UNIVERSIDADE DO MINDELO | UNIVERSIDADE DA BEIRA INTERIOR

CURSO DE MESTRADO EM ENGENHARIA INFORMÁTICA

RELATÓRIO DE DISSERTAÇÃO DE MESTRADO
ANO LETIVO 2020/2021 – 2º ANO

Tema: Desenvolvimento de um Chatbot para Universidade do Mindelo (UMBOT)

Autor: Emanuel Vieira de Pina, N.º ME1006

Orientador: Prof. Doutor Hugo Proença

Mindelo, 2021

UNIVERSIDADE DO MINDELO | UNIVERSIDADE DA BEIRA INTERIOR

Emanuel Vieira de Pina

Desenvolvimento de um Chatbot para Universidade do Mindelo

Mestrado em Engenharia Informática

Dissertação de Mestrado

Trabalho efetuado sob orientação do

Professor Doutor Hugo Proença

Abril 2021

Dedico a presente dissertação a minha filha **Emanuelle Virgínia Gomes Vieira de Pina** e a minha mãe **Virgínia Andrade Vieira**

AGRADECIMENTOS

Primeiramente agradeço a Deus por ter me abençoado com saúde e capacidade para realizar este trabalho.

Um agradecimento especial ao meu orientador, Professor Doutor **Hugo Proença**, pela orientação, atenção, disponibilidade, sensatez e valor das suas opiniões.

Ao Magnifico Reitor da Universidade do Mindelo Professor Doutor **Albertino Emanuel Lopes da Graça**, pela sua dedicação, esforço na realização do curso, pelo apoio, incentivo para a conclusão do trabalho.

À Universidade do Mindelo e respetivos docentes e funcionários, pela constante aprendizagem e evolução tanto a nível de conteúdos escolares como pessoais durante o curso.

À minha família, pelas condições e possibilidade de dia após dia, trabalhar sem interrupções.

Obrigado a todos.

RESUMO

Os *chatbots* têm sido alvo de grande estudo por parte da comunidade de Inteligência Artificial, em particular nos domínios do Processamento de Linguagem Natural e da Interação Homem-Máquina, sendo o seu futuro considerado promissor. A ideia de automatizar conversas através de autómatos é bastante interessante para muitas empresas, visto que acarreta vários benefícios a um preço relativamente baixo. Por exemplo, pode dar apoio ao cliente a tempo inteiro, visto que consegue comunicar com um grande número de pessoas em simultâneo e pode ser utilizado como ferramenta de automação de trabalho repetitivo.

Esta dissertação tem como foco a sugestão de uma arquitetura relativa a um sistema conversacional (*chatbot*), a operar em contexto académico, com o objetivo dar resposta aos alunos da Universidade do Mindelo. O *chatbot* proposto tem o nome de *UMbot* e é representativo de uma abordagem híbrida, pela combinação de técnicas de processamento de linguagem natural e de um modelo Deep Learning, na regação das suas respostas.

De forma a provar que o sistema apresentado representa uma solução prática viável, procedeu-se ao desenvolvimento de uma fase primária do motor conversacional do sistema, expondo as diferentes abordagens realizadas de forma que, pela análise dos seus resultados, fosse possível inferir sobre a sua melhor implementação.

Adicionalmente, este trabalho reporta um estudo sobre o funcionamento dos *chatbots*, algumas ferramentas para a sua construção e uma efetiva implementação para o uso em um aplicativo de mensagens, o *Telegram*. Por fim, esta aplicação tem como objetivo sanar dúvidas e responder perguntas dentro do contexto universitário da Universidade do Mindelo. O serviço foi implementado e depurado durante o desenvolvimento desta Dissertação, restando agora ser colocado em produção.

Palavras-chave: *Chatbots, Inteligencia Artificial, Processamento de Linguagem Natural, Sistemas Conversacionais, Deep Learning.*

ABSTRACT

The chatbots have been the subject of a great study by the Artificial Intelligence community, particularly in the domains of Natural Language Processing and Human-Machine Interaction, and their future is considered promising. The idea of automating conversations using automata is quite interesting for many companies, since it has several benefits at a relatively low price. For example, it can provide full-time customer support, as it can communicate with a large number of people simultaneously and can be used as an automation tool for repetitive work.

This dissertation focuses on the suggestion of an architecture related to a conversational system (chatbot), operating in an academic context, in order to respond to the students of the University of Mindelo. The proposed chatbot is called UMbot and is representative of a hybrid approach, through the combination of natural language processing techniques and a Deep Learning model, in the regulation of your responses.

In order to prove that the presented system represents a practical practical solution, a primary phase of the system's conversational engine was developed, exposing the different approaches carried out in such a way that, through the analysis of its results, it was possible to infer about its better implementation.

In addition, this work reports a study on the functioning of chatbots, some tools for their construction and an effective implementation for use in a messaging application, Telegram. Finally, this application aims to answer questions and answer questions within the university context of the University of Mindelo. The service was implemented and debugged during the development of this Dissertation, and now it remains to be put into production.

Keywords: Chatbots, Artificial Intelligence, Natural Language Processing, Conversational Systems, Deep Learning.

ÍNDICE

RESUMO v
ABSTRACT VI
NOTAÇÃO, ACRÓNIMOS E GLOSSÁRIO IX
CAPÍTULO I 1
1.1 Introdução 1
1.2 Motivação 3
1.3 Objetivos 4
1.3.1 Objetivo Geral 4
1.3.2 Objetivos Específicos 4
1.4 Estrutura do Documento 5
CAPÍTULO II 7
2.1 Chatbots - Definições e Aplicações 7
2.2 Classificação do Chatbot 9
2.2.1 Os Diferentes Tipos de Chatbot 10
2.2.1.1 Modelos Baseados na Extracção de Informação e Modelos Generativos 10
2.2.1.2 Conversas Curtas *Versus* Conversas Longas 11
2.2.1.3 Domínio Aberto e Domínio Fechado 12
2.2.1.4 Chatbots Conversacionais e Chatbots Orientados a uma Tarefa 12
2.2.1.5 Perspectiva Geral da Estrutura de um Chatbot 13
2.2.2 Arquiteturas de Chatbots 15
2.2.3 Pattern Matching 15
2.2.3.1 Modelos de Rede Neuronal 15
2.3 Estado da Arte 16
2.4 Crescimento da Popularidade De chatbots 23
2.5 Tendências que Influenciam o Desenvolvimento da Tecnologia de Chatbot 24
CAPÍTULO III 27
3.1 Processamento de Linguagem Natural 27
3.1.1 Módulos de Processamento em Sistemas de NLP 28
3.1.2 Fases de Processamento de Linguagem Natural 30
3.1.2.1 Segmentação de Documentos e Frases 30
3.1.2.2 Atomização 30
3.1.2.3 Part-of-Speech Tagging 31
3.1.2.4 Normalização de Palavras 31
3.1.2.5 Parsing 32
3.1.3 Processamento de linguagem natural baseado em embeddings 33
3.1.3.1 Word Embeddings 34

3.1.3.2 Modelos de Linguagem Pré-treinados ... 36
3.1.4 Natural Language Understanding (NLU) ... 38
3.1.5 Natural Language Generation (NLG) ... 39
3.1.6 Tensorflow Embedding: RASA ... 40
3.1.7 Recurrent Neural Networks ... 42
3.1.8 Long Short Term Memory Networks ... 43
3.1.9 Arquiteturas de Transformador ... 47
3.1.9.1 Transformador como Política de Diálogo ... 47
3.2 Plataformas e Frameworks para Construção de Chatbots ... 50
3.2.1 Rasa Framework ... 52
3.2.1.1 Rasa NLU ... 54
3.2.1.2 Rasa Core ... 63
3.2.1.3 Rasa X ... 70
CAPÍTULO IV ... 75
4.1 Configuração e Desenvolvimento ... 75
4.1.1 Requisitos ... 75
4.1.2 Arquitetura ... 75
4.1.3 Preparação do Ambiente e Criação do Projeto com o Rasa ... 76
4.1.4 Migração de Dados ... 77
4.1.5 Configurações do Rasa ... 79
4.1.5.1 Definição do Pipeline ... 81
4.1.5.2 Políticas de Treinamento ... 81
4.1.5.3 Deploy no Servidor ... 84
4.1.5.4 Instalação do Rasa X ... 82
CAPÍTULO V ... 87
5.1 Resultados da Interface Conversacional. ... 87
5.1.1 Intent Classification ... 87
5.1.1.1 Pré-processando ... 87
5.1.1.2 Métricas de Desempenho ... 88
5.2 Avaliação e Resultados com Utilizadores ... 96
5.2.1 Metodologia ... 96
5.2.2 Teste de Conversação e do Survey ... 97
5.2.2.1 Resultados ... 98
5.2.2.2 Discussão ... 100
CAPÍTULO VI ... 103
6.1 Conclusão ... 103
6.2 Limitações ... 104
6.3 Trabalho Futuro ... 104
REFERÊNCIAS ... 106
APÊNDICE ... 114

ÍNDICE DE FIGURAS

Figura 1.1: Tipos de comunicação virtual e exemplos 2
Figura 2.1: Resultados do estudo “The 2016 Mobile Messaging Report”. Fonte: [14] 9
Figura 2.2: Estrutura dos modelos generativos e baseados em extração. 11
Figura 2.3: Estrutura de conversação do chatbot. 14
Figura 2.4: Interação com o ELIZA. Fonte: https://en.wikipedia.org/wiki/ELIZA. 17
Figura 2.5: Exemplo de interação conversacional presente no chatbot PARRY. 19
Figura 2.6: Trecho de Conversa com ALICE. Fonte: Traduzido de Prize (2004). 20
Figura 2.7: Trecho de Conversa com ROSE. Fonte: Traduzido de Prize (2015). 20
Figura 2.8: Aplicações de mensagens móveis globais mais populares desde janeiro de 2021. Fonte: (Statista, 2021) 25
Figura 3.1: Diagrama NLP com as subseções NLG e NLU. Fonte: [50] 28
Figura 3.2: Exemplo de NER obtido utilizando o toolkit StanfordNLP. Fonte: [59] 31
Figura 3.3: Projeto Penn Treebank [81] part-of-speech tagging [80] 31
Figura 3.4: Exemplo de análise de dependência. Fonte: [66] 33
Figura 3.5: Arquitetura de modelo CBOW. Fonte: Word Embeddings e seu uso em tarefas de classificação de frases [70]. 35
Figura 3.6: Arquitetura do modelo Skip-gram. Fonte: Word Embeddings e seu uso em tarefas de classificação de frases [70]. 35
Figura 3.7: RASA tensorflow embedding em profundidade. Fonte: Tobias Wochinger, Rasa NLU em profundidade [75]. 40
Figura 3.8: Um pedaço de um RNN. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 42
Figura 3.9: Loop desenrolado de um RNN. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 42
Figura 3.10: Módulo de repetição em um RNN padrão. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 44
Figura 3.11: Módulo de repetição nos LSTMs. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 44
Figura 3.12: Unidade de porta de esquecimento LSTM. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 45
Figura 3.13: Camada de porta de entrada LSTM. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 45
Figura 3.14: Atualização do estado da célula LSTM. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 46
Figura 3.15: Porta de saída da célula LSTM. Fonte: Cristopher Olah, Understanding LSTM Networks [79]. 46
Figura 3.16: Uma representação esquemática de duas etapas de tempo da política TED. Autor: [92]. 48
Figura 3.17: F-scores para os diferentes serviços NLU, agrupados por corpus. Fonte: [97]. 51
Figura 3.18: Análise das intenções previstas e suas respetivas confianças NLU no Rasa CLI. 56
Figura 3.19: Comparando componentes para dois pipelines pré-configurados. Fonte: [100]. 57

Figura 3.20: Exemplo de representação de frases com bag-of-words. Fonte: [100] 61
Figura 3.21: Representação do processamento feito com o EmbeddingIntentClassifier. Fonte: [100] 62
Figura 3.22: Exemplo de um diálogo no formato de história Rasa. Fonte [101] 64
Figura 3.23: Arquivo domain.yml de um projeto inicial criado pelo Rasa. Fonte: [102] 65
Figura 3.24: Representação do armazenamento da informação em slot. Fonte: [102] 67
Figura 3.25: Controle de versão integrado ao Rasa X. Fonte: [103].....71
Figura 3.26: Aba conversations do Rasa X. Fonte: [104] 73
Figura 3.27: Janela de compartilhar o assistente com usuários testadores. Fonte: [104].....73
Figura 4.1: Arquitetura do chatbot construída usando a estrutura Rasa, tanto o Rasa Core quanto o Rasa NLU são utilizados para gestão de diálogo e compreensão de linguagem natural. O chatbot é hospedado em uma instância de Amazon Web Services para testes de utilizador.....76
Figura 4.2: Ferramenta presente no Facebook para extração de dados relativos ao utilizador. 78
Figura 4.3: Componentes que compõem o supervised_embeddings_pipeline.....79
Figura 4.4: Configuração final das políticas no ficheiro config.yml.....83
Figura 4.5: Configuração de Instance no AWS para a instalação do Rasa X. 83
Figura 4.6: Instalação do Rasa X através do Prompt de Comando 83
Figura 4.7: Instância criada e seu DNS e endereço de IP.....84
Figura 4.8: Estrutura de um projeto padrão Rasa.....85
Figura 4.9: UI do Rasa X demonstrando que a integração com o repositório foi bem-sucedida. 86
Figura 5.1: Matriz de confusão para a abordagem de pre-trained embedding 89
Figura 5.2: Distribuição de confiança do histograma para a abordagem de pre-trained embedding 91
Figura 5.3: Matriz de confusão para a abordagem de rasa tensorflow embedding 92
Figura 5.4: Distribuição de confiança do histograma para a abordagem de rasa tensorflow embedding 94
Figura 5.5: Gráfico com F1-Score do modelo spacy pré-treinado vs rasa tensorflow embedding. ..95

ÍNDICE DE TABELAS

Tabela 3.1: Exemplo de stemming utilizando o algoritmo de Porter [62]. 32
Tabela 3.2: Critérios adicionais para a avaliação NLU. 52
Tabela 5.1: Valores métricos para a abordagem de pre-trained embedding 90
Tabela 5.2: Valores métricos para a abordagem do rasa tensorflow 93
Tabela 5.3: Resultado das Questões Realizadas no Survey..... 99
Tabela 5.4: Resultado em Relação ao Desempenho e a Utilidade de UMbot. 99
Tabela 5.5 – Comentários dos estudantes no Survey Realizado. 101

NOTAÇÃO, ACRÓNIMOS E GLOSSÁRIO

NOTAÇÃO GERAL

ACRÓNIMOS

IA	INTELIGÊNCIA ARTIFICIAL
IM	INSTANT MESSAGING
NLP	NATURAL LANGUAGE PROCESSING
DL	DEEP LEARNING
UM	UNIVERSIDADE DO MINDELO
PLN	PROCESSAMENTO DE LINGUAGEM NATURAL
NLU	NATURAL LANGUAGE UNDERSTANDING
AC	AGENTE CONVERSACIONAL
HCI	HUMAN-COMPUTER INTERACTION
NER	NAMED ENTITY RECOGNITION
POS	PART-OF-SPEECH
CBOW	CONTINUOUS BAG-OF-WORDS
RNN	RECURRENT NEURAL NETWORKS
LSTM	LONG SHORT TERM MEMORY
SDK	SOFTWARE DEVELOPMENT KIT
REDP	RECURRENT EMBEDDING DIALOGUE POLICY

GLOSSÁRIO

Dataset – Conjunto de dados.

Deep Learning – Subárea de *Machine Learning*, responsável pela aprendizagem de vários níveis de representação e abstração, que permitam fazer com que o conjunto de dados obtenha um sentido, utilizando para tal redes neuronais artificiais.

Epochs – Número de ciclos completos efetuados na rede, para todos os casos de treino.

DIET - Transformador de dupla intenção e entidade. A arquitetura NLU padrão utilizada pelo *Rasa Open Source*, que executa tanto a classificação de intenções quanto a extração de entidades.

Pipeline - A lista de componentes NLU que define o sistema NLU de um assistente *Rasa*. Uma mensagem do usuário é processada por cada componente, um a um, antes de retornar a saída estruturada final.

Slot - Um armazenamento de valor-chave que o Rasa usa para rastrear informações durante uma conversa.

TED Policy - Política de diálogo de incorporação do transformador. TED é a política de diálogo padrão baseada em aprendizado de máquina utilizada pelo *Rasa Open Source*. O TED complementa as políticas baseadas em regras tratando de situações nunca antes vistas, onde nenhuma regra existe para determinar a próxima ação.

CAPÍTULO I

1.1 INTRODUÇÃO

Um *chatbot* é um software de inteligência artificial simula uma conversa numa linguagem percetível ao utilizador. A tecnologia, que cada vez reúne mais interesse, tem como principal propósito a otimização de recursos na interação entre humanos e máquinas, nomeadamente como forma de impulsionar os negócios entre empresas. Do ponto de vista tecnológico, um *chatbot* representa a evolução natural de um sistema de pergunta-resposta que utiliza *Natural Language Processing*. A formulação de respostas a perguntas em linguagem natural é um dos exemplos mais típicos de NLP aplicado ao software que as empresas apresentam aos consumidores.

A ideia subjacente ao fornecimento da capacidade conversacional a uma máquina remonta há cerca de 60 anos, altura em que o famoso matemático *Alan Turing* questionou se "Podem as máquinas pensar?", na sua famosa obra, "*Computing Machinery and Intelligence*", conceptualizando o problema no que foi chamado de jogo de imitação, atualmente amplamente conhecido como teste de *Turing* [1].

Atualmente já é frequente encontrarmos os *bots* embebidos em aplicações de *messaging*, sites ou até em aplicações móveis.

Ora, como todas estas aplicações aumentam, em teoria, a rapidez com que as tarefas são realizadas e diminuem o trabalho que tais tarefas requerem e também a complexidade inerente, não é de estranhar que várias áreas do mundo dos negócios estejam a apostar na tecnologia.

Tal como ilustrado na figura 1.1, podemos distinguir dois grandes tipos de comunicação virtual, quanto ao seu conceito de troca de mensagens: Troca de mensagens de maneira instantânea (em inglês, *Instant Messaging*) e troca de mensagens de maneira não instantânea (em inglês, *Non-Instant Messaging*) [2].

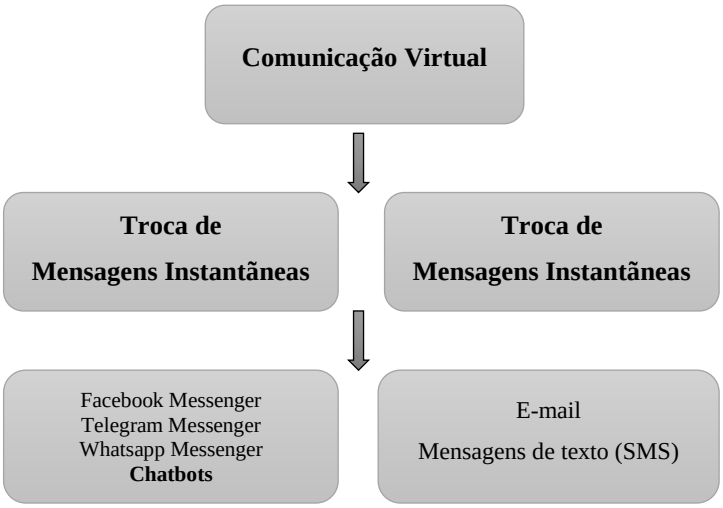

Figura 1.1: Tipos de comunicação virtual e exemplos.

A troca de mensagens instantânea diz respeito a uma forma de comunicação em tempo real, sob a forma de texto ou voz, entre dois ou mais sistemas em rede, onde se incluem aplicações amplamente conhecidas como o *Whatsapp* ou o *Facebook Messenger*, estando também incluídos neste grupo, os sistemas que representam o foco desta dissertação: os diversos *chatbots* desenvolvidos e implementados nas diferentes áreas [3].

A maior parte dos seus utilizadores utilizavam estas aplicações de IM no seu computador pessoal, sendo que, nos últimos anos, estes sistemas se tornaram cada vez mais frequentes noutro tipo de dispositivos, maioritariamente móveis, levando assim ao seu crescimento exponencial em termos do número de utilizadores [2].

No entanto, como nem tudo se traduz em vantagens, um dos problemas que a introdução desta nova forma de comunicar trouxe foi o facto de, com o passar do tempo, esta garantia de comunicação em tempo real, ter levado a que os utilizadores, como consumidores, possuíssem a expectativa de que as empresas ou serviços devessem estar sempre disponíveis, acessíveis e com assistência imediata. Estas três "exigências" revelavam-se difíceis de ultrapassar e, consequentemente, não eram fornecidas por todos os serviços devido à falta de recursos exigidos pela sua implementação.

Foi aqui que o desenvolvimento de programas computacionais, disponíveis 24 horas por dia, 7 dias por semana, capazes de simular a conversa humana, entrou em ação [4].

O crescimento recente das áreas associadas à inteligência artificial, levou a que o desenvolvimento deste tipo de sistemas fosse para outro patamar, mudando totalmente o paradigma de como um serviço é disponibilizado.

Do ponto de vista negativo, os *chatbots* continuam a gerar alguma desconfiança, quer por parte da comunidade científica, quer dos utilizadores [5], pelas carências que revelam ao nível da fiabilidade e capacidade de improviso, sendo tipicamente considerado que não garantem uma simulação de consciência humana positiva e confiável.

1.2 MOTIVAÇÃO

A utilização de *chatbots* está em claro crescimento. No entanto, como se estima que no futuro tenham um peso cada vez maior na vida das pessoas, este é um excelente momento para apostar na ideia, já que é mais fácil ter influência no mercado por diferenciação quando ainda não há sobrelotação. O facto das aplicações de mensagens estarem a suplantar as redes sociais como principal plataforma de utilização na Internet realça ainda mais as vantagens resultantes do desenvolvimento de um *chatbot*.

No contexto académico, o apoio fornecido aos alunos é uma área fundamental para a sua fidelização, sendo cada vez mais vital para as universidades fornecerem este serviço. A forma mais comum de apoio ao aluno na área das telecomunicações é a utilização de *call centres* para lidar com as questões dos alunos, estando esta solução normalmente associada a um alto custo: exige despesas operacionais, de recursos humanos, e limitações recorrentes da necessidade de funcionamento constante.

As aplicações dos *chatbots* podem estar embutidas nos sites e *apps* particulares das universidades ou nas plataformas já existentes, como o *Facebook Messenger*, *Skype*, *Whatsapp* etc.

O objetivo principal desta dissertação é fornecer uma maneira mais conveniente para que o aluno obtenha ajuda, bem como as respostas de que necessita a qualquer momento, através do desenvolvimento de um *chatbot* para apoio à Universidade do Mindelo. A melhoria substancial a que se vem assistindo nas áreas de Inteligência Artificial, mais precisamente na área de *Deep Learning*, conjugada com os avanços presentes nas áreas de compreensão e processamento linguístico, permitiram desenvolver uma nova abordagem ao

destes sistemas conversacionais, fornecendo-lhe, na verdadeira essência da palavra, inteligência, algo que era tido como ilusão até à data [6].

Perante todos estes avanços e desenvolvimentos, a questão principal a ser respondida é a seguinte:

Com base nas tecnologias de DL e NLP, até que ponto pode a vida dos utilizadores-alvo identificados nesta dissertação, melhorar, usando um *chatbot* devidamente treinado para o efeito?

1.3 OBJETIVOS

O trabalho descrito neste documento centra-se num objetivo geral, decomposto em seis objetivos específicos:

1.3.1 OBJETIVO GERAL

Desenvolver um *chatbot*, baseado em NLP, para a Universidade do Mindelo, que aja como um funcionário dos SAA (Serviços Académicos e Administrativos). O nosso objetivo é implementar um *chatbot* para os alunos que desejam ingressar na UM, bem como para os alunos já existentes. Uma vez que os todos têm que pesquisar muito na Internet, este torna-se um trabalho muito moroso. Muitas vezes, um aluno acha muito difícil descobrir as informações relacionadas à UM. Este projeto apresenta a técnica e implementação de um *chatbot* que pode responder a todos os tipos de perguntas relacionadas com a UM num único lugar. É possível obter todos os tipos de respostas num só lugar com muita facilidade. Este é um grande problema que os alunos enfrentam, mas não há nenhum *chatbot* em funcionamento na UM. Este *chatbot* terá como função fornecer todas as informações num único lugar mais algumas informações adicionais. Assim, este projeto deverá economizar tempo e fornecer as informações com muita facilidade.

1.3.2 OBJETIVOS ESPECÍFICOS

- Criar uma plataforma que permita utilizar recursos de NLP para realizar a interação com o utilizador;
- Disponibilizar o *chatbot* em site e *apps*;

- Criar entradas numa base de dados para simular os dados das contas dos utilizadores;
- Realizar testes para assegurar um bom funcionamento do *chatbot* com todos os recursos empregados;
- Avaliar o desempenho e a utilidade do *chatbot* criado; e
- Possuir capacidade para a qualquer momento aceder às informações da universidade.

Para a resolução destas tarefas e com vista ao cumprimento dos objetivos definidos, foram utilizadas as tecnologias *Python*, *TensorFlow* e *Rasa*. Para a representação interna estruturada das mensagens em língua natural trocadas entre o humano e a máquina foi utilizada a notação *JSON*.

1.4 Estrutura do Documento

Esta Dissertação está estruturada em seis capítulos. Após esta Introdução, mais cinco capítulos descrevem o trabalho realizado, sendo eles:

- No **Capítulo II** descrevem-se os principais conceitos associados a um *chatbot*, relevantes para a compreensão do presente trabalho. Nele está contida toda a revisão da literatura, onde se inclui a sua definição e os seus diferentes tipos de implementação, terminando o Capítulo com o estado da arte, onde são descritos o funcionamento e a lógica implementada em todos os *chatbots* referenciados, permitindo inferir acerca da evolução assistida no desenvolvimento deste tipo de sistemas.
- O **Capítulo III** fornece uma apresentação teórica sobre as tecnologias utilizadas para o desenvolvimento do tipo de **chatbot** projetado nesta dissertação, havendo uma secção sobre o processamento da linguagem natural, onde se apresenta em que consiste esta área bem como as suas técnicas mais importantes.
- De seguida, no **Capítulo IV**, apresentamos a arquitetura do *chatbot* proposto e descrevemos todo o processo para a criação do *chatbot UMbot*. Esse capítulo aborda desde a estruturação dos diálogos para o *chatbot* até os testes realizados no sistema desenvolvido.

- No **Capítulo V**, são apresentados testes realizados no sistema e os resultados obtidos a partir do desenvolvimento do *chatbot*, discutindo e avaliando o seu comportamento final.
- A dissertação termina com o **Capítulo VI**, onde são apresentadas as conclusões e efetuada uma análise relativa ao trabalho futuro.

CAPÍTULO II

CHATBOTS: ESTADO DA ARTE

Neste Capítulo, abordamos os conceitos de Processamento de Linguagem Natural (PLN) e dos *chatbots*, necessários para uma melhor compreensão deste trabalho. O Capítulo termina com a apresentação do estado da arte.

2.1 CHATBOTS - DEFINIÇÕES E APLICAÇÕES

Atualmente existem diferentes termos para nos referirmos a um *chatbot*: agentes ou sistemas conversacionais, *chatterbots*, *chat robots* ou simplesmente *bots*, sendo que, todos estes termos são tratados como equivalentes devido ao facto da semântica associada e propósito serem os mesmos: simular a habilidade conversacional humana [7].

Devido à complexidade destes sistemas, é difícil encontrar uma definição universal para o termo *chatbot*, podendo estes ser projetados de inúmeras maneiras e com diferentes propósitos, dependendo sempre da visão do seu desenvolvedor. Para efeito desta dissertação, definimos *chatbot* como sendo [7]:

"*Um agente conversacional baseado em regras e/ou métodos de inteligência artificial, com o objetivo de simular uma conversa humana real, através do uso de linguagem natural para comunicar com os seus utilizadores*".

Os *chatbots* são criados para vários propósitos e, como exemplos, temos: assistentes virtuais, agentes tutores e companheiros virtuais. Também são empregados em diferentes domínios, como no comércio, no entretenimento, na educação e na saúde [8]. No site chatbots.org[1] estão disponíveis vários exemplos de *chatbots* e o seu campo de aplicação.

No presente, os *chatbots* têm vindo a receber atenção crescente, sendo boa parte dela devida a recentes pronunciamentos das empresas Microsoft e Facebook que, em 2016, declararam que os *chatbots* mudarão a forma como interagimos com as aplicações [9]. Vários

[1] https://www.chatbots.org/

chatbots estão sendo incorporados a programas populares de troca de mensagens como o Messenger[2] e Telegram[3]. Os assistentes virtuais, uma forma derivada dos *chatbots*, estão promovendo essa revolução e, como exemplos, citamos: Alexa da Amazon[4], Cortana da Microsoft, Siri da Apple, Google Now[5] como representantes de *bots* que [10].

Wm [11], *Sansonnet et. al.* fornecem uma estrutura básica comportamental, indicativa das propriedades expectáveis num chatbot moderno:

- **Agente de aprendizagem dialógica** – Deve entender o utilizador, isto é, deve fornecer a função de compreensão através da sua capacidade em extrair o contexto e o significado acoplado às diferentes entradas fornecidas pelo utilizador.
- **Agente racional** – Deve ter acesso ou deve ser alimentado por uma base de conhecimento externa e representativa do senso comum e conhecimento geral, de maneira a fornecer um certo tipo de competência aquando da formulação da resposta a devolver ao utilizador.
- **Agente personificado/corporificado** – Deve fornecer a função de presença, sendo uma das maiores características que um *chatbot* deve possuir. A conceção de uma personalidade por parte de um sistema conversacional é um grande desafio, mas que se revela importante no estabelecimento da confiança entre o utilizador e o sistema, sendo o seu objetivo final a perceção deste, por parte do utilizador, como uma pessoa real.

Mas qual a razão para tanto foco e para este súbito interesse neste tipo de sistemas, atualmente?

Para além do constante crescimento na área de Inteligência Artificial e de Processamento de Linguagem Natural, houve recentemente um aumento do interesse nestes sistemas, maioritariamente devido a duas razões [12]:

A primeira prende-se com o aumento dos serviços de troca de mensagem, que inclusive se tornaram mais populares e mais utilizados quando comparados com as redes sociais. A título de curiosidade, as quatro principais aplicações de troca de mensagens

[2] https://messengerplatform.fb.com/

[3] https://telegram.org/blog/bot-revolution

[4] https://developer.amazon.com/public/solutions/alexa

[5] https://www.google.com/landing/now/

(Messenger, WhatsApp, WeChat e Viber) possuem mais utilizadores ativos por mês que qualquer umas das quatro maiores redes socais (Facebook, Twitter, Instagram, Linkedin) [12], [13].

A segunda razão reside no facto de os utilizadores, no papel de consumidores, não aceitarem horários de funcionamento para os serviços de que desejam usufruir. A figura 2.1 ilustra os resultados de um estudo efetuado no ano de 2016, intitulado *The 2016 Mobile Messaging Report*, onde foi demonstrado que 51% dos utilizadores clamam que um serviço devia estar disponível 24 horas por dia, 7 dias por semana, sendo que por esta altura, esse número já deve ter aumentado substancialmente [14].

A eliminação de horários de funcionamento constitui uma das motivações principais para o desenvolvimento de um *chatbot*, pois permite a interação dos utilizadores com o serviço a ele acoplado independentemente da hora e local onde a interação é realizada.

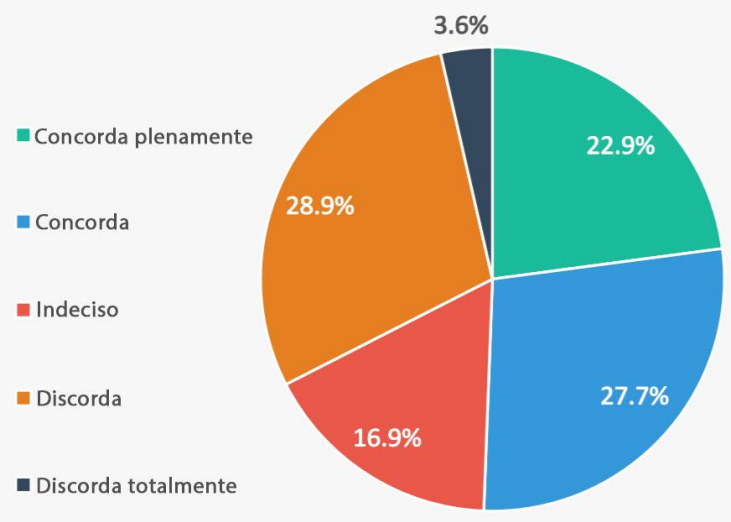

Figura 2.1: Resultados do estudo "The 2016 Mobile Messaging Report". Fonte: [14]

2.2 Classificação do Chatbot

Os *chatbots* podem ser classificados em várias categorias com base em diferentes critérios, como por exemplo o modo de interação, o domínio de conhecimento, o seu uso e as técnicas de geração de resposta que estão normalmente associados ao desenvolvimento desses *chatbots*. Esses critérios podem incluir a estratégia de aprendizagem principal dos *chatbots* ou a extensão em que o contexto precisa ser armazenado e considerado para

compreender a conversa ou o tipo e o propósito da conversa para a qual o *chatbot* está a ser projetado [15].

