come up with a component that can do these three tasks, which is very helpful in a chatbot. The only field we configured in this component was the *epochs* field, which represents the number of times the algorithm will see the training data. The bigger the number, the more information the algorithm might learn. However, after some epochs the algorithm is not learning any more useful information and it is costing computational time. Therefore there needs to be a balance between feeding information to the algorithm and the velocity in which the model is trained. The number we chose was 100.

The last component of the pipeline is **ResponseSelector**. This component basically uses the DIETClassifier architecture, but only the parts that are needed for the intent classification. Therefore, all the useful parts for calculating the intent loss are kept and slightly modified and the rest is not used in this component. This component returns, for each sentence given as input, an array with the intent that it classified the sentence with the confidence level as well as a ranking of the top candidates for intents with their rankings.

In section 7.2 we have the results of using this pipeline in our chatbot.

6.2.2 Word2vec Pipeline

This pipeline is called Word2vec pipeline because it uses the word2vec model created by us as a component. The creation of our word2vec model is explained with detail in section 6.3. The pipeline is present in the following listing:

Listing 6.10: Word2vec NLU pipeline

```
– name: WhitespaceTokenizer
– name: LexicalSyntacticFeaturizer
– name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
– name: rasa_nlu_examples.featurizers.dense.GensimFeaturizer
  cache_path: path\to\model.kv
– name: DIETClassifier
  epochs: 100
```

The first component of this pipeline is **Whitespace Tokenizer** and it simply creates tokens by separating character sequences by white spaces.

The second and third components of this pipeline are **WhiteSpaceTokenizer** and **CountVectorsFeaturizer** and these components are already explained in section 6.2.1 related to the Spacy pipeline.

The important part of this pipeline is the **GensimFeaturizer**, the fourth component. This component is not present in Rasa itself but can be imported to rasa from a github repository present in `https://github.com/RasaHQ/rasa-nlu-examples` - this repository contains machine learning components that are compatible with Rasa. This particular component, the Gensim Featurizer, contains the Gensim python library which is a library that is very helpful when a developer wants to create its own word vectors. Among other important features, the Gensim library provides word2vec, making it possible for developers to create their own word2vec model. That is why this component is crucial in this pipeline - without it it would not be possible to use our word2vec model in a RASA pipeline. Therefore, this component was added to the pipeline and it was specified which model we wanted to use. The model used was a model that was developed during the course of this thesis and it is described in section 6.3.

Last, there is the **DIETClassifier** component, which is also a component explained in section 6.2.1.

6.3 Word2Vec

Word2vec is a technique that is used for natural language understanding and therefore it is a good resource for this project. It works by learning word associations using a dataset and is helpful to predict words' synonyms. Word2vec is explained with detail in section 2.5.1.

In this project, our first goal related to word2vec was to use the Google News dataset model, provided by Google in [20]. As mentioned before, this model contains 3 million mappings of words and sentences that were obtained through news present in Google.

First, the model from Google had to be downloaded from [20] and KeyedVectors had to be imported from the gensim Python Library, which is a library that contains a lot of resources for natural language understanding [17]. **KeyedVectors** is a module that essentially maps keys to vectors. This model was important because word2vec creates vectors for each word on a sentence. Therefore, using KeyedVectors it was possible to map a word to its respective word vector created by the word2vec model. This module was used due to the fact that the Google News dataset model contains a lot of data and by using the KeyedVectors it only has to load the vectors that exist in the model. This saves up computational time because the module only uses the necessary data structures.

Following the step of importing KeyedVectors, the Google News dataset model had to be loaded and specified that it was in the word2vec format, using the method **load_word2 vec_format**. Then, experiments with the model were done to see if it contains information useful for our project. The results of these experiments are described in 7.1. The code to load the Google News model is present in listing 6.11.

Listing 6.11: Load Google News model

```
from gensim.models import KeyedVectors
model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-
negative300.bin', binary=True)
```

The second step of trying to use word2vec in our project was to implement our own model using data collected that was related to the topic of burnout. Aside from the data obtained from the initial forms, the datasets used for this implementation are described in section 5.1.1.

To implement our model, it was necessary to analyse our data to decide what it was useful for our project. After analysing it, we created an excel file with all the answers that could be used as data for our model. Because our dataset was small, we had to get more data and the rest of the data was obtained by dowloading the datasets mentioned in section 5.1.1: the **Dreaddit** dataset was downloaded from [53] and the **Stress Annotated Dataset (SAD)** dataset was downloaded from [33].

All of our datasets came in the excel format, so they had to be converted to json format. To do that, it was used the pandas module from Python. **Pandas** is an open-source module from Python that is used for manipulation and analysis of data [41]. Using pandas, the three excel files needed were selected and concatenated into one excel file, called *final_output.xlsx*. After that, this last excel created was converted to json and stored in a file called *final_data.json*, using the method **.to_json('./final_data.json')**. The final json, that is the data used for our word2vec model, contains **11129 examples** of words and sentences related to stress and burnout. The code for the creation of the final json containing the data for the model is present in listing 6.12.

Listing 6.12: Creating data file

```
import pandas as pd

df1 = pd.read_excel('all_answers.xlsx', sheet_name='answers')
df2 = pd.read_excel('SAD_v1.xlsx', sheet_name='SAD', usecols=
['text'])
df3 = pd.read_excel('dreaddit.xlsx', sheet_name='Sheet1',
usecols=['text'])


df = pd.concat([df1, df2, df3])
df.to_excel('final_output.xlsx', index=False)

excel_data_df = pd.read_excel('final_output.xlsx', sheet_name=
'Sheet1')
excel_data_df.to_json('./final_data.json')
```

After having the file with the data for the model, it was possible to create our word2vec model since we had data to train the model.

The first step to achieve our model was to import both pandas and gensim modules, since there were features from both modules that were used. Secondly, we loaded the json file that was created called **final_data.json**, containing the data. After this, the data was "cleaned" - that is, all the examples that had empty values were eliminated. This has to be done in order for the next step to work - the step where the data preprocess was applied. Gensim comes with a method (**simple_preprocess**) that preprocesses the data before building the word2vec model - this method lowercases and tokenizes the data, turning our json file into a list of tokens to be used as data for the model. However, this method does not work with empty values, so these empty values had to be eliminated first.

Following the steps to structure the data to a correct format, the model had to be developed. To develop our word2vec model with our data we used the **Word2vec** model that comes with gensim. Using this model, we specified our sentences, window, min_count, workers and epochs:

- **sentences** - this field contains the data that the model will use to train - our final json file containing the data of our 3 datasets;

- **window** - this field indicates the maximum distance between the current and the predicted word within a sentence. This field varies from 2 to 10 and, after tests, it was decided that the best results came when using the value 5;

- **min_count** - this field indicates a value that means that the training will ignore all words with total absolute frequency lower than the chosen value (this value can be between 2 and 100). The value chosen was 10. Since we do not have a lot of data, it was best to choose a small value;

- **workers** - the number of working threads to train the model. The bigger the value, the faster the training. We chose 4 to be the value for this field since the system has 4 cpus (as checked using the method **cpu_count()** from the Python os module);

- **epochs** - this field represents the number of iterations over the corpus. The value chosen was 20.

After specifying the model, the key vectors were saved in a file called **model.kv**. The results of using this model are described in section 7.1. The code for the creation of our model can be seen in listing 6.13.

Listing 6.13: Creating our word2vec model

```
import pandas as pd
import gensim

df = pd.read_json('final_data.json')

sentences = df.text
clean_data = sentences.dropna()

reviewText = clean_data.apply(gensim.utils.simple_preprocess)

model = gensim.models.Word2Vec(sentences=reviewText, window=5,
          min_count=10,
          workers=4,
          epochs=20)

model.save("model.kv")
```

Last, the goal was to connect the Google News model with our own word2vec model that contains our data. This was planned to see if adding our model more specified to stress and burnout to the Google News model would increase performance. However, this step was not achievable since the method that did the intersection between word2vec models (**gensim.models.Word2Vec.intersect_word2vec_format**) was discontinued. Therefore, it was decided to keep both models and see which one would behave better in our field of study. This comparison is shown in section 7.1.

6.4 HTML & JavaScript

The last part of the implementation of this project was to develop an agreeable interface so that the user could feel comfortable talking to our chatbot.

To do this, we used Javascript and HTML. The HTML developed for the website's main page was done by us. However, the code for the chatbot present in the main page of the website was implemented using the code present in the following github: Rasa Webchat.

In the final HTML code, there are two Javascript scripts. The first one is used to add calming music to the website. This music can be played and stopped whenever the user wants, since it is activated and deactivated through a *Play* button. The second script contains the chatbot's configurations:

- **initPayload** - the initial payload contains the first message that the chatbot sends when the user opens the chatbot;
- **customData** - here, it is specified that the language used was english;
- **socketURL** - this field contains the url that is going to be fetched to have the bot working. It is set to localhost since the chatbot is not working on an internet domain;
- **title** and **subtitle** - these fields contain the title and subtitle shown on the widget;
- **other configurations** - there are other optional fields that can added. In our case, we added:
    1. **hideWhenNotConnected** - which declares that the chatbot widget is hidden when the connection to the url cannot be established;
    2. **showFullScreenButton** - which shows the button to set the widget to full size;
    3. **showMessageDate** - which shows when the message was sent.

Our chatbot widget works using a socket that works as an endpoint for sending and receiving the messages between the user and the bot in real-time. To connect to a socket, we have to specify in the *config.yml* file that a socket is being used. The first two arguments of this specification define the event names that RASA uses when sending or receiving messages through socket.io [38] and the third argument defines whether the session is to be restarted or not when the user reloads the page (in this case, due to privacy matters, we believe it is important to restore the session in each reload). The following code contains the configuration of the socket:

Listing 6.14: Socket Configuration

```
socketio:
  user_message_evt: user_uttered
  bot_message_evt: bot_uttered
  session_persistence: false
```

7 Results

In this section, the results for each part of the development of this thesis are presented and explained.

7.1 Word2vec

One of the goals of this project was to study whether word2vec was a good option to be used as a model for natural language processing in our system. To do that, some tests were applied.

First, we had to decide which model to use in our final pipeline. The two models we had available are explained in more detail in section 6.3, but essentially there are two models for us to test - the model developed by us and the Google News model. The first tests compare these two models in order to help us decide which one to choose.

7.1.1 Most Similar

The first test done was to check the most similar words to the word **stress**. The word stress was chosen due to the fact that the topic of this thesis falls within the topic of stress, since stress is what leads to burnout. This was done by using the **.wv.most_similar("stress")** method from the gensim module. This method fetches the word vectors from the model being tested and checks which ones are most similar to the word vector of the word we want - in this case **stress**.

The results for the word **stress** are present in table 7.1.

After analysing the results, we can conclude that Google News model has words more related to stress than our model. This is due to the fact that the Google News model contains more data examples than our model. However, the model created by us also returned some promising results, like the words *pressure*, *frustration*, *emotional*, *stressing*, *fatigue*, *drama* and *tension*, since all these words are related to experiencing stress. The results of our model were also promising due to the fact that the data we fed to our model is related to the topic of stress and burnout and a lot of our examples mentioned stress.

Word2vec "Stress" Results	
Google News Model	[(**'Stress'**, 0.6845571398735046), (**'anxiety'**, 0.6142929792404175), (**'stresses'**, 0.6005389094352722), (**'stressors'**, 0.5955410003662109), (**'stressful'**, 0.5830803513526917), (**'stressful_situations'**, 0.5716186165809631), (**'stressor'**, 0.533774197101593), (**'stress_hormones'**, 0.5286791920661926), (**'inborn_anti_Americanism'**, 0.523517370223999), (**'relieve_stress'**, 0.5232334136962891)]
Our Model	[(**'frustration'**, 0.5721256732940674), (**'fatigue'**, 0.551192581653595), (**'drama'**, 0.5494061708450317), (**'pressure'**, 0.5302340388298035), (**'extreme'**, 0.5282014012336731), (**'gained'**, 0.5197038054466248), (**'causing'**, 0.5191066861152649), (**'emotional'**, 0.514659583568573), (**'tension'**, 0.5055704712867737), (**'stressing'**, 0.503849983215332)]

Table 7.1: Word2vec models: similarity with word "Stress"

It was also decided to test the word "burnout" for each model since it is the main topic of this thesis, and the results are in table 7.2.

Analysing table 7.2, we can conclude that for the word "burnout", the Google News model presents much better results. This is due to the fact that it is a much bigger model comparing to ours and additionally, even though ours focuses on the topic of stress and burnout, there are not many examples of sentences containing the word burnout. There are even some words that do not make sense in the main topic of our chatbot (burnout in college students), like *homelessness* and *survivors*. This happens because the dataset containing the data from the initial forms is substantially smaller than the other two datasets and these two datasets that were downloaded contain a lot of examples mentioning other problems that can lead to burnout. Examples of problems mentioned in the datasets are homelessness, being an ex-military or victim of a war and being a victim of a traumatic situation like abuse.

7.1.2 Similarity

It was also decided to test similarity between important words. The words chosen take into consideration the topic of this thesis, which is to talk about burnout in college students. Therefore, these words are all related to stress and burnout and to reasons that lead students to feel distressed, like college. The words chosen were: **stress, burnout, pressure, college, deadlines, coronavirus, disease, covid, exams** and **projects**.

Word2vec "Burnout" Results	
Google News Model	[(**'burnouts'**, 0.5292092561721802), (**'fatigue'**, 0.4874576926231384), (**'overtraining'**, 0.4661228656768799), (**'overwork'**, 0.46278584003448486), (**'boredom'**, 0.4624118208885193), (**'Overtraining'**, 0.45657920837402344), (**'demotivation'**, 0.45091530680656433), (**'exhaustion'**, 0.4361668825149536), (**'overworking'**, 0.43293362855911255), (**'tiredness'**, 0.41212061047554016)]
Our Model	[(**'illnesses'**, 0.6788585186004639), (**'levels'**, 0.6719637513160706), (**'disorders'**, 0.6684169769287109), (**'psychological'**, 0.6596361398696899), (**'illness'**, 0.6523089408874512), (**'emotional'**, 0.6452532410621643), (**'homelessness'**, 0.6358888149261475), (**'survivors'**, 0.6337337493896484), (**'chronic'**, 0.6286095380783081), (**'circumstances'**, 0.6250165104866028)]

Table 7.2: Word2vec models: similarity with word "Burnout"

Comparison between words in word2vec models		
(Word1, Word2)	**Google News Model**	**Our Word2vec Model**
('stress', 'burnout')	0.38552284	0.43331093
('stress', 'pressure')	0.5010789	0.5747037
('college', 'deadlines')	0.08692381	0.4412732
('coronavirus', 'disease')	0.43146974	0.3678412
('covid', 'disease')	Key 'covid' not present	0.33725443
('exams', 'college')	0.3348187	0.54530555
('projects', 'college')	0.09290687	0.4044958

Table 7.3: Comparison between words in word2vec models

To test the similarity between words we used the method **.similarity('word1', 'word2')** from the gensim module. Table 7.3 contains the words used to test similarity in both models, as well as the results of this test for each model.

If we compare the results in table 7.3, we can see that the majority of the results were better in our model. The reason for this is the fact that our model only contains data that is focused on the same topic, the topic of burnout and stress. Therefore, the word vectors for these words are closer in our model than in the Google News model, which contains many examples with irrelevant data. Additionally, we can see that the similarities between words related to college are higher in our model - this happens because the data we feed our model contains examples that are answers given by college students when talking about stress and burnout whereas in the Google News model the data comes from a lot of different news related to various topics. Lastly, we can also conclude that the topic of covid-19 was not present in the data of the Google News model since the word **covid** does not exist in the vocabulary of that model (we can also confirm this since the model was built in 2013, time where the *coronavirus* were already known as viruses but the covid-19 pandemic had not happened yet).

7.1.3 Does Not Match

The third test performed was checking, in a list of words, which word did not match. The words chosen for this test are yet again words that we considered important in the context of the project being developed. Therefore, all of the words are either related to the topic of stress and burnout or to reasons that lead college students to experience burnout, like college and work.

To check which word did not match in a li of words, we used the method **doesnt_match ([list of words])** also from the gensim module. The results are presented in table 7.4.

Words That Do Not Match			
List Of Words	**Wrong Word**	**Google Model**	**Our Model**
projects, college, coronavirus	coronavirus	correct	correct
burnout, stress, happy	happy	correct	correct
college, work, relax	relax	wrong (college)	correct
disease, coronavirus, school	school	correct	correct
nervous, stress, relax	relax	wrong (nervous)	correct
depression, burnout, joy	joy	correct	wrong (burnout)
relaxed, calm, stressed	stressed	correct	correct
sad, depressed, happy	happy	wrong (depressed)	correct
projects, work, friends	friends	correct	correct
burnout, stress, content	content	correct	wrong (stress)

Table 7.4: Wor2vec models: Words That Do Not Match

Analysing the results in table 7.4, it can be concluded that both models predict the wrong word correctly most of the times (in this set of examples). However, both fail as well, in different scenarios. The first time our model fails is when predicting the wrong word in *depression, burnout, joy* - instead of predicting joy, the model predicts burnout. This can happen because both depression and joy are sentiments and therefore the model might have predicted its vectors closer than the burnout and depression vectors.