2.2.1 Os Diferentes Tipos de Chatbot

Esta subsecção tem como propósito apresentar as principais diferenças e decisões a tomar aquando do desenvolvimento de um *chatbot*, identificando os principais aspetos a ter em conta durante a sua projeção.

2.2.1.1 Modelos Baseados na Extracção de Informação e Modelos Generativos

A distinção mais importante a realizar e a ter em conta no desenvolvimento de um chatbot reside no tipo de modelo implementado para a devolução da resposta para o utilizador: se o sistema é baseado na **extração de informação** ou se é um **modelo generativo** [16].

Os *chatbots* baseados no modelo de extração de informação fornecem respostas predefinidas com base na entrada e no contexto do utilizador. Para determinar a resposta, uma variedade de heurísticas pode ser usada, desde a correspondência de expressão baseada em regras simples até uma combinação mais complexa utilizando classificadores de *machine learning*. Os *chatbots* baseados no modelo de extração de informação não criam novas respostas por conta própria, eles escolhem a partir de um *pool* de respostas predefinidas que o desenvolvedor cria. Contrariamente aos modelos baseados na extração de informação, os *chatbots* generativos são capazes de criar novas respostas que são construídas a partir da aplicação de um conjunto de tradução automática e técnicas de inteligência artificial [15].

Ambos os modelos têm vantagens e desvantagens distintas. Os *chatbots* baseados na extração de informação são muito mais fáceis de desenvolver e cometem menos erros gramaticais, porque escolhem as respostas a partir de um conjunto de respostas predefinidas e validadas. No entanto, eles são incapazes de lidar com entradas desconhecidas ou entender o contexto. Assim, informações mencionadas anteriormente numa conversa, como nomes ou lugares, ou conceitos ligados à actualidade não podem ser referidas novamente neste modelo. Os modelos gerativos, por sua vez, são considerados mais inteligentes porque podem entender o contexto e criar suas próprias respostas. O desafio é que eles são muito difíceis de

construir, sujeitos a erros gramaticais e geralmente requerem grandes quantidades de dados de aprendizagem [15].

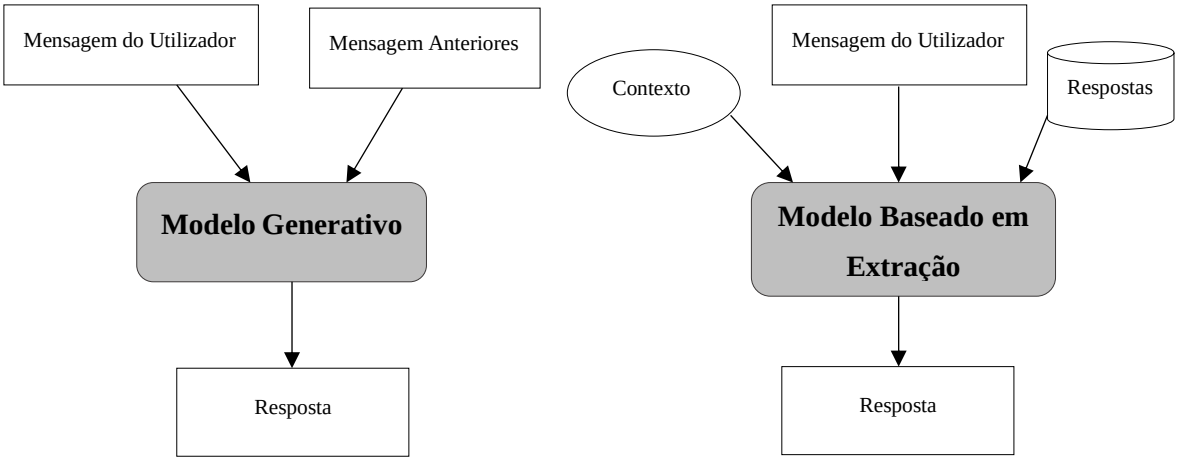

Figura 2.2: Estrutura dos modelos generativos e baseados em extração.

2.2.1.2 CONVERSAS CURTAS *VERSUS* CONVERSAS LONGAS

Conversas longas *versus* curtas estão relacionadas com a complexidade das conversas que o *chatbot* precisa lidar. Quanto mais longa é uma conversa, mais difícil é a sua automação. Aquando da elaboração da arquitetura de um *chatbot,* é comum depararmo-nos com o problema do quão extensas serão as frases envolvidas no diálogo a simular, pois estas podem influenciar, por si só, toda a adesão e qualidade associada a um *chatbot*. Enquanto isso, as conversas de texto curto, com poucos termos em cada interação, são mais fáceis de implementar, sendo o objetivo destes *chatbots*, criar uma resposta única e objetiva para cada entrada recebida. Neste tipo, o sistema devolve uma resposta genérica, procurando evitar detalhes nem efetuando mais perguntas acerca da questão recebida. As respostas são baseadas em objetividade, sendo este o tipo de implementação mais escolhido numa fase inicial de desenvolvimento, devido ao facto de permitir ao sistema moldar e interligar os termos mais facilmente [17].

Para além disso, existem sistemas de conversas extensas, mais difíceis de implementar devido ao facto de necessitarem algum mecanismo de memória a si associados. Esta capacidade de memorização resulta do estabelecimento por parte do sistema de um certo tipo de questionário acerca da entrada fornecida pelo utilizador, de maneira que este consiga obter informação suficiente para retornar a resposta mais adequada, sendo necessário, para

tal, que o sistema se consiga manter a par do que está a ser referido pelo utilizador. Os sistemas de apoio ao cliente são normalmente um exemplo de sistemas que utilizam este tipo de lógica [17].

2.2.1.3 Domínio Aberto e Domínio Fechado

A estrutura e forma das conversas humanas muitas vezes variam de um assunto para o outro, sem qualquer razão em particular. Essas conversas que podem mudar de domínio são chamadas de conversas de domínio aberto, em que os modelos de *chatbot* não são projetados para servir a um propósito específico. As conversas nas redes sociais no Facebook e Twitter são exemplos de conversas de domínio aberto. Elas exigem uma quantidade considerável de treinamento e dados para gerar respostas razoáveis. Um sistema de domínio aberto é normalmente mais difícil de projetar e implementar.

Pelo contrário, as conversas de domínio fechado concentram-se num conjunto limitado de conhecimentos para um tópico específico, a fim de fornecer uma resposta apropriada. Um exemplo é um sistema de reserva de restaurante, onde o sistema tem um propósito específico de reservar uma mesa, sendo a conversa baseada nessa tarefa. Esses sistemas geralmente usam dados específicos de domínio para treinamento. Estes sistemas não possuem a capacidade de responder a temas que não estejam no seu universo de discurso, possuindo uma certa tendência para a devolução de respostas genéricas para tudo aquilo que se encontra deslocado do seu espectro conversacional [15].

2.2.1.4 Chatbots Conversacionais e Chatbots Orientados a uma Tarefa

Outra distinção frequente nos *chatbots* reside no seu propósito: se este é um sistema meramente conversacional ou se é construído de maneira a realizar uma determinada tarefa, i.e., com o intuito de atingir um certo objetivo.

Os *chatbots* orientados para tarefas (os declarativos) são sistemas de propósito único que se concentram na execução de uma função. Usando essencialmente regras NLP e muito pouco ML, eles geram respostas automaticas, mas conversacionais, às perguntas dos usuários. As interações com esses *chatbots* são altamente específicas e estruturadas e são mais aplicáveis às funções de suporte e serviço. Os *chatbots* orientados a tarefas podem lidar com perguntas comuns, como consultas sobre o horário de funcionamento ou transações

simples que não envolvem uma variedade de variáveis. Embora eles usem a NLP para que os utilizadores finais possam interagir de uma maneira coloquial, os seus recursos são bastante básicos. Atualmente, esses são os *chatbots* mais usados [18].

Os *chatbots* preditivos e orientados por dados (os conversacionais) são frequentemente chamados de assistentes virtuais ou assistentes digitais, e são muito mais sofisticados, interativos e personalizados do que os *chatbots* orientados para tarefas. Esses *chatbots* são cientes do contexto e aproveitam a *natural language understanding* (NLU), NLP e ML para aprender à medida que avançam. Eles aplicam inteligência preditiva e análise para permitir a personalização com base em perfis de utilizador e comportamento anterior do utilizador. Assistentes digitais podem aprender as preferências de um utilizador ao longo do tempo, fornecer recomendações e até mesmo antecipar necessidades. Além de monitorar dados e intenções, eles podem iniciar conversas. O Siri da Apple e o Alexa da Amazon são exemplos de *chatbots* preditivos orientados para o consumidor, sendo orientados por dados [18].

2.2.1.5 Perspectiva Geral da Estrutura de um Chatbot

Conforme descrito nas subsecções acima, existem diferentes abordagens que um desenvolvedor pode adotar aquando do desenvolvimento e projeção de um *chatbot*, devendo a sua escolha basear-se no propósito do sistema a desenvolver.

As arquiteturas correspondentes aos sistemas conversacionais podem ser diferenciadas principalmente pela sua esfera operacional (domínio conversacional) e pelo método de geração da resposta a devolver ao utilizador. Diferentes arquiteturas possuem diferentes graus de dificuldade associados à sua implementação, estando ilustradas na figura 2.3 as combinações passíveis de serem implementadas num *chatbot* com base nos dois atributos referidos anteriormente.

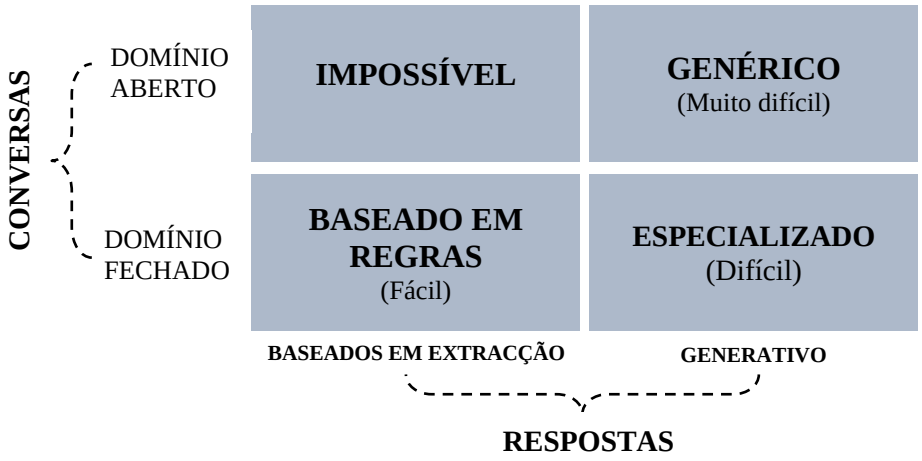

Figura 2.3: Estrutura de conversação do *chatbot*.

Certamente que um *chatbot* que possuí um modelo baseado em extração de informação como lógica de devolução de resposta, nunca poderá implementar um domínio conversacional aberto, sendo este totalmente restrito àquilo que é definido pelo utilizador. Este tipo de *chatbots* é maioritariamente implementado em empresas ou serviços, estando associados a dois domínios distintos de informação: um domínio de resposta e um domínio de informação sobre o serviço/empresa para o qual foi projetado.

Nestes casos, construir um *chatbot* de domínio aberto, revela-se algo desnecessário devido ao facto de a intenção se resumir a que este seja apenas capaz de responder e esclarecer os temas envolvidos e relacionados com esse serviço ou empresa, devendo, portanto, ser projetado como um sistema de domínio fechado [19].

Quanto aos *chatbots* incorporadores de modelos generativos para a devolução de resposta, estes representam a tarefa bastante mais árdua em termos de desenvolvimento quando comparados com os sistemas baseados em extração. Um modelo generativo ser projetado com um domínio fechado é algo contraditório, devido ao facto de as respostas devolvidas pelo sistema não poderem ser limitadas, sendo este um caminho que não se deve tomar nestes casos. Em último caso temos os sistemas que representam a verdadeira essência de inteligência num chatbot: os sistemas genéricos, incorporadores de modelos generativos para a formulação de resposta, e sem qualquer restrição de área conversacional.

2.2.2 ARQUITETURAS DE CHATBOTS

A maioria dos sistemas de *chatbot* atuais historicamente emprega técnicas de reconhecimento de padrões, que utilizam regras para gerar respostas apropriadas. Com as melhorias na inteligência artificial e nas técnicas de *machine learning*, a arquitetura do *chatbot* começa a implementar técnicas de modelagem de rede neuronal mais sofisticadas. As características principais destas duas abordagens são apresentadas nesta seção.

2.2.3 PATTERN MATCHING

A correspondência de padrões (*pattern matching*) é a metodologia mais utilizada para projetar/desenvolver *chatbots*, pois alguma forma de algoritmo de correspondência de padrões está presente em quase todos os sistemas de *chatbot*. A combinação de padrões consiste em identificar a estrutura de uma frase e fornecer uma resposta pré-definida que pode mudar de acordo com as variáveis características da frase [15]. Embora a correspondência direta de padrões seja relativamente simples de implementar, ela sofre de respostas que podem se tornar previsíveis e repetitivas.

2.2.3.1 MODELOS DE REDE NEURONAL

Para superar as limitações de respostas predefinidas de correspondência direta de padrões, abordagens modernas usando redes neuronais têm sido utilizadas na arquitetura de *chatbot* nos últimos anos. O objetivo geral é gerar uma sequência de destino examinando a sequência de origem. No contexto dos *chatbots*, a sequência de origem é a mensagem proveniente do utilizador, enquanto a sequência de destino é a resposta da máquina.

Um modelo *sequence-to-sequence* usa duas redes neuronais recorrentes e um codificador-descodificador que usa uma sequência como entrada e gera outra sequência como saída. O codificador mapeia a sequência de entrada para uma representação codificada dessa sequência, que é então usada pelo descodificador para gerar uma sequência de saída. A utilização da técnica de modelo de rede neuronal permite que os *chatbots* criem as suas próprias respostas com base em informações anteriores, porém esses modelos são muito difíceis de treinar, são propensos a erros gramaticais e geralmente exigem grandes quantidades de dados de treinamento [15].

2.3 Estado da Arte

Os *chatbots* com variados graus de inteligência têm vindo a ser desenvolvidos desde os anos 60: os primeiros eram baseados em correspondências simples, baseando-se no reconhecimento de padrões e em palavras chave para a geração da resposta a ela associada. Com o passar dos anos, assistiu-se a um crescimento exponencial na área de sistemas conversacionais inteligentes, levando a que fossem desenvolvidos diversos tipos de *chatbots*, cada um adotando uma técnica própria e inovadora, desde o uso de técnicas de processamento de linguagem natural até à utilização de metodologias de *Machine Learning* e de *Deep Learning*.

Teste de Turing

Em 1950, a meio do século passado, Alan Turing[6] publicou o artigo *Computing Machinery and Intelligence*, no qual partilhou pela primeira vez o *Turing Test*. O Teste de Turing original, também conhecido como Jogo da Imitação, requer três terminais - cada um deles fisicamente separado dos outros dois. Um terminal é operado por um computador, enquanto os outros dois são operados por humanos. Durante o teste, um dos humanos funciona como questionador, enquanto o segundo humano e o computador funcionam como respondentes. O questionador interroga os respondentes dentro de uma determinada área de assunto, usando um formato e um contexto especificados. Após um período predefinido ou um número de perguntas, o questionador é solicitado a decidir qual respondente é humano e qual é o computador. O teste é repetido muitas vezes. Se o questionador fizer a determinação correta em metade dos testes, ou menos, o computador é considerado como tendo inteligência artificial, porque o questionador considerava-o "tão humano“ quanto o respondente humano. Este teste permitia distinguir, então, o comportamento de um programa de computador do de um humano, e acabou por influenciar toda a área da Inteligência Artificial [20].

[6] Alan Mathison Turing (1912-1954) foi um matemático, lógico, criptoanalista e cientista de computação britânico.

ELIZA

A partir desta data, os especialistas em *Computer Science* que queriam criar *software* inteligente começaram a ter sempre em conta o Teste de Turing. O professor Joseph Weizenbaum[7], criou em 1966 aquele que ficou conhecido como o primeiro *chatbot* da história, o ELIZA. Apesar do sucesso do *bot* e de parecer genuinamente inteligente, já que conversava como um humano, o produto de Weizenbaum não era mais do que um conjunto de respostas pré-programadas que eram emitidas sempre que alguma palavra-chave era referida e tinha apenas 204 linhas de código.

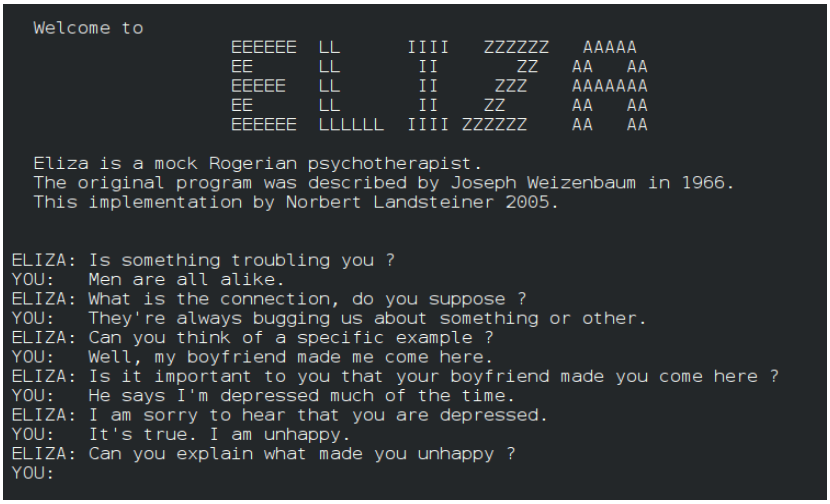

Figura 2.4: Interação com o ELIZA. Fonte: https://en.wikipedia.org/wiki/ELIZA.

O *chatbot* ELIZA tem algumas limitações. ELIZA não possui memória, ou seja, não mantém um histórico das conversas realizadas com os utilizadores. Desta forma, um utilizador pode repetir um trecho do diálogo e ELIZA responderá naturalmente, como se estivesse analisando aquela frase pela primeira vez. Por exemplo: o utilizador pode dirigir um cumprimento a ELIZA ao iniciar uma conversa, por exemplo: “Olá, tudo bem?” e, no meio da conversa, repetir este mesmo cumprimento, que ELIZA responderá da mesma forma, como se ainda não tivesse recebido o mesmo cumprimento antes. Outro problema de ELIZA é que algumas respostas que dá são confusas. Por exemplo, o utilizador poderia dizer “Eu

[7] Joseph Weizenbaum (1923-2008) foi um escritor e cientista da computação norte-americano. Professor emérito do MIT, também criou a linguagem de programação SLIP.

estou bem obrigado", e ELIZA responder algo como, "Há quanto tempo você está bem obrigado?". Embora com limitações, ELIZA conseguiu iludir muitos utilizadores. Um *chatbot*, THE PC THERAPIST [21], baseado na arquitetura de ELIZA, conquistou quatro prêmios *Loebner* logo que a competição foi instituída.

PARRY

Outro sistema de sucesso, que usa técnicas de casamento de padrão parecidas com as que ELIZA utilizava, é o PARRY [22]. Esse *chatbot* foi criado para simular o comportamento paranoico de uma pessoa com esquizofrenia. Ao contrário de ELIZA, que se fazia passar por psicoterapeuta, PARRY foi criado para que as pessoas conversassem com um paciente paranoico.

Colby, o criador de PARRY, elaborou a sua própria versão do *Imitation Game* para validar a sua criação. Ele convidou psiquiatras para conversar com PARRY e com três pacientes humanos diagnosticados como paranoicos. O resultado do teste comprovou o bom desempenho de PARRY, uma vez que os psiquiatras não conseguiram distinguir quem eram os humanos e quem era o sistema. Entretanto, é possível que PARRY tenha conseguido enganar os juízes porque simular um comportamento paranoico não deve seguir uma certa lógica o que, mas mesmo assim, não é trivial [23]. Para [23], PARRY é um avanço sobre ELIZA porque ele possui personalidade. Existe uma versão de PARRY no site *botlibre*[8].

PARRY tem implementado no seu mecanismo uma representação de estado emocional, e reproduz esse estado através de alguns comportamentos como o medo, a raiva e a desconfiança. PARRY possui variáveis para guardar cada emoção que ele pode expressar. Essas variáveis são incrementadas ou decrementadas de acordo com o texto digitado pelo utilizador. As respostas de PARRY são selecionadas de acordo com os valores dessas variáveis. Na Figura 2.5 temos um trecho da conversa com PARRY. PARRY também apresenta algumas limitações. Ele tem problemas em determinar a raiz das palavras, o corretor ortográfico não é eficiente e há um problema clássico com anáforas [24].

[8] https://www.botlibre.com/browse?id=857177

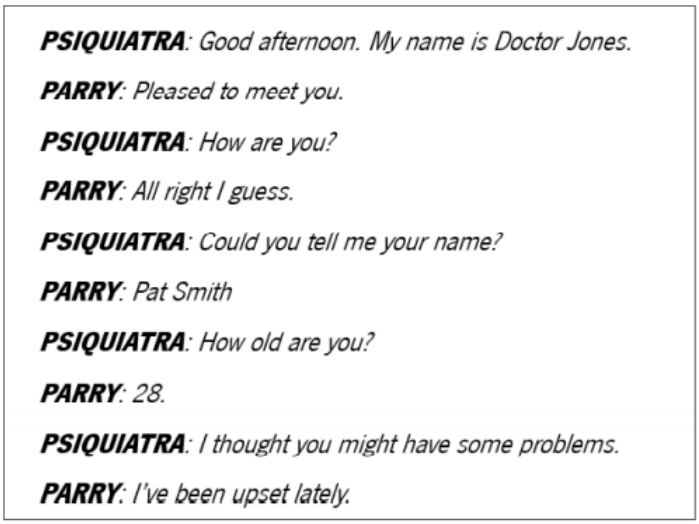

Figura 2.5: Exemplo de interação conversacional presente no *chatbot* PARRY.

ALICE

A.L.I.C.E - Acrônimo para *Artificial Linguistic Internet Computer Entity*, é um *chatbot* de código aberto, criado inicialmente por Wallace em 1995 tendo, mais tarde, recebido contribuições de vários desenvolvedores espalhados pelo mundo [25]. ALICE é o primeiro *chatbot* escrito na linguagem AIML. Ela é um dos *chatbots* mais conhecidos e premiados do mundo. ALICE já ganhou o prêmio Loebner três vezes, nas competições dos anos 2000, 2001 e 2004.

Devido ao seu grande sucesso, ALICE foi a responsável pela difusão do AIML, tendo influenciado a criação de centenas de *chatbots* em diferentes línguas para as mais variadas aplicações. Na Figura 2.6 podemos ver o trecho de uma conversa, traduzida para o português, de ALICE quando ela participou e ganhou o prémio Loebner em 2004. Em AbuShawar e Atwell (2015), é feita uma crítica ao *chatbot* ALICE [25] em relação ao *chatbot* AskJeeves [26] (atualmente o AskJeeves é conhecido como *ask.com*, e não funciona mais como um *chatbot*, mas sim como um motor de busca). O autor relata que o AskJeeves utiliza técnicas sofisticadas de NLP, tais como processamento morfológico, sintático e semântico, enquanto ALICE depende de um número muito grande de regras para realizar o casamento de padrão com a frase do utilizador. Para o autor, ALICE privilegia o tamanho da base de regras ao invés da sofisticação de NLP. ALICE compensa a falta dos processamentos morfológico, sintático e semântico, por ter uma grande base de regras.

ALICE: Olá, meu nome é Alice e eu sou humana.
JUIZ: Olá Alice, como você está?
ALICE: Olá. Tudo está indo muito bem.
JUIZ: Estou contente de estar aqui. Onde você mora?
ALICE: Obrigada pela informação. Estou localizada em Nova York. Onde você está?
JUIZ: Em Nova York. Como você chegou aqui hoje?
ALICE: Conte-me um pouco sobre sua vida em Nova York. Ah! Tudo que eu preciso saber eu aprendi com meu instrutor.

Figura 2.6: Trecho de Conversa com ALICE. Fonte: Traduzido de Prize (2004).

ROSE

O *chatbot* ROSE, criado através da linguagem *ChatScript*, também é um *chatbot* bastante premiado. Ele foi vencedor do prêmio Loebner nas competições dos anos 2014 e 2015, e obteve bons resultados nos anos de 2013 e 2016, onde conquistou o terceiro lugar [27]. ROSE assume a personalidade de uma jovem que trabalha e tem um bom salário. Às vezes, ela pode apresentar um comportamento reservado em relação a alguns assuntos, pois o seu trabalho fez com que ela aprendesse o quanto todos nós estamos sendo observados [28]. A Figura 9 mostra um trecho da conversa de ROSE quando ela foi vencedora do prêmio Loebner em 2015.

JUIZ: Oi, eu sou Andrew. Quem é Você?
ROSE: Sim, eu já sei que o seu nome é Andrew.
JUIZ: Como você está hoje?
ROSE: Bem, obrigada. Como você está?
JUIZ: Você é humana?
ROSE: Claro que sou humana. Não somos todos?
JUIZ: Você pode responder a essa pergunta?

Figura 2.7: Trecho de Conversa com ROSE. Fonte: Traduzido de Prize (2015).

Outros *chatbots* antecessores a ROSE, criados através do *ChatScript*, também obtiveram excelentes resultados. Suzette ganhou o prêmio Loebner em 2010 e Rosette ganhou em [29]. O *chatbot* Angela ficou em segundo lugar no ano de 2012 devido, segundo o criador Bruce Wilcox, a um erro introduzido no protocolo da competição Loebner, mas não no seu mecanismo [30]. Também em 2012, Angela ficou em terceiro lugar na competição ChatterbotBattles e ganhou o prémio, nessa mesma competição, dos melhores 15 minutos de conversa [31].

Os primeiros *bots* com inteligência apareceram apenas no século XXI e já envolviam Processamento de Linguagem Natural [32], uma característica que se tornou indispensável para que estes reunissem condições de aprendizagem. O desenvolvimento da indústria ganhou, então, uma forte componente tecnológica, mas foi, finalmente, em 2016 que o estudo e construção dos *chatbots* sofreu um grande aumento de interesse e tudo graças a uma mudança de paradigma no mundo das aplicações de conversação.

JABBERWACKY

Jabberwacky foi um sistema conversacional criado por Rollo Carpenter em 1997, com o objetivo de, segundo o autor, "simular a conversa natural humana, de uma maneira interessante, divertida e bem-humorada" [33], ou seja, era um sistema de domínio aberto, sem qualquer personalidade adjacente e sem qualquer público-alvo previamente estabelecido. Este sistema revolucionou de certa forma, a tecnologia de desenvolvimento dos *chatbots* até aquela data, devido ao facto de ter sido o primeiro sistema a fugir às bases de conhecimento estáticas, procurando uma abordagem mais dinâmica e generativa [34]. O sistema tinha a capacidade de colecionar e adicionar os vários pares de perguntas e respostas, efetuados entre si e os seus utilizadores, à sua própria base de conhecimento, tornando esta dinâmica e capaz de retornar diferentes respostas às mesmas perguntas, algo que nunca tinha sido realizado até à data [34]. No entanto, o termo generativo não é o mais correto para caracterizar o sistema devido ao facto de este não ser capaz de gerar respostas autónomas, mas sim apenas devolver algo pré-declarado no seu universo de conhecimento, mesmo que este seja mais amplo e diversificado quando comparado com os sistemas apresentados anteriormente. Foi esta nova abordagem, que permitiu a este sistema vencer o prémio *Loebner*, uma competição anual, em tudo semelhante ao teste de *Turing*, responsável por premiar os sistemas conversacionais, considerados pelos jurados como os que melhor representam a capacidade conversacional humana.

CHATBOTS MODERNOS

Em abril de 2016 o Facebook anunciou que o Messenger passaria a suportar conversações individuais com *bots* personalizados, o que exponenciou, de uma forma assinalável, o mediatismo destes simuladores.

"*Cerca de 90% do nosso tempo no telemóvel é passado em plataformas de e-mail e mensagens. Adoraria formar equipas que criassem coisas para sítios onde os consumidores vão!*" - **Niko Bonatsos**, Diretor Geral na *General Catalyst*.

"*As pessoas estão agora a passar mais tempo em aplicações de mensagens do que nas redes sociais, o que é um grande ponto de viragem. As aplicações de mensagens são as plataformas do futuro e os bots vão ser o meio para aceder a todos os tipos de serviço*." - **Peter Rojas**, empresário e trabalhador da *Betaworks*.

"*As mensagens são os locais onde passamos uma grande parte do tempo e esperamos comunicar. É ridículo que ainda tenhamos que ligar para a maioria dos negócios*." - **Josh Elman**, sócio parceiro na *Greylock.*

Ao trazer os *chatbots* para os locais onde as pessoas passavam mais tempo no telemóvel, a adesão dos mesmos sofreu um enorme incremento, transformando de uma forma gigantesca a forma como as empresas chegavam aos seus consumidores. Atualmente, outras plataformas como o Skype, o Slack ou o Telegram também já permitem o lançamento e implementação de *bots* nos seus serviços e é provável que mais ferramentas de *messaging* se juntem à lista no futuro.

Nos dias de hoje, os *chatbots* já se encontram num estado muito mais avançado e são o foco de grandes empresas mundiais como demonstram, por exemplo, os casos dos sistemas *Echo* da *Amazon*, da *Siri* criada pela *Apple*, ou do sistema desenvolvido pela *Microsoft*, *Cortana*. Todas estas assistentes pessoais possuem a capacidade de responder às diferentes perguntas do seu utilizador, tendo como base mecanismos de reconhecimento de áudio e de extração de informação na Web. Recentemente, a sua capacidade foi melhorada para efetuar pesquisa baseada em reconhecimento de imagem.

Na área da saúde, são inúmeros os *chatbots* desenvolvidos e implementados para diversas aplicações: O *chatbot Your.MD*, desenvolvido em 2012, é um sistema conversacional que possui associadas ao seu desenvolvimento e manutenção, cerca de 50 pessoas, o que demonstra o seu grau de complexidade [35]. Este *chatbot* tem como base a utilização de técnicas de IA e de ML, de maneira a fornecer informação médica genérica, bem como a divulgação de novos produtos e serviços clínicos [36]. Neste sistema, os algoritmos nele implementados são treinados sobre uma série de obras literárias medicinais

previamente validadas, permitindo ao sistema aprender diversos sintomas comuns e fornecer recomendações ao seu utilizador [36].

Outro *chatbot* que tem vindo a obter imenso sucesso, aquando da sua implementação e disponibilização, tem o nome de *Florence* e diz respeito a um sistema que tem como propósito funcionar como enfermeira virtual [37]. A sua aplicabilidade baseia-se na capacidade do sistema em relembrar os seus utilizadores da hora a que estes devem tomar a sua medicação, sendo, portanto, um sistema bastante utilizado por utilizadores com idade acima dos 65 anos [38]. Para além desta funcionalidade, *Florence* pode ainda efetuar rastreios de saúde relativos ao peso corporal e ao estado emocional do utilizador, interagindo com este quando são verificadas oscilações comportamentais quando comparado com aquilo que é considerado normal pelo sistema [38].

Estes são apenas alguns exemplos de *chatbots* desenvolvidos e já implementados nos dias de hoje, demonstrativos da evolução e da aplicabilidade destes sistemas conversacionais quer na área da saúde, quer na vida pessoal em geral dos diferentes utilizadores.

2.4 Crescimento da Popularidade De chatbots

Com o surgimento da internet e da computação móvel, pesquisadores de HCI em 2008 previram uma presença crescente de "computadores cada vez mais inteligentes" na vida das pessoas e uma mudança de simulação de comportamento natural para fornecimento de utilidade [39]. Os populares assistentes digitais baseados em voz, incluindo o *Alexa* da *Amazon* e *Siri* da *Apple*, agora são apreciados por imensas pessoas nas suas casas para ajudar a verificar o tempo, encomendar itens e tocar música. O crescimento de *chatbots* baseados em plataformas de mensagens também disparou, com o *Facebook Messenger* sozinho hospedando mais de 300.000 *chatbots* desde 2016. Combinado com *Slack* e *Skype*, as três plataformas hospedam atualmente mais de um milhão de *chatbots* [40]. Esses *chatbots* fornecem utilidade e realizam uma variedade de casos de uso, desde pedidos de pizza (*Domino's*) até reservas de voos (*Kayak*) e compras (*Burberry*) [41].

Estudos recentes têm mostrado que o crescimento do *chatbot* deve continuar a aumentar. De acordo com o *Global Market Insights*, "O tamanho geral do mercado de *chatbots* em todo o mundo seria superior a US $1,3 bilhão em 2024." Portanto, é inevitável

que a indústria de *chatbot* se torne a força motriz das comunicações comerciais. O tamanho do mercado de *chatbot* é projetado para crescer de US $2,6 bilhões em 2021 para US $9,4 bilhões em 2024 a uma taxa composta de crescimento anual (CAGR) de 29,7% [50*]. A *Juniper Research*, sediada no Reino Unido, prevê que em 2022 os *bots* economizarão US $8 bilhões por ano para as empresas [42].