The second situation where the model fails is the last example on the table and it might happen due to the fact that not many examples in the training data fed to the model contained the word content, therefore the vector for this word might have been badly calculated.

After analysing the results from all tests, we can conclude that our word2vec model is a good option for our project, since the results are very satisfying in each test, even if the Google News model performs better in particular situations. However, there are a few good and bad aspects worth mentioning. The bad aspects are the following:

- There are words that will not get any results since they're not present in our dataset;
- Sometimes, the results are not the ones expected and in a topic as sensitive as the one we have in hands this can be a crucial matter since one mistake can lead to a bad answer.

These two problems mentioned could be addressed and fixed if our data had more examples, specially examples related to stress and burnout, because the word vectors would be calculated with more precision.

The advantages of using our model are:

- We have data that is focused on the topic we have in hands, which is helpful because we are not calculating unnecessary word vectors;
- Our predictions are correct most of the time;
- In terms of computation cost it is less expensive than the Google News model, since it takes less time making predictions due to its size.

7.2 RASA with Spacy Pipeline

This section analyses the results of using the NLU pipeline with Spacy components explained in 6.2.1.

To test our nlu pipeline, we used the command **rasa shell nlu** from RASA. This command allows the developer to see and study the results that the NLU pipeline returns when doing the natural language processing of the message sent by the user.

Since one of the intents that we have in our chatbot is the **greet** intent, we decided to start with the simple test of sending the sentence **hi there** to be processed by the NLU pipeline. The results are present in the following listing:

Listing 7.1: Results of the message "Hi there" from the NLU pipeline with spacy components

```
Next message:
hi there
{
  "text": "hi there",
  "intent": {
    "name": "greet",
    "confidence": 0.9960948824882507
  },
  "entities": [],
  "text_tokens": [
    [
      0,
      2
    ],
    [
      3,
      8
    ]
  ],
  "intent_ranking": [
    {
      "name": "greet",
      "confidence": 0.9960948824882507
    },
    {
      "name": "goodbye",
      "confidence": 0.0016894647851586342
    },
    {
      "name": "mood_great",
      "confidence": 0.0009747259318828583
    },
    {
      "name": "mood_pressured",
      "confidence": 0.0006875027320347726
    },
    {
      "name": "mood_college",
      "confidence": 0.00019966621766798198
    },
    {
      "name": "user_confirm",
      "confidence": 8.877684740582481e-05
    },
    {
      "name": "mood_sad",
      "confidence": 8.512935164617375e-05
    },
    {
      "name": "mood_p_problems",
      "confidence": 8.462707774015144e-05
    },
    {
      "name": "mood_covid",
      "confidence": 3.090366590186022e-05
    },
    {
      "name": "say_name",
      "confidence": 2.9049861041130498e-05
    }
  ],
}
```

By analysing the results present in listing 7.1, we can see that the pipeline was able to predict the correct intent for the sentence "hi there", which is **greet**. It is also shown the confidence level of this prediction, which was of ≈ 0.996. Not only is this a great result, but it left no doubts as to what intent was chosen since the second best guess has a confidence level of ≈ 0.002.

Following the test to the sentence "Hi there", we tested more complex sentences. Table 7.1 illustrates the sentences we analysed, the intent chosen, the confidence level of the intent chosen and also the second best intent in the ranking with its confidence level result. We also evaluated if the intent was correct or incorrect. It is important to highlight that all the examples we used to test the pipeline were not present in the training data. However, the examples chosen were sentences that we believe are important for the topic of our project and could be sentences said by users when using the system.

By evaluating the results in table 7.1, we can see that the majority of the intents were predicted correctly and with high values. There are, however, some examples of wrong predictions. This can be due to the fact that some intents don't have many examples in the training data. For instance, for the intent **bot_challenge** we only have four examples in the training data to illustrate the intent.

7.3 RASA with Word2vec Pipeline

This section analyses the results of using the NLU pipeline with word2vec component explained in section 6.2.2.

Spacy Pipeline Classification			
Sentences	**Classified Intent**	**Second best option**	**Classification**
My parents are always pressuring me to be better at school and make me feel guilty for taking some time to myself	explain_pressure_reason (0.9997)	explain_college_reason (0.0002)	correct
school demands a lot of work	explain_college_reason (0.8761)	mood_great (0.0284)	correct
you are a machine right?	goodbye (0.5210)	bot_challenge (0.2675)	incorrect
everyone around me	reason_for_pressure (0.9359)	bot_challenge (0.0277)	correct
happy	mood_great (0.3070)	user_confirm (0.1762)	correct
so horrible	mood_sad (0.3889)	goodbye (0.1243)	correct
School takes most of my time and leaves me with no time to relax and enjoy myself	explain_college_reason (0.9930)	explain_pressure_reason (0.0059)	correct
all the time	explain_pressure_reason (0.9153)	user_confirm (0.0217)	incorrect
im always thinking my parents might die if they catch it and i'm afraid for them	explain_covid_reason (0.9953)	explain_pressure_reason (0.0021)	correct
absolutely not	user_oppose (0.5866)	explain_covid_reason (0.2228)	correct
I keep thinking about catching and then dying	explain_pressure_reason (0.5060)	explain_college_reason (0.2505)	incorrect

Figure 7.1: Spacy Pipeline: Classification of intents

The command **rasa shell nlu** mentioned before in section 7.2 was also used to test this nlu pipeline.

The sentences chosen to be analysed in this section were also sentences not present in the training data but were related to this thesis' topic and possible sentences that could be said by a user when talking with our chatbot, like the sentences chosen in section 7.2. Table 7.2 illustrates these sentences that were analysed and also contains information about the intent predicted and the second best intent, as well as if the predictions were well made.

By analysing table 7.2, it is possible to conclude that four out of eleven times the system made a wrong prediction. This might be due to the fact that our training data is related to the topics of stress and burnout so when there are examples provided that do not contain a lot of words or sentences regarding those topics, the system might get confused and predict the wrong intent since it does not have many examples to illustrate those intents.

Word2vec Pipeline Classification			
Sentences	**Classified Intent**	**Second best option**	**Classification**
My parents are always pressuring me to be better at school and make me feel guilty for taking some time to myself	explain_pressure_reason (0.9961)	bot_challenge (0.0013)	correct
school demands a lot of work	explain_college_reason (0.8793)	bot_challenge (0.0292)	correct
you are a machine right?	bot_challenge (0.3412)	goodbye (0.2870)	correct
everyone around me	goodbye (0.6208)	reason_for_pressure (0.2742)	incorrect
happy	mood_great (0.1703)	user_confirm (0.1702)	correct
so horrible	greet (0.2389)	explain_covid_reason (0.1695)	incorrect
School takes most of my time and leaves me with no time to relax and enjoy myself	explain_college_reason (0.9062)	explain_pressure_reason (0.0575)	correct
all the time	explain_pressure_reason (0.8782)	explain_college_reason (0.0211)	incorrect
im always thinking my parents might die if they catch it and i'm afraid for them	explain_covid_reason (0.9963)	mood_pressured (0.0005)	correct
absolutely not	user_oppose (0.8810)	user_confirm (0.0315)	correct
I keep thinking about catching and then dying	explain_college_reason (0.4599)	explain_covid_reason (0.4172)	incorrect

Figure 7.2: Word2vec Pipeline: Classification of intents

7.4 Spacy vs Word2vec Pipelines

In this section, it is present a comparison between the two pipelines used - spacy pipeline, described in section 6.2.1, and word2vec pipeline, described in section 6.2.2.

Table 7.3 contains 20 examples of sentences not present in our training data. These examples were chosen based on the topic that our project focuses on, which is burnout and stress. Additionally, the examples correspond to sentences that a user might say when using our chatbot.

Both the **Word2vec** pipeline and the **spacy** pipeline analyse each sentence and give a prediction as to what is the intent that the sentences correspond to. There is also one column of table 7.3 that indicates which was the correct intent that the models should have predicted for each sentence.

The results present in table 7.3 show that our **word2vec** model behaved better in this test with this particular set of examples. This model only got three predictions wrong

Spacy vs Word2vec pipelines			
Sentences	**Word2vec**	**Spacy**	**Correct intent**
i feel like everyone puts pressure on me	pressure_reason (0.9713)	pressure_reason (0.9535)	pressure_reason
i hate having to go to classes	explain_college_ rea son (0.9434)	explain_pressure_rea son (0.8301)	explain_college _reason
i am unhappy	explain_college_rea son (0.2137)	mood_sad (0.6572)	mood_sad
i believe the motive is pressure	explain_pressure_rea son (0.2420)	mood_pressured (0.4178)	explain_pressure_ rea son
i am excited	mood_great (0.1795)	mood_sad (0.5176)	mood_great
too many deadlines make me not want to study anymore	explain_college_rea son (0.9905)	explain_college_rea son (0.9983)	explain_college_ reason
are you a machine?	bot_challenge (0.4122)	bot_challenge (0.9420)	bot_challenge
is it a person talking?	bot_challenge (0.5498)	bot_challenge (0.7099)	bot_challenge
i work and study so the pressure to succeed in both is huge	explain_pressure_rea son (0.9879)	explain_pressure_rea son (0.9995)	explain_pressure_ reason
i don't want my parents to catch covid	explain_covid_rea son (0.8884)	explain_covid_rea son (0.8648)	explain_covid_ reason
it is a disease that can kill you and I'm scared	explain_covid_rea son (0.9648)	explain_covid_rea son (0.8839)	explain_covid_ reason
i am always thinking i have to do great to have a good future	explain_pressure_rea son (0.9552)	explain_pressure_rea son (0.6260)	explain_pressure_ reason
i am experiencing burnout	mood_sad (0.9880)	mood_sad (0.8670)	mood_sad
burnout is killing me	mood_sad (0.9729)	mood_sad (0.5827)	mood_sad
my peers	explain_college_rea son (0.4813)	reason_for_pre ssure (0.9077)	reason_for_ pressure
my granny can die from it	explain_covid_rea son (0.3298)	explain_college_rea son (0.1696)	explain_covid_ reason
being so tired makes me want to not attend classes	explain_college_rea son (0.8982)	explain_college_rea son (0.6379)	explain_college_ reason
i feel like you aren't a person	mood_great (0.1610)	bot_challenge (0.6592)	bot_challenge
feelin good like i should	mood_great (0.7670)	mood_great (0.6380)	mood_great
covid is a scary disease	explain_covid_rea son (0.9309)	explain_covid_rea son (0.8473)	explain_covid_ reason

Figure 7.3: Spacy vs Word2vec: Classification of intents

(rows number 3, 15 and 18). There are two conclusions that we were able to make from these examples of sentences:

The first conclusion is that the pipeline that uses word2vec does not behave well when being presented with examples that are not related to the topic that the system is focusing on. In this case, the pipeline did not behave well when evaluating the sentences *"my peers"* and *"I feel like you aren't a person"*, which are vague sentences that do not contain any words related to stress or burnout. These results were expected since our training data was small and the examples present in the training data were very focused on the topics of stress and burnout. Therefore, we expected the model to work well when being presented with sentences related to the thesis' topic and not that well when presented with random examples.

The second conclusion we can take from this experience is that the word2vec model sometimes does not perform well on small sentences, containing few vague words. For example, when predicting the sentence *"I am unhappy"*, not only did the pipeline predicted it wrong but also gave a really small level of confidence, which demonstrates that the model was not that sure of what it was doing.

7.5 Preparation of the validation

This section explains how the experience of testing our chatbot was presented to the users.

First, in figure 7.4 it is present the design of the page that is shown to the user. This design is well explained in section 6.4. There is a header with the title of the page and then, in the body of the page there is a text explaining what is the aim of this project. The text is the following:

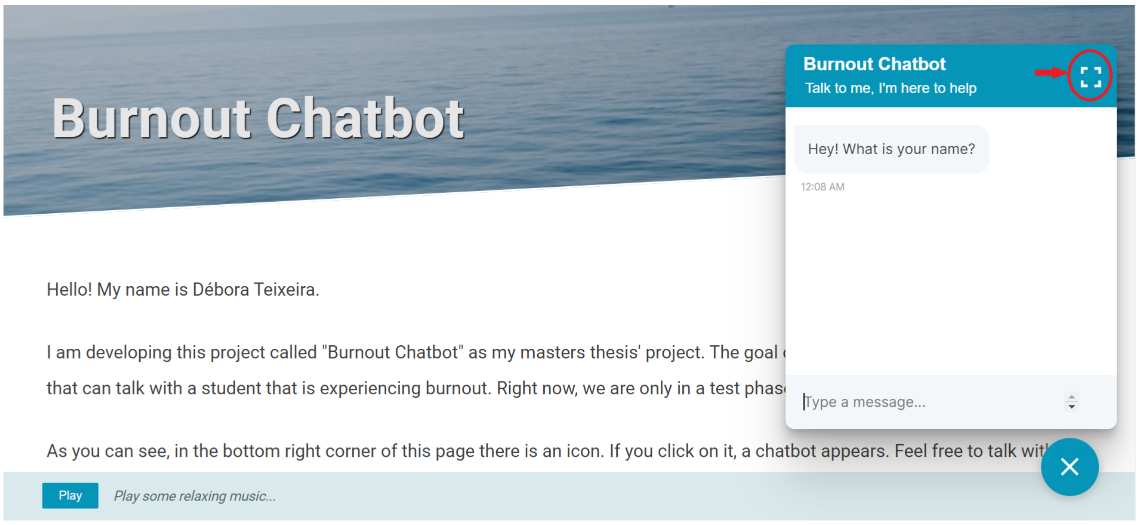
I am developing this project called "Burnout Chatbot" as my masters thesis' project. The goal of this project is to develop a chatbot that can talk with a student that is experiencing burnout. Right now, we are only in a test phase.

As you can see, in the bottom right corner of this page there is an icon. If you click on it, a chatbot appears. Feel free to talk with this chatbot and please provide me some feedback at the end of your session.

Thank you for participating!

After the text explaining the aim of this project, there is a footer which contains a button that says **Play** - if the user presses this button, calm music will start with the aim of calming the user during the conversation.

Additionally, in the footer of the page there is a button with a chat widget. That is the crucial part of this page, since it loads the chatbot when the user clicks on the widget. The chatbot window is small and is located at the bottom right corner of the page, as it can be seen in figure 7.4. However, the chatbot can occupy the whole page if the user clicks on the expand widget present in the chatbot (circled in red in figure 7.4). The chatbot in full window size is present in figure 7.5.



Hello! My name is Débora Teixeira.

I am developing this project called "Burnout Chatbot" as my masters thesis' project. The goal

that can talk with a student that is experiencing burnout. Right now, we are only in a test phas

As you can see, in the bottom right corner of this page there is an icon. If you click on it, a chatbot appears. Feel free to talk wit

Figure 7.4: Interface - part 1

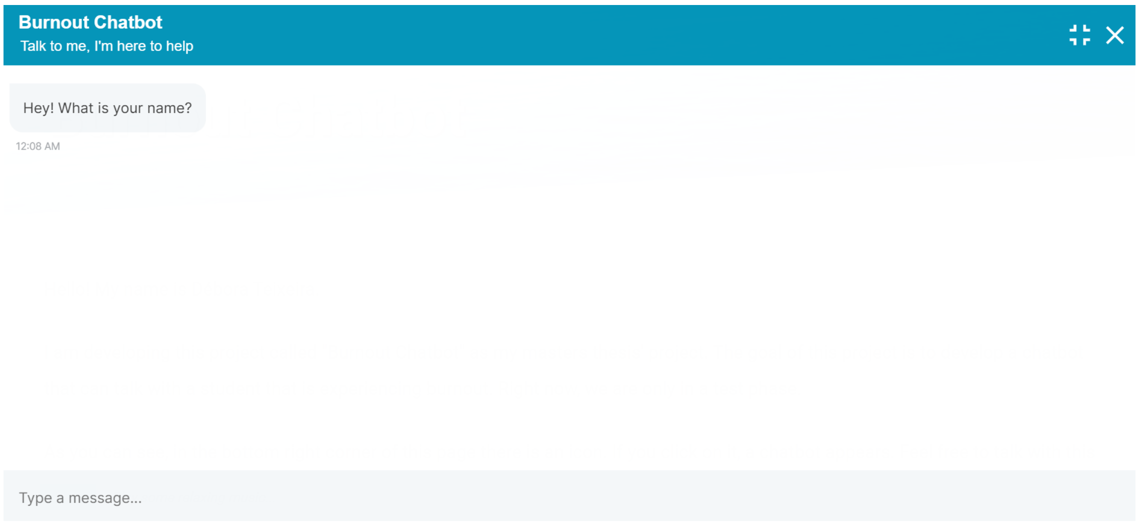


Figure 7.5: Interface - part 2

In order for the chatbot to run, we have to run two different commands, both at the same time and in the same directory as the project. The first one - **rasa run -m models –enable-api –cors "*"** - enables the web server api, starts the input channel and allows requests from the page of the directory. The second one - **rasa run actions** - "activates" the actions present in RASA - which means that throughout the whole conversation, RASA will check when the actions we declared in our python file are needed and run them. If this last command is not running, the conversation will not continue when an action is needed since that action will not load, and if the first command is not running, the chatbot won't load at all.