2.5 Tendências que Influenciam o Desenvolvimento da Tecnologia de Chatbot

2.5.1 Avanços em Inteligência Artificial, Aprendizagem Automática, Processamento de Linguagem natural e Tecnologias relacionadas

As tecnologias mencionadas prometem melhorias significativas na capacidade dos *chatbots* de entender a fala humana, incluindo recursos aprimorados de interpretação e previsão em linguagem natural. Além disso, o progresso na modelagem de conversação já permite que os *chatbots* baseados em redes neuronais superem os *chatbots* de correspondência baseados em padrões tradicionais em eficácia por uma grande margem [43]. Resumindo, os avanços da tecnologia estão ajudando os *chatbots* a se tornarem mais inteligentes e eficazes para entender e responder às informações humanas.

2.5.2 Adoção de plataformas de Mensagens

A forma como as pessoas se comunicam mudou drasticamente desde a chegada do iPhone da Apple e da computação móvel em geral. Cerca de 4,5 bilhões de pessoas em todo o mundo utilizam aplicações de mensagens móveis, como *Facebook Messenger*, *WhatsApp* e *WeChat*, como seus meios de comunicação preferidos [44].

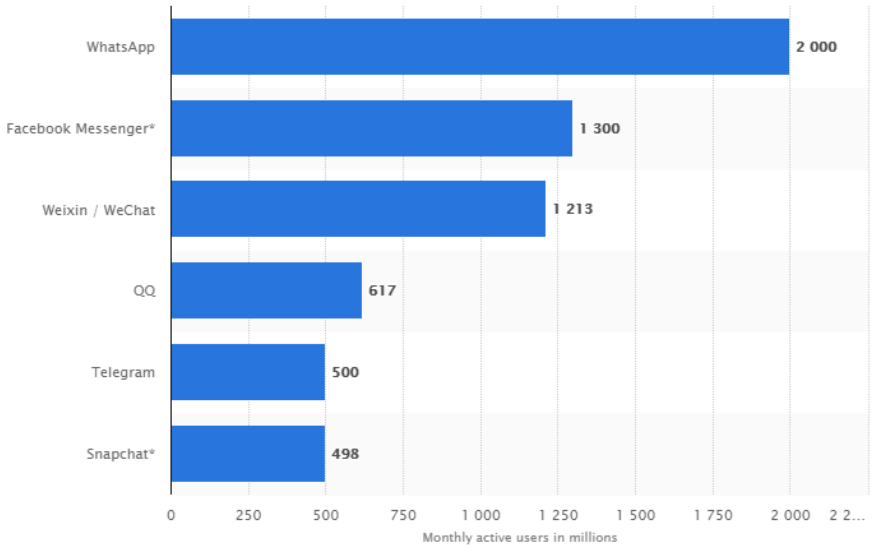

Figura 2.8: Aplicações de mensagens móveis globais mais populares desde janeiro de 2021.

Fonte: (Statista, 2021)

De acordo com [45], aplicativos de mensagens são um ambiente perfeito para *chatbots* devido ao facto de a omnipresença da interface de mensagens ser efetivamente uma interface sem atrito. Em vez de exigir o download, a instalação e a abertura de um novo programa apenas para reservar um restaurante, os utilizadores podem fazer isso nas aplicações de mensagens existentes que já usam diariamente.

2.5.3 Estruturas de *Chatbot* e Custos de Desenvolvimento

O desenvolvimento de *chatbots* tornou-se mais barato e acessível do que antes. Nos últimos anos, os principais gigantes da tecnologia, incluindo *Microsoft*, *Facebook* e *Google*, criaram estruturas para ajudar a acelerar o desenvolvimento, permitindo que os engenheiros utilizem o que há de mais recente em inteligência artificial e tecnologias relacionadas. Por exemplo, depois de lançar o *Bot Framework* em 2016, a *Microsoft* afirmou que já tinha mais de 30.000 desenvolvedores criando *bots* para a sua empresa na plataforma *Skype* [46]. Quando comparados aos aplicativos móveis tradicionais, os *chatbots* são normalmente mais baratos de construir e manter, pois são essencialmente aplicações do lado do servidor com uma interface de utilizador simples. Os *chatbots* também podem ser implantados uma vez sem se preocupar com sua adaptação a vários tamanhos de tela ou sistemas operacionais.

CAPÍTULO III

TECNOLOGIA DE UM CHATBOT

Esta seção apresentará brevemente as técnicas de *deep learning* mais promissoras utilizadas na NLP, particularmente técnicas como *word embedding*, uma nova técnica de *tensorflow embedding pipeline* introduzido pelo *framework Rasa* e RNN. As primeiras e últimas técnicas estão presentes em quase todos os *frameworks*, a fim de fornecer um ecossistema decente para iniciar o desenvolvimento de novos sistema de conversação. Para finalizar o capítulo uma descrição teórico-técnica das tecnologias e conceitos utilizados na presente dissertação, para a projeção e construção do tipo de *chatbot* nela proposto.

3.1 PROCESSAMENTO DE LINGUAGEM NATURAL

Processamento de Linguagem Natural (ou, *Natural Language Processing*) é uma subárea de pesquisa da Inteligência Artificial (IA) que estuda a comunicação, através da linguagem natural, entre o Homem e o Computador [47]. A motivação para esse estudo é fundamentada no aperfeiçoamento da interação homem - computador, a qual ainda é motivo de grande preocupação para os pesquisadores, tendo em vista a sua complexidade. De um modo mais formal, NLP pode ser definida como um ramo da IA que tem por objetivo analisar, interpretar ou produzir texto em uma língua natural (e.g., Português, Inglês, Espanhol) [48].

NLP possibilita que os seres humanos se comuniquem com os computadores de uma forma mais natural, utilizando sua língua nativa. O seu uso permite que os utilizadores não precisem aprender uma linguagem artificial para realizar a interação com o computador, a qual a sintaxe costuma ser um pouco mais difícil, como as linguagens de consulta de banco de dados [49].

Para ser considerado um sistema baseado em NLP, ele deve atender a duas condições [49]:

- Uma parte do sistema deve ser codificado em linguagem natural;
- O processamento de entrada e/ou saída é baseado em aspetos sintáticos, semânticos e/ou pragmáticos de uma língua natural.

O NLP é geralmente feito em duas etapas, *Natural Language Understanding* (NLU) e *Natural Language Generation* (NLG), que envolve, respetivamente, as tarefas de entender e de gerar o texto. Abaixo uma representação do relacionamento entre esses campos de estudo.

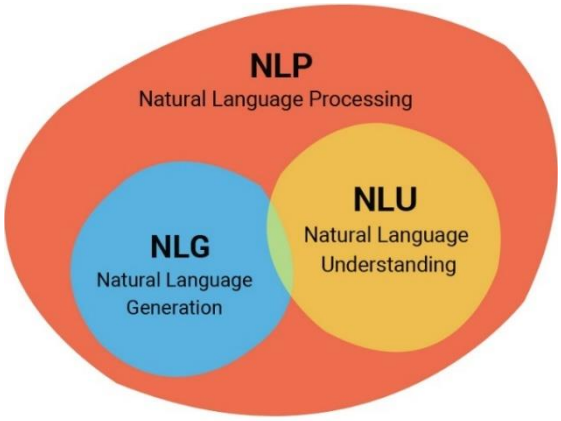

Figura 3.1: Diagrama NLP com as subseções NLG e NLU. Fonte: [50]

3.1.1 Módulos de Processamento em Sistemas de NLP

Para que um sistema computacional interprete uma frase e/ou realize a geração de texto em linguagem natural, é necessário que ele execute alguns processos. Por exemplo, processamento morfológico, sintático, semântico, do discurso e/ou pragmático. Geralmente, os sistemas em NLP são modularizados, e cada um desses módulos é responsável por realizar um desses processamentos. Descrevemos a seguir, esses módulos de processamento em NLP [51], [52], [53], [49]:

- **Processamento Morfológico** - O processamento morfológico trata as palavras quanto à sua estrutura, forma, flexão e classificação, que correspondem às estruturas estáticas (radicais) e variantes (afixos). Para o desenvolvimento de *chatbots*, o processamento morfológico é importante, pois pode resultar em um número bem menor de regras necessárias para realizar o casamento de padrão. Esse processamento permite reduzir as palavras aos seus radicais, e isso faz com que não precisemos criar uma regra para cada variação da palavra (afixos diferentes). Por exemplo, para realizar o casamento de padrão com a palavra "amigo" e algumas de suas variações (e.g.,"amiga", "amigos" e "amigas"), não é necessário a criação de várias regras para o casamento de padrão, sendo apenas

necessário construir a regra com o radical “amig”. O processo para realizar a redução da palavra ao seu radical é conhecido em NLP como *Stemming*. Outra atribuição do processamento morfológico é determinar a classe gramatical das palavras (e.g., substantivo, adjetivo, verbo, advérbio). Esse procedimento é conhecido em NPL como *Parts-of-Speech*, ou *POS-Tagging*.

- **Processamento Sintático** - Esse módulo de processamento se encarrega de verificar as relações formais entre as palavras, ou seja, qual é o papel de cada palavra numa frase e a sua correlação com as demais palavras (e.g., sujeito, predicado, objeto direto, objeto indireto). De uma outra perspetiva, o objetivo do processamento sintático é determinar os componentes estruturais das frases. Esse processo mapeia uma frase na sua estrutura sintática. Essa etapa também é conhecida como *Parsing*.
- **Processamento Semântico** - O processamento semântico atribui o sentido das palavras que foram identificadas pelo processamento sintático. A representação do significado é um grande problema em NLP, pois o sentido das palavras em uma frase pode ser interpretado de forma ambígua. Por exemplo, a palavra banco em “Eu irei ao banco”, “Eu vou sentar no banco” ou “Eu banco o jantar”. É interessante lembrar que não existe uma correspondência direta entre sintaxe e semântica. Uma mesma estrutura sintática pode originar diferentes representações semânticas. Por exemplo, “Achei minha caneta de ouro”. A palavra “ouro” pode se referir a um metal precioso, como também a um objeto estimado.
- **Processamento do Discurso** - Essa etapa tenta identificar qual é a influência de uma ou mais frases na interpretação das frases subsequentes, ou seja, ela extrapola a estrutura da frase que está sendo verificada atualmente pelo processador. O processamento do discurso é essencial para a resolução de pronomes (e.g., eu, você, esta, aquela) e de dêiticos (e.g., aqui, hoje).
- **Processamento Pragmático** - O processamento pragmático verifica as relações entre as mensagens e os sujeitos, ou seja, analisa a intenção de quem fala e o reconhecimento desta intenção pelo ouvinte, levando em consideração o contexto em que o emissor e o ouvinte estão inseridos.

3.1.2 FASES DE PROCESSAMENTO DE LINGUAGEM NATURAL

Com o objetivo de extrair informações relevantes de amostras de texto, um *pipeline* de mineração de dados e tarefas de processamento deve ser seguido. Dessa forma, os computadores são capazes de usar a linguagem humana e produzir os resultados necessários. As subseções a seguir exploram as principais etapas usadas na PLN para preparação de dados e extração de recursos.

3.1.2.1 SEGMENTAÇÃO DE DOCUMENTOS E FRASES

A primeira etapa ao analisar dados de texto é dividir o documento em diferentes seções, de acordo com a estrutura lógica. Esta etapa é seguida pela segmentação da seção em blocos elementares que tratam do mesmo tópico [54]. Essas são as unidades padrão de análise em sistemas de PLN e muitas vezes correspondem a sentenças. Esse processo ocorre utilizando um modelo que examina dicas sintáticas, como sinais de pontuação e letras maiúsculas [55]. Além disso, existem alguns sistemas que aplicam medidas de similaridade para agrupar alguns desses blocos [54].

3.1.2.2 ATOMIZAÇÃO

Os blocos elementares são então segmentados mais uma vez em *tokens*, que são o menor elemento dos dados de entrada. Nesse caso, os *tokens* podem ser palavras, números, sinais de pontuação ou mesmo grupos de palavras que não devem ser divididos como nomes de cidades (por exemplo, São Francisco ou Nova *York*) [55]. Portanto, o processo de tokenização deve levar em consideração as dependências de palavras relacionadas e o conhecimento de domínio específico, bem como o tratamento de especificidades linguísticas morfossintaticamente relevantes. Naturalmente, a tokenização requer um modelo treinado ou um modelo baseado em regras, para o qual o conhecimento específico do domínio e regras gramaticais são necessários [56]. Além disso, os algoritmos de tokenização dependem do idioma. Idiomas diferentes têm regras gramaticais distintas, incluindo particularidades ortográficas e lexicais, e a tokenização pode não ser tão tácita quanto na língua inglesa. Um exemplo claro disso são as línguas orientais que não têm limites explícitos de palavras [57]. Também é importante mencionar que pode ser vantajoso remover palavras de interrupção,

como "o" e "a", que fornecem pouca ou nenhuma informação. Após a obtenção dos *tokens* é possível rotulá-los com uma entidade nomeada, permitindo o reconhecimento das diferentes classes semânticas presentes na frase. Essa operação é chamada de *Named Entity Recognition* (NER) e é um dos principais procedimentos nas tarefas de recuperação de informação e *text mining* [58] (veja o exemplo na Figura 3.2). No âmbito deste trabalho, o NER permitiria a identificação das entidades, tais como pessoas, localização, horas / datas, entre outros.

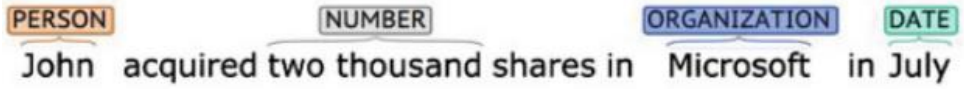

Figura 3.2: Exemplo de NER obtido utilizando o *toolkit StanfordNLP*. Fonte: [59]

3.1.2.3 PART-OF-SPEECH TAGGING

A etapa a seguir no *pipeline* de PLN é a marcação de parte da fala (*PoS tagging*) em que um PoS é atribuído a cada *token*. Uma *PoS tag* resulta da análise lexical de um elemento em uma frase. Este processo é considerado um problema de rotulagem sequencial [57]. Os resultados obtidos com a marcação PoS podem se tornar muito importantes na extração de informações para o contexto específico [60]. Por exemplo, em uma receita de ovos cozidos, o sistema deve reconhecer a palavra "ferver" como um verbo que se refere ao substantivo "ovo". Nesse caso, os verbos têm grande probabilidade de estarem relacionados a métodos de cozimento, enquanto os substantivos estão mais relacionados a ingredientes. Um exemplo de uma frase marcada pode ser observado na Figura 3.3.

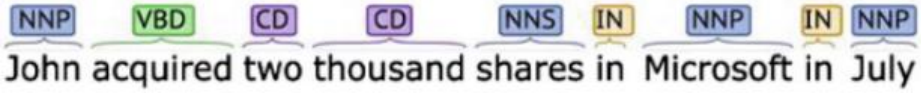

Figura 3.3: Projeto Penn Treebank [81] part-of-speech tagging [80]

3.1.2.4 NORMALIZAÇÃO DE PALAVRAS

Após o processo de tokenização, pode ser necessário normalizar os dados para que sejam facilmente processados. O *stemming* pode ser usado para o propósito referido. O objetivo desse mecanismo é agrupar palavras semelhantes de acordo com seu significado (encontre a raiz de cada palavra). Por exemplo, "*stir*", "*stirs*", "*stirred*" e "*stirring*" têm origem semântica semelhante. A abordagem de derivação mais conhecida é o algoritmo de *Porter*. Este lematizador contém 60 sufixos, uma regra contextual que orientou a remoção

dos sufixos e 2 regras para recodificação de palavras [61]. Após o lematizador *Porter*, todas as palavras do exemplo dado seriam transformadas em "*stir*". Mais exemplos podem ser vistos na Tabela 3.1.

Tabela 3.1: Exemplo de *stemming* utilizando o algoritmo de *Porter* [62].

Original words	***Stems***
studied, studying, student, studies, study	*studi, studi, student, studi, studi*
miner, mining, mine	*miner, mine, mine*
vegetable, vegetarian, vegetate	*veget, vegetarian, veget*
eating, ate, eats, eater	*eat, ate, eat, eater*

A normalização de palavras também pode ser realizada transformando as palavras em sua "forma de dicionário" ou lema [63], em vez de remover o sufixo. As primeiras abordagens para lematização utilizaram tabelas de pesquisa para substituição, que funcionou muito bem para formas de palavras complexas, como as formas "*worse*" e "*worst*" da palavra "*bad*". Mais recentemente, abordagens utilizando tabelas de pesquisa e sistemas baseados em regras com base na combinação de recursos feitos à mão e *machine learning* [64]. Este processo muitas vezes requer a etiqueta da classe gramatical para produzir melhores resultados, uma vez que a transformação que vai ser aplicada a cada palavra pode mudar de acordo com a função morfológica [65]. Adicionalmente, lematizadores podem ser desenvolvidos para um domínio específico, aumentando a qualidade dos resultados produzidos, como o trabalho apresentado em [64].

3.1.2.5 PARSING

As informações sintáticas são extraídas pelo processo de análise. O resultado desse processo é uma árvore estruturada que representa a relação gramatical entre os diversos componentes da frase [57]. De acordo com a integridade da tarefa, é possível distinguir a análise superficial da análise completa. O primeiro método tenta identificar grupos de palavras que geralmente andam juntas. Para fazer isso, o algoritmo foca em uma palavra "principal" de uma determinada classe gramatical a partir da qual ele pode criar a estrutura. Por outro lado, a análise completa tenta determinar a estrutura completa de uma sentença adicionando as relações entre os grupos previamente identificados [60]. Além disso,

analisadores completos podem ser divididos em duas classes: analisadores de constituintes e analisadores de dependências. Enquanto o primeiro grupo se concentra em dominância e precedência e relações frasais na frase, analisadores de dependência discernem e classificam as relações palavra-a-palavra (ver Figura 3.4). No último caso, os rótulos produzidos são, por exemplo, sujeito ou objeto e, portanto, analisadores de dependência são considerados como uma abordagem mais direta para a semântica de uma frase [60].

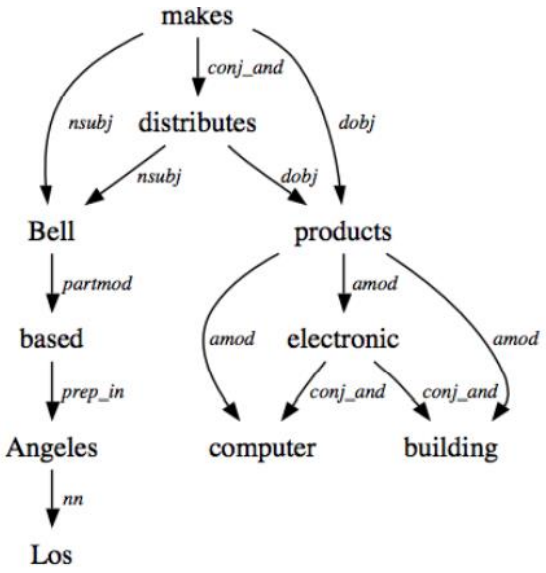

Figura 3.4: Exemplo de análise de dependência. Fonte: [66]

3.1.3 PROCESSAMENTO DE LINGUAGEM NATURAL BASEADO EM EMBEDDINGS

Na seção anterior, as técnicas utilizadas para extrair recursos textuais do *corpus* foram descritas. No entanto, mais recentemente, surgiram abordagens baseadas em representações vetorizadas de palavras. Esses métodos tentam mapear matematicamente cada palavra (ou frase) em um espaço vetorial, que também considera as relações com outras palavras [63]. Este vetor é uma forma aproximada de representar o significado da palavra e deve manter a relação relativa entre as palavras. Essas representações podem ser geradas a partir de dados de texto anotados com relações semânticas e sintáticas. No entanto, este tipo de dados é muito difícil de obter (requer uma força de trabalho muito especializada, como linguistas), tornando virtualmente impossível gerar um conjunto de dados grande o suficiente. Diante dessa limitação, os métodos não supervisionados apareceram como as

principais formas de gerar vetores de palavras. Existem duas maneiras principais de obter *embeddings* que serão exploradas nas seções a seguir.

3.1.3.1 WORD EMBEDDINGS

Esses modelos baseiam-se na premissa de que palavras que coocorrem devem ter uma semântica semelhante. Dessa forma, consegue captar as relações contextuais das palavras que aparecem próximas umas das outras nas frases. Os vetores de palavras são criados de forma que as palavras que compartilham contextos no texto sejam posicionadas próximas umas das outras no espaço vetorial.

Um algoritmo bem conhecido utilizado neste processo é o *Word2vec*. Este modelo utiliza uma rede neuronal para processar um modelo vetorial baseado em documento para gerar um modelo vetorial baseado em palavras [67].

O *Word2Vec* é de particular interesse para a presente tese, pois já provou funcionar de forma eficiente na área de interfaces conversacionais. Mais especificamente, essa técnica vem com dois modelos, o modelo *Continuous Bag-of-Words* (CBOW) e o modelo *skip-gram*. A técnica do modelo *skipgram* está presente na biblioteca *Keras*[9], que é utilizada em vários *frameworks* para a criação de agentes conversacionais.

- ***Continuous Bag-of-Words Model***

O objetivo da técnica CBOW é bastante simples: calcular a probabilidade condicional de uma palavra-alvo dadas as palavras de contexto que a cercam em uma janela de tamanho k [68]. Nessa técnica, a camada oculta não linear é removida e a camada de projeção é compartilhada por todas as palavras, assim, todas as palavras são projetadas na mesma posição (seus vetores são calculados). Chamamos essa arquitetura de modelo de saco de palavras (ou, *Bag-of-words*), pois a ordem das palavras na história não influencia a projeção [69].

[9] https://keras.io/

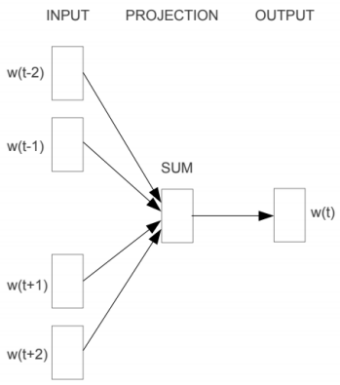

Figura 3.5: Arquitetura de modelo CBOW. Fonte: *Word Embeddings* e seu uso em tarefas de classificação de frases [70].

- ***Skip-Gram model***

Inicialmente introduzido em Estimativa eficiente de representações de palavras no espaço vetorial [69], onde o princípio subjacente é simples, prevendo as palavras do contexto circundante dadas a palavra alvo central. que é exatamente o oposto da técnica CBOW.

Mais precisamente, cada palavra é utilizada como entrada para um classificador log-linear com uma camada de projeção contínua e, em seguida, as palavras são previstas dentro de um intervalo antes e depois da palavra atual [69]. Aumentar o intervalo melhora a qualidade dos vetores de palavras aplicados, mas também aumenta a complexidade computacional. Pois as palavras mais distantes estão menos relacionadas com a palavra atual do que as palavras mais próximas dela [69].

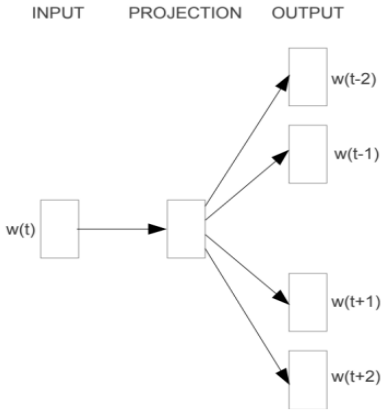

Figura 3.6: Arquitetura do modelo *Skip-gram.* Fonte: *Word Embeddings* e seu uso em tarefas de classificação de frases [70].

Com referência à figura 3.6, exemplos de treinamento essencialmente artificiais são gerados na forma de (*wt,[wt-2, wt-1, wt+1, wt+2]*), considerando que o tamanho da janela é 5. O princípio passa pelo sequência de palavras, o meio é o tamanho da palavra-alvo (entrada) e as palavras que a precedem, enquanto as palavras seguintes formam o contexto da palavra-alvo. O objetivo principal é extrair a camada interna e usar como uma representação vetorial da palavra para o vocabulário treinado, se as palavras estiverem em codificação *one-hot* (um processo pelo qual variáveis categóricas são convertidas de forma a que possam ser fornecidas ao algoritmos de ML para fazer um trabalho melhor na previsão), então ele servirá apenas como uma tabela de pesquisa para as representações vetoriais.

Na prática, o vocabulário é convertido em sequências de índice primeiro, depois o modelo *skip-gram* é treinado nessas mesmas sequências e, finalmente, essas sequências de índice são convertidas em sequências vetoriais, para que o algoritmo final possa ser treinado.

É possível encontrar léxicos pré-treinados criados por meio da aplicação do *Word2vec* em grandes *corpora* (como o *Google News Dataset*). Esses léxicos têm a vantagem de não exigir sessões de treinamento e são genéricos o suficiente para capturar os significados das palavras. No entanto, isso pode ser visto como uma de suas maiores desvantagens, que é o fato de não terem sido treinados especificamente para um determinado domínio ou contexto e podem não representar bem a semântica desse domínio / contexto. As palavras em um domínio específico podem ser agrupadas de forma diferente dos textos onde o modelo foi treinado. Para contornar esse problema, é possível usar essa técnica para criar representações de palavras para *corpora* específicos, mas é importante ter uma grande coleção de dados para obter bons resultados. Outra desvantagem desses métodos é o fato de que os *embeddings* são estáticos, independentemente do contexto. Isso significa que, independentemente do contexto de uma frase, cada palavra terá a mesma representação - o contexto não é levado em consideração após o treinamento. Por exemplo, nas frases: "*I love apple pie*" e "*I want to buy a Apple laptop*", o vetor da palavra *apple* terá os mesmos valores, embora possuam significados totalmente distintos.

3.1.3.2 MODELOS DE LINGUAGEM PRÉ-TREINADOS

Mais recentemente, em 2018, alguns pesquisadores produziram modelos que utilizam uma abordagem diferente para lidar com a falta de contexto nos *embeddings*. Esses

modelos são chamados de modelos de linguagem e empregam um mecanismo de autoatenção para alterar os encaixes de cada palavra de acordo com seu contexto. Nesse caso, a mesma palavra em contextos diferentes será representada por um vetor diferente. Além disso, eles permitiram uma grande melhoria de desempenho em várias tarefas da PLN, como *Question Answering* ou *Natural Language Inference*. O modelo de linguagem mais popular é o *BERT* (*Bidirectional Encoder Representations from Transformers*) do Google [71]. Como afirmado acima, esses modelos aplicam uma arquitetura de autoatenção, ou seja, o módulo *Transformer*, também desenvolvido pelo *Google* [91], que é responsável por aprender as relações contextuais da palavra em um texto. Esta arquitetura é composta por um codificador e um decodificador. No caso do *BERT*, apenas o codificador será considerado ao utilizá-lo, pois o objetivo é produzir um modelo de linguagem. O par codificador-decodificador é utilizado durante o treinamento, que acontece de forma bidirecional, ou seja, cada frase é analisada sem direção preferencial. Isso permite uma compreensão mais profunda do contexto da palavra, tanto à esquerda quanto à direita dela. Para uma determinada frase de entrada, o *BERT* funciona primeiro incorporando os *tokens* e, em seguida, processando-os usando uma rede neuronal. Este *pipeline* produz uma sequência de vetores em que cada um corresponde a um *token* de entrada.

Outro aspeto fundamental do *BERT* é seu processo de treinamento. O *BERT* é treinado utilizando duas abordagens diferentes: previsão do *token* mascarado e previsão da próxima frase. Dessa forma, consegue compreender tanto o contexto de cada palavra, quanto o contexto das frases do documento. A criação de modelos de transformadores em PLN é considerada o equivalente do *ImageNET*[10] para visão computacional. Ele melhorou drasticamente quase todas as tarefas relacionadas ao processamento de texto. Voltando ao exemplo dado acima ("*I love apple pie*" e "*I want to buy a Apple laptop*"), este modelo não irá mais gerar a mesma incorporação para ambas as instâncias da palavra "*apple*", uma vez que estão em diferentes contextos. Além disso, o *BERT* também é capaz de gerar um *embedding* para uma frase inteira embutida em um *token* *__CLS__* especial, que captura o significado dele. Isso pode ser muito útil na construção de *chatbot*. Esses modelos são geralmente aplicados por meio de transferência de aprendizagem e ajuste fino. A premissa é

[10] http://image-net.org/about-overview

que se um modelo for treinado em um *corpora* grande o suficiente, ele será capaz de generalizar para praticamente todos os contextos.

3.1.4 NATURAL LANGUAGE UNDERSTANDING (NLU)

Um dos maiores desafios de sistemas conversacionais é o de entender o texto produzido por um utilizador humano. As razões pelas quais isso é difícil se estendem por amplas razões, mas no centro delas está a infinita variabilidade da linguagem natural. Existem vários caminhos diferentes que um utilizador pode seguir para expressar um único conceito. Diante dessa variabilidade, a capacidade de um humano de extrair significado é algo que talvez nunca seja completamente duplicado.

O *Natural Language Understanding* (NLU) tem como objetivo lidar com esses insumos não necessariamente estruturados que são governados por regras flexíveis, e convertê-los em uma forma estruturada que uma máquina pode entender e agir sobre estes [72]. O NLU busca o entendimento do significado do que o utilizador ou a entrada que é dada quer dizer.

No contexto de um *chatbot*, o objetivo básico do NLU é determinar a conexão entre o texto recebido de um utilizador e a resposta apropriada do sistema. Essa resposta pode ser para fornecer uma resposta simples a uma pergunta, ou envolver uma ação iniciada pelo utilizador ou armazenar as informações fornecidas pelo utilizador em resposta a uma pergunta. A maioria das abordagens ao NLU nos *kits* de ferramentas do *chatbot*, portanto, divide o NLU em duas subtarefas: **classificação de intenção** e **extração de entidades**.

Classificar a intenção do texto de um utilizador é parte de um turno de diálogo iniciado pelo mesmo: o utilizador expressou uma solicitação, ou fez uma pergunta, e o *chatbot* precisa classificar essa solicitação ou questão como pertencente a uma das solicitações que ele sabe como responder ou rejeitá-la, como um evento "sem correspondência". Isso determinará quais das ações ou respostas o *chatbot* irá fornecer.

Extrair entidades do texto do utilizador é tipicamente parte de um turno de diálogo que foi (pelo menos implicitamente) iniciado pelo sistema. O utilizador está fornecendo informações em resposta a uma pergunta. O trabalho da extração é selecionar as informações do texto e possivelmente normalizar essas informações quando, por exemplo, houver várias maneiras de expressar as mesmas informações.

Outro termo para essa tarefa de extrair entidades da tarefa do utilizador para concluir as informações necessárias para um determinado caso de uso é o preenchimento de *slots*. Esse tipo de artifício pode ser interessante para que se evitem repetições de perguntas para diferentes fluxos de conversa, ao reter uma informação que pode ser reutilizada em outros momentos.

3.1.5 NATURAL LANGUAGE GENERATION (NLG)

Geração de Linguagem Natural (ou, *Natural Language Generation* - NLG) é o processo de produzir frases, sentenças e parágrafos que são significativos a partir de uma representação interna. É uma etapa do Processamento de Linguagem Natural e acontece em três fases: identificar os objetivos, planejar como os objetivos podem ser alcançados, avaliando a situação e as fontes comunicativas disponíveis, e efetivando os planos na forma de um texto. Os componentes do NLG são os seguintes [73]:

Gerador - para gerar um texto, é preciso ter um gerador ou um programa que exiba as respostas da aplicação, em frases fluentes relevantes para a situação.

Componentes e Níveis de Representação - O processo de geração de linguagem envolve as seguintes tarefas interligadas:

- Seleção de conteúdo: as informações devem ser selecionadas e incluídas na configuração. Dependendo de como essa informação é analisada em unidades representacionais, partes das unidades podem ter que ser removidas, enquanto outras podem ser adicionadas por padrão.
- Organização textual: a informação deve ser organizada textualmente de acordo com a gramática, deve ser ordenada sequencialmente e em termos de relações linguísticas.
- Recursos linguísticos: para apoiar a realização da informação, os recursos linguísticos devem ser escolhidos. O que engloba a escolhas de palavras específicas, expressões idiomáticas, construções sintáticas e etc.
- Realização: os recursos selecionados e organizados devem ser transformados em um texto real ou uma saída de voz.
- Aplicação: armazena o histórico, estrutura o conteúdo que é potencialmente relevante e implanta uma representação do que realmente sabe.

3.1.6 TENSORFLOW EMBEDDING: RASA

A abordagem *bag-of-words* provou ser uma boa linha de base, podemos encontrar alguns resultados no seguinte artigo "*Baselines and Bigrams: Simple, Good Sentiment and Topic Classification*", mas pode ter algumas limitações no caso prático, falta de vetor palavras de algumas palavras importantes, por exemplo, é um dos problemas mais conhecidos, especialmente se trabalharmos com outros idiomas além do inglês [74].

Enquanto isso, o *framework RASA*, introduziu uma nova técnica, chamada *Tensorflow Embedding Pipeline*, que em vez de usar *embeddings* pré-treinados e treinar um classificador em cima disso, ele treina *embeddings* de palavras do zero. É normalmente usado com uma técnica de saco de palavras para contar com que frequência palavras distintas dos dados de treinamento aparecem em uma mensagem e fornece isso como uma entrada para o classificador, posteriormente. A figura 3.7 explica como os vetores de contagem seriam diferentes [75].