Due to the fact that our project only works in localhost, in order to test our project the users had to use the computer where the project is and test it there, one by one. This is not the ideal situation for testing, especially due to the fact that the user might not feel 100% comfortable sharing its problems in a computer that does not belong to him/her. Due to the restrictions in the environment for the user experience tests, such tests should be contextualized as focused in the behaviour of the chatbot rather than the conversation itself.

7.6 User Experience

The last results that were analysed in this thesis were the opinions of the users that tested our chatbot. It is important to remind that the chatbot is still in an early stage of development and the product evaluated by the users still lacks a lot of information regarding the psychological field. The way users were presented with the experience is explained in section 7.5.

To test our chatbot, we presented this first stage of development to **37** university students. These students were contacts that were close to the developer, which can lead to misleading conclusions due to the following reasons:

- the participants may not fully understand the topic of burnout since it was not required to have experienced burnout to test the chatbot;
- the participants might feel pressured to give good answers since the developer and the participants were acquainted;
- the participants tested this chatbot in an environment chosen by the developer, since the chatbot is in a local server. Therefore, they might not feel the most comfortable sharing information and therefore provide misleading information.

Even though the results might be misleading, they were still analysed.

The first question analysed was if the user was comfortable sharing information with the chatbot and the results are present in figure 7.6. In **37** answers, **34** people answered **Yes**, which means that 92% of the users felt comfortable while sharing information with the chatbot. Regarding the 3 users that answered **No**, 2 said that there was nothing that could be done in the implementation of the system that could help them feel more comfortable sharing their personal information, meaning that there is a group of people who might not feel comfortable with this new approach of trying to develop a digital complement to psychological treatment. There was, however, a user that mentioned that he/she would feel more comfortable sharing information with the chatbot if there is a more explicit way of explaining what is done in the chatbot that guarantees security of the user's data.

It was also analysed if the users considered the flow of the conversation as something that made sense or not, and this graphic is present in figure 7.7. Out of **35** participants, **30** said that the flow of the conversation made sense to them. However, **5** participants

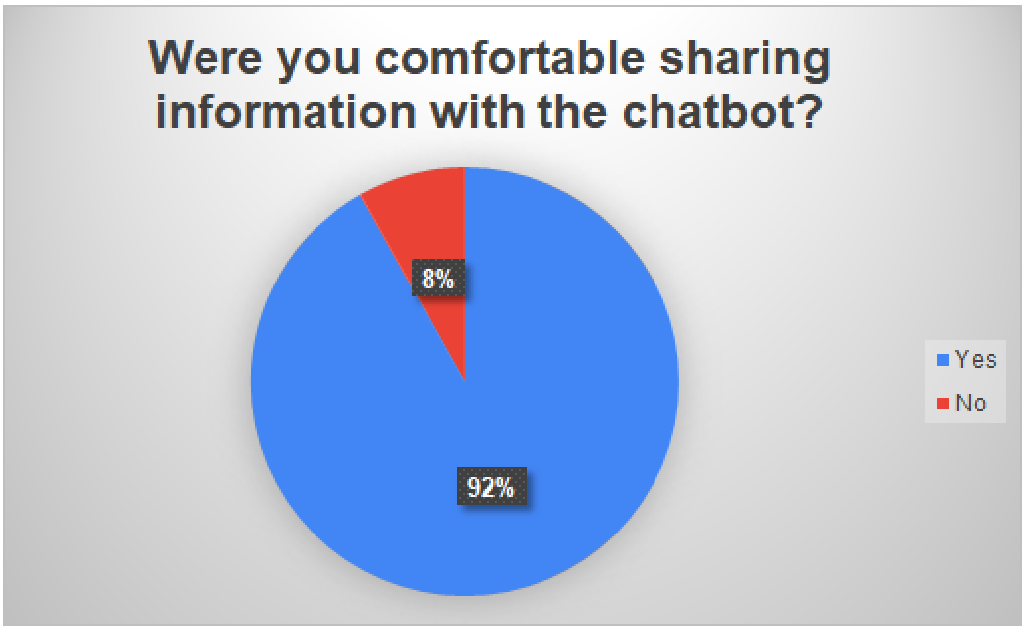


Figure 7.6: Results of question "*Were you comfortable sharing information with the chatbot?*"

said that the flow of the conversation did in fact make them confused. This might be due to the fact that the chatbot might have made a wrong prediction of the intent of the user and therefore gave an incorrect answer to the user, making him/her confused as to why the system was returning that particular response.

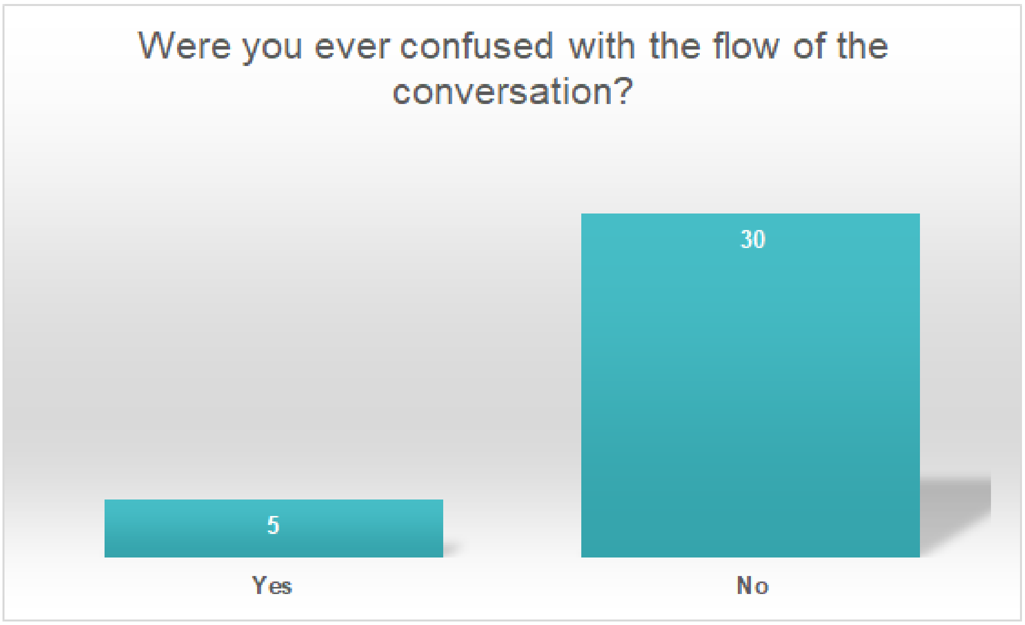


Figure 7.7: Results of question "*Were you ever confused with the flow of the conversation?*"

The third question analysed was if the user believed that our project could actually help students in a distressed mood if it contained more information about how to deal with burnout. This information is present in figure 7.8. Out of **35** answers, **34** participants said that they would find this chatbot a helpful tool if it contained more information related to the psychological field. Only one person considered the chatbot not helpful, for reasons unknown.

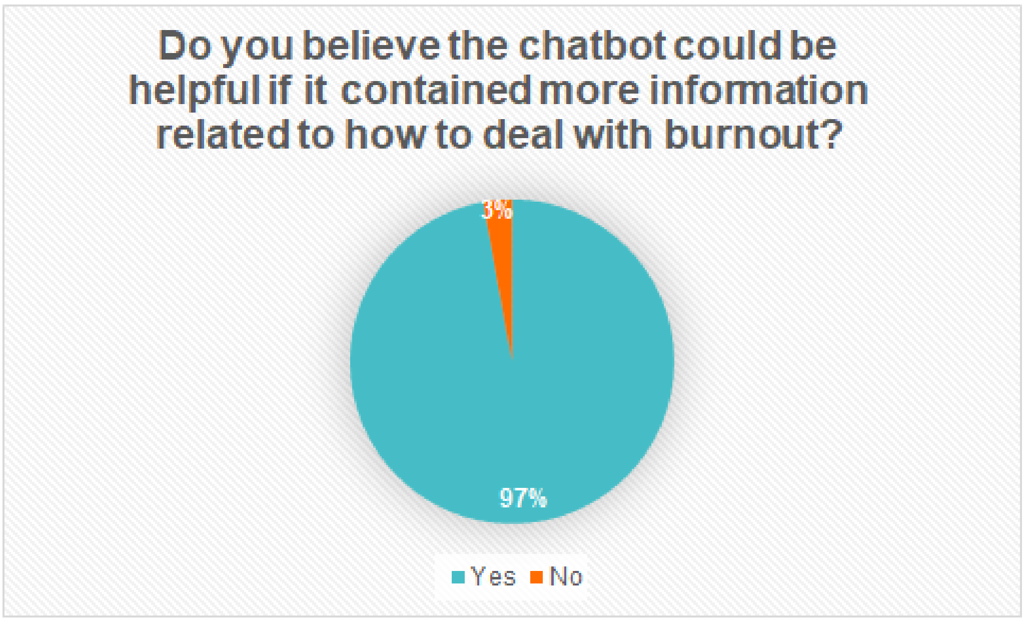


Figure 7.8: Results of question "*Do you believe the chatbot could be helpful if it contained more information related to how to deal with burnout?*"

Finally, the last question analysed was whether the users felt understood by the chatbot or not. The results for this question are present in figure 7.9. **30** out of **35** users felt understood, **4** felt understood at times and **1** was not sure if it felt understood. The main reason that the users who did not feel understood all the time gave for not feeling understood was that the answers given by the chatbot were too generic and not that specific to the problem that the user was describing.

Finally, the users also provided advice that can be divided into three topics:

- **Security** - Users believe that the security rules established for this project should be mentioned in a way that could provide reassurance to the users;

  Information - Users defend that the chatbot should contain more information regarding the psychological field so that it could give them more advice;

  Infer feelings - Users claim that the bot should try to predict their feelings more instead of asking directly if the user is feeling burned out, since it might be something that the user is not yet sure about.

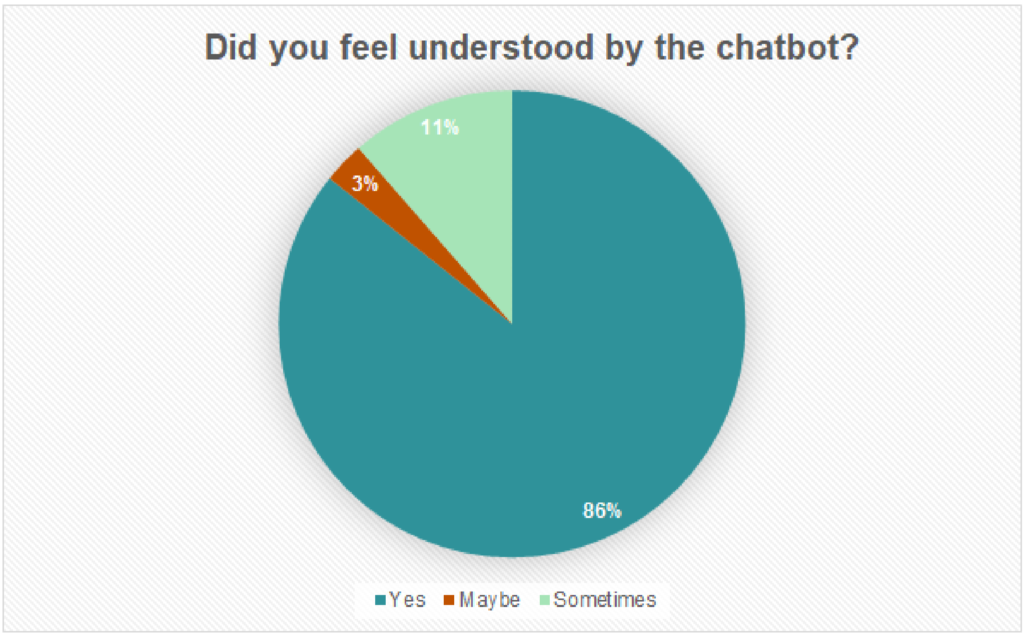


Figure 7.9: Results of question "*Did you feel understood by the chatbot?*"

8 Conclusions

This research aimed to develop the early stage of a chatbot that deals with burnout in college students. Based on the results of a form developed to ask college students about burnout, it was possible to identify four main reasons for burnout in college students - covid-19, college, pressure and personal problems -, and with this information it was possible to develop four scripts for the chatbot's conversations.

Additionally, a word2vec model developed with training data related to stress and burnout was also tested and compared with another word2vec model, the Google News model present in [61]. This can be seen in section 7.1. After testing both models, we were able to see that for our particular project it was better to use our model since it contained more data related to the topic of burnout and spent less time predicting the correct answer to respond to the user. It is important to highlight that these observations were made only with the examples we provided and are related only to our project.

Another important test done in section 7.4 was the comparison between our word2vec model in Rasa's nlu pipeline versus the use of spacy components in the same pipeline. With this comparison we observed that in our project in situations where word2vec was well trained due to a great amount of examples provided in the training data, word2vec performed better than the spacy components. Therefore, following the examples that we evaluated it is feasible to think that in chatbots that need great precision in the answers they return to the user (which is the case of this project), word2vec with a great amount of training data related to the chatbot's topic is a good model to use.

It is important to highlight that in general situations where a great precision is not necessary, both spacy 7.2 and word2vec 7.3 work well.

Finally, a final project with a graphic interface and the chosen word2vec model was developed, using RASA, python and Javascript. This project can be seen in: Thesis' Folder [1]. This folder is really important since it includes all the important aspects and features of the development of this thesis.

Results from the users experience shown in section 7.6 demonstrate that users would be willing to use this chatbot as a way to help deal with burnout.

[1] https://drive.google.com/drive/folders/1t26Lohbs2BT1gs2MeOHaCTS-q8ZddqmD?usp=sharing

Finally, we found evidence that technology can be a great tool to help with psychological problems, namely burnout, although this evidence should be better evaluated with further work. There are several techniques that can be used to develop tools such as the chatbot studied on this project, and this thesis shows that it is feasible to integrate rule-based and neural-based techniques and that RASA allows for these techniques to work well alongside each other.

8.1 Future Work

Addressing mental health issues is a topic that is highly sensitive and therefore it needs to be approached with great care.

By analysing the results from the natural language understanding (NLU) pipeline present in section 7.4, we can conclude that is is very important to have a huge amount of training data for this chatbot due to the fact that sometimes, when there isn't enough data to illustrate the user intents, the system does a wrong prediction of the intent of the user and therefore returns a wrong answer to the user. To a person that is in a distressful state, it is very important to be careful with the type of response that the person receives since it can lead to an even more distressful state. Therefore, one step that could be done in future work for this project would be to obtain more data related to the psychological field to feed the chatbot for it to learn more information.

Additionally, BERT was studied but not used in this phase of development. In future work, BERT or similar related models may be included in this project and tested to see if the results were better than applying word2vec. The developer should have in mind that the results might be better but the response time of the chatbot might increase significantly, which can be a problem in a system that is conversing with a human.

Another feature that was studied but not used in this phase of development due to lack of time was sense2vec, a feature that can be added in spacy pipelines. This feature can be added and studied in future phases of development of this project.

It is also important to mention that for further development of this project, there should be a team of experienced psychologists to help develop the answers that the bot should give, due to the fact that they have more knowledge of what to be said in certain situations. Additionally, this project could be moved to an online website in order to test with more participants.

Bibliography

[1] B. AbuShawar and E. Atwell. “ALICE chatbot: Trials and outputs”. In: *Computación y Sistemas* 19.4 (2015), pp. 625–632 (cit. on p. 7).

[2] J. B. Ahire. *Introduction to Word Vectors*. Feb. 2018. URL: https://dzone.com/articles/introduction-to-word-vectors (cit. on p. 5).

[3] *AI in Software Testing: Rule-Based Testing vs. Learning Systems*. July 2020. URL: https://www.tricentis.com/artificial-intelligence-software-testing/ai-approaches-rule-based-testing-vs-learning/ (cit. on pp. 4, 5).

[4] I. A. Alshawwa et al. “An Expert System for Depression Diagnosis”. In: (2019) (cit. on pp. 7, 8, 10).

[5] R. W. Andrew Chin et al. “Investigating validity evidence of the Malay translation of the Copenhagen Burnout Inventory”. In: *Journal of Taibah University Medical Sciences* 13.1 (2018), pp. 1–9. ISSN: 1658-3612. DOI: https://doi.org/10.1016/j.jtumed.2017.06.003. URL: https://www.sciencedirect.com/science/article/pii/S1658361217301002 (cit. on p. 11).

[6] P. Ataee. *Word2Vec Models are Simple Yet Revolutionary | Towards Data Science*. Dec. 2020. URL: https://towardsdatascience.com/word2vec-models-are-simple-yet-revolutionary-de1fef544b87 (cit. on p. 12).