Figura 3.7: *RASA tensorflow embedding* em profundidade. Fonte: Tobias Wochinger, *Rasa* NLU em profundidade [75].

Além disso, outro vetor de contagem é criado para o rótulo de intenção. Este método aprende *embeddings* separados para vetores de recurso e intenção, e ambos os vetores têm as mesmas dimensões, o que torna possível medir a distância vetorial entre os recursos embutidos e os rótulos de intenção embutidos usando similaridade de cosseno [75].

Esta nova técnica apresentada em 2018 foi desenvolvida pela equipa RASA, mas foi inspirada no artigo "*StarSpace: Embed All The Things!*".

O *StarSpace* é um modelo neuronal geral com o objetivo de *embeddings* de entidades de aprendizagem eficientes para resolver uma ampla variedade de problemas, um dos casos que pode ser aplicado é na classificação de intenção para um sistema de IA conversacional [76].

O modelo *StarSpace*, criado pelo *Facebook*, consiste em entidades de aprendizagem, cada uma das quais é descrita por um conjunto de recursos discretos (*bag-off-features*) provenientes de um dicionário de comprimento fixo, onde uma entidade como um documento ou uma frase pode ser descrito por uma *bag-of-words* ou *n-gramas*, uma entidade como esse utilizador pode ser descrita pelo saco de documentos, filmes ou itens de que gostou. O modelo é livre para comparar entidades de diferentes tipos. Uma entidade de utilizador pode ser comparada com uma entidade de item (recomendação) ou uma entidade de documento com entidades de rótulo (classificação de texto) e assim por diante. Isso é possível aprendendo a incorporá-los no mesmo espaço de forma que as comparações sejam significativas [77].

Resumindo, o *StarSpace* incorpora entidades de diferentes tipos em um espaço de *vectorial embedding*, então a palavra estrela (*), significando todos os tipos e o nome do espaço, naquele espaço comum os compara entre si [76].

Para um agente de conversação AI, o *embedding intent classifier*, incorpora entradas do utilizador e *intent labels* no mesmo espaço conforme já explicado, as entradas do utilizador podem ser consideradas por um saco de palavras. Além disso, durante o modelo de treinamento, as entradas do utilizador devem ser comparadas e a seguinte perda deve ser minimizada [76].

$$\sum L^{batch}(sim(a,b), sim(a,b_1^-), ..., sim(a,b_k^-)) \tag{3.1}$$

Na equação 3.1 podemos identificar os seguintes pontos, se aplicados em um sistema conversacional de IA [76].

- *a* são os documentos (saco de palavras).
- *b* são os rótulos (*intents*) do conjunto de treinamento.
- As entidades negativas *b* são amostradas a partir do conjunto de rótulos possíveis.
- (*a*, *b*) são pares de entidades positivas, vem diretamente de um conjunto de treinamento de dados rotulados especificando pares (*a*, *b*).
- *sim* (· , ·) é a função de similaridade. No caso, o padrão Rasa utiliza similaridade de cosseno.

- *L* é a função de perda que compara o par positivo (*a*, *b*) com os pares negativos.

3.1.7 RECURRENT NEURAL NETWORKS

O RNNs ou *Recurrent Neural Network* é um modelo de rede neuronal proposto na década de 80 por (Rumelhart et al., 1986; Elman, 1990; Werbos, 1988) para modelagem de séries temporais [78]. A principal característica desse tipo é que ele pode reter informações passadas de entradas recebidas, permitindo descobrir correlações temporais entre eventos que poderiam estar distantes entre si de entrada nos dados. Também pode ser dito que eles são uma rede neuronal com *loops* permitindo que a informação persista.

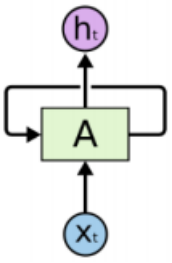

Figura 3.8: Um pedaço de um RNN. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

Na figura 3.8, *A* observa para alguma entrada, neste caso, X_t e produz um valor h_t. Este *loop* permite que as informações sejam passadas de uma etapa da rede para a próxima. Um RNN pode ser considerado como várias cópias da mesma rede, cada uma passando uma mensagem para o sucessor [79].

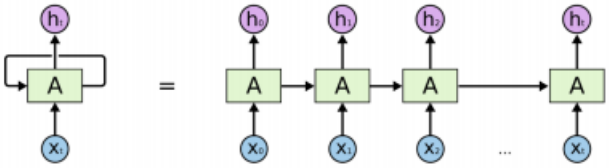

Figura 3.9: *Loop* desenrolado de um RNN. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

Olhando para a figura 3.9, que mostra um *loop* desenrolado de um RNN, podemos perceber que a natureza dessa rede neuronal está relacionada às sequências e listas. Eles são a arquitetura natural da rede neuronal a ser usada para esses dados.

Uma escolha comum nos últimos anos tem sido usar uma rede neural recorrente (RNN) para processar a sequência de curvas de diálogo anteriores, tanto para domínio aberto [80], [81] quanto para sistemas orientados a tarefas [82]. Com dados de treinamento suficientes, um RNN deve ser capaz de aprender qualquer comportamento desejado. No entanto, em uma configuração típica de poucos recursos, onde nenhum grande corpus para treinar uma tarefa específica está disponível, não há garantia de que um RNN irá de fato aprender a generalizar esses comportamentos. O trabalho anterior sobre a modificação da estrutura básica do RNN para incluir tendências indutivas para este comportamento em uma política de diálogo foi conduzido por Vlasov et al. [83] e Sahay et al. [84]. Esses trabalhos visam superar uma característica dos RNNs indesejável para a modelagem de diálogos. RNNs, por padrão, consomem toda a sequência de elementos de entrada para produzir uma codificação, a menos que uma estrutura mais complexa como uma célula de *Long Short-Term Memory* (LSTM) seja treinada em dados suficientes para aprender explicitamente que deve "esquecer" partes de uma sequência.

3.1.8 LONG SHORT TERM MEMORY NETWORKS

Long Short Term Memory Networks são utilizadas extensivamente na PLN, elas foram introduzidas pela primeira vez em 1997 por Hochreiter e Schmidhuber [85]. Os LSTMs são projetados para evitar um problema presente na versão RNN já explicado na seção 3.1.7, o problema de dependência de longo prazo, em poucas palavras, os RNNs simples podem conter uma lacuna no transporte da informação contextual entre as redes, este problema foi explorado em profundidade por Bengio [86], que encontrou algumas razões fundamentais pelas quais deve ser difícil.

Se compararmos o módulo de repetição do RNN padrão como o Cristopher Colah [79] menciona, podemos observar uma estrutura tão simples, com uma única camada de *tanh*, referindo-se na imagem 3.10.

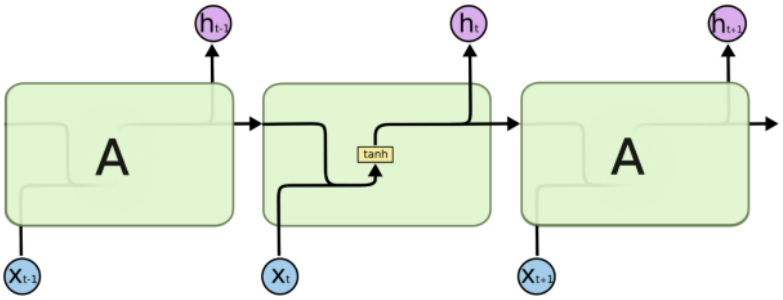

Figura 3.10: Módulo de repetição em um RNN padrão. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

Porém, se compararmos com a estrutura em cadeia dos LSTMs, podemos observar as diferenças no módulo de repetição, ao invés de ter uma única camada de rede neuronal, são quatro, interagindo de forma muito especial, essas camadas são apresentadas na figura 3.11.

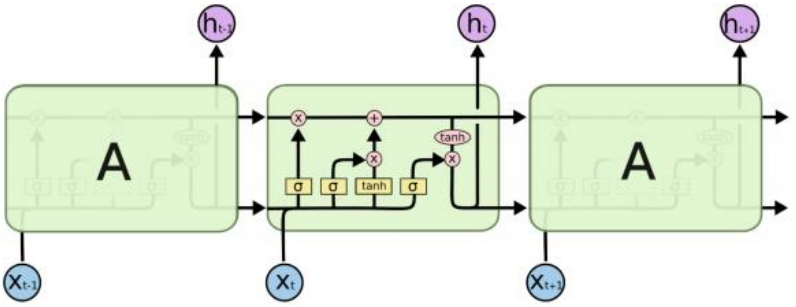

Figura 3.11: Módulo de repetição nos LSTMs. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

Os principais componentes dos LSTMs são a porta de esquecimento, a porta de entrada, o estado da célula e a porta de saída, todos eles serão explicados nesta seção para uma melhor compreensão do fluxo na rede neuronal dos LSTMs.

O primeiro estado, denominada camada de porta de esquecimento, é responsável por decidir quais informações devem ser descartadas do estado da célula, e é fornecido pela equação 3.2.

$$f_t = (W_f \cdot [h_t{-}1, x_t] + b_f) \tag{3.2}$$

Ele olha para $h_t - 1$ e x_t e, em seguida, produz um número entre *0* e *1* para cada número no estado de célula $C_t - 1$. [79] Logicamente, o número *1* representa para manter o valor, enquanto o número *0* representa para esquecer o valor. A figura 3.12 ilustra o processo.

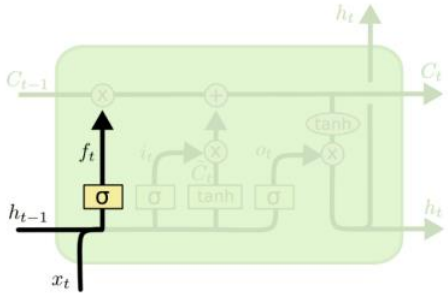

Figura 3.12: Unidade de porta de esquecimento LSTM. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

As portas de entrada decidirão quais informações devem ser armazenadas no estado da célula, que é segmentado em duas seções. A primeira que é chamada pela camada de porta de entrada decide quais valores devem ser atualizados, fornecida pela equação 3.3, denotada por $\tilde{c}_t$ escalado por i_t como a figura 3.13 mostrada [79].

$$i_t = (Wi \cdot [h_t{-}1, x_t] + b_i) \tag{3.3}$$

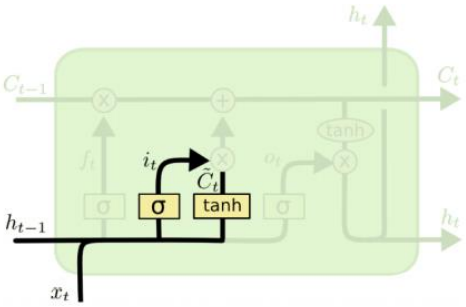

Figura 3.13: Camada de porta de entrada LSTM. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

O antigo estado da célula é então atualizado de acordo com a equação 3.4, doado por $\tilde{c}_t - 1$ para o novo estado doado por C_t. Os passos anteriores já decidiram isso, então devemos apenas executá-lo. Deve-se multiplicar o estado antigo por f_t, esquecendo os valores

que decidimos na primeira etapa, então adicionamos $i_t * C_t$. Esses devem ser os novos valores do estado da célula. As conexões desta atualização estão destacadas na figura 3.14 [79].

$$C_t = f_t C_t{-}1 + i_t tanh(W_C \cdot [h_t{-}1, x_t] + b_C) \tag{3.4}$$

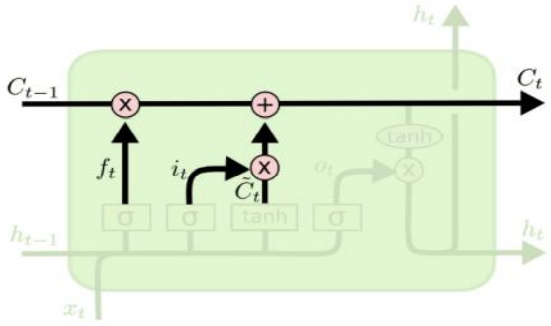

Figura 3.14: Atualização do estado da célula LSTM. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

Na etapa final, ele deve decidir qual deve ser a saída. A saída é baseada no estado atual da célula, mas em uma versão filtrada. Ele executa uma camada *sigmóide* que decide quais partes do estado ela deve produzir, e então, o estado da célula deve ser colocado em *tanh* e multiplicado pela saída da função de porta *sigmóide*. É revelado pelas equações 3.5, 3.6 e a figura 3.15 [79].

$$o_t = (W_o \cdot [h_t{-}1, x_t] + b_o) \tag{3.5}$$

$$h_t = o_t * tanh(C_t) \tag{3.6}$$

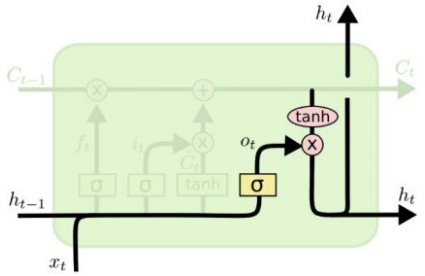

Figura 3.15: Porta de saída da célula LSTM. Fonte: Cristopher Olah, *Understanding LSTM Networks* [79].

3.1.9 ARQUITETURAS DE TRANSFORMADOR

A arquitetura do transformador (ou, *transformer*) substituiu nos últimos anos as redes neuronais recorrentes como o padrão para modelos de linguagem de treinamento, com modelos como *Transformer-XL* [87] e GPT-2 [88] alcançando perplexidades muito menores em uma variedade de corpora e produzindo representações que são útil para uma variedade de tarefas *downstream* [89], [90]. Além disso, recentemente os transformadores mostraram ser mais robustos a entradas inesperadas [91]. Intuitivamente, como o mecanismo de autoatenção pré-seleciona quais *tokens* contribuirão para o estado atual do codificador, um transformador pode ignorar *tokens* não informativos (ou adversários) em uma sequência. Para fazer uma previsão em cada etapa de tempo, um LSTM precisa atualizar sua célula de memória interna, propagando essa atualização para etapas de tempo posteriores. Se uma entrada na etapa de tempo atual for inesperada, o estado interno é perturbado e na etapa de tempo seguinte a rede neural encontra um estado de memória diferente de qualquer coisa encontrada durante o treinamento. Um transformador contabiliza a história do tempo por meio de um mecanismo de autoatenção, tornando as previsões em cada etapa de tempo independentes umas das outras. Se um transformador receber uma entrada irrelevante, ele pode ignorá-la e usar apenas as entradas anteriores relevantes para fazer uma previsão.

Uma vez que um transformador escolhe quais elementos em uma sequência usar para produzir um estado de codificador em cada etapa, segundo os autores de [92] poderia ser uma arquitetura útil para o processamento de histórias de diálogo. A sequência de declarações em uma conversa pode representar vários tópicos intercalados, e o mecanismo de autoatenção do transformador pode simultaneamente aprender a desembaraçar esses segmentos de discurso e também a responder de forma adequada.

3.1.9.1 TRANSFORMADOR COMO POLÍTICA DE DIÁLOGO

No artigo *Dialogue Transformers* [92], os autores propuseram a política do *Transformer Embedding Dialogue* (TED), que simplifica muito a arquitetura do REDP. Semelhante ao REDP, os autores não utilizaram um classificador para selecionar uma ação do sistema. Em vez disso, treinaram conjuntamente *embeddings* para o estado de diálogo e cada uma das ações do sistema, maximizando uma função de similaridade entre eles. No momento da inferência, o estado atual do diálogo é comparado a todas as ações possíveis do

sistema e aquele com a maior similaridade é selecionado. Uma abordagem semelhante é adotada por [93], [84], [81] no treino de modelos de recuperação para o diálogo orientado a tarefas.

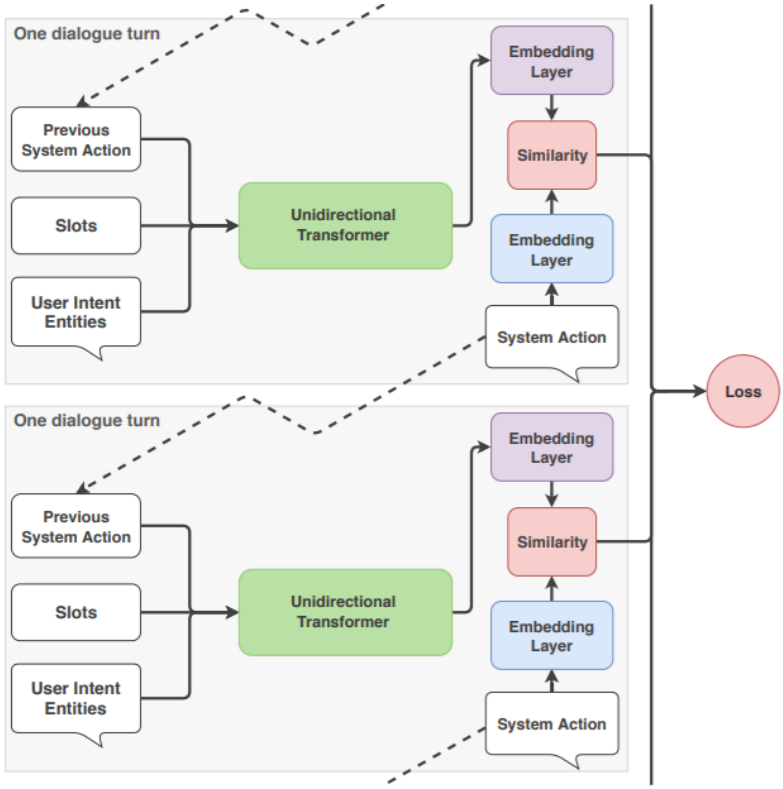

Figura 3.16: Uma representação esquemática de duas etapas de tempo da política TED. Autor: [92].

Duas etapas de tempo (ou seja, turnos de diálogo) da política TED são ilustradas na Figura 3.16. Uma etapa consiste em várias partes principais [92]:

- ***Featurization***

Em primeiro lugar, a política caracteriza a entrada de utilizador, ações do sistema e *slots*.

A política TED pode ser usada de ponta a ponta ou de forma modular. A abordagem modular é semelhante àquela adotada em políticas de diálogo baseadas em POMDP [94] ou Redes de Código Híbrido [82], [95]. Um sistema externo de compreensão de linguagem natural é utilizado e a entrada de utilizador é caracterizada como um vetor binário indicando

a intenção reconhecida e as entidades detetadas. A política de diálogo prevê uma ação a partir de uma lista fixa de ações do sistema. As ações do sistema são caracterizadas como vetores binários que representam o nome da ação, seguindo a abordagem REDP explicada em detalhes em [83].

Segundo eles, não há supervisão além da sequência de enunciados. Ou seja, não há rótulos dourados para a saída de NLU ou os nomes de ação do sistema. A política TED ponta a ponta ainda é um modelo de recuperação e não gera novas respostas. Na configuração ponta a ponta, as expressões de utilizador e do sistema são codificadas como vetores de pacote de palavras.

Os *slots* são sempre caracterizados como vetores binários, indicando sua presença, ausência ou que o valor não é importante para o utilizador, a cada etapa do diálogo. Utilizaram um método de rastreamento de *slot* simples, sobrescrevendo cada *slot* com o valor especificado mais recentemente.

- **Transformador**

A entrada para o transformador é a sequência de entradas de utilizador e ações do sistema. Portanto, aproveitaram mecanismo de autotenção presente no transformador para acessar diferentes partes da história do diálogo de forma dinâmica em cada turno do diálogo. A relevância dos turnos de diálogo anteriores é aprendida a partir de dados e calculada novamente a cada turno do diálogo. Crucialmente, isso permite que a política de diálogo leve em consideração a expressão de utilizador em um turno, mas a ignore completamente em outra.

- **Semelhança**

Todos os contextos de diálogo e ações do sistema são incorporados em um único espaço vetorial semântico e utilizaram a perda de produto escalar [77], [83], [96] para maximizar a similaridade com o rótulo alvo e minimizar similaridades com os incorretos amostrados. Aplicamos uma perda de entropia cruzada, onde os exemplos corretos são rotulados como 1 e os incorretos como 0. Uma vez que *softmax* não pode atribuir zeros à sua saída, usamos 1,5-*entmax* [93] na função de perda. A perda global é uma média de todas as funções de perda de todas as etapas de tempo. No momento da inferência, a similaridade do

produto escalar serve como um classificador para o próximo problema de recuperação de enunciado.

No treinamento modular, utilizaram uma estratégia de lote balanceado. O lote balanceado foi introduzido no *Rasa* 1.3. para mitigar o desequilíbrio de classes, pois algumas ações do sistema são muito mais frequentes do que outras.

3.2 PLATAFORMAS E FRAMEWORKS PARA CONSTRUÇÃO DE CHATBOTS

Há, atualmente, várias ferramentas, plataformas e *frameworks* para a criação de *chatbots* disponíveis no mercado. Elas variam em alguns fatores, como por exemplo, preço, funcionalidades, algoritmos, facilidade de extensão e integração com outros serviços e utilização de interface gráfica ou somente de código para o desenvolvimento do *bot*.

As quatro principais empresas de tecnologia, ou seja, *Google*, *Facebook*, *Microsoft* e *Amazon* estão todas representadas no espaço de *chatbots* e *frameworks* de NLP, já que algumas das plataformas mais populares, como *Dialogflow*, *Wit.ai*, *LUIS* e *LEX NLU* são de propriedade dessas empresas respetivamente.

A seleção da estrutura do *bot* é altamente dependente da arquitetura e vice-versa, pois as diferentes estruturas implicam em abordagens diferentes para o desenvolvimento. Como é usual no desenvolvimento de software atualmente, não é apenas uma decisão por um *framework*, mas por uma pilha inteira.

Três ofertas foram consideradas para avaliação. *Google Dialogflow*[11] e *Microsoft LUIS*[12] são serviços em nuvem e, portanto, não requerem nenhuma infraestrutura ou instalação. No entanto, eles não têm a possibilidade de inspecionar ou adaptar o código-fonte. O *Dialogflow* contém um NLU e um gerenciador de diálogo. LUIS é simplesmente um NLU, portanto, não contém um gerenciador de diálogo. No entanto, a *Microsoft* oferece o *Azure Bot Service*[13], onde um gerenciador de diálogo pode ser desenvolvido utilizando um SDK. A terceira oferta que consideramos para avaliação é *Rasa*, um projeto de código aberto popular, que oferece um *pipeline* de NLU e um gerenciador de diálogo chamado *Rasa Core*. O *Rasa*

[11] https://dialogflow.com/

[12] https://www.luis.ai/home

[13] https://azure.microsoft.com/en-us/services/bot-service/

NLU tem um *pipeline* do *Tensorflow* configurável que permite aos desenvolvedores personalizar os parâmetros do modelo para um determinado domínio.

Todos os gerenciadores de diálogo listados acima suportam diálogos não lineares de várias voltas de várias maneiras. No *Rasa Core* e no *Dialogflow*, esses gerenciadores de diálogo seguem uma abordagem baseada no aprendizado de máquina e precisam ser treinados. Utilizando o SDK fornecido pelo serviço de *bot* do *Microsoft Azure*, o suporte ao diálogo *multi-turn* deve ser desenvolvido manualmente.

Com base na pesquisa de Braun, Mendez e Matthes [97], que examinou e avaliou os serviços NLU em inglês, o *Rasa NLU* teve uma boa pontuação em comparação com seus concorrentes *LUIS*, *Watson Conversation*[14], *API.ai*, *wit.ai*[15] e *Amazon Lex*[16] e ficou em segundo lugar. Os produtos foram avaliados quanto à funcionalidade básica, com base em dois corpora de dados distintos, como ilustra a figura 3.16.

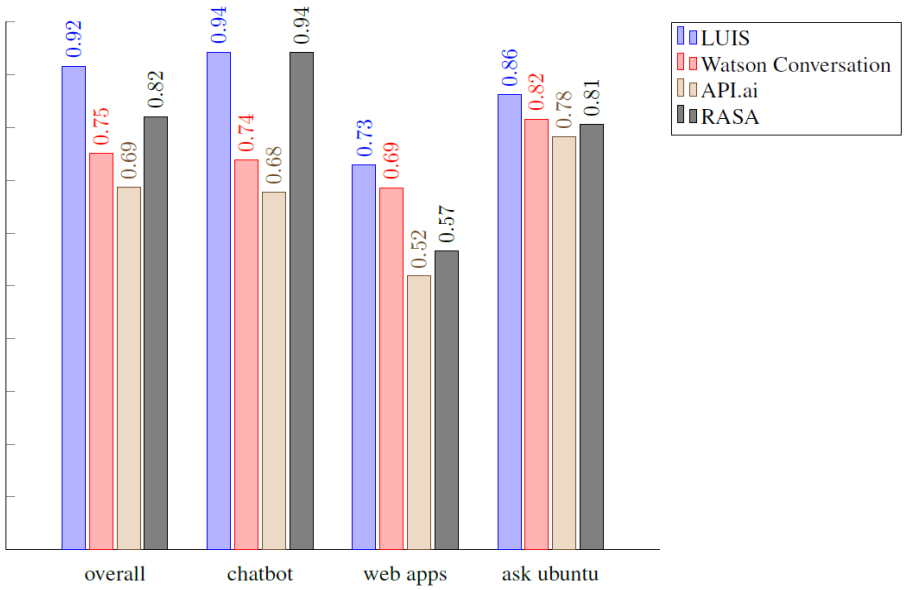

Figura 3.17: *F-scores* para os diferentes serviços NLU, agrupados por corpus. Fonte: [97]

Para os dois corpora utilizados, *LUIS* mostrou os melhores resultados, no entanto, a alternativa de código aberto *RASA* poderia alcançar resultados semelhantes. Dadas as

[14] https://www.ibm.com/watson

[15] https://wit.ai/

[16] https://aws.amazon.com/lex/

vantagens das soluções *open source* (principalmente adaptabilidade), pode muito bem ser possível obter resultados ainda melhores com *RASA*, após alguma personalização [97].

Como *LUIS* e *RASA* tiveram um desempenho bom, consideramos critérios adicionais para a avaliação, que podem ser vistos na Tabela 3.2.

Tabela 3.2: Critérios adicionais para a avaliação NLU.

CRITERIO	**LUIS**	**RASA**
Source availability	Closed Source	**Open Source**
Cost	**None (within limits)**	Hosting cost
Rate limit	5 transactions per second	**No limit**
Data processing location	USA/EU	**CH**
Installation	**None**	Needs installation
Customize pipeline	Not possible	**Possible**
Multiple models	**Yes**	**Yes**
Web GUI	**Yes**	Third party project

Um projeto de código aberto tem grandes vantagens em comparação com o software de código fechado. Mesmo que seja necessário investir tempo na configuração de ambientes, a possibilidade de inspecionar, solucionar problemas e personalizar os *pipelines* favorece uma abordagem de código aberto. Por essas razões, decidimos contra o *Microsoft LUIS* e selecionamos o *NLU da Rasa* e o gerenciador de diálogo para este projeto.

De acordo com Raj [98], "*Rasa NLU* não é qualquer outra biblioteca com um monte de métodos", ele também dá aos desenvolvedores a capacidade de desenvolver quase qualquer tipo de *chatbots* com ele.

3.2.1 RASA FRAMEWORK

O *Rasa*[17] é um *framework* em *Python* de IA conversacional de código aberto. O robô criado com o *Rasa* fornece suporte a qualquer linguagem e fácil integração a diversas

[17] http://rasa.com/docs/getting-started/

plataformas de *chat*, como *Facebook*, *Slack* e *Telegram*, além de uma documentação bem informativa e detalhada.

O Rasa possui dois componentes principais, *Rasa NLU* e *Rasa Core*. O *Rasa NLU* é responsável pelo entendimento da linguagem natural, que identifica as intenções e extrai as entidades da frase de entrada. Enquanto que o trabalho do *Rasa Core* é essencialmente gerar a mensagem de resposta para o *chatbot*. Este obtém a saída do *Rasa NLU* (intenção e entidades), utiliza as informações do ficheiro de configuração, do ficheiro de domínio e das histórias de exemplo, e aplica os modelos de Aprendizagem Automática para gerar uma resposta.

Outra funcionalidade bastante útil é o modo de aprendizagem interativa. Nele o programador fornece *feedback* ao seu *bot* enquanto fala com ele. Trata-se de uma maneira poderosa de explorar o que o *bot* pode fazer e uma maneira fácil de corrigir qualquer erro que ele cometa. É possível literalmente ensinar o *bot* a como reagir a cada situação.

O *Rasa* foi entendido como a melhor opção para o desenvolvimento deste projeto. Em virtude de ser um *framework* gratuito, possuir boa documentação, uma organização onde as informações e funcionalidades conseguem ser bem distinguidas, uma alta capacidade de customização, e efetiva participação no aprendizado do *bot*, fizeram com que esta fosse a escolha mais apropriada.

Recentemente, lançaram ainda o *Rasa X*, baseado na necessidade de levar aos seus utilizadores mais do que algoritmos, os criadores do *framework* desenvolveram uma ferramenta poderosa para coletar e anotar dados de treinamento [99]. Trata-se de uma ferramenta para visualizar e filtrar conversas entre humanos e seu assistente *Rasa*, para transformar essas conversas em dados de treinamento, para gestão e versionamento de modelos e para facilitar o acesso de utilizadores de teste a seus assistentes.

Maior detalhamento sobre o funcionamento da biblioteca, assim como de seus componentes, que serão comentadas a seguir, podem ser encontradas no Blog Oficial[18] e na documentação[19].

[18] https://blog.rasa.com/

[19] https://rasa.com/docs/

3.2.1.1 RASA NLU

O *Rasa NLU* é utilizado para extrair significado da entrada de texto. Recebe informações de utilizador na forma de linguagem humana não estruturada e extrai dados estruturados na forma de intenções e entidades.

Intenções podem ser entendidas como rótulos para cada entrada de utilizador que representam o objetivo ou o significado de suas mensagens. Por exemplo, um utilizador que dá como entrada "Olá", pode possuir uma intenção de saudação, porque o objetivo dessa entrada é cumprimentar.

Entidades são partes de uma informação que o assistente pode precisar em certo contexto. Por exemplo, na entrada "Olá! Meu nome é Emanuel.", é dito um nome. O assistente deve ser capaz de fracionar o nome em uma entidade e lembrar disso durante a conversa, para fazer com que a interação seja natural.

O treinamento de um modelo NLU nos dados de treinamento permite que o modelo faça previsões sobre as intenções e entidades em novas mensagens de utilizador, mesmo quando a mensagem não corresponde a nenhum dos exemplos que o modelo já viu antes. Para designar o assistente a entender as intenções e extrair as entidades que são definidas nos ficheiros de dados de treinamento é necessário se construir um modelo NLU, que é criado a partir de um *pipeline* de treinamento, também conhecido como *pipeline* de processamento.

3.2.1.1.1 CADEIA DE PROCESSAMENTO

Um cadeia de processamento (*pipeline)* pode ser entendido como uma sequência de etapas de processamento usadas para extrair recursos de textos específicos e treinar determinados componentes que permitem ao modelo aprender os padrões subjacentes dos exemplos fornecidos.

No Rasa, há a possibilidade de criar um *pipeline* personalizado ou utilizar um dos pré-configurados. Existem dois *pipelines* pré-configurados a escolha, um deles é chamado *pretrained_embeddings_spacy*. Esse *pipeline* usa uma biblioteca chamada *Spacy*[20], que carrega modelos de linguagem pré-treinados para representar cada palavra em uma frase.

[20] SpaCy é uma biblioteca de software de código aberto para processamento avançado de linguagem natural, escrita nas linguagens de programação Python e Cython.

Trata-se de uma representação vetorial das palavras, o que significa que cada palavra em uma mensagem de utilizador é convertida em um vetor numérico denso.

Os vetores capturam o aspeto semântico e sintático das palavras, o que significa que palavras semelhantes devem ser representadas por vetores semelhantes. Os aspetos semânticos e sintáticos conhecidos das palavras aumentarão o desempenho dos modelos, mesmo que haja uma amostra de dados de treinamento muito pequena. Além disso, já que o treinamento não começa completamente do zero, o treinamento dos modelos costuma ser rápido, com tempos de iteração curtos.

Há algumas deficiências nesse tipo de configuração. Primeiro, as boas combinações de palavras não estão disponíveis para todos os idiomas, porque geralmente são treinadas em conjuntos de dados disponíveis ao público, em sua maioria em inglês, o que pode ser um fator limitante. Outro ponto é que as combinações de palavras geralmente são treinadas a partir de conjuntos de dados de treinamento bastante genéricos, por exemplo, artigos da *Wikipedia* e similares, o que significa que eles não cobrem palavras específicas de um domínio como acrônimos, siglas e etc.

Em situações em que seja necessário solucionar esses problemas, é recomendado utilizar o segundo tipo de *pipeline* pré-configurado, chamado de *supervised_embeddings*. A principal diferença desse *pipeline* para o anterior é que, em vez de utilizar um vocabulário pronto, este *pipeline* aprende tudo do zero, usando os exemplos que forem fornecidos no ficheiro de dados de treinamento, da forma totalmente personalizada. Como não estaria usando nenhum conhecimento pré-aprendido, naturalmente, é preciso de mais exemplos de treinamento para que os modelos realmente aprendam e comecem a generalizar as entradas de utilizador.