[7] A. Banafa. *What is Affective Computing?* June 2016. URL: https://www.bbvaopenmind.com/en/technology/digital-world/what-is-affective-computing/#:%5C%7E:text=Affective%5C%20computing%5C%20is%5C%20the%5C%20study,%5C%2C%5C%20psychology%5C%2C%5C%20and%5C%20cognitive%5C%20science.&text=A%5C%20motivation%5C%20for%5C%20the%5C%20research%5C%20is%5C%20the%5C%20ability%5C%20to%5C%20simulate%5C%20empathy. (cit. on p. 10).

[8] O. Ben-Kiki, C. Evans, and B. Ingerson. “Yaml ain’t markup language (yaml™) version 1.1”. In: *Working Draft 2008* 5 (2009), p. 11 (cit. on p. 12).

[9] T. Bocklisch et al. “Rasa: Open source language understanding and dialogue management”. In: *arXiv preprint arXiv:1712.05181* (2017) (cit. on pp. 19–21).

[10] *Chatbot market in 2022: Stats, trends, and companies in the growing AI chatbot industry*. Feb. 2022. URL: https://www.businessinsider.com/chatbot-market-stats-trends?IR=T#:%5C%7E:text=Nearly%5C%2040%5C%25%5C%20of%5C%20internet%5C%20users,increase%5C%20in%5C%20popularity%5C%20moving%5C%20forward. (cit. on p. 6).

[11] *Component Diagram Tutorial*. URL: https://www.lucidchart.com/pages/uml-component-diagram (cit. on p. 36).

[12] *Conversational Agent*. May 2019. URL: https://deepai.org/machine-learning-glossary-and-terms/conversational-agent (cit. on p. 4).

[13] A. Deshpande et al. "A survey of various chatbot implementation techniques". In: *International Journal of Computer Engineering and Applications* 11.7 (2017) (cit. on p. 8).

[14] J. Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on pp. 12, 14).

[15] P. Dugar. *Attention - Seq2Seq Models*. July 2019. URL: https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263 (cit. on p. 5).

[16] *Featurizers — deepchem 2.6.2.dev documentation*. URL: https://deepchem.readthedocs.io/en/latest/api_reference/featurizers.html?highlight=tokenizer (cit. on p. 6).

[17] *Gensim: topic modelling for humans*. May 2022. URL: https://radimrehurek.com/gensim/intro.html (cit. on p. 54).

[18] B. S. George and A. S. Gillis. *What is the Turing Test?* June 2021. URL: https://www.techtarget.com/searchenterpriseai/definition/Turing-test (cit. on p. 5).

[19] *Global Study: 82% of People Believe Robots Can Support Their Mental Health Better Than Humans*. URL: https://www.oracle.com/emea/news/announcement/artificial-intelligence-supports-mental-health-2020-10-07.html (cit. on p. 1).

[20] *Google Code Archive - Long-term storage for Google Code Project Hosting*. July 2013. URL: https://code.google.com/archive/p/word2vec/ (cit. on pp. 37, 54).

[21] Great Learning Team. *An Introduction to Bag of Words in NLP using Python | What is BoW?* May 2022. URL: https://www.mygreatlearning.com/blog/bag-of-words/#sh1 (cit. on p. 6).

[22] L. Gupta. *Differences Between Word2Vec and BERT - The Startup*. Nov. 2020. URL: https://medium.com/swlh/differences-between-word2vec-and-bert-c08a3326b5d1#:%5C%7E:text=Word2Vec%5C%20will%5C%20generate%5C%20the%5C%20same,vectors%5C%20like%5C%20beach%5C%2C%5C%20coast%5C%20etc. (cit. on pp. 24, 25).

[23] S. Gupta. *Sentiment Analysis: Concept, Analysis and Applications*. Jan. 2018. URL: https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17 (cit. on p. 5).

[24] M. C. Holcombe. *According to our Google searches, 2021 was about getting better*. Dec. 2021. URL: https://edition.cnn.com/2021/12/08/health/google-year-in-search-2021-wellness/index.html (cit. on p. 1).

[25] M. Honnibal. *Introducing spaCy ·*. Feb. 2015. URL: https://explosion.ai/blog/introducing-spacy/ (cit. on p. 15).

[26] T. Hu et al. "Touch your heart: A tone-aware chatbot for customer care on social media". In: *Proceedings of the 2018 CHI conference on human factors in computing systems*. 2018, pp. 1–12 (cit. on p. 9).

[27] S. Hussain, O. Ameri Sianaki, and N. Ababneh. "A survey on conversational agents/chatbots classification and design techniques". In: *Workshops of the International Conference on Advanced Information Networking and Applications*. Springer. 2019, pp. 946–956 (cit. on pp. 4, 5).

[28] D. Ireland, H. Hassanzadeh, and S. N. Tran. "Sentimental analysis for AIML-based e-health conversational agents". In: *International Conference on Neural Information Processing*. Springer. 2018, pp. 41–51 (cit. on p. 10).

[29] P. Kandru. *Guide to Sense2vec – Contextually Keyed Word Vectors for NLP*. Mar. 2021. URL: https://analyticsindiamag.com/guide-to-sense2vec-contextually-keyed-word-vectors-for-nlp/ (cit. on p. 19).

[30] J. Kelly. *Indeed Study Shows That Worker Burnout Is At Frighteningly High Levels: Here Is What You Need To Do Now*. Dec. 2021. URL: https://www.forbes.com/sites/jackkelly/2021/04/05/indeed-study-shows-that-worker-burnout-is-at-frighteningly-high-levels-here-is-what-you-need-to-do-now/ (cit. on p. 1).

[31] Lk. *A.L.I.C.E Alicebot Artificial Intelligence (AI) Chatbot*. Jan. 2020. URL: https://techjourney.net/alice-alicebot-artificial-intelligence-ai-chatbot/ (cit. on p. 7).

[32] A. Malhotra. *Introduction to Libraries of NLP in Python — NLTK vs. spaCy*. June 2018. URL: https://medium.com/@akankshamalhotra24/introduction-to-libraries-of-nlp-in-python-nltk-vs-spacy-42d7b2f128f2 (cit. on pp. 25, 26).

[33] M. L. Mauriello et al. "Sad: A stress annotated dataset for recognizing everyday stressors in sms-like conversational systems". In: *Extended abstracts of the 2021 CHI conference on human factors in computing systems*. 2021, pp. 1–7 (cit. on pp. 12, 54).

[34] T. Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013) (cit. on p. 13).

[35] H. Nguyen, D. Morales, and T. Chin. “A neural chatbot with personality”. In: *Semantic Scholar* (2017) (cit. on p. 9).

[36] L. Ni et al. “Mandy: Towards a smart primary care chatbot application”. In: *International symposium on knowledge and systems sciences*. Springer. 2017, pp. 38–52 (cit. on p. 9).

[37] M. . T. . Nietzel. *Almost Half Of Americans Don’t Seek Professional Help For Mental Disorders*. May 2021. URL: https://www.forbes.com/sites/michaeltnietzel/2021/05/24/why-so-many-americans-do-not-seek-professional-help-for-mental-disorders/?sh=531807a13de7 (cit. on p. 1).

[38] *Open source conversational AI*. Nov. 2021. URL: https://rasa.com (cit. on pp. 27, 57).

[39] A. Pai. *Pretrained Word Embeddings | Word Embedding NLP*. Mar. 2020. URL: https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/ (cit. on p. 5).

[40] E. W. Pamungkas. “Emotionally-aware chatbots: A survey”. In: *arXiv preprint arXiv:1906.09774* (2019) (cit. on pp. 6, 7, 10).

[41] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (visited on 09/11/2022) (cit. on p. 54).

[42] R. W. Picard. *Affective computing*. MIT press, 2000 (cit. on p. 10).

[43] A. . Prasad. *Conditional Random Fields Explained - Towards Data Science*. Dec. 2021. URL: https://towardsdatascience.com/conditional-random-fields-explained-e5b8256da776 (cit. on p. 6).

[44] B. Raj. *Understanding BERT: Is it a Game Changer in NLP? - Towards Data Science*. Oct. 2019. URL: https://towardsdatascience.com/understanding-bert-is-it-a-game-changer-in-nlp-7cca943cf3ad#:%5C%7E:text=1%5C%2C%5C%20BERT%5C%20achieves%5C%2093.2%5C%25%5C%20F1,Language%5C%20Understanding%5C%20(NLU)%5C%20tasks. (cit. on p. 14).

[45] H. Ritchie and M. Roser. “Mental health. Our world in data”. In: *Retrieved May* 19 (2018), p. 2020 (cit. on p. 1).

[46] Seng. *Chatbot Intents – Simple Explanation Fundamentals*. Apr. 2020. URL: https://chatbotbusinessframework.com/chatbot-intents/#:%5C%7E:text=An%5C%20intent%5C%20is%5C%20the%5C%20goal,level%5C%20description%5C%20of%5C%20the%5C%20intent. (cit. on p. 6).