Para treinar um bom modelo com o *pipeline supervised_embeddings* serão necessários mais exemplos de treinamento do que em um *pipeline pretrained_embeddings_spacy*. Não há um número estrito para a quantidade de dados de treinamento; uma quantidade recomendada de exemplos para o uso do *pipeline supervised_embeddings* é de mil exemplos rotulados ou mais.

Na prática, uma configuração de *pipeline* de processamento é definida em um ficheiro *config.yml* do projeto. Este ficheiro é criado automaticamente no momento de criação de um novo projeto, ao executar o comando *rasa init* no *Rasa Command Line*

Interface (Rasa CLI). Ao abrir a pasta do projeto inicial há um ficheiro de exemplo de configuração, que consiste no indicador de idioma que corresponde ao idioma no qual está sendo construído o assistente e o nome do *pipeline*.

Para treinar um modelo NLU utilizando o *pipeline supervised_embeddings*, basta defini-lo no ficheiro *config.yml* e executar o comando *rasa train nlu* no Rasa CLI. Este comando treinará o modelo junto a seus dados de treinamento e o salvará em um diretório chamado *models*.

Para testar o modelo recém-treinado executando o comando *rasa shell nlu* no *Rasa CLI*, que carrega o modelo NLU treinado mais recentemente e permite que seja testado seu desempenho, conversando com o assistente na linha de comando. Enquanto estiver no modo de teste é possível testar o modelo com várias entradas e ver como o modelo se comporta, por exemplo, verificando como a saída do modelo se parece com uma mensagem “Olá”. A saída do modelo consiste na intenção mais provável que foi prevista para esta mensagem de entrada e, ao lado, a confiança. Neste exemplo, obteve como retorno o *JSON* {“name: saudacao”, “confidence: 0.8844845294952393”}. Isso significa que o modelo tem 88% de certeza de que "Olá" é uma saudação. Se houver outras intenções, mostrará também uma lista de *intent_rankings*. Esses resultados mostram a classificação de intenções para todas as outras intenções definidas nos dados de treinamento.

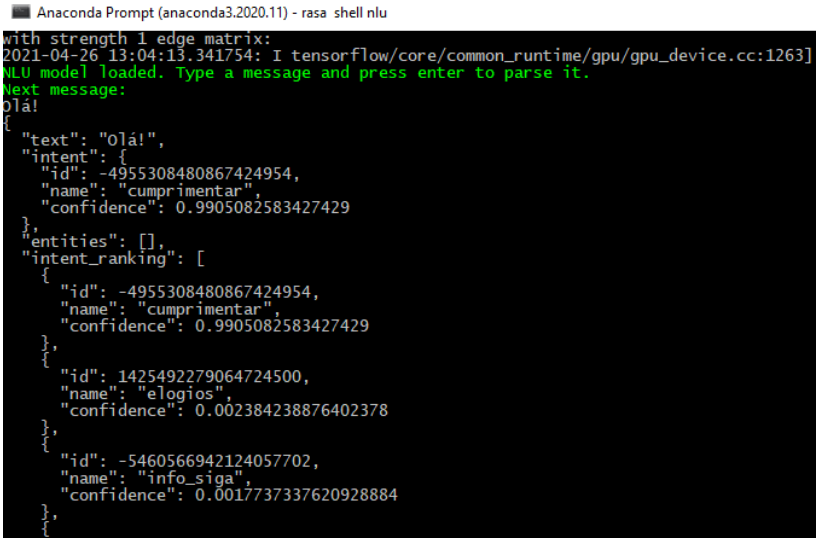

Figura 3.18: Análise das intenções previstas e suas respetivas confianças NLU no *Rasa CLI*.

A medida que o projeto cresce, é natural a necessidade de alterar aspetos específicos do modelo, e que seja preciso explorar o que compõe o *pipeline*. Os *pipelines* pré-configurados são na realidade um atalho para uma lista de diferentes componentes responsáveis por etapas específicas do processamento, como podemos visualizar abaixo:

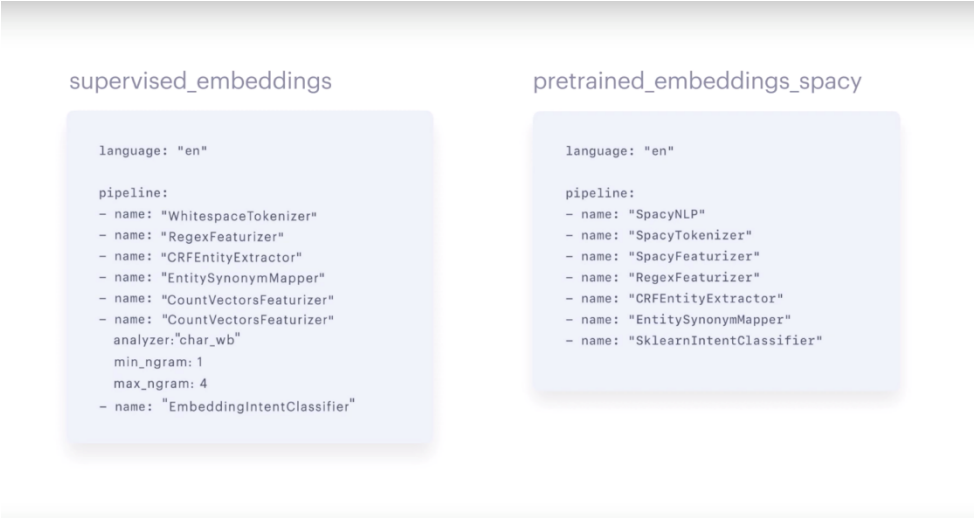

Figura 3.19: Comparando componentes para dois *pipelines* pré-configurados. Fonte: [100].

Para que seja possível a criação e configuração de *pipelines* personalizados, que melhor se adequem ao projeto, as definições dos componentes precisam estar claras. A seguir, a fim de melhor compreender porque os modelos estão se comportando de determinada maneira e como projetar os dados de treinamento para obter o melhor desempenho, entenderemos o que cada componente é, o que faz e porque é importante no processamento dos dados.

3.2.1.1.2 COMPONENTES

As mensagens recebidas são processadas por uma sequência de componentes. Esses componentes são executados um após o outro no supracitado *pipeline* de processamento, que permite personalizar o modelo e ajustá-lo ao conjunto de dados. Existem componentes para extração de entidade, para classificação de intenção, seleção de resposta, pré-processamento e outros. É permitida também a criação de novos componentes, customizados de acordo com as necessidades do projeto.

Cada componente processa a entrada e cria uma saída. A saída pode ser utilizada por qualquer componente que vem depois desse componente no *pipeline*. Existem componentes que produzem apenas informações utilizadas por outros componentes no *pipeline* e outros que produzem atributos de saída que serão retornados após o término do processamento.

Anteriormente, foi comentando sobre os dois *pipelines* pré-configurados, *supervised_embeddings* e *pretrained_embedding_spacy*. Comparando, esses dois *pipelines* consistem em um conjunto de componentes responsáveis por tarefas específicas. De recurso de processamento de texto e extração de características, até definir o atual modelo que foi utilizado para classificar as intenções e extrair as entidades. Vamos analisar os componentes e entender melhor, pois uma vez que se é entendido o que eles fazem e o porque são necessários torna-se mais fácil de se criar *pipelines* customizados e fazer as devidas mudanças nos que já existem.

- **SpacyNLP**

O *pipeline pretrained_embedding_spacy* começa com um competente chamado *SpacyNLP* que é responsável por carregar modelos de linguagem *Spacy* e ter isso pronto para os próximos passos do processo. Esse componente só é importante na utilização de modelos de linguagem *Spacy* e não é necessário para outras configurações de *pipeline*.

- **Tokenizer**

Usando combinações de palavras pré-estabelecidas ou aprendendo-as do zero, o *pipeline* deverá definir como as mensagens de utilizadores são processadas. É um passo importante para que os modelos aprendam os padrões de significado, os exemplos que estão sendo utilizados no treino devem ser divididos em menores unidades.

No *Rasa*, a frase é tratada utilizando um *tokenizer* de palavras, um componente que obtém uma entrada de utilizador, de uma simples não estruturada linguagem humana, e a transforma em pedaços pequenos, *tokens*. Um *token*, por exemplo, pode ser uma palavra. O componente utilizado por padrão no *pipeline supervised_embeddings* é o *WhitespaceTokenizer*. O espaço em branco em uma frase vai criar um *token* para cada sequência de caracter. Este é situado para processar a maioria das linguagens.

Existem alguns *tokens* que podem ser escolhidos dependendo da abordagem utilizada. Caso o assistente esteja sendo construído em uma linguagem que requer um processo de *tokenizer* mais específico, é possível escolher outro *token* disponível ou criar um personalizado. Por exemplo, para processar a linguagem chinesa pode ser utilizado o *tokenizer* chamado *Jieba*, que foi construído especificamente para processar esta linguagem. Se a escolha for utilizar *Spacy* em modelos de linguagem prontos, pode ser utilizado o *SpacyTokenizer* que vem com regras de organização para estes modelos.

Uma vez que *tokenization* é um dos primeiros passos do processo aplicado na entrada dos utilizadores, no processo de organização do *pipeline*, este deve estar entre um dos primeiros componentes. Os *tokens* criados pelo *tokenizer* são geralmente utilizados por outros componentes subsequentes no *pipeline*, como por exemplo, um classificador de intenção ou um extrator de entidade, entre outros.

Para extrair as entidades o modelo necessita de componentes para reconhecimento de nomes. Assim como todo os outros componentes, o *Rasa* fornece uma gama de opções a escolha.

- **CRFEntityExtractor**

O componente mais robusto é chamado *CRFEntityExtractor* que define um modelo, chamado *Conditional Random Field*, que aprende a identificar quais palavras em uma sentença são entidades e qual tipo de entidade elas são, observando as sequências de *tokens*. Escolhe uma palavra alvo e verifica palavras no entorno, extraindo características de texto de todas essas palavras ao redor. Um componente CRF produz a saída que é diretamente adicionada no conjunto de saída do NLU *model*, as palavras em uma sentença que forem entidades e quais são suas etiquetas, quão confiante o modelo foi de fazer essas previsões e qual modelo foi utilizado.

Se houver uma pequena quantidade de dados de treinamento e estiver sendo utilizado modelos de linguagem *Spacy*, utilizando combinações de palavras pré-treinadas, para melhorar a performance pode ser utilizado o *SpacyEntityExtractor*, que pode aproveitar parte da marcação de fala e outros recursos para localizar as entidades nos exemplos de treinamento.

- **DucklingHttpExtractor**

Algumas entidades podem seguir padrões bastante específicos, isso se aplica para entidades como data, números, códigos postais, entre outros. Há a opção de habilitar os modelos para extrair tal entidade usando componentes como o *DucklingHttpExtractor*. Uma biblioteca designada especificamente para extrair informações como datas, números, códigos postais, números de telefone, e-mails e outras informações sem treinar um modelo de extração de entidade do zero.

- **RegexFeaturizer**

Para casos avançados de extração de entidades onde modelos baseados em *Machine Learning* não são o suficiente, é possível incrementar a extração de entidade do *CRFEntityExtractor* utilizando expressões regulares ou tabelas de pesquisa.

Expressões regulares combinam padrões de escrita difíceis, por exemplo, como pode ser feito o reconhecimento de códigos postais de 8 dígitos. Tabelas de pesquisa são usadas quando as entidades forem pré-definidas como um conjunto de valores, por exemplo, caso um país seja uma entidade, este pode ter 195 valores pré-definidos.

Para usar expressões regulares e tabelas de pesquisa é necessário adicionar o componente *RegexFeaturizer* antes do componente *CRFEntityExtractor* no *pipeline*. Expressões regulares e a tabela de pesquisa são adicionadas aos recursos do *CRFEntityExtractor*, que passa a marcar as palavras em que suas combinações resultem no conteúdo esperado pelas expressões regulares ou da tabela de pesquisa.

- **Featurizer**

Para ativar os modelos para classificarem as intenções inicialmente são necessários dois componentes principais, um *featurizer* e um real modelo de classificação. O *featurizer* é utilizado para criar novas *features* (atributos ou variáveis de entrada) a partir dos *tokens* que podem ser usados pelo modelo de classificação para aprender padrões subjacentes e fazer as previsões.

Se não for utilizado um *pipeline pretrained_embeddings_spacy*, então um componente chamado *CountVectorsFeaturizer* é uma boa escolha para *featurizer* como

componente. Este *featurizer* cria uma representação de *bag-of-words* da mensagem de um utilizador utilizando o *CountVectorizer* do *Sklearn*[21]. O modelo *bag-of-words* desconsidera a ordem das palavras em um corpo de texto e, em vez disso, concentra-se no número de vezes que as palavras aparecem. Assim, o *CountVectorizer* conta quanta vezes palavras distintas aparecem nas mensagens e as utiliza como entrada para o modelo de classificação de intenção.

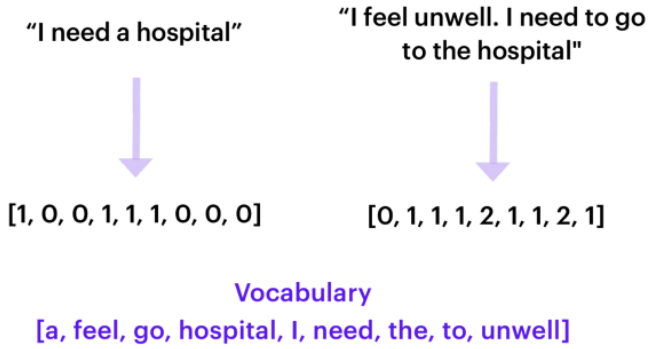

Figura 3.20: Exemplo de representação de frases com *bag-of-words*. Fonte: [100]

Em algumas situações o modelo vai funcionar melhor se em vez de contar palavras utilizar *n-grams*. Um *n-grams* é uma sequência de n itens em dados de texto, em que n representa as unidades linguísticas utilizadas para dividir os dados, por exemplo, por caracteres, sílabas ou palavras. Utilizar *n-grams* pode fazer com que o modelo seja mais robusto contra erros de digitação, mas ao mesmo tempo incrementa o tempo de treinamento porque o vetor de *features* fica maior. Este fator pode ser especificado customizando o parâmetro *analyser* na configuração do componente, possuindo as opções 'word', 'char' ou 'char_wb', o último cria caracteres *n-grams* somente a partir do texto dentro dos limites da palavra. Para este tipo de *featurizer*, um componente chamado *EmbeddingIntentClassifier* é a melhor opção para a classificação das intenções.

Se houver o uso de combinações de palavras pré-treinadas, o *SpacyFeaturizer* é o componente *featurizer* recomendado. Ele retorna vetores de palavras *SpaCy* para cada *token* que é passado ao *SklearnIntentClassifier* para a classificação de intenção.

[21] Scikit-learn é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python.

- **Classificador de intenção**

As *features* criadas pelo *CountVectorizer* alimentará o modelo de classificação de intenção e os resultados serão produzidos. O *EmbeddingIntentClassifier* funciona alimentado por entradas de mensagens de utilizador e rótulos de intenção dos dados de treinamento em duas redes neuronal separadas. As similaridades de cosseno entre o processamento das mensagens de entrada e os rótulos de intenção incorporados são calculadas. É feita a maximização das semelhanças com o rótulo de destino e a minimização entre as semelhanças com os incorretos. Os resultados são previsões de intenção que são expressas na saída final do modelo NLU.

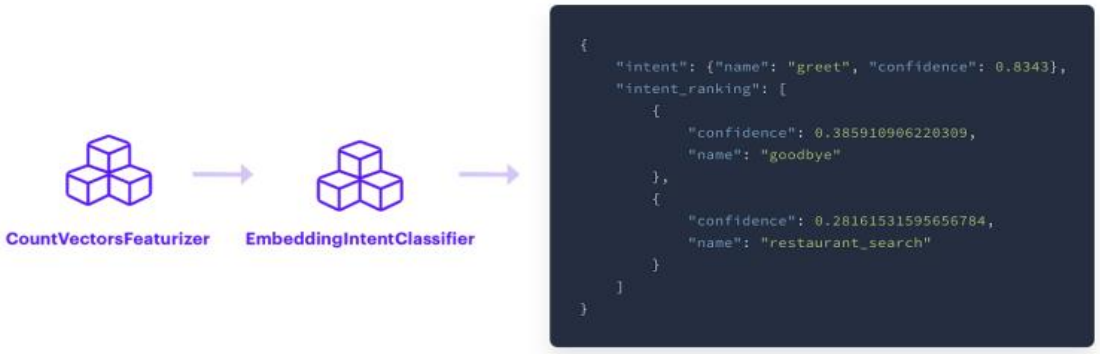

Figura 3.21: Representação do processamento feito com o *EmbeddingIntentClassifier*.
Fonte: [100]

Ao utilizar combinações de palavras pré-treinadas, a recomendação é a de utilizar o componente *SklearnIntentClassifier* para classificação de intenção. Este componente usa os recursos extraídos pelo *SpacyFeaturizer*, bem como combinações de palavras pré-treinadas para treinar um modelo chamado SVM (*Support Vector Machine*). O modelo SVM prevê a intenção da entrada de utilizador com base nos recursos de texto observados. A saída é um objeto que mostra a principal intenção classificada e uma matriz listando as classificações de outras possíveis intenções.

Como os componentes de classificação de intenção aprendem a partir das *features* extraídas do *featurizer*, as *features* devem sempre vir antes do classificador como componente na configuração *pipeline*. Ao mesmo tempo, uma vez que as *features* usam os símbolos produzidos pelo *tokenizer*, um *tokenizer* deve vir antes do *featurizer*. Os componentes de classificação produzem os resultados que são diretamente adicionados às

saídas do modelo NLU. Isso significa que o componente de classificação de intenção pode ser especificado no final do *pipeline* ou em qualquer lugar, contanto que os componentes responsáveis pelos processos de *featurizer* e *tokenizer* venham primeiro.

Construir processos customizados de *pipelines* e escolher componentes é um processo que geralmente leva a alguns passos para frente e para trás quando se constrói modelos. O que pode necessitar de atenção e serem necessárias as realizações de mudanças a medida que o sistema crescer. O treinamento também é uma parte crucial quando se quer construir bons modelos. Quanto mais treinamento puder ter e mais puder ser aprendido, melhor o modelo poderá ser, o que tem uma influência direta sobre quais componentes devem ser incluídos no *pipeline*.

3.2.1.2 RASA CORE

Foi discutido primeiramente como a ferramenta consegue habilitar o assistente para entender entradas de utilizadores, por meio de modelos NLU. Outro fator importante está relacionado a gestão de diálogo e como habilitar o assistente a responder e dirigir a conversa, mantendo o contexto.

Atualmente, a abordagem mais popular para construir uma conversa com assistente na indústria é utilizando um conjunto de regras ou um uma máquina de estados. Essa abordagem consegue lidar bem com o caminho feliz, situações onde os utilizadores perfeitamente seguem a conversa predefinida por um assistente. Entretanto, as coisas ficam confusas uma vez que utilizadores desviam do caminho. Acontece que é impossível escrever regras para qualquer diálogo possível. E uma vez que os utilizadores ficam à deriva do caminho feliz, o que cedo ou mais tarde vai acontecer com utilizadores reais, ocorrem quebras na conversa, o que vai resultar em bastante desapontamento na experiência do utilizador. Além disso, assistentes baseados em regras são difíceis de trabalhar, pois quando ficam grandes por causa do longo conjunto de regras, fica difícil de gerenciar e manter.

Rasa tomou diferentes abordagens no gerenciamento de diálogo. Antes de criar regras e impor elas aos utilizadores, se ensina a máquina a aprender padrões de conversa de um dado conjunto de exemplos de conversa e prevê como o assistente deve responder em específicas situações baseado no contexto da história da conversa e em outros detalhes. O componente responsável pelo gerenciamento de diálogo no *Rasa* é chamado *Rasa Core*.

O *Rasa Core* usa Aprendizagem Automática para captar padrões de conversação em conversas de exemplo. Com base nesses padrões, o assistente pode generalizar, permitindo que o modelo preveja a próxima melhor ação a ser realizada em uma conversa. Isso permite que o modelo forneça uma resposta apropriada, mesmo quando a conversa não corresponder exatamente a nenhum dos exemplos de treinamento vistos antes.

É importante ressaltar que isso significa que não é preciso programar todas as conversas possíveis para o assistente com antecedência. É possível fornecer dados de treinamento contendo alguns exemplos de conversa quando criar o assistente pela primeira vez e, em seguida, reunir novos dados de conversa diretamente de interações reais do usuário. Usando o Aprendizagem Automática, o assistente pode melhorar com o tempo, com base no que os utilizadores estão realmente dizendo.

3.2.1.2.1 STORY

No gerenciamento de diálogo os dados de treinamento são chamados de histórias (em inglês, *stories*), exemplos de conversas entre utilizadores e assistentes escritas em um determinado formato. Esse formato utiliza a entrada de utilizador, expressa como a intenção correspondente enquanto que as respostas dos assistentes são expressas como nomes de ações.

Abaixo, um exemplo de história, que começa com duplo hashtag que marca o nome da história. Não é obrigatório escrever os nomes das histórias, entretanto, recomenda-se providenciar nomes de histórias descritivas, alguma coisa que descreva um exemplo do que a conversa é. O final de uma história é marcado com um novo nome, e então uma nova história começa com uma *hashtag* dupla.

```
## greet + location/price + cuisine + num people    <!-- name of the story - just for debugging -->
* greet
   - action_ask_howcanhelp
* inform{"location": "rome", "price": "cheap"}  <!-- user utterance, in format intent{entities} -->
   - action_on_it
   - action_ask_cuisine
* inform{"cuisine": "spanish"}
   - action_ask_numpeople        <!-- action that the bot should execute -->
* inform{"people": "six"}
   - action_ack_dosearch
```

Figura 3.22: Exemplo de um diálogo no formato de história Rasa. Fonte [101]

As entradas de utilizadores são expressas como intenções rotuladas igualmente definidas nos dados de treinamento e dentro das chaves a entidade extraída deve ser definida provisionando o nome e o valor da entidade. As histórias de treinamento não têm que incluir a real mensagem do utilizador. Com isso, torna-se possível alavancar os resultados do modelo NLU, por meio de intenções e entidades generalizadas.

As respostas do assistente são expressas como nomes de ações. São dispostas como linhas começando com um hífen, contendo o nome da ação. Há dois tipos de ação que podem ser utilizadas com o *Rasa*: expressões e ações customizadas. Expressões são cadeias de texto codificadas que representam o que o assistente irá dizer ao utilizador. Dentro de uma história, uma expressão é rotulada com o prefixo 'utter_'. Ações personalizadas executam código personalizado, como busca de dados de uma API. Nas histórias, ações personalizadas são rotuladas com o prefixo 'action_'.

3.2.1.2.2 DOMAIN

Um domínio é uma parte essencial para começar a construir um modelo de gerenciamento de conversa com o *Rasa*. Nele são definidos os universos em que o assistente vai operar: quais intenções e entidades vão estar disponíveis para aprender, quais ações customizadas e expressões o assistente deve responder e quais informações o assistente deve lembrar durante a conversa. Um domínio é geralmente especificado em um ficheiro onde temos 3 principais seções: *Intents*, *Responses* e *Actions*. Abaixo um exemplo automaticamente criado num projeto inicial.

Figura 3.23: Arquivo *domain.yml* de um projeto inicial criado pelo Rasa. Fonte: [102]

A seção chamada *Intents* define uma lista de intenções que o assistente é capaz de entender. Estes detalhes, devem vir do modelo NLU, logo, nessa seção devem ser fornecidos todos os rótulos de todas as intenções que existem no modelo. As entidades também podem ter influência em como o assistente vai decidir como responder as entradas dos usuários, por isso que as entidades também devem ser incluídas no ficheiro de domínio, caso existam. Para fazer isso é preciso criar uma sessão chamada *Entitys* e uma lista de todas as entidades que existem no modelo que foi treinado para extrair.

A seção chamada *Actions* deve conter uma lista de todas as expressões e ações customizadas que o assistente deve usar para responder as entradas dos usuários. Isto deve ser preenchido com as *actions* utilizadas no ficheiro de histórias.

A terceira seção no domínio é chamada *Responses*. Nela serão definidas as respostas do assistente, utilizada para responder com específicas respostas quando determinadas expressões são previstas. Pode haver mais de um *response* para cada expressão. Isso pode ir além de uma simples mensagem, podem incluir objetos como botões, imagens customizadas, etc.

Responder os *responses* diretamente no ficheiro do domínio é o jeito mais fácil de definir qual mensagem um assistente vai mandar de volta ao utilizador quando ele fizer alguma declaração, entretanto, existe outro jeito de conseguir o mesmo resultado criando ações customizadas. Ações customizadas são respostas do assistente que incluem algum código customizado, esse código pode ser um simples texto de resposta de volta para o utilizador, algum tipo de processo, fazer uma chamada a alguma API ou conectar na base de dados, de um modo geral, algum detalhe importante ou atípico que o assistente precisa lidar. Ações customizadas são definidas em um ficheiro *actions.py*, assim como o nome da extensão sugere, será necessário escrever algum código em *Python* para a sua criação. Todas as ações customizadas devem ter nomes incluídos no ficheiro de domínio para poderem ser utilizadas.

Os ficheiros de domínios de todos os dados de treinamento e o de histórias são estritamente conectados. Não tem nenhuma regra específica para cada um e sobre qual deve ser criado primeiro, mas em geral, mudanças em um ficheiro vai resultar em alguma mudança em outras partes de outros ficheiros. No desenvolvimento de assistentes AI com *Rasa* são

normalmente utilizadas algumas iterações e a expectativa é que esses ficheiros sejam constantemente alterados.

- **Slots**

Também podem ser incluídos nos ficheiros de configuração e podem ser bem úteis para o gerenciamento de diálogos. Os *slots* podem ser utilizados para disponibilizar ao assistente como lembrar de importantes detalhes e usar eles em um certo contexto onde deve se dirigir a conversação. Como, por exemplo, conteúdos de identidades extraídas do modelo NLU, ou mesmo se a informação estiver fora, como em resultados de extrações dos dados de bases externas.

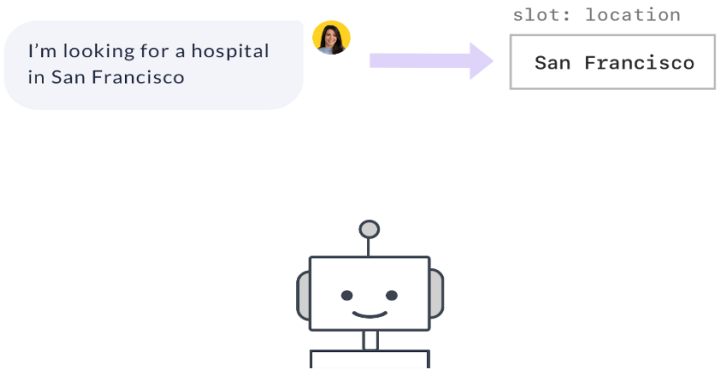

Figura 3.24: Representação do armazenamento da informação em *slot*. Fonte: [102]

Os tipos de *slots* possuem uma influência direta em como o gerenciador de diálogos faz previsões. Em algumas situações apenas a presença ou ausência dos *slots* vai importar, mas em outros tipos de *slots* o conteúdo vai importar também. A seguir comentaremos os tipos de *slots* disponíveis no Rasa:

- *Text Slot*: os textos serão criados quando o *rasa* visualizar *slots* específicos. O valor real deste *slot* não faz nenhuma diferença, apenas é importante saber se o dado foi fornecido ou não. *Slots* com tipo texto são úteis quando o diálogo deve alternar entre turnos, dependendo se os utilizadores forneceram ou não detalhes específicos. Exemplo: se o *slot location* não tiver sido definido, um assistente deve solicitar esses detalhes; caso contrário, avance e leve a conversa adiante.
- *Boolean Slots*: é utilizado quando está se lidando com detalhes, onde há duas possibilidades, ser verdadeiro ou falso. Aqui não é só a presença ou ausência

que vai importar, mas sim o conteúdo. O modelo de gerenciamento vai verificar se esse *slot* é verdadeiro e vai utilizar essa informação quando estiver fazendo as previsões para a próxima ação.

- *Categorical Slots*: para este tipo, a presença e também os conteúdos vão ser levados em conta. Os *slots* categórico são utilizados quando um pedaço da informação puder assumir de 1 a N possíveis valores. Por exemplo, baixo, médio e grande. O gerenciador de diálogos vai verificar o conteúdo do *slot* e levar as previsões até a próxima ação.
- *Float Slots*: utilizado para armazenar informações contínuas, como números de ponto flutuante. Aqui de novo a presença e o conteúdo irão importar, com os parâmetros de *min_value* e *max_value* é possível definir os valores mínimo e máximo possíveis para o *slot*. Tudo o que estiver acima de *max_value* será definido como *max_value*, enquanto tudo que estiver abaixo de *min_value* será definido como *min_value*.
- *List Slots*: se o modelo NLU extrair mais de um valor para uma entidade, convém armazenar todos os valores fornecidos. Um *slot* com a lista de tipos foi projetado para armazenar detalhes com vários valores.
- *Unfeaturized Slots*: algumas vezes pode ser útil usar os slots para histórias com algumas informações em que o assistente as use apenas em específicas ações customizadas. Usar *Unfeaturized Slots* significa que o conteúdo do *slot* não vai ter influência em como o modelo do gerenciamento de diálogos vai realizar as previsões.

- **Policy**

A política (em inglês, *policy*) no Rasa é um componente responsável por fazer a decisão de como assistente vai responder a próxima mensagem. Políticas de treinamento podem ser muito úteis em modelos de conversas mais sofisticados, uma vez que são capazes de prever a próxima ação baseada em diversos de detalhes, no histórico da conversa e no contexto.

No geral, as configurações de política consistem nos nomes das políticas e no conjunto de parâmetros que são utilizados para treinar os modelos que podem ser

configurados pelo desenvolvedor. Abaixo, alguns exemplos de políticas que podem ser utilizadas:

- *MemoizationPolicy*: essa política é uma das mais simples que estão disponíveis os históricos e são treinadas dependendo do parâmetro *max_history*. Isto tenta combinar fragmentos do histórico atual com o histórico previsto nos dados de treinamento procurando previsões nos ficheiros para as próximas ações. Esta política apenas memoriza as conversas dos seus dados de treinamento. Ela prevê a próxima ação com confiança "1.0" se essa conversa exata existir nos dados de treinamento, caso contrário, prevê "None" com confiança "0.0".
- *TEDPolicy*: É uma política de diálogo baseada em uma arquitetura transformador (ou, *transformer*). é uma arquitetura multitarefa para previsão da próxima ação e reconhecimento de entidade. A arquitetura possui vários codificadores de transformador que são compartilhados para ambas as tarefas. Uma sequência de rótulos de entidade é prevista por meio de uma camada de marcação de campo aleatório condicional (CRF) no topo da saída do codificador do transformador de sequência do utilizador correspondente à sequência de entrada de *tokens*. Para a próxima previsão de ação, a saída do codificador do transformador de diálogo e os rótulos de ação do sistema são incorporados em um único espaço vetorial semântico.
- *FallbackPolicy*: essa política lida com o gerenciamento de conversas com inteligência artificial, em situações onde é impossível prever as situações que os utilizadores vão perguntar, por isso, o assistente não é designado para realmente fazer alguma coisa quando os utilizadores perguntarem. Essa política permite que sejam respondidas frases como "Desculpa, não compreendi", ou "Eu não entendi", ou qualquer outra a escolha, que o assistente poderia dizer nesse tipo de situação.
- *RulePolicy*: é uma política que usam lógica predefinida para selecionar uma resposta fixa e aquelas que aprendem a prever a próxima ação do assistente a partir dos dados. Isso remove a suposição de selecionar uma configuração de política e torna mais fácil aplicar a lógica de negócios.

- *Transformer Embedding Dialogue Policy* (TEDP): utiliza o *Transformer*, um modelo de *Deep Learning* que produz resultados precisos em uma variedade de conjunto de dados e também lida bem com entradas inesperadas do utilizador.

3.2.1.2.3 APRENDIZAGEM INTERATIVA

No modo de aprendizado interativa, é fornecido o *feedback* do administrador ao seu *bot* enquanto fala com ele. Essa é uma forma efetiva de explorar o que o *bot* pode fazer e a forma mais fácil de corrigir os erros que ele cometa. Uma vantagem do diálogo baseado em Aprendizagem Automática é que, quando o *bot* ainda não sabe fazer algo, ele poderá ser ensinado.

No modo interativo, o *Rasa* solicitará que seja confirmada todas as previsões feitas pelo NLU e pelo *Core* antes de continuar. O histórico do bate-papo e os valores das intenções previstas são impressas na tela, onde deve conter todas as informações necessárias para decidir qual será a próxima ação correta. Ao final, é permitido ao administrador do *bot* salvar as novas anotações, dados de treinamento e histórias criadas durante a aprendizagem interativa, adicionando de forma automática os dados obtidos aos ficheiros com os dados de treinamento iniciais.

3.2.1.3 RASA X

No desenvolvimento de assistentes de conversas com AI, um dos maiores desafios é ter um bom dado de treinamento. Tudo começa com o desenvolvedor manual que gera alguns exemplos de conversas para iniciar o processo de desenvolvimento. Porém, é muito difícil ter uma gama de variedade nos exemplos de treinamento NLU.

Quando isso chegar na generalização de dados de treinamento, humanos subconscientemente tendem a ter padrões específicos. Conversas hipotéticas não refletem realmente o mundo real, isso pode diferenciar drasticamente os utilizadores reais e como esses utilizadores querem conversar com seus assistentes. É por isso que utilizadores reais devem ser envolvidos no processo de desenvolvimento o mais cedo possível, porque as conversas que eles têm com seu assistente é o melhor treinamento de dados possível para alcançar a aprovação do assistente.

Ter alguns *feedbacks* de utilizadores reais vai ajudar a levar o assistente para o outro nível. Melhorar o modelo para um dado do mundo real adicionando novas habilidades ou introduzindo algumas mudanças para as que já existem. Um jeito de alcançar tudo isso é introduzir *Rasa X* no desenvolvimento.

O *Rasa X* é uma ferramenta que permite manipular assistentes construídos com o *Rasa*. Isso permite que o assistente seja compartilhado com utilizadores reais, colete conversas que eles têm com o assistente, as reveja e decida a arquitetura que deva ser utilizada. Uma vez que o *Rasa X* se tornou uma ferramenta para alcançar assistentes existentes, o único pré-requisito para usar o *Rasa* X é que haja um assistente construído com o *Rasa*.

Após a abertura no *Web Browser* da página do *Rasa X* é disponibilizada a *feature* de controle de versão e há as seções "Talk to your bot", "Conversations", "Models" e "Training".

3.2.1.3.1 CONTROLE DE VERSÃO

Controladores de versão permitem o desenvolvimento de versões dos dados e dos códigos a fim de armazenamento, revisão de mudanças antes de irem para produção e execução de testes integrados com as mudanças propostas.

O *Rasa X* permite a conexão do assistente a plataformas que disponibilizam um controlador de versão, como o *GitHub* ou qualquer outro servidor remoto. Para isso, é necessário ter o assistente no servidor remoto com todos os dados e configurações de ficheiros, assim como se tem em um computador local. Uma vez que o assistente está conectado ao *GitHub* pode-se habilitar o controle de versão no *Rasa X*.

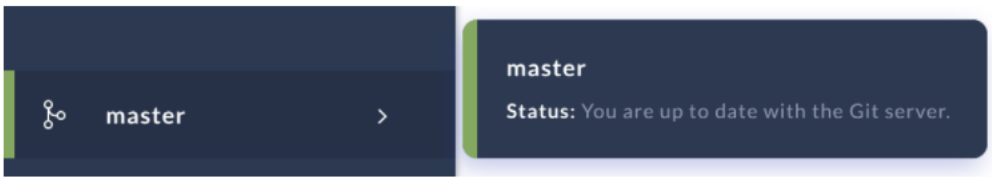

Figura 3.25: Controle de versão integrado ao Rasa X. Fonte: [103]

Uma vez que houver uma mudança e houver uma nova anotação nos dados de treinamento, se forem alteradas as configurações do modelo ou qualquer outro aspeto, o

processo de controle de versão integrada ficará laranja, sugerindo que novas mudanças foram feitas e não foram integradas ao servidor. Para incluir essas mudanças, é possível fazer o upload destas na *branch* master, ou criar uma nova *branch*. Uma vez que isso for feito, deverá ser encontrada uma solicitação de requerimento no repositório do assistente que sugere mudanças.

3.2.1.3.2 MÓDULO TALK TO YOUR BOT

Esta seção possui dois modos: *Talk* e *Interactive Learning*. O modo *Talk* refere-se propriamente a um modo de conversa com a versão atual do *bot*, em que há, a cada interação, a visualização na interface de como o *bot* identifica a intenção da frase de entrada e que ação ele executa em seguida como resposta. Apresenta também a história apresentada e os *slots* que tenham sido preenchidos.

Enquanto o modo *Interactive Learning*, é um modo de aprendizagem do assistente virtual. Nele é possível validar cada passo da conversação, como cada frase deverá ser identificada, qual deverá ser a próxima ação do *bot* e qual deverá ser a história para um determinado fluxo de conversação. Ao final, todas informações deste treinamento poderão ser salvas nos ficheiros de configuração do *bot*.

3.2.1.3.3 MÓDULO CONVERSATIONS

Depois que a interface de utilizador do *Rasa X* é iniciada, é possível visualizar as conversas realizadas na guia *Conversations*. Nesta guia, encontram-se as conversas do *bot* com quaisquer utilizadores e há a possibilidade de marcar partes das conversas para futura visualização e anotar corretamente a intenção de uma determinada frase. A aba *Conversations* também permite filtrar conversas por duração, por intenções e ações, o que pode ser útil caso haja um grande número de conversas.

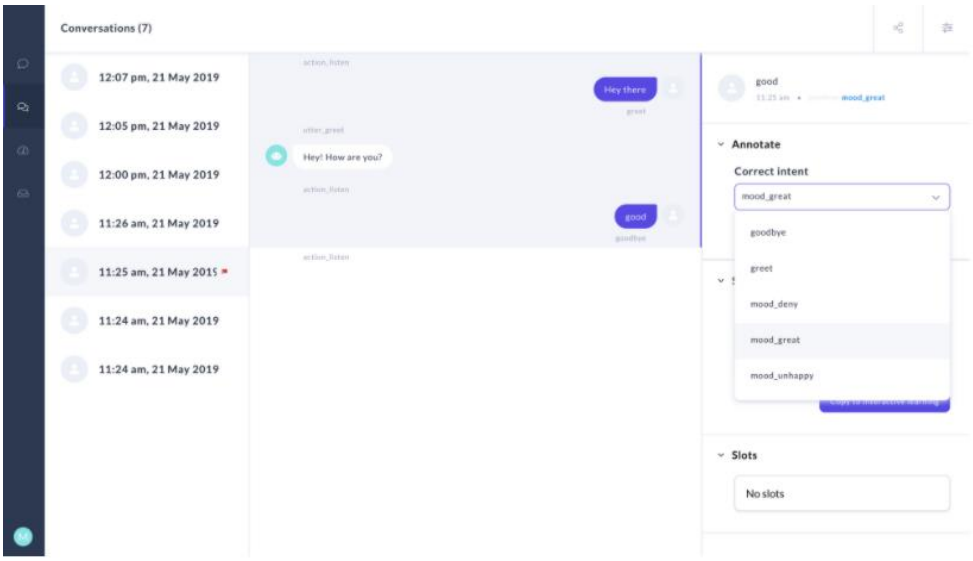

Figura 3.26: Aba conversations do Rasa X. Fonte: [104]

A sugestão do *blog* da *Rasa* [104] é que o *bot*, após a geração de uma versão básica, seja entregue a utilizadores reais o mais rápido possível, por ser uma das únicas formas do assistente ser capaz de aprender diferentes padrões de conversação. Ao clicar no botão *Share your assistant with Guest Testers*, o *bot* poderá ser testado por utilizadores externo.

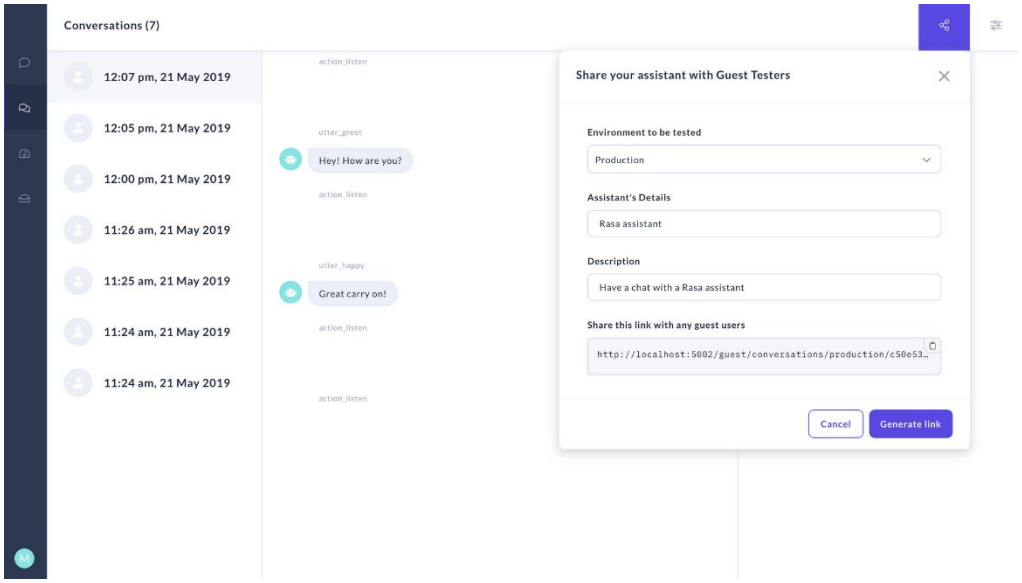

Figura 3.27: Janela de compartilhar o assistente com usuários testadores. Fonte: [104]

Ao clicar em *Generate Link* será gerada uma *url*, em que os testadores do assistente poderão conversar com ele imediatamente, sem instalar ou configurar nada, e as conversas serão exibidas na guia *Conversations*. Lá, como dito no início da sessão, pode-se anotar os dados incorretos, criar mais histórias de treinamento e usá-los para melhorar continuamente o assistente ao longo do tempo.

3.2.1.3.4 MÓDULO MODELS

Após a realização das configurações e a inserção dos dados de treinamento, com um *click* no botão *Train* é possível gerar um novo modelo. Ao finalizar o treinamento, o modelo será exibido na aba *Models*. Nesta aba é exibido o nome de cada modelo, sua data de criação e seu status. A nova versão do *bot* gerada pode começar a ser utilizada após o *click* nos três pontos à direita do modelo, e o tornando ativo.

3.2.1.3.5 MÓDULO TRAINING

Nesta seção é feita toda a configuração e treinamento do *bot*. É subdivida em 5 subseções:

- *NLU training* - É onde é feita a adição e a anotação de novos dados a serem recebidos pelo *bot*. Realiza-se a classificação da intenção do conjunto de dados de treinamento referentes ao que pode ser dito ao *bot*.
- *Responses* - Aqui são configurados os *templates* e dados de resposta que serão reproduzidos quando determinada ação do *bot* for ativada.
- *Stories* - Para a visualização, criação e comparação de histórias, trata-se de caminhos que o *bot* irá seguir durante a conversa com o utilizador. Pode ser definida a ação a ser executada a partir da identificação de uma dada intenção de entrada e possíveis desdobramentos e fluxos.
- *Configuration* - Define as regras de treinamento do modelo.
- *Domain* - Define o universo em que o assistente trabalha. Especifica as intenções, entidades, *slots*, *templates* e ações que o *bot* deve conhecer.

CAPÍTULO IV

IMPLEMENTAÇÃO DE UMBOT

No presente capítulo é apresentada a abordagem prática que permitiu o desenvolvimento do *chatbot (UMBot)*, desde a componente relativa à extração inicial de dados, que permitiu a construção do *dataset* utilizado, até à implementação prática do motor conversacional, expondo todos os passos desde o pré-processamento sobre o conjunto de dados formulado até ao desenvolvimento e treino do modelo de *Deep Learning*.

4.1 CONFIGURAÇÃO E DESENVOLVIMENTO

4.1.1 REQUISITOS

O desenvolvimento prático do sistema *UMbot* foi construído utilizando o *framework* de *machine learning Rasa*, cuja escolha se deveu maioritariamente a vantagens apresentados no capítulo 3.2, entre outras razões:

Rasa é escrito em *Python*, por ser a linguagem mais utilizada para processamento de dados textuais, fornecendo bibliotecas, que permitem a aplicação de técnicas de NLP ao nosso conjunto de dados, de forma simples e abstrata, possibilitando-nos preocupar apenas o propósito do seu uso e não lógica e metodologias que as compõe.

Rasa oferece recursos de infraestrutura, como persistência de modelo ou acesso HTTP, que são necessários em soluções de conversação no mundo real.

4.1.2 ARQUITETURA

A Figura 4.1 mostra a arquitetura do *chatbot* construído utilizando o *framework Rasa*. O *chatbot* foi construído utilizando o *framework Rasa* de código aberto, que consiste em dois componentes principais: *Rasa Core* e *Rasa* NLU. *Rasa Core* é o mecanismo de diálogo e o ficheiro de domínio define as configurações do *chatbot*, como o que ele deve entender, o que poderia usar para responder e assim por diante. O *Rasa* NLU adquiriu dados de treino em NLU e exemplos de conversas para treinar o *chatbot*, fazendo autónomamente a classificação de *intent* e a extração de entidade. A arquitetura do fluxo de conversação e

outras funcionalidades, como armazenamento e busca de informações externamente de um *dataframe*, foram definidos no ficheiro de ação. Este *chatbot* foi implementado com a ajuda da ferramenta *Rasa* X, que ajudou a melhorar o *chatbot*, à medida que ele interagia com os utilizadores.

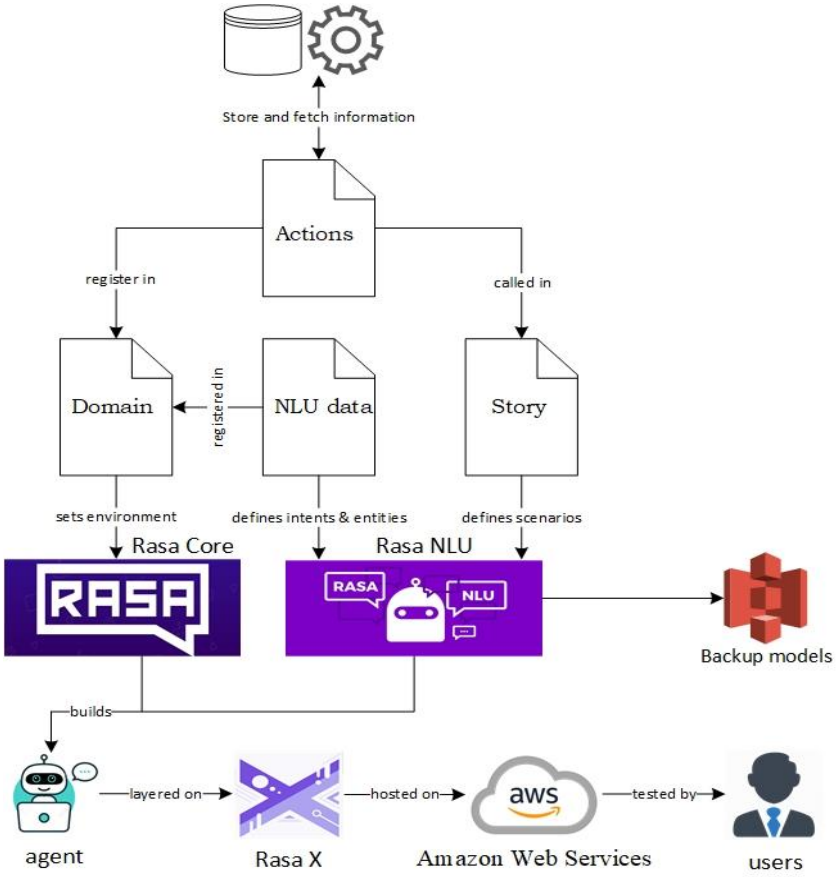

Figura 4.1: Arquitetura do *chatbot* construída usando a estrutura *Rasa*, tanto o *Rasa Core* quanto o *Rasa NLU* são utilizados para gestão de diálogo e compreensão de linguagem natural. O *chatbot* é hospedado em uma instância de Amazon Web Services para testes de utilizador.

4.1.3 Preparação do Ambiente e Criação do Projeto com o Rasa

Inicialmente, foi feita a instalação do *Rasa* numa máquina local, com sistema operacional Windows 10. A instalação do *Rasa* tem como pré-requisitos o gerenciador de pacotes *Pip* e a presença do *Python* nas versões 3.8 [105]. Foi instalado, portanto, o *Anaconda Distribution* para *Python* 3.8, que é um instalador *open souce* que reúne uma série de utilitários, como o próprio *Python* 3.8, ferramentas utilizadas por desenvolvedores *Python*,

como *Jupyter Notebook*, a IDE *Spyder*, além de famosas bibliotecas no ramo da Ciência de Dados, como *NumPy*, *Pandas* e *Scikit-learn* [106].

Após a instalação do *Anaconda*, no *Anaconda Prompt*, um *prompt* onde é possível a execução de comandos em *Python* e comandos do *Anaconda*, foi executado o comando *pip3 install rasa* para a instalação do *Rasa*. Em seguida, foi feita a criação de um projeto *Rasa* inicial, com o comando *rasa init --no-prompt*. O comando cria todos os ficheiros que um projeto *Rasa* precisa e treina um *bot* simples com alguns dados de amostra. Sem a *flag* de *--no-prompt* serão feitas algumas perguntas sobre como é desejado que o projeto seja configurado, como por exemplo, o diretório destino de extração dos ficheiros. Após a execução do comando, os seguintes ficheiros são criados:

__init__.py	um ficheiro vazio que ajuda o *python* a encontrar as ações
actions.py	código para suas ações personalizadas
config.yml	configuração dos modelos NLU e Core
credentials.yml	detalhes para conectar-se a outros serviços
data/nlu.md	os dados de treinamento NLU
data/stories.md	as histórias
domain.yml	o domínio do assistente
endpoints.yml	para definição de endpoints, URLs onde o serviço possa ser acessado por outras aplicações clientes
models/.tar.gz	o modelo inicial

4.1.4 MIGRAÇÃO DE DADOS

Para efetuar a alimentação inicial e consequente treino primário do modelo de *Deep Learning* implementado, revelou-se necessário efetuar a construção de um *dataset* do robô específico, que tivesse a capacidade de representar comportamentos conversacionais estruturados e consistentes.

A utilização direta de dados de conversas reais, sem qualquer tipo de filtro ou tratamento, revela-se normalmente, um obstáculo difícil de ultrapassar aquando do seu processamento no treino do modelo, resultando na maior parte dos casos numa má aprendizagem devido ao facto de estes conterem bastante ruído, sob a forma de erros ortográficos, informação errada, acentuação, entre outros, originando um mau desempenho na formulação de resposta por parte do sistema.

Adicionalmente a este problema, a falta de dados estruturados e que representassem conversas reais, também se revelou uma tarefa árdua de superar, não havendo nenhum *dataset* referente a conversas na língua portuguesa, disponível para utilização, pelo que a solução adotada se baseou na extração destes dados a partir das nossas próprias conversas, localizadas em redes sociais, por exemplo *Facebook*.

O Facebook fornece a possibilidade ao utilizador, deste descarregar todos os dados a si referentes, desde imagens partilhadas até às conversas efetuadas no chat.

Como se pode verificar na imagem 4.2, os dados podem ser criados em dois tipos de formatos diferentes: HTML e JSON.

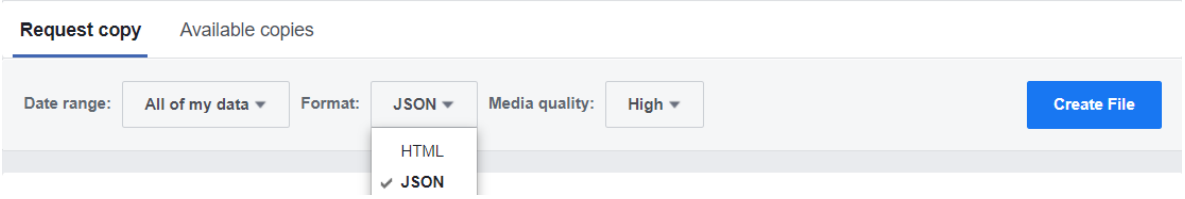

Figura 4.2: Ferramenta presente no Facebook para extração de dados relativos ao utilizador.

Entretanto, extraídos do Facebook e o *Rasa* são diferentes, na questão do formato em que as informações devem ser dispostas no ficheiro e da forma em si como as configurações são realizadas. Havia não só a necessidade de registrar possíveis perguntas e respostas no formato em que o *Rasa* identificasse, como também era demandada a criação do nome de cada intenção referente a cada pergunta, assim como a criação das histórias com as ações que o *bot* deverá executar em cada fluxo. Assim, houve a transformação dos dados exportados do Facebook, para que fossem adaptados aos requisitos do *Rasa*, de modo que ao final foram registadas 38 histórias, 31 intenções, 33 ações e 39 *responses* de respostas.

Atualmente, a documentação do *Rasa* [107] fornece a possibilidade personalizar como o *Rasa* importa os dados de treinamento. Onde é possível, por exemplo, criar um *script*

em *Python* que consiga carregar dados de treinamento, estando estes em outros formatos, ou mesmo obter dados de treinamento de diferentes origens.

4.1.5 CONFIGURAÇÕES DO RASA

Diante de uma variada gama de possíveis configurações, uma etapa que demandou bastante testes foi a definição do ficheiro de configurações. No final, mesmo com o entendimento do funcionamento dos componentes do *pipeline* e das políticas de treinamento, tornou-se necessário ir modificando aos poucos as combinações de componentes, políticas e hiperparâmetros para obtenção da configuração atual, de modo que as necessidades do projeto fossem atendidas.

4.1.5.1 DEFINIÇÃO DO PIPELINE

Primeiramente, foi pensado na escolha de um dos *pipelines* pré-configurados. O *pipeline supervised_embeddings* seria o mais adequado, uma vez que seriam construídos vetores de palavras personalizados para o domínio do projeto. O outro *pipeline*, o *pretrained_embeddings_spacy*, faz uso de modelos pré-treinados na obtenção da resposta final, o que poderia utilizar muitas palavras que não fazem parte do contexto de um FAQ acadêmico da UM, atrapalhando a previsão do modelo.

Contudo, como já comentado, um *pipeline* pré-configurado é composto por vários componentes, que não necessariamente são utilizados. Assim, em virtude de otimizar o processo de treinamento e obtenção de respostas do *bot*, foi iniciado um processo de verificar o que poderia ser removido/adicionado dessa listagem de componentes.

```
pipeline:
   - name: WhitespaceTokenizer
   - name: RegexFeaturizer
   - name: LexicalSyntacticFeaturizer
   - name: CountVectorsFeaturizer
   - name: CountVectorsFeaturizer
     analyzer: char_wb
     min_ngram: 1
     max_ngram: 4
   - name: DIETClassifier
     epochs: 100
   - name: EntitySynonymMapper
   - name: ResponseSelector
     epochs: 100
   - name: FallbackClassifier
     threshold: 0.3
     ambiguity_threshold: 0.1
```

Figura 4.3: Componentes que compõem o *supervised_embeddings_pipeline*

O primeiro componente, o *WhitespaceTokenizer*, tem como responsabilidade a criação de um *token* para cada sequência de caracteres separados por espaços em branco. Este é essencial para que os modelos consigam decompor as frases em partes menores e aprendam os padrões de significado das palavras. Este componente é *case sensitive* (diferencia letras minúsculas de maiúsculas) por padrão, e nesta implementação, entende-se que as palavras não precisam ser diferenciadas dessa forma. Portanto, além de manter esse componente, foi adicionado um atributo, *case_sensitive: false*, indicando que o processamento não diferencie letras maiúsculas de minúsculas.

Os componentes de *RegexFeaturizer* e *EntitySynonymMapper* não foram utilizados. Até o momento não foi feito nenhum uso de *regex* para deteção de algum padrão, ou de entidades que precisassem ser extraídas das frases para alguma finalidade, muito menos de sinônimos de entidades. Isso pode ser pensado e adicionado no futuro, mas no momento esses componentes não teriam utilidade.

O componente *LexicalSyntacticFeaturizer*, cria recursos para extração de entidade. Move-se com uma janela deslizante sobre cada *token* na mensagem de utilizador e cria recursos de acordo com a configuração.

O próximo componente da lista é o *CountVectorFeaturizer*, que é o *featurizer* responsável por criar novas *features* a partir dos *tokens*, para que possam ser utilizados pelo modelo de classificação, de modo a aprender padrões subjacentes e fazer as previsões. O *pipeline* usa duas instâncias de *CountVectorsFeaturizer*. O primeiro cria *features* com base em palavras. O segundo cria *features* com base em caracteres *n-grams*. O segundo *featurizer* tende a ser mais poderoso, mas é recomendado pela documentação a manutenção dos dois *featurizers* para tornar a *featurization* mais robusta.

O componente *DIETClassifier*, é uma arquitetura utilizado para classificação de intenções e reconhecimento de entidades. A arquitetura é baseada em um transformador que é compartilhado para ambas as tarefas. Uma sequência de rótulos de entidade é prevista por meio de uma camada de marcação de campo aleatório condicional (CRF) no topo da sequência de saída do transformador correspondente à sequência de entrada de *tokens*. Para os rótulos de *intent*, a saída do transformador para o enunciado completo e os rótulos de *intent* são incorporados em um único espaço vetorial semântico. Usamos a perda de produto escalar

para maximizar a semelhança com o rótulo alvo e minimizar as semelhanças com amostras negativas.

O componente *ResponseSelector* é utilizado para construir um modelo de recuperação de resposta para prever diretamente uma resposta do *bot* a partir de um conjunto de respostas candidatas. A previsão deste modelo é utlizada pelo gerenciador de diálogo para emitir as respostas previstas. Ele incorpora entradas de utilizador e rótulos de resposta no mesmo espaço e segue exatamente a mesma arquitetura de rede neuronal e otimização do *DIETClassifier*.

Por fim, o componente *FallbackClassifier* que classifica uma mensagem de utilizador com o *intent nlu_fallback* no caso de o classificador de *intent* anterior não conseguir classificar um *intent* com uma confiança maior ou igual ao limite do *FallbackClassifier*. Ele também pode prever a intenção de *fallback* no caso em que as pontuações de confiança das duas principais *intents* classificadas estão mais próximas do que o *ambiguity_threshold*. *FallbackClassifier* pode ser utilizado para implementar uma ação de *Fallback* que lida com mensagens com previsões de NLU incertas.

4.1.5.2 POLÍTICAS DE TREINAMENTO

As políticas, como já comentado, ajudam a definir qual ação tomar a cada passo da conversa. Foram utilizadas as políticas *MemoizationPolicy, TEDPolicy* e *RulePolicy*.

A *MemoizationPolicy* e a *TEDPolicy* apresentaram um desempenho mais próximo do esperado durantes os testes (quando comparada a *KerasPolicy*). A MemoizationPolicy lembra as histórias de seus dados de treinamento. Ele verifica se a conversa atual corresponde às histórias em seu ficheiro *stories.yml*. Nesse caso, ele irá prever a próxima ação a partir das histórias correspondentes de seus dados de treinamento com uma confiança de 1.0. Se nenhuma conversa correspondente for encontrada, a política prevê nenhum com confiança de 0.0.

TEDPolicy é uma arquitetura multitarefa para previsão da próxima ação e reconhecimento de entidade. A arquitetura possui vários codificadores de transformador que são compartilhados para ambas as tarefas.

Por fim, há a política *RulePolicy*, que lida com partes da conversa que seguem um comportamento fixo (por exemplo, lógica de negócios). Ele faz previsões com base em quaisquer regras que você tenha em seus dados de treinamento.

```
policies:
   - name: MemoizationPolicy
   - name: TEDPolicy
     max_history: 5
     epochs: 100
   - name: RulePolicy
```

Figura 4.4: Configuração final das políticas no ficheiro *config.yml*

4.1.5.3 DEPLOY NO SERVIDOR

Após ter realizado o *set up* inicial do projeto, o mesmo foi hospedado no *Github*. Com isso, foi iniciado o processo de *deploy* do *Rasa* X no servidor.

4.1.5.4 INSTALAÇÃO DO RASA X

Como já atrás referido, o *Rasa* X é uma ferramenta que permite o manuseio de assistentes construídos com o *Rasa*. Para o *deploy* do *Rasa* X no servidor foi utilizado o passo a passo contido na documentação, que utiliza o *Docker*. O *Docker* é uma plataforma de virtualização que faz uso de *containers* para facilitar a criação, implementação e execução de aplicações. Os *containers* permitem o empacotamento de uma aplicação com todas as suas partes necessárias, o que possibilita a execução da mesma em qualquer outra máquina, uma vez que a imagem gerada inclui todas as dependências necessárias para executar o código.

O *deploy* que foi realizado utilizando o *Docker-Compose*. O *Compose* é uma ferramenta para definir e executar aplicativos *Docker* de vários containers. Com o *Compose*, se é utilizado um ficheiro *YAML* para configurar os serviços da aplicação. Em seguida, com um único comando, é criado e iniciado todos os serviços definidos na configuração.

O recomendado é o uso de sistema operacional Linux para a configuração do *Rasa* X, mas pode-se usar outro sistema operacional se houver o suporte para *Docker*. O *Rasa* X foi instalado em um uma instância do *Amazon Web Services* (AWS). No AWS, após efetuado o login no console, foi criada uma nova *Instance* no caminho *Compute Instances* > *Running instances*.

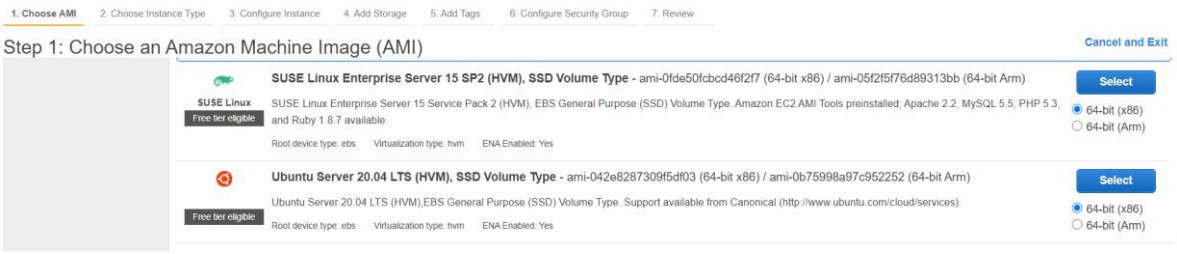

Figura 4.5: Configuração de *Instance* no AWS para a instalação do Rasa X.

Inicialmente, foi configurada a instalação de *Instance*, definindo aspetos como nome, região que será localizada, tipo de máquina e CPU que será usado. O próximo passo, foi especificar o sistema de inicialização. No AWS há uma variação de diferentes distribuições Linux. Foi utilizada a versão Ubuntu 20.04 LTS, e, nas configurações do Firewall, foram permitidos o tráfego HTTP e HTTPS.

Uma vez que a máquina foi configurada e criada, iniciou-se o processo de instalação do *Rasa* X. Foi feito download e instalado o *Rasa* X utilizando os comandos conforme imagem abaixo. O "0.35.1" refere-se a versão do Rasa X a ser instalada.

```
login as: ubuntu
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1038-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information disabled due to load higher than 2.0

 * Introducing self-healing high availability clusters in MicroK8s.
   Simple, hardened, Kubernetes for production, from RaspberryPi to DC.

     https://microk8s.io/high-availability

41 updates can be installed immediately.
27 of these updates are security updates.
To see these additional updates run: apt list --upgradable


Last login: Sun Apr  4 12:42:33 2021 from 41.221.196.181
ubuntu@ip-172-31-82-71:~$ curl -sSL -o install.sh https://storage.googleapis.com/rasa-x-releases/0.35.1/install.sh
ubuntu@ip-172-31-82-71:~$ sudo bash ./install.sh
```

Figura 4.6: Instalação do Rasa X através do Prompt de Comando

O processo de instalação geralmente pode levar alguns minutos e iniciará após o aceite dos termos e condições para utilizar o *Rasa* X. Os ficheiros vão ser instalados, por padrão, no diretório "etc/rasa/" no servidor, o que também pode ser customizado. Esses ficheiros consistem em tudo que o *Rasa* X precisa para fornecer informações aos assistentes. Por exemplo, possui ficheiros como: credenciais, o diretório de modelos, um ficheiro de

banco de dados (que será utilizado para armazenar as conversas entre utilizador e o *bot*), *logs* de *contêiner* e um ficheiro do *docker-compose*.

No diretório onde estão os ficheiros instalados, bastou executar o comando *sudo docker-compose up -d* para que os containers fossem baixados e o *Rasa* X fosse executado. Uma vez que os containers estejam em funcionamento, é necessário definir a senha de utilizador administrador com o seguinte comando *sudo python rasa_x_commands.py create --update admin me*.

Com essas configurações, o *Rasa* X fica pronto para ser utilizado, com capacidade de conexão com os assistentes e implementação de melhorias. Para abrir o *Rasa* X, conectar os assistentes e trabalhar nas melhorias deve-se abrir o servidor utilizando o DNS ou o endereço de IP.

Figura 4.7: Instância criada e seu DNS e endereço de IP.

Como já comentado anteriormente, o *Rasa* X possui um controle de versões integrado que permite conectar e controlar o assistente que está hospedado no *GitHub* ou qualquer outro servidor remoto. Este controle de versões faz com que o *Rasa* X torne muito mais fácil a conexão de assistente existente ao servidor, assim como o trabalho nas melhorias. Os requerimentos para utilizar essa funcionalidade é possuir um projeto *Rasa* com uma estrutura espelhada na estrutura *default* de um projeto criado pelo comando *rasa init* no *Rasa* CLI.

Figura 4.8: Estrutura de um projeto padrão Rasa

Uma vez organizado é necessário autenticar o servidor. Primeiro, gerar uma SSL *key* para o repositório remoto utilizando o seguinte comando: *ssh-keygen -t rsa -b 4096 -f git-deploy-key*. O comando irá gerar uma chave pública e uma privada. A partir disso, é necessário copiar a chave guardada no ficheiro *git-deploy-key.pub* (que corresponde a uma chave pública) e colar isso nas sessões de chaves de *deploy* do projeto no *GitHub*.