[47] I. V. Serban et al. “A deep reinforcement learning chatbot”. In: *arXiv preprint arXiv:1709.02349* (2017) (cit. on p. 9).

[48] H. Shah et al. “Can machines talk? Comparison of Eliza with modern dialogue systems”. In: *Computers in Human Behavior* 58 (2016), pp. 278–295 (cit. on p. 7).

[49] *spaCy · Industrial-strength Natural Language Processing in Python*. 2016. URL: https://spacy.io/ (cit. on pp. 15, 16, 18).

[50] *Transformer (machine learning model)*. URL: https://en.wikipedia.org/wiki/Transformer_(machine_learning_model) (cit. on p. 5).

[51] A. Trask, P. Michalak, and J. Liu. "sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings". In: *arXiv preprint arXiv:1511.06388* (2015) (cit. on pp. 12, 18, 19).

[52] M. Trotter. *Transfer Learning for Machine Learning*. June 2021. URL: https://www.seldon.io/transfer-learning/ (cit. on p. 6).

[53] E. Turcan and K. McKeown. *Dreaddit: A Reddit Dataset for Stress Analysis in Social Media*. 2019. DOI: 10.48550/ARXIV.1911.00133. URL: https://arxiv.org/abs/1911.00133 (cit. on pp. 11, 54).

[54] E. Turcan and K. McKeown. "Dreaddit: A Reddit Dataset for Stress Analysis in Social Media". In: *Proceedings of the Tenth International Workshop on Health Text Mining and Information Analysis (LOUHI 2019)*. Hong Kong: Association for Computational Linguistics, Nov. 2019, pp. 97–107. DOI: 10.18653/v1/D19-6213. URL: https://aclanthology.org/D19-6213 (cit. on p. 11).

[55] *UML Class Diagram Tutorial*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/ (cit. on p. 38).

[56] A. N. Vaidyam et al. "Chatbots and conversational agents in mental health: a review of the psychiatric landscape". In: *The Canadian Journal of Psychiatry* 64.7 (2019), pp. 456–464 (cit. on pp. 10, 11).

[57] V. Warmerdam. *Custom SpaCy 3.0 models in RASA*. June 2021. URL: https://rasa.com/blog/custom-spacy-3-0-models-in-rasa/ (cit. on p. 27).

[58] V. Warmerdam. *Intents Entities: Understanding the Rasa NLU Pipeline*. Mar. 2021. URL: https://rasa.com/blog/intents-entities-understanding-the-rasa-nlu-pipeline/ (cit. on p. 21).

[59] V. Warmerdam. *Zooming in on Dialogue Management in Rasa 2.0*. Dec. 2020. URL: https://rasa.com/blog/dialogue-policies-rasa-2/ (cit. on p. 21).

[60] A. Xu et al. "A new chatbot for customer service on social media". In: *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017, pp. 3506–3510 (cit. on pp. 9, 12).

[61] *Year in Search 2021 - Google*. 2021. URL: https://about.google/intl/ALL_uk/stories/year-in-search-2021/trends/mental-health/ (cit. on pp. 1, 2, 37, 74).

A

Form *Burnout - General*

This appendix contains the *Burnout - General* form presented to the participants of this study.

Burnout - General

Greetings, my name is Débora Teixeira and I'm a last year student of Computer Science in NOVA School of Science and Technology. For my masters thesis I'm studying a chatbot that assists people feeling burned out. The goal of this form is to understand how burnout is affecting people and if it is related to work. It is anonymous and the answers are only used to help on the development of my thesis. Thank you for your participation!

*Required

1. Have you been feeling stressed the last couple of days? *

Mark only one oval.

- Never
- Rarely
- Sometimes
- Frequently
- All the time

2. Is your stress related to your job? *

Mark only one oval.

- Never
- Rarely
- Sometimes
- Frequently
- All the time

3. Do you feel emotionally drained from your job? *

Mark only one oval.

- ◯ Never *Skip to question 7*
- ◯ Rarely *Skip to question 7*
- ◯ Sometimes
- ◯ Frequently
- ◯ All the time
- ◯ I do not think this question is relevant *Skip to question 5*
- ◯ I would ask this question in a different way *Skip to question 6*

4. Why do you feel emotionally drained from your work? *

Skip to question 7

5. What question would you ask instead of "Do you feel emotionally drained from your job?"

Skip to question 7

6. How would you change the question "Do you feel emotionally drained from your job?" ? *

7. Does you job make you frustrated? *

Mark only one oval.

- ◯ Never *Skip to question 9*
- ◯ Rarely *Skip to question 9*
- ◯ Sometimes
- ◯ Frequently
- ◯ All the time

8. Why does your job make you frustrated? *

9. Do you feel emotionally exhausted? *

Mark only one oval.

- Never *Skip to question 11*
- Rarely *Skip to question 11*
- Sometimes
- Frequently
- All the time

10. How would you describe your exhaustion? (e.g. tired and without energy to set goals and stick with them) *

11. Do you feel physically exhausted? *

Mark only one oval.

- Never *Skip to question 13*
- Rarely *Skip to question 13*
- Sometimes
- Frequently
- All the time

12. What do you feel as symptoms of physical exhaustion? *

13. Do you feel tired in the morning when you think about another day of work? *

Mark only one oval.

- ◯ Never *Skip to question 15*
- ◯ Rarely *Skip to question 15*
- ◯ Sometimes
- ◯ Frequently
- ◯ All the time

14. Describe your thoughts in the morning when you think about going to work. *

15. Do you consider every hour of work tiresome? *

Mark only one oval.

- ◯ Never
- ◯ Rarely
- ◯ Sometimes
- ◯ Frequently
- ◯ All the time

16. Is there any thought you have that helps you deal with being tired from work?

17. Do you ever think about quitting your job? *

Mark only one oval.

- ⬭ Never *Skip to question 19*
- ⬭ Rarely *Skip to question 19*
- ⬭ Sometimes
- ⬭ Frequently
- ⬭ All the time

18. How do you deal with the thoughts of quitting your job? *

19. Do you feel satisfied with your results in your work? *

Mark only one oval.

- ⬭ Never
- ⬭ Rarely
- ⬭ Sometimes *Skip to question 21*
- ⬭ Frequently *Skip to question 21*
- ⬭ All the time *Skip to question 21*

20. Why are you not satisfied with your results? *

21. Do you feel like you work too much when comparing to the benefits of your job? *

Mark only one oval.

- ◯ Never *Skip to question 23*
- ◯ Rarely *Skip to question 23*
- ◯ Sometimes
- ◯ Frequently
- ◯ All the time

22. In what way could the ratio between work and benefits improve? (e.g. work less hours or gain more benefits) *

23. Do you feel weak or susceptible to getting sick? *

Mark only one oval.

- ◯ Never *Skip to question 27*
- ◯ Rarely *Skip to question 27*
- ◯ Sometimes *Skip to question 26*
- ◯ Frequently *Skip to question 26*
- ◯ All the time *Skip to question 26*
- ◯ I do not consider this question relevant *Skip to question 25*
- ◯ I consider this question somewhat relevant, but should be written in a different way

24. How would you change the question "Do you ever feel weak or susceptible to getting sick?" *

25. What question would you ask instead of "Do you ever feel weak or susceptible to getting sick?"

26. Why do you think you feel weak or susceptible to getting sick? *

____
____
____
____
____

27. Do you have enough energy for your family and friends during your leisure time? *

Mark only one oval.

- ◯ Never
- ◯ Rarely
- ◯ Sometimes
- ◯ Frequently
- ◯ All the time

28. Do you think about your job when you're having your leisure time? *

Mark only one oval.

- ◯ Never *Skip to question 30*
- ◯ Rarely *Skip to question 30*
- ◯ Sometimes
- ◯ Frequently
- ◯ All the time

29. Describe the type of thoughts you have about your job when you're having your leisure time. *

30. Do you think the main reason for not quitting your job is your salary? *

Mark only one oval.

- Never
- Rarely
- Sometimes
- Frequently
- All the time

31. Do you take medication to deal with stress or anxiety from your job? *

Mark only one oval.

- Never
- Rarely
- Sometimes
- Frequently
- All the time

32. Do you consider your workplace a safe and calm environment? *

Mark only one oval.

- ( ) Never
- ( ) Rarely
- ( ) Sometimes *Skip to question 34*
- ( ) Frequently *Skip to question 34*
- ( ) All the time *Skip to question 34*

33. How would you change the environment you work in? *

34. Do you consider your colleagues stressful? *

Mark only one oval.

- ( ) Never
- ( ) Rarely
- ( ) Sometimes
- ( ) Frequently
- ( ) All the time

35. Do you believe your colleagues influence your stress? How? *