Após isso é preciso criar um ficheiro *repository.json* com seguintes parâmetros:

- *repository*_url: é uma SSH URL para o repositório remoto. Isso pode ser encontrado na página do repositório no *Github*;
- ssh_*key* é a chave privada que foi gerada (no ficheiro *git-deploy-key*);
- *target_branch*: é o parâmetro que vai apontar o servidor *Rasa* X para essa *branch*, que foi definido como sendo a própria master.

Deve-se atualizar todos os detalhes e salvar o ficheiro. Uma vez feito, para realizar a autenticação com o *Rasa* X deve utilizado o seguinte comando:

```
curl --request POST \
     --url https://<Rasa X server
host>/api/projects/default/git_repositories?api_token=<your
api token> \
     --header 'content-type: application/json' \
     --data-binary @repository.json
```

Para executar esse comando será necessário fornecer o *Host* URL do servidor (que neste caso vai ser o servidor IP) e o *token* da API. O jeito mais fácil de encontrar o *token* é

navegando na página de modelos *Rasa* X e clicando no botão *Upload Model*. Se a autenticação tiver sido bem-sucedida, *Rasa* X vai dispor da versão atual dos ficheiros que está no repositório de assistente.

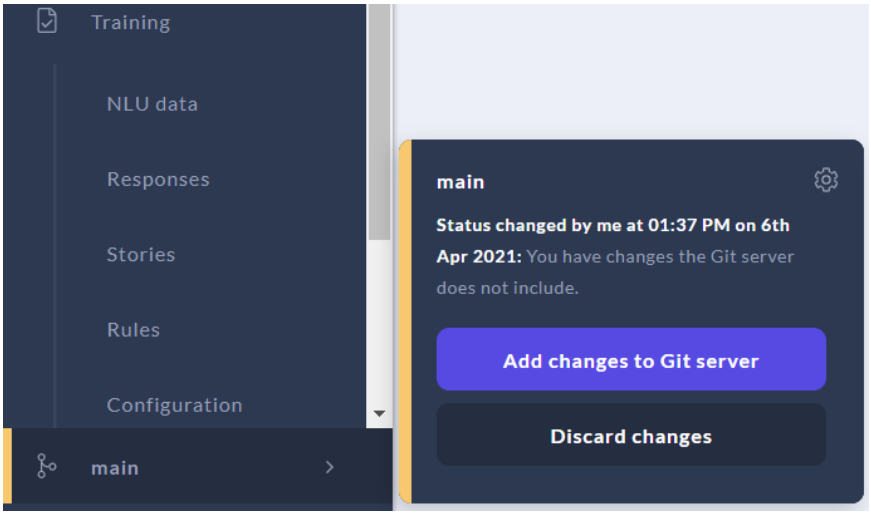

Figura 4.9: UI do Rasa X demonstrando que a integração com o repositório foi bem-sucedida.

Para poder implantar o assistente a amplo público de utilizador conectando-o ao canais externos, existem algumas pendências que precisam ser configuradas em relação ao servidor do *bot*, a partir do momento em que servidor *Rasa* X foi hospedado e possui um endereço IP. É necessário definir o nome de domínio, configurar DNS e obter um certificado SSL para que o servidor possa comunicar com alguns aplicativos da *Web*, como o *Telegram*.

4.1.5.5 INTEGRAÇÃO COM O TELEGRAM

O *Telegram* é um aplicativo de mensagens gratuito que pode ser utilizado em uma variedade de diferentes dispositivos, como tablets, smartphones ou laptops [108]. Suporta aplicações de terceiros como *chatbots*.

Os processos de conexão de um assistente do *Rasa* a diferentes plataformas são semelhantes, neste projeto o foco foi a integração com o *Telegram*. O primeiro passo, foi a configuração da conta do assistente no Telegram através do chamado *BotFather*, o gerenciador de *bots* do *Telegram*.

CAPÍTULO V

EXPERIÊNCIAS E RESULTADOS

Neste capítulo são apresentados e discutidos os resultados, obtidos a partir da tarefa de previsão sob o modelo treinado, permitindo assim a apresentação de uma perspetiva global acerca do seu funcionamento e desempenho.

5.1 RESULTADOS DA INTERFACE CONVERSACIONAL.

Este subcapítulo enfocará as experiências, resultados e desempenho dos módulos do sistema conversacional, bem como os resultados obtidos com o uso das técnicas e métodos diferentes.

Mais precisamente, vamos experimentar dois métodos diferentes para a classificação de *intent*, uma abordagem de embedding pré-treinada para as palavras em português e a incorporação de fluxo de *rasa tensorflow* que treina as palavras do zero, em que uma visão elevada mostra resultados mais aceitáveis neste domínio particular do sistema de conversação.

5.1.1 INTENT CLASSIFICATION

O primeiro problema a resolver é a classificação da intenção, já que deveria ser do texto em português, seria necessário afinar da melhor forma possível, é necessário testar com os diferentes métodos ou técnicas.

5.1.1.1 PRÉ-PROCESSANDO

Antes do treinamento real dos dados e devido ao seu design, geralmente os dados textuais não podem ser alimentados diretamente em uma rede neuronal, eles exigem uma etapa extra de pré-processamento. Em profundidade, eles precisam ser transformados em sequências de inteiros com o mesmo comprimento. As etapas de pré-processamento estão listadas abaixo.

- Transforme todos os caracteres em minúsculas
- Filtre os caracteres de pontuação insignificantes. (por exemplo, “?!,^…)

- *Tokenize* as expressões usando palavras como uma lista de *tokens*.
- Converta os *tokens* em vetores de palavras, no final desta etapa teremos uma lista apenas de números.

Ao final dessas etapas o resultado obtido será adequado para treinar no algoritmo de aprendizado de máquina.

5.1.1.2 MÉTRICAS DE DESEMPENHO

Para atingir ou identificar os principais problemas / erros do componente NLU na interface conversacional, logicamente seria necessário quantificar a qualidade dos dados de treinamento. Além disso, para avaliar o desempenho do modelo seria necessário obter mais avaliação para cada *intent* de pessoas reais que estão no contexto com o domínio do sistema conversacional, seria perfeito para testar com um número maior de avaliações em cada *intent* no conjunto de dados, mas como o domínio é específico, achamos que esses enunciados seriam suficientes para identificar os problemas e compare as soluções.

5.1.1.2.1 ABORDAGEM DE PRE-TRAINED EMBEDDING COM SPACY PORTUGUESE MODEL

Como queríamos comparar mais de uma abordagem para identificar o melhor *pipeline* de classificador com nosso componente NLU, testamos o teste de dados com a abordagem de vetor pré-treinada do modelo que pode ser encontrada em [109].

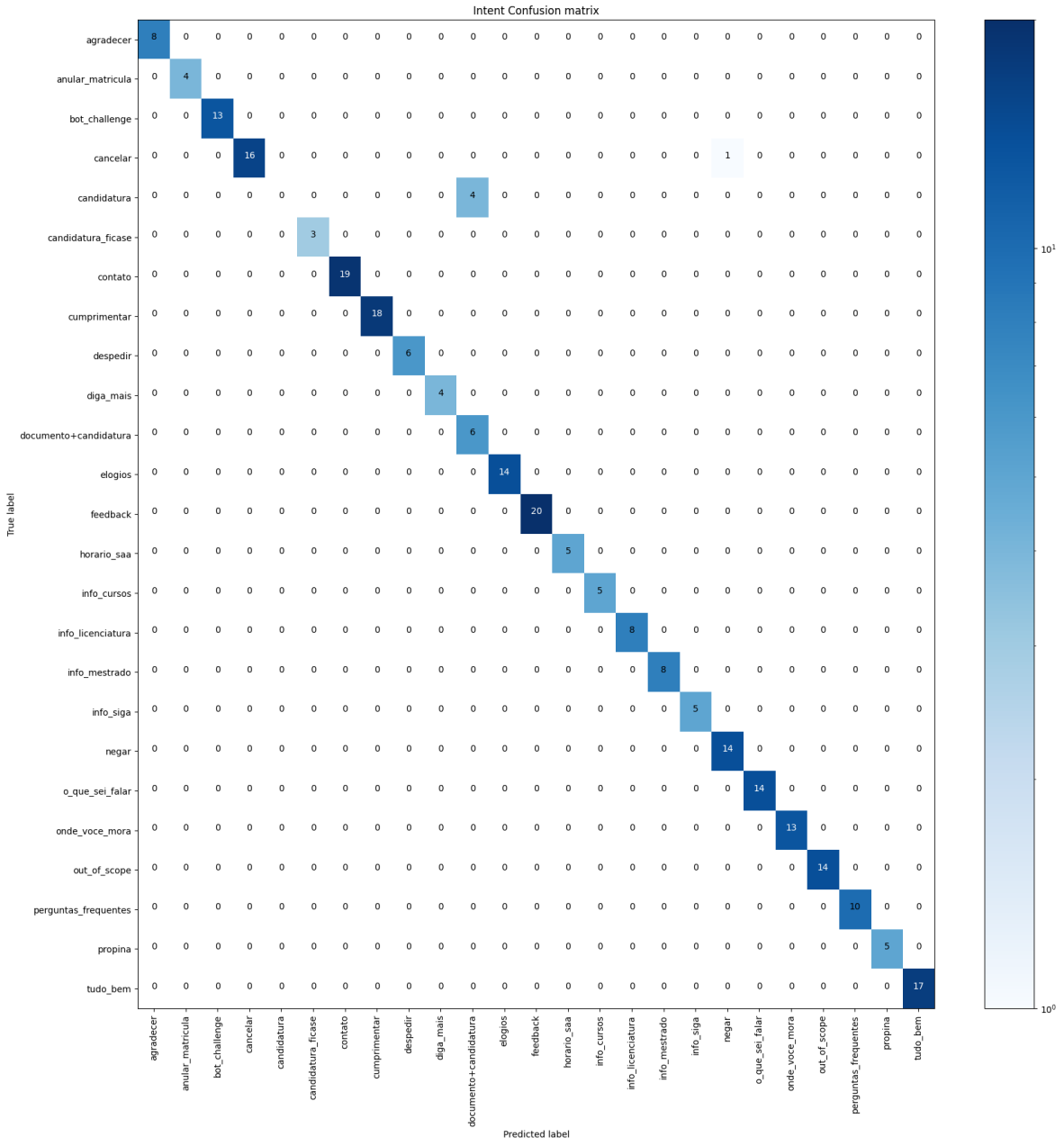

Figura 5.1: Matriz de confusão para a abordagem de pre-trained embedding

Se observarmos a figura 5.1, o classificador parece identificar corretamente os *intents* para o conjunto de dados de teste, no entanto, se observarmos com mais precisão, podemos descobrir alguns erros em alguns *intents*, também alguns dos *intents* treinados no modelo (candidatura) aparece sem uma expressão do conjunto de dados de teste que define que alguma outra intenção foi prevista incorretamente. No geral, essa abordagem não é

perfeita, mas parece viável e robusta o suficiente para trabalhar com os valores obtidos na matriz de confusão, seria interessante, de facto, buscar mais resultados e testá-la.

A Tabela 5.1 mostra os valores de *precision*, *recall* e *f1-score* obtidos para cada *intent* utilizado no *dataset* de teste utilizando esta abordagem atual.

Intent	**precision**	**recall**	**f1-score**	**Support**
info_mestrado	1.0	1.0	1.0	8
info_licenciatura	1.0	1.0	1.0	8
cumprimentar	1.0	1.0	1.0	18
bot_challenge	1.0	1.0	1.0	13
out_of_scope	1.0	1.0	1.0	14
feedback	1.0	1.0	1.0	20
diga_mais	1.0	1.0	1.0	4
info_cursos	1.0	1.0	1.0	5
cancelar	1.0	0.941	0.969	17
documento+candidatura	0.6	1.0	0.749	6
anular_matricula	1.0	1.0	1.0	4
negar	0.933	1.0	0.965	14
elogios	1.0	1.0	1.0	14
horario_saa	1.0	1.0	1.0	5
info_siga	1.0	1.0	1.0	5
perguntas_frequentes	1.0	1.0	1.0	10
o_que_sei_falar	1.0	1.0	1.0	14
propina	1.0	1.0	1.0	5
agradecer	1.0	1.0	1.0	8
despedir	1.0	1.0	1.0	6
contato	1.0	1.0	1.0	19
onde_voce_mora	1.0	1.0	1.0	13
tudo_bem	1.0	1.0	1.0	17
candidatura	0.0	0.0	0.0	4
candidatura_ficase	1.0	1.0	1.0	3
macro average	0.941	0.957	0.947	254
weighted average	0.971	0.980	0.974	254

Tabela 5.1: Valores métricos para a abordagem de pre-trained embedding

Se observarmos detalhadamente a tabela 5.1, esses valores são de fato muito precisos, porém, para tornar a solução totalmente viável, devemos agora observar os níveis de confiança das intenções, que serão introduzidos na figura 5.2.

No entanto, apesar dos bons resultados na matriz de confusão de intenções para a abordagem de *pre-trained embedding*, se avaliarmos a confiança nas intenções de distribuição no conjunto de dados de teste, como podemos observar na figura 5.2, é visualmente percetível que a confiança não é tão bom como já esperávamos, isso porque há muitas palavras já pré-treinadas do Modelo *Spacy* onde serão confundidas com algumas palavras que esperamos que sejam significativas neste contexto específico.

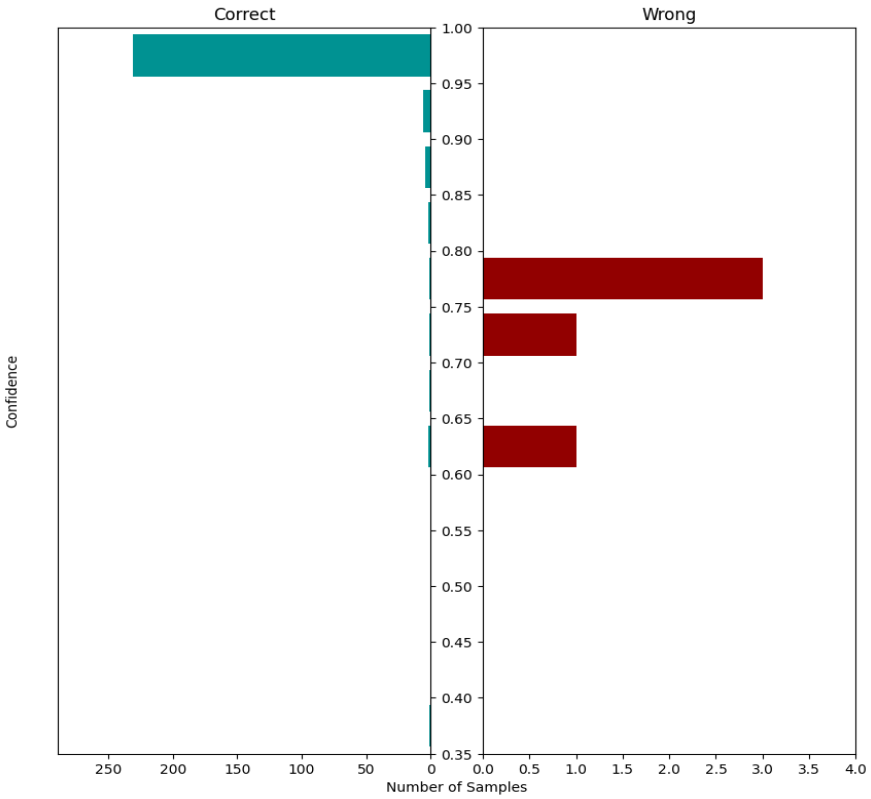

Figura 5.2: Distribuição de confiança do histograma para a abordagem de pre-trained embedding

Em um caso real de palavras, seria muito perigoso identificar uma *intent* com nível de confiança em torno de 0.5 (50%), isso poderia gerar uma conversa fora do contexto, e também há muitos erros em prever a *intent* corretamente. Portanto, se definirmos o limite para uma ação de *fallback* acima de 0.7/0.8 (deve ser a ação a ser executada quando o componente NLU não é capaz de identificar corretamente a intenção definida pelo limite no sistema), esta abordagem não deve ser tão boa para o nosso domínio de sistema conversacional.

5.1.1.2.2 RASA TENSORFLOW EMBEDDING

Uma das principais vantagens de usar esta técnica, o fato de ser inerentemente independente da linguagem e não depender de boa incorporação de palavras para um determinado idioma, também é adotável no domínio específico, uma vez que treina as palavras do zero, nós podemos esperar bons resultados do contexto atual do sistema de conversação, no entanto, também podemos esperar algumas lacunas quando o utilizador disser alguma palavra nova que não foi treinada no conjunto de dados atual.

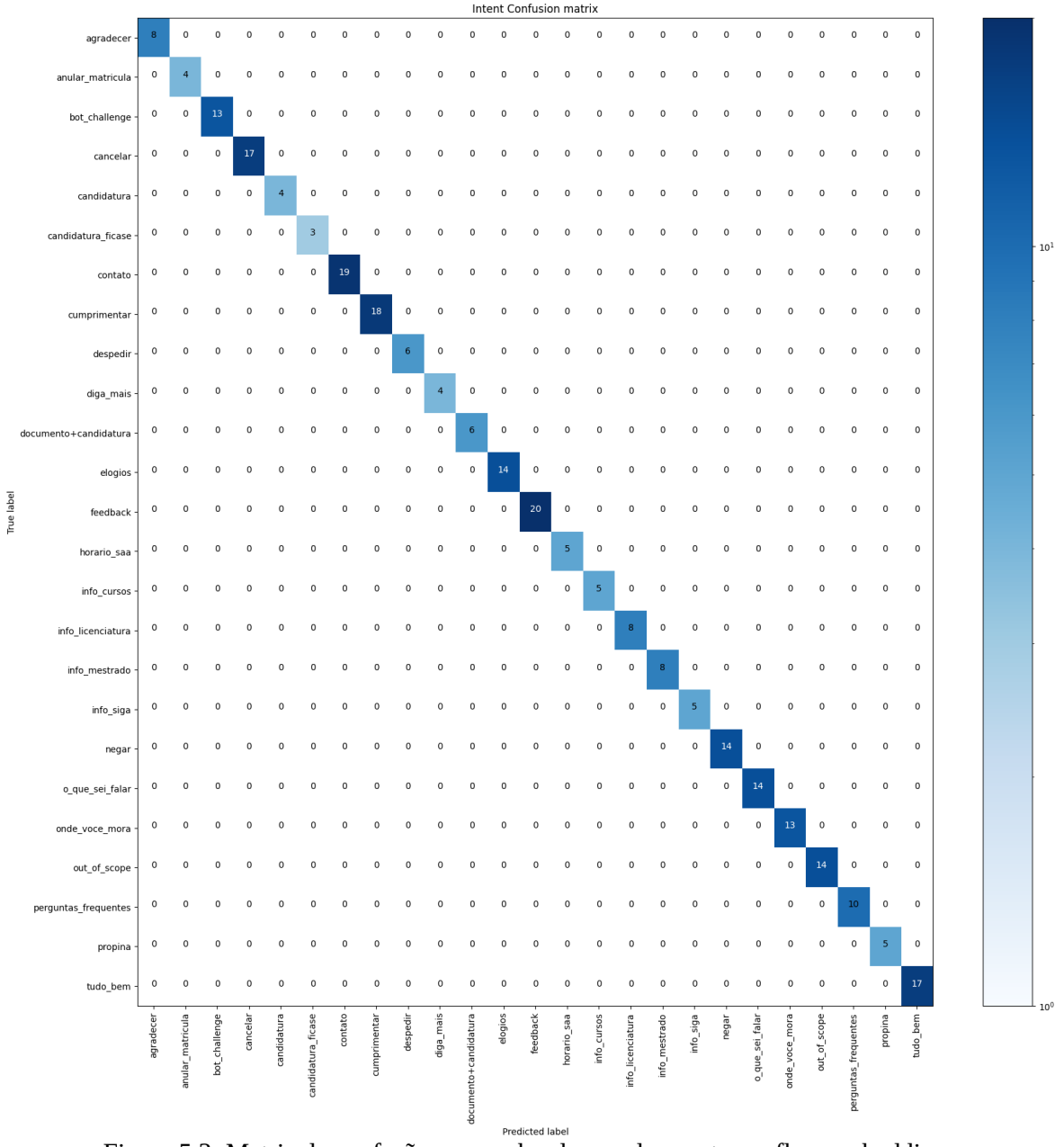

Figura 5.3: Matriz de confusão para a abordagem de rasa tensorflow embedding

Observando a figura 5.3, novamente o classificador apresenta bons resultados na identificação dos *intents* corretos do conjunto de dados de teste, na verdade, também parece mais preciso do que a abordagem do modelo pré-treinado da figura 5.1. No geral, esperamos ter um bom desempenho no caso de uso real para este domínio específico do agente conversacional.

A tabela apresentada em 5.2 mostra os valores de *precision*, *recall* e *f1-score* obtidos para cada *intent* utilizado no *dataset* de teste.

Tabela 5.2: Valores métricos para a abordagem do rasa tensorflow

Intents	**precision**	**recall**	**f1-score**	**support**
Despedir	1.0	1.0	1.0	6
anular_matricula	1.0	1.0	1.0	4
perguntas_frequentes	1.0	1.0	1.0	10
Negar	1.0	1.0	1.0	14
diga_mais	1.0	1.0	1.0	4
info_mestrado	1.0	1.0	1.0	8
info_cursos	1.0	1.0	1.0	5
documento+candidatura	1.0	1.0	1.0	6
info_licenciatura	1.0	1.0	1.0	8
Elogios	1.0	1.0	1.0	14
out_of_scope	1.0	1.0	1.0	14
onde_voce_mora	1.0	1.0	1.0	13
Propina	1.0	1.0	1.0	5
bot_challenge	1.0	1.0	1.0	13
Cancelar	1.0	1.0	1.0	17
candidatura_ficase	1.0	1.0	1.0	3
candidatura	1.0	1.0	1.0	4
cumprimentar	1.0	1.0	1.0	18
o_que_sei_falar	1.0	1.0	1.0	14
Feedback	1.0	1.0	1.0	20
tudo_bem	1.0	1.0	1.0	17
Agradecer	1.0	1.0	1.0	8
horario_saa	1.0	1.0	1.0	5
info_siga	1.0	1.0	1.0	5
Contato	1.0	1.0	1.0	19
macro average	1.0	1.0	1.0	254
weighted average	1.0	1.0	1.0	254

Observando em profundidade a tabela 5.2 os valores também são bastante decentes a partir dessas métricas, de fato, se compararmos com a tabela da abordagem do modelo pré-treinado em 5.1, podemos notar uma melhora significativa.

Observando a figura 5.4, podemos verificar uma melhora percetível sobre os resultados obtidos com a abordagem do modelo pré-treinado na figura 5.2, tendo todas as intenções do conjunto de dados de teste sido previstas com 1.0 (100%) de confiança, o que é um excelente valor preciso para um caso de uso real.

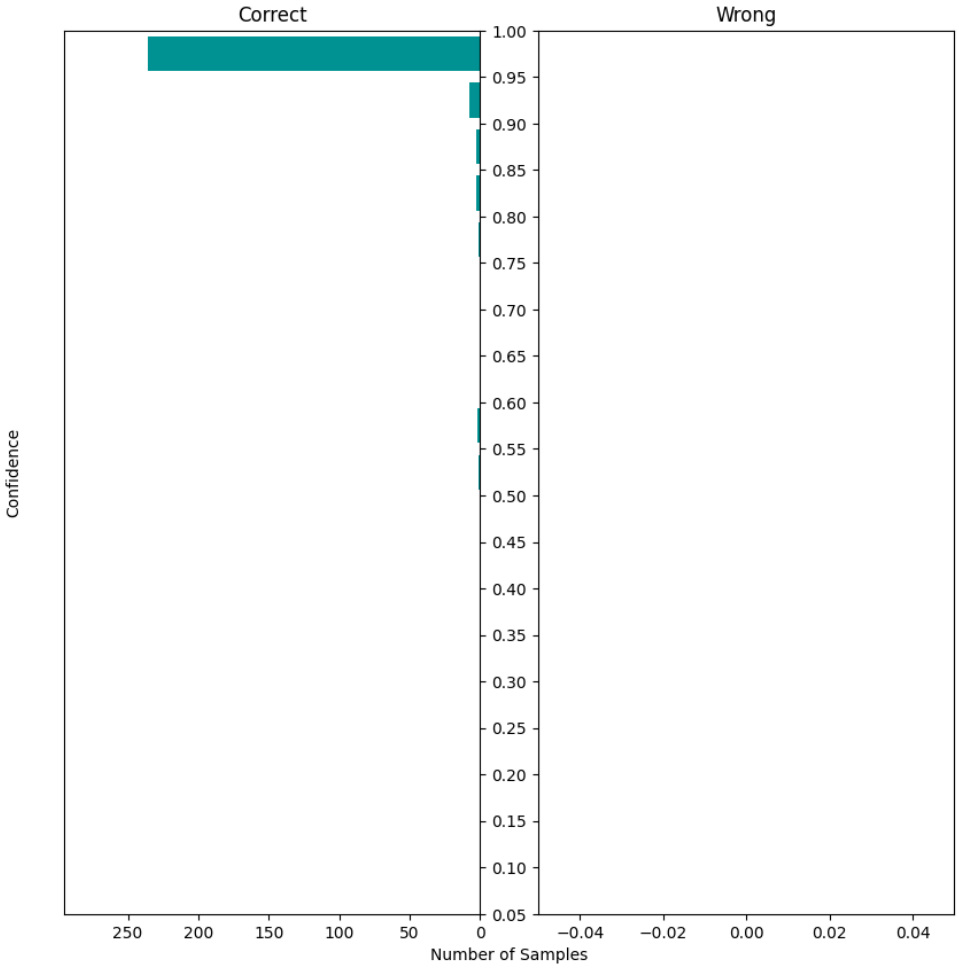

Figura 5.4: Distribuição de confiança do histograma para a abordagem de rasa tensorflow embedding

5.1.1.2.3 COMPARANDO PRE-TRAINED EMBEDDING E AS ABORDAGENS DE RASA TENSORFLOW EMBEDDING

Na figura 5.5, observamos o gráfico *F1-Score* para a classificação de *intents*, o processo para fazer este gráfico foi o seguinte, criamos / dividimos um novo conjunto de dados de teste do conjunto de dados de treinamento original cinco vezes (0%, 25%, 50%, 75%, 90%) e então treinamos várias vezes cada *pipeline* com a percentagem de dados excluídos do conjunto de dados de treinamento. Finalmente, obtemos o *F1-Score* de cada percentagem de exclusão registada.

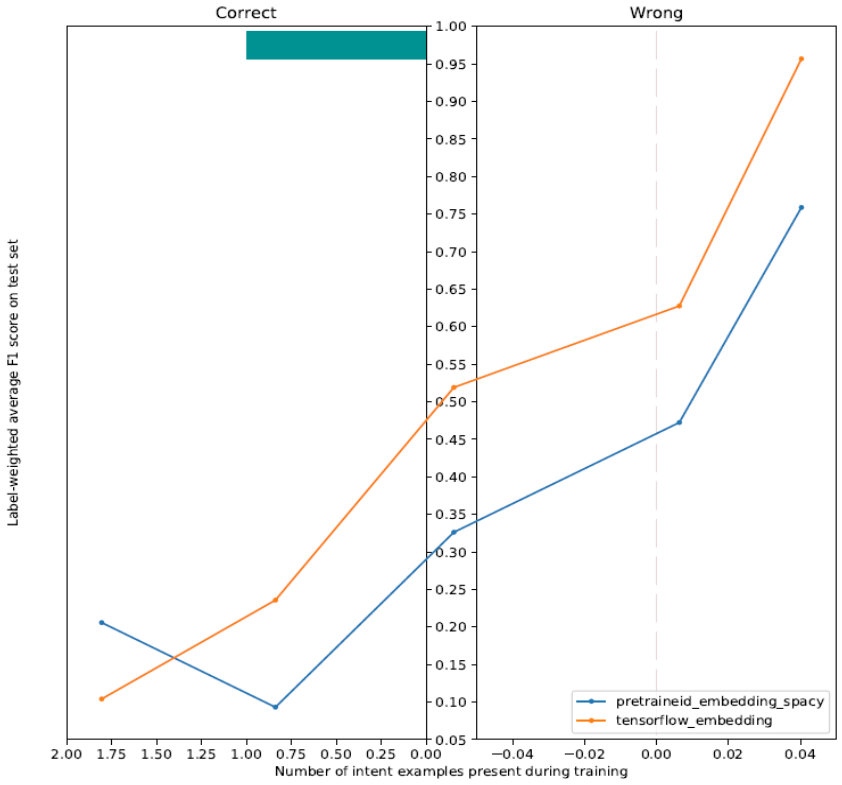

Figura 5.5: Gráfico com F1-Score do modelo spacy pré-treinado vs rasa tensorflow embedding.

Se observarmos os detalhes na figura 5.5, obviamente aumentando a percentagem de treinamento de dados a partir dos dados divididos, o *F1-Score* também aumentará gradualmente, no entanto, é claramente visível que *rasa tensorflow embedding* obtém melhores resultados com este mesmo crescimento do que abordagem pré-treinada de saco de palavras treinada. Isto afirma que a abordagem da *embedding words* do zero é mais estimulante para este domínio de contexto específico e para a língua portuguesa do sistema conversacional.

5.2 AVALIAÇÃO E RESULTADOS COM UTILIZADORES

Neste subcapítulo, apresentamos e discutimos o *Survey* realizado para avaliar o desempenho e a utilidade do *chatbot UMbot* após um teste de conversação realizado com estudantes. Na seção 5.2.1, explanamos a metodologia adotada para realização do *Survey* com os estudantes. Na seção 5.2.2, descrevemos o teste realizado e discutimos os resultados obtidos através do *Survey*. Por fim, a seção 5.2.3 traz as considerações finais sobre o que foi visto neste subcapítulo.

5.2.1 METODOLOGIA

Nesta seção, descrevemos a metodologia adotada para avaliar o desempenho e a utilidade do *chatbot UMBot*.

Para avaliar *UMbot* com os estudantes, definimos a Questão de Pesquisa a seguir.

Questão de Pesquisa

O desempenho e a utilidade de UMbot são satisfatórios em uma conversação com estudantes?

Para responder a essa questão, elaboramos um *Survey* que foi aplicado após um teste de conversação com *UMbot*.

Survey é um método utilizado para descrever, comparar ou explicar conhecimentos, atitudes e comportamentos. Através de um *Survey*, podemos coletar dados de indivíduos que respondem a uma série de questões sobre comportamentos e opiniões, muitas vezes sob a forma de um questionário. Esse é um dos métodos de pesquisa mais utilizados, buscando garantir aos indivíduos um alto nível de anonimato para que eles se sintam livres para expressar suas ideias e opiniões pessoais, sem o risco de serem identificados.

O *Survey* foi elaborado com 5 questões, são elas:

1. Você achou o *chatbot UMbot* fácil de usar?
2. Você considera que foi interessante conversar com *UMbot*?
3. *UMbot* conseguiu entender as suas perguntas e respostas?
4. Ao executar as tarefas solicitadas, você prefere usar o *chatbot* ou a interface tradicional?
5. Você conversaria novamente com *UMbot*?

Para os estudantes responderem às questões, utilizamos a escala *Likert*[22] com um nível de gradação de cinco pontos sobre o grau de conformidade. Assim, utilizamos os valores de 1 a 5 para cada resposta, sendo: (1) Discordo Totalmente, (2) Discordo, (3) Não Tenho Certeza, (4) Concordo e (5) Concordo Totalmente. Para possibilitar a aquisição de um perfil mínimo do estudante, além de perguntarmos o sexo e a idade, fizemos também a seguinte pergunta: *Você já tinha conversado com um sistema parecido com UMbot?*. Essa pergunta foi realizada para tentarmos verificar se o fato de o estudante já ter tido uma experiência com *chatbots* influenciaria o resultado do *Survey*. Além disso, também incluímos no *Survey* um campo para comentários / sugestões, para que eles fornecessem suas opiniões sobre a conversa realizada com *UMbot* e apresentar sugestões de melhorias. Este pode ser um bom ponto para continuar desenvolvendo as habilidades da interface de conversação, detetando o que os utilizadores gostariam de ver em um *chatbot*.

5.2.2 TESTE DE CONVERSAÇÃO E DO SURVEY

Nesta seção, explanamos como o teste de conversação foi realizado, e discutimos os resultados obtidos com o *Survey*.

Realização do Teste

O Teste foi realizado com 18 estudantes da Universidade do Mindelo dos cursos de EISC (Engenharia Informática e Sistemas Computacionais) e LRE (Línguas e Relações Empresariais), nos dias 19, 20 e 22 de abril de 2021.

Inicialmente, foi explicado para os estudantes sobre o projeto, o tempo de realização do teste (em média 35min), a aplicação do questionário no final da conversação, e sobre o anonimato. No decorrer do teste, notamos lentidão no funcionamento de *UMbot*, e em alguns equipamentos o *UMbot* parou de funcionar (possivelmente devido a problemas com a Internet). Após utilizar a opção recarregar a página do Navegador Web, o *chatbot* voltou a funcionar corretamente.

[22] A escala Likert ou escala de Likert é um tipo de escala de resposta psicométrica usada habitualmente em questionários, e é a escala mais usada em pesquisas de opinião.

5.2.2.1 RESULTADOS

O teste foi realizado por estudantes, 11 do sexo masculino e 7 do feminino.

Apresentamos agora os resultados das questões realizadas no *Survey*.

Questão 1: *Você achou o chatbot UMbot fácil de usar?*

Um total de 40.5% dos estudantes concordaram que foi fácil usar *chatbot UMbot*, e 58.32% concordaram totalmente com essa questão. Um percentual de 1.18% respondeu não ter certeza se foi fácil usar *chatbot UMbot*. Assim, podemos concluir que 98.82% dos estudantes concordaram que foi fácil usar *UMbot*.

Questão 2: *Você considera que foi interessante conversar com UMbot?*

Um total de 50.25% dos estudantes concordaram que foi interessante conversar com *UMbot*, e 47.55% concordaram totalmente com essa questão. Um percentual de 2.2% respondeu não ter certeza se foi interessante conversar com o *chatbot*. Assim, podemos concluir que 97.8% dos estudantes concordaram que foi interessante conversar com *UMbot*.

Questão 3: *UMbot conseguiu entender as suas perguntas e respostas?*

Do total de estudantes, 57% concordaram que *UMbot* conseguiu entender suas perguntas e respostas, 29.5% concordaram totalmente que *UMbot* entendeu suas perguntas e respostas, 11.2% não têm certeza que *UMbot* conseguiu entender e 2.3% discordaram que *UMbot* entendeu suas perguntas e respostas. Portanto, 86.5% dos estudantes concordaram que *UMbot* conseguiu entender suas perguntas e respostas, e apenas 2.3% discordaram. Nenhum dos estudantes discordou totalmente com a questão 3.

Questão 4: *Ao executar as tarefas solicitadas, você prefere usar o chatbot ou a interface tradicional?*

Um total de 37.5% dos estudantes concordaram que preferem usar *chatbot* em vez de *interface* tradicional e 46.88% dos estudantes concordaram totalmente em usar *chatbot*, 9.38% não têm certeza, e 6.25% discordaram com essa questão. Portanto, concluímos que 84.38% dos estudantes concordaram que preferem usar *chatbot*. Nenhum dos estudantes discordou totalmente com a questão 4.

Questão 5: *Você conversaria novamente com UMbot?*

Um percentual de 50.88% dos estudantes concordaram que conversariam com *UMbot* novamente, 35.50% dos estudantes concordaram totalmente com essa questão, 6.25% não têm certeza se conversariam com *UMbot* novamente, 5.5% discordaram dessa pergunta

e 1.87% discordaram totalmente que repetiriam essa atividade. Assim, podemos concluir que 86.38% dos estudantes concordaram que conversariam com *UMbot* em uma nova oportunidade e apenas 7.37% não conversariam novamente. Acreditamos que esse resultado é muito positivo.

Na Tabela 5.3, podemos ver os resultados das 5 perguntas realizadas no *Survey*. Assim, **DT** significa Discordo Totalmente, **D** significa Discordo, **NTC** quer dizer Não Tenho Certeza, **C** representa Concordo e **CT** significa Concordo Totalmente. Para cada coluna de resposta, podemos ver o percentual de estudantes que as responderam. Na última linha da tabela, temos uma média calculada dos percentuais das respostas pelo número de questões realizadas.

Tabela 5.3: Resultado das Questões Realizadas no *Survey*.

Questão	**DT**	**D**	**NTC**	**C**	**CT**
1	0%	0%	1.18%	40.5%	58.32%
2	0%	0%	2.2%	50.25%	48.55%
3	0%	2.3%	11.2%	57%	29.5%
4	0%	6.25%	9.38%	37.5%	46.88%
5	1.87%	5.5%	6.25%	50.88%	35.50%
Média	0.37%	2.81%	6.04%	47.23%	43.75%

A **Questão de Pesquisa** (*O desempenho e a utilidade de UMbot são satisfatórios em uma conversação com estudantes?*) que norteou o *Survey* realizado com os estudantes, foi dividida em dois elementos principais, o desempenho e a utilidade. Em relação a esses elementos, temos os seguintes resultados conforme evidencia a Tabela 4.4.

Tabela 5.4: Resultado em Relação ao Desempenho e a Utilidade de UMbot.

Elemento	**D**	**NTC**	**C**
Desempenho	3.22%	6.21%	90.56%
Utilidade	3.13%	5,79%	91.59%

Para calcular o desempenho de *UMbot*, somamos os resultados das respostas da questão 1 (*Você achou o chatbot UMbot fácil de usar?*), da questão 3 (*UMbot conseguiu*

entender as suas perguntas e respostas?) e da questão 5 (*Você conversaria novamente com UMbot?*), e calculamos a média. Agrupamos as respostas *Discordo Totalmente* e *Discordo* no resultado *Discordo*. A resposta *Não tenho Certeza* permaneceu com a mesma nomenclatura de resultado e, *Concordo* e *Concordo Totalmente* em *Concordo*.

Conforme mostra a Tabela 4.4, 90.56% dos estudantes concordaram que *UMbot* teve um desempenho satisfatório, contra 3.22% dos estudantes que discordaram. O elemento "desempenho" foi traduzido em relação a facilidade dos estudantes na utilização do *chatbot*, e nos erros de interpretação das perguntas e do processamento das respostas cometidos por *UMbot*.

Para calcular a utilidade de *UMbot* realizamos o processo similar ao cálculo do resultado do desempenho. Somamos os resultados das respostas da questão 2 (*Você considera que foi interessante conversar com UMbot?*) e da questão 4 (*Ao executar as tarefas solicitadas, você prefere usar o chatbot ou a interface tradicional?*) do *Survey*, e calculamos a média.

De acordo com a Tabela 4.4, 91.59% dos estudantes concordaram que *UMbot* foi útil para eles, e 3.13% dos estudantes discordaram da sua utilidade. O elemento "utilidade" foi traduzido em relação ao interessante dos estudantes na utilização de *UMbot* e na sua escolha.

5.2.2.2 DISCUSSÃO

Acreditamos que o ideal também seria realizar a avaliação de *UMbot* com diferentes tipos de utilizadores como os alunos que desejam ingressar na UM e utilizadores que estão familiarizados com este tipo de software.

Na verdade, acreditamos que esses resultados e números obtidos ao final poderiam ter várias alterações, se a interface de conversação no caso tivesse mais recursos e se o número de utilizadores ativos na experiência fosse maior.

Um fator que prejudicou a realização do teste de conversação foi a lentidão do processamento das respostas por *UMbot*. Essa lentidão foi causada pela baixa velocidade da Internet e recursos da máquina que executa o software.

A base de diálogos atual de *UMbot* precisa ser melhorada. Contudo, apesar das dificuldades, os resultados do *Survey* foram consistentemente positivos.

Como visto anteriormente, o *Survey* oferecia um espaço para comentários / sugestões. Vejamos alguns deles na Tabela 5.5.

Tabela 5.5 – Comentários dos estudantes no Survey Realizado.

Comentário / Sugestões
"Foi muito bom conversar com UMbot"
"Gostei muito da experiencia"
"Gostaria que fosse implementado no Facebook da UM"
"Eu imagino que a maioria das pessoas aprenderia a usar este sistema muito rapidamente"
"foi interessante conversar com bot, mas acho que deveria ter mais resposta para as perguntas"
"foi ótimo gostaria de falar com ela de novo"
"sugiro que o chatbot apresentasse opções de perguntas em alguns casos em vez do utilizador escrever perguntas"

Em relação à pergunta: *Você já tinha conversado com um sistema parecido com UMbot?*, um total 90.6% dos estudantes nunca haviam conversado com um sistema parecido com *UMbot*, enquanto 9.4% responderam positivamente.

Apesar das dificuldades na realização da avaliação de *UMbot*, consideramos os resultados satisfatórios. O desempenho e a utilidade de *UMbot* mostraram um bom nível de aceitação. Em relação ao desempenho, 90.56% dos estudantes ficaram satisfeitos, contra 3.22% que não ficaram satisfeitos. Em relação à utilidade, 91.59% dos estudantes disseram que foi interessante conversar com *UMbot* e que utilizavam *chatbot* para conversar em relação a outras *interfaces* tradicionais.

CAPÍTULO VI

O capítulo final desta dissertação resume o trabalho que foi realizado para atingir o objetivo da dissertação e delineia as possíveis direções para trabalhos futuros, com o objetivo de melhorar o sistema proposto.

6.1 CONCLUSÃO

Esta dissertação teve como objetivo principal construir um *chatbot* baseado em Processamento de Linguagem Natural, para a Universidade do Mindelo, visando conversar com os alunos que desejam ingressar na UM, bem como os alunos já existentes, respondendo às suas perguntas relativamente a Universidade.

No capítulo II apresentamos uma revisão de todos os conceitos associados ao termo *chatbot* e Processamento de Linguagem Natural, apresentamos as características diferenciadoras deste tipo de sistemas, expondo para tal, os seus diferentes tipos de projeção e terminamos o capítulo com a apresentação do seu estado de arte. No capítulo III apresentamos brevemente as técnicas de *deep learning* mais promissoras utilizadas na NLP, particularmente técnicas como *word embedding*, uma nova técnica de *tensorflow embedding pipeline* introduzido pelo *framework Rasa* e RNN. As primeiras e últimas técnicas estão presentes em quase todos os *frameworks* e para finalizar o capítulo uma descrição teórico-técnica das tecnologias para a projeção e construção do tipo de *chatbot* proposto. No capítulo IV apresentamos a abordagem prática que permitiu o desenvolvimento do *chatbot UMBot*, desde a componente relativa à extração inicial de dados, que permitiu a construção do *dataset* utilizado, até à implementação prática do motor conversacional. Finalmente, no capítulo V fizemos a comparação entre modelo *pre-trained embedding* e *rasa tensorflow embedding* com analise de resultados obtidos a partir da tarefa de previsão sob o modelo treinado, permitindo assim a apresentação de uma perspetiva global acerca do seu funcionamento e desempenho, chegamos a conclusão que *rasa tensorflow embedding* obtém melhores resultados em relação ao modelo *pre-trained embedding* conforme é apresentado no subsessão 5.1.1.2.3. Para avaliar o desempenho e a utilidade do *chatbot* criado, realizamos um teste de conversação e um *Survey* com 18 estudantes da Universidade do Mindelo. Através das avaliações realizadas, concluímos que

UMbot apresentou resultados satisfatórios. Em relação ao desempenho, 90.56% dos estudantes ficaram satisfeitos com *UMbot* e, em relação à utilidade, *UMbot* teve um resultado de 91.59% de aprovação por parte dos estudantes.

O futuro reserva resultados promissores no campo do Processamento de Linguagem Natural, novos métodos nascem (como o *rasa tensorflow embedding* criado recentemente e utilizado na implementação do *bot*), na verdade o futuro do sucesso ou falha no *chatbot*, depende desse campo. No entanto, o crescimento do *chatbot* deixou uma marca na história, e também na mente dos consumidores, pode ser inteiramente possível em um futuro próximo ver os papéis invertidos entre os *bots* e os humanos, poderemos ver um dia um *bot* interação com um humano usando a linguagem humana para cuidar das tarefas diárias da vida humana.

Em qualquer caso, depois das dificuldades que encontrei em trabalhar com essa tecnologia, ainda estou otimista com esses tipos de implementações e desenvolvimentos e como eles moldariam o futuro de nossas sociedades. Acho que veremos um dia esse tipo de tecnologia indispensável para a vida humana e, obviamente, melhorando substancialmente a experiência geral de utilizador.

6.2 LIMITAÇÕES

Listamos, a seguir, algumas limitações deste trabalho:

- Não foi possível realizar uma avaliação do *chatbot* com mais utilizadores como os estudantes que desejam ingressar na UM e utilizadores que estão familiarizados com este tipo de software;
- Por questão de tempo, a quantidade de padrões de diálogo (entrada-saída) não está satisfatória;
- Não realizamos uma avaliação da velocidade (performance) de *UMbot* em conversar com um número maior de utilizadores ao mesmo tempo.

6.3 TRABALHO FUTURO

O desenvolvimento de um sistema baseado em NLP, incluindo o desenvolvimento de um *chatbot*, não é uma tarefa fácil e requer tempo e dedicação. De seguida, apontamos a

algumas extensões que podem ser implementadas, com o intuito de aumentar a robustez da solução. De entre essas propostas, destacamos:

- Possibilidade de estender os componentes atuais no sistema conversacional, por exemplo, um analisador de sentimento pode ser implementado para identificar as respostas positivas ou negativas de utilizador.
- Estender o canal atual implementado para outras Plataformas de Mensagens disponíveis, como *Slack*, *Facebook Messenger*.
- Se o utilizador não conseguir ser esclarecido, em vez de ir para uma ação de *fallback*, antes disso, o *chatbot* pode sugerir possíveis problemas ao utilizador na forma de uma lista de seleção com problemas comuns.
- Explorar outros métodos e ferramentas para avaliar a melhoria da experiência de utilizador.

REFERÊNCIAS

[1] A. M. Turing, “Computing machinery and intelligence,” *Mind,* vol. 59, nº 236, p. 433–460, Out. 1950.

[2] S. Hosseini, “Chat bots,” 2017.

[3] M. B. Hoy, “Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants,” *Medical Reference Services Quarterly,* vol. 37, nº 1, pp. 81-88, Janeiro 2018.

[4] “Chatbot As An Enterprise Virtual Assistant,” Abril 2018. [Online]. Available: https://chatbotsmagazine.com/chatbot-as-an-enterprise-virtual-assistant-1b2b1c19180f. [Acedido em 10 Out. 2020].

[5] C. B. Nordheim, “Trust in chatbots for customer service – findings from a questionnaire study,” M.S. thesis, 2018.

[6] A. Celikyilmaz, L. Deng e D. Hakkani-Tür, “Deep Learning in Spoken and Text-Based Dialog Systems,” em *Deep Learning in Natural Language Processing*, Springer Singapore, 2018, p. 49–78.

[7] J. Kříž, “Chatbot for Laundry and Dry Cleaning Service,” M.S. thesis, 2017.

[8] MARTÍNEZ-MIRANDA, “J. Towards an empathic virtual agent to support the treatment of major depression,” 2010.

[9] R. DALE, “The return of the chatbots,” *Natural Language Engineering,* vol. 22, nº 5, p. 811–817, 2016.

[10] M. YAN, P. CASTRO, P. CHENG e V. ISHAKIAN, “Building a chatbot with serverless computing,” *ACM,* p. 5:1–5:4, 2016.

[11] J.-P. Sansonnet, D. Leray e J.-C. Martin, “Architecture of a Framework for Generic Assisting Conversational Agents,” Springer, Berlin, Heidelberg, 2006.

[12] “Chatbot Report 2019: Global Trends and Analysis – Chatbots Magazine,” 2019. [Online]. Available: https://chatbotsmagazine.com/chatbot-report-2019-global-trends-and-analysis-a487afec05b. [Acedido em 17 Nov. 2020].

[13] G. Richardson, “chatamo,” 2018. [Online]. Available: https://chatamo.com/rise-of-the-chatbot. [Acedido em 23 Nov. 2020].

[14] Ubisend, *Mobile Messaging Report 2016,* 2016.

[15] K. Ramesh, S. Ravishankaran, A. Joshi e K. Chandrasekaran, “A survey of design techniques for conversational agents,” em *International Conference on Information, Communication and Computing Technology*, Springer, Singapore, 2017.

[16] “Deep Learning chatbot - analysis and implementation - Sigmoidal,” [Online]. Available: https://sigmoidal.io/chatbots-for-b2c-and-deep-learning/. [Acedido em 25 Nov. 2018].

[17] “Chatbot Building Best Practices—Why Message Length Matters,” [Online]. Available: https://uxdesign.cc/chatbot-building-best-practices-why-message-length-matters-e951bed1b550. [Acedido em 10 Jan. 2021].

[18] “How Do Chatbots Work?,” [Online]. Available: https://www.oracle.com/middleeast/chatbots/what-is-a-chatbot/. [Acedido em 15 Jan. 2021].

[19] “A Comparative Analysis of Chatbots APIs – ActiveWizards: machine learning company – Medium,” [Online]. Available: https://medium.com/activewizards-machine-learning-company/a-comparativeanalysis-of-chatbots-apis-f9d240263e1d. [Acedido em 15 Jan. 2021].

[20] S. Russell e P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, Egnlewood Cliffs: Pearson, 1995, p. 946.

[21] J. WEINTRAUB, “History of the PC THERAPIST,” Loebner Prize, 2017. [Online]. Available: http://loebner.net/Prizef/weintraub-bio.html. [Acedido em 26 Jan. 2021].

[22] K. M. COLBY, Artificial Paranoia: A Computer Simulation of Paranoid Processes, New York, NY, USA: Elsevier Science Inc., 1975.

[23] M. L. MAULDIN, “Chatterbots, tinymuds, and the turing test: Entering the loebner prize,” *ACM*, vol. 94, p. 16–21, 1994.

[24] R. C. PARKINSON, K. M. COLBY e W. S. FAUGHT, “Conversational language comprehension using integrated pattern-matching and parsing. Artificial Intelligence,” *Elsevier*, vol. 9, nº 2, p. 111–134, 1977.

[25] R. S. WALLACE, “The anatomy of alice,” *Springer, Dordrecht*, p. 181–210, 2009.

[26] S. COFFMAN e S. MCGLAMERY, “The Librarian and Mr. Jeeves,” *American Libraries*, vol. 31, nº 5, p. 66–69, 2000.

[27] “Loebner Prize,” [Online]. Available: https://www.wikiwand.com/en/Loebner_Prize. [Acedido em 27 Jan. 2021].

[28] B. WILCOX, “Brillig Understanding,” Brillig Understanding, Inc, [Online]. Available: http://brilligunderstanding.com/. [Acedido em 27 Jan. 2021].

[29] L. BRADEŠKO e D. MLADENIĆ, “A survey of chatbot systems through a loebner prize,” p. 34–37, 2012.

[30] B. WILCOX, “ChatScript Engine. 2017,” [Online]. Available: https://sourceforge.net/projects/chatscript/. [Acedido em 27 Jan. 2021].

[31] S. WORSWICK, “Chatbot Battles. [S.l.]: Chatbot Battles, 2012,” [Online]. Available: http://chatbotbattles.com/.

[32] S. Bird, E. Klein e E. Loper, Natural language processing with Python: analyzing text with the natural language toolkit, O'Reilly Media, Inc., 2009.

[33] R. C. a. J. Freeman, "Computing machinery and the individual: the personal Turing test," 2005.

[34] S. L. Lim and O. S. Goh, "Intelligent Conversational Bot for Massive Online Open Courses (MOOCs)," *Arxiv.Org*, vol. 1, p. 1689–1699, 2016.

[35] "Chatbots for Healthcare - Comparing 5 Current Applications," [Online]. Available: https://www.techemergence.com/chatbots-for-healthcare-comparison/. [Acedido em 28 jan. 2021].

[36] "Your.MD - Health Guide and Symptom Checker," [Online]. Available: https://www.your.md/. [Acedido em 28 Jan. 2021].

[37] "The Top 12 Health Chatbots - The Medical Futurist," [Online]. Available: https://medicalfuturist.com/top-12-health-chatbots. [Acedido em 28 Jan. 2021].

[38] "Florence - Your health assistant," [Online]. Available: https://florence.chat/. [Acedido em 28 Jan. 2021].

[39] Harper, E. R., Rodden, T., Rogers, Y., Sellen, A., & Human, B., Human-Computer Interaction in the year 2020, 2008.

[40] L. M. M. &. A. H. Piccolo, "Chasing the chatbots: Directions," *ORO*, p. 157–169, 2019.

[41] Jain, M., Kota, R., Kumar, P., & Patel, S. N., "Convey: Exploring the Use of a Context View for Chatbots," *ACM*, p. 468, 2018.

[42] "What is a Chatbot? The Tech Responsible for $8 Billion in Savings," CX Today, 2019. [Online]. Available: https://www.cxtoday.com/contact-centre/what-is-a-chatbot/. [Acedido em 28 Jan. 2021].

[43] Brandtzaeg, P. B., & Følstad, A., "Why people use chatbots. In International Conference on Internet Science," *Springer*, pp. 377-392, 2017.

[44] "Most popular global mobile messenger apps as of January 2021," Statista, 2021. [Online]. Available: https://www.statista.com/statistics/258749/most-popular-globalmobile-messenger-apps/. [Acedido em 29 Jan. 2021].

[45] R. Dale, "The return of the chatbots. Natural Language," *Cambridge University Press*, vol. 22, nº 5, pp. 811-817, 2016.

[46] G. Pall, "Progress in the shift to conversational computing," Official Microsoft Blog, 2016. [Online]. Available: https://blogs.microsoft.com/blog/2016/08/03/. [Acedido em 29 Jan. 2021].

[47] M. MALDONADO, D. ALULEMA, D. MOROCHO e M. PROAÑO, "System for monitoring natural disasters using natural language processing in the social network twitter," *IEEE*, p. 1–6, 2016.

[48] J. F. ALLEN, “Natural language processing,” *ACM,* p. 1218–1222, 2003.

[49] F. A. OLIVEIRA e P. O. A. NAVAUX, “Processamento de linguagem natural: princípios básicos e a implementação de um analisador sintático de sentenças da língua portuguesa,” 2004.

[50] “The Past and the Present of Natural Language Generation,” INSIGHTS, 2019. [Online]. Available: https://insights.ai-jobs.net/the-past-and-the-present-of-natural-language-generation/. [Acedido em 30 Jan. 2021].

[51] J. ALLEN, Natural Language Understanding, New York, NY, USA: Benjamin-Cummings, 1995.

[52] F. A. BARROS e P. C. A. R. TEDESCO, “Agentes inteligentes conversacionais: Conceitos básicos e desenvolvimento,” *Sociedade Brasileira de Computação,* vol. 1, nº 4, p. 169–218, 2016.

[53] D. N. MÜLLER, Processamento de linguagem natural, Porto Alegre, 2003.

[54] Choi, F. Y. Y.; Wiemer-hastings, P.; Moore, J., “Latent semantic analysis for text segmentation,” *Proceedings of the 2001 Conference on Empirical Methods in Natural Language,* vol. 102, p. 109–117, 2001.

[55] C. D. Manning, H. Schütze e G. Weikurn, “Foundations of Statistical Natural Language Processing,” *SIGMOD Record,* vol. 31, nº 3, p. 37–38, 2002.

[56] M. Hassler e G. Fliedl, “Text preparation through extended tokenization,” *WIT Transactions on Information and Communication Technologies,* vol. 37, p. 13–21, 2006.

[57] S. Sun, C. Luo e J. Chen, “A review of natural language processing techniques for,” *Information Fusion,* vol. 36, p. 10–25, 2017.

[58] A. Goyal, V. Gupta e M. Kumar, “Recent Named Entity Recognition and Classification techniques: A systematic review,” *Elsevier Inc.,* vol. 29, pp. 21-43, 2018.

[59] M. McTear, Z. Callejas e D. Griol, The Conversational Interface. Cham, Springer, 2016.

[60] K. B. Cohen, “Biomedical Natural Language Processing and Text Mining,” em *Methods in Biomedical Informatics,* 2013, p. 141–177.

[61] M. F. Porter, “An algorithm for suffix stripping,” *ACM,* p. 313–316, 1997.

[62] J. M. Moreira, A. C. P. L. F. d. Carvalho e T. Horváth, A General Introduction to Data Analytics, 2018.

[63] P. H. Chen, “Essential Elements of Natural Language Processing: What the Radiologist Should Know,” *Academic Radiology,* vol. 27, pp. 6-12, 2019.

[64] H. Liu, T. Christiansen, W. A. Baumgartner e K. Verspoor, "BioLemmatizer: a lemmatization tool for morphological processing of biomedical text," *Journal of Biomedical Semantics*, vol. 3, nº 1, p. 3, 2012.

[65] C. D. Manning, P. Raghavan e H. Schütze, "Introduction to information retrieval," *Cambridge University Press*, p. 482, 2008.

[66] M.-C. D. Marneffe e C. D. Manning, "Stanford typed dependencies manual," Tech. Rep., 2008. [Online]. Available: http://www.universaldependencies.org/.

[67] T. Mikolov, K. Chen, G. Corrado e J. Dean, "Efficient estimation of word representations in vector space," em *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, International Conference on Learning Representations, ICLR, 2013, 2013.

[68] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing," *CoRR, abs/1708.02709*, 2017.

[69] T. Mikolov et al., "Efficient estimation of word representations in vector space," 2013.

[70] A. S. Amit Mandelbaum, "Word embeddings and their use in sentence classification tasks," *CoRR, abs/1610.08229*, 2016.

[71] J. Devlin, M.-W. Chang, K. Lee, K. T. Google e A. I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Tech. Rep., [Online]. Available: https://github.com/tensorflow/tensor2tensor.

[72] 2. ASPECT SOFTWARE, "Customer Service Chatbots and Natural Language," 2019. [Online]. Available: https://www.aspect.com/globalassets/microsite/nlu-lab/images/Customer-Service-Chatbots-and-Natural-Language-WP.pdf. [Acedido em 31 Jan. 2021].

[73] D. Khurana, A. Koli, K. Khatter e S. Singh, "Natural Language Processing: State of The Art, Current Trends and Challenges," *CoRR abs/1708.05148*, vol. 1, 2017.

[74] A. Nichol, "Supervised word vectors from scratch in rasa nlu," [Online]. Available: https://blog.rasa.com/supervised-word-vectors-from-scratch-in-rasa-nlu/. [Acedido em 31 Jan. 2021].

[75] T. Wochinger, "Rasa nlu in depth: Part 1 intent classification," [Online]. Available: https://blog.rasa.com/rasa-nlu-in-depth-part-1-intent-classification/. [Acedido em 31 Jan. 2021].

[76] T. Parshina, "Understanding rasa tensorflow intent classifier," 2019. [Online]. Available: https://medium.com/@tatiana.parshina/understanding-rasa-tensorflow-intent-classifier-e9d4ef019c6. [Acedido em 31 Jan. 2021].

[77] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston, "Starspace: Embed all the things!," *CoRR, abs/1709.03856*, 2017.

[78] T. M. Y. B. Razvan Pascanu, “On the difficulty of training recurrent neural networks,” 2012.

[79] C. Olah, “Understanding lstm networks,” [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Acedido em 01 Fev. 2021].

[80] A. Sordoni, Y. Bengio, H. Vahabi, C. Lioma, J. Grue Simonsen, and J. Nie, “A hierarchical recurrent encoder-decoder for generative context-aware query suggestion,” *In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management,* vol. 1, p. 553–562, 2015.

[81] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau, “Building end-to-end dialogue systems using generative hierarchical neural network models,” *In Thirtieth AAAI Conference on Artificial Intelligence,* vol. 3, 2016.

[82] J. D. Williams, K. Asadi, and G. Zweig, “Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning,” *arXiv preprint arXiv:1702.03274,* vol. 2, 2017.

[83] V. Vlasov, A. Drissner-Schmid, and A. Nichol, “Few-shot generalization across dialogue tasks,” *arXiv preprint arXiv:1811.11707,* vol. 1, 2018.

[84] S. Sahay, S. H. Kumar, E. Okur, H. Syed, and L. Nachman, “Modeling intent, dialog policies and response adaptation for goal-oriented interactions,” *In Proceedings of the 23rd Workshop on the Semantics and Pragmatics of Dialogue,* 2019.

[85] S. Hochreiter e J. Schmidhuber, “ Long short-term memory,” *Neural Computation,* vol. 9, nº 8, p. 1735–1780, 1997.

[86] Y. Bengio, P. Simard, P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks,,* vol. 5, nº 2, p. 157–166, 1994.

[87] Z. Dai, Z. Yang, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, R. Salakhutdinov, “Transformer-xl: attentive language models beyond a fixed-length context,” *arXiv preprint arXiv:1901.02860,* vol. 3, 2019.

[88] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI Blog 1 (8).*

[89] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “Glue: a multi-task benchmark and analysis platform for natural language understanding,” *arXiv preprint arXiv:1804.07461,* vol. 3, 2019.

[90] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “Bert: pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805,* vol. 2, 2019.

[91] Y. Hsieh, M. Cheng, D. Juan, W. Wei, W. Hsu, and C. Hsieh, "On the robustness of self-attentive models," *In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 1520–1529, 2019.

[92] Vladimir Vlasov, Johannes E. M. Mosig, Alan Nichol, "Dialogue Transformers," *arXiv:1910.00486*, vol. 3, 2020.

[93] B. Peters, V. Niculae, and A. F. Martins, "Sparse sequence-to-sequence models," *arXiv preprint arXiv:1905.05702*, vol. 2, 2019.

[94] Young, J. D. Williams and S., "Partially observable markov decision processes for spoken dialog systems," *Computer Speech & Language 21 (2)*, p. 393–422, 2007.

[95] T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, "Rasa: open source language understanding and dialogue management," *arXiv preprint arXiv:1712.05181*, vol. 2, 2017.

[96] M. Henderson, I. Vulić, D. Gerz, I. Casanueva, P. Budzianowski, S. Coope, G. Spithourakis, T. Wen, N. Mrkšić, and P. Su, "Training neural response selection for task-oriented dialogue systems," *arXiv preprint arXiv:1906.01543*, vol. 2, 2019.

[97] D. Braun, Hernandez-Mendez, M. F. A. e M. Langen, "Evaluating Natural Language Understanding Services for Conversational Question Answering Systems," *Association for Computational Linguistics*, vol. Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, p. 174–185, 2017.

[98] S. Raj, Building Chatbots with Python, Apress, Berkeley, CA, 2019, p. 192.

[99] "Algorithms alone won't solve conversational AI — Introducing Rasa X," [Online]. Available: https://medium.com/rasa-blog/algorithms-alone-wont-solve-conversational-ai-introducing-rasa-x-b2767d1964de. [Acedido em 02 Fev. 2021].

[100] "The Rasa Masterclass Handbook: Episode 4," [Online]. Available: https://blog.rasa.com/the-rasa-masterclass-handbook-episode-4/. [Acedido em 2 Fev. 2021].

[101] "Stories," [Online]. Available: https://rasa.com/docs/rasa/core/stories/. [Acedido em 3 Fev. 2021].

[102] "The Rasa Masterclass Handbook: Episode 6," [Online]. Available: https://blog.rasa.com/the-rasa-masterclass-handbook-episode-6-2/. [Acedido em 3 Fev. 2021].

[103] "Rasa X," [Online]. Available: https://rasa.com/docs/rasa-x/0.27.8/installation-and-setup/integrated-version-control/. [Acedido em 3 Fev. 2021].

[104] "Rasa X," [Online]. Available: https://blog.rasa.com/rasa-x-getting-started-as-a-current-rasa-user/. [Acedido em 3 Fev. 2021].

[105] "Installation," [Online]. Available: https://rasa.com/docs/rasa/user-guide/installation/. [Acedido em 10 Fev. 2021].

[106] “Anaconda Distribution,” [Online]. Available: https://www.anaconda.com/distribution/. [Acedido em 10 Fev. 2021].

[107] “Training Data Importers,” [Online]. Available: https://rasa.com/docs/rasa/api/training-data-importers/. [Acedido em 11 Fev. 2021].

[108] “Telegram FAQ,” [Online]. Available: https://telegram.org/faq. [Acedido em 15 Fev. 2021].

[109] “Spacy. Available pretrained statistical models for portuguese.,” [Online]. Available: https://spacy.io/models/pt. [Acedido em 18 Fev. 2021].

APÊNDICE

Formulário de Avaliação do Chatbot UMbot

Objetivo: Este questionário será utilizado para avaliar o desempenho e a utilidade do *chatbot UMbot* para seus utilizadores.

Tempo estimado: 45 minutos.

Consentimento de participação e confidencialidade: Respondendo este questionário, você consente com a sua participação nesta pesquisa. Esse questionário é anônimo.

Endereço de acesso ao *chatbot*:

http://ec2-3-237-22-170.compute-1.amazonaws.com/conversations

Perfil de utilizador:

Idade:

☐ 18 - 25 anos
☐ 26 - 35 anos
☐ 36 - 45 anos
☐ 46+ anos

Sexo:

☐ Masculino
☐ Feminino

Você já tinha ouvido falar ou conversou com um *chatbot*?

☐ Sim
☐ Não

Questões:

1. **Você achou o *chatbot UMbot* fácil de usar?**

☐ Discordo Totalmente
☐ Discordo
☐ Não Tenho Certeza

☐ Concordo

☐ Concordo Totalmente

2. Você considera que foi interessante conversar com *UMbot*?

☐ Discordo Totalmente

☐ Discordo

☐ Não Tenho Certeza

☐ Concordo

☐ Concordo Totalmente

3. UMbot conseguiu entender as suas perguntas e respostas?

☐ Discordo Totalmente

☐ Discordo

☐ Não Tenho Certeza

☐ Concordo

☐ Concordo Totalmente

4. Ao executar as tarefas solicitadas, você prefere usar o chatbot ou a interface tradicional?

☐ Discordo Totalmente

☐ Discordo

☐ Não Tenho Certeza

☐ Concordo

☐ Concordo Totalmente

5. Você conversaria novamente com UMbot?

☐ Discordo Totalmente

☐ Discordo

☐ Não Tenho Certeza

☐ Concordo

☐ Concordo Totalmente

Comentários: