INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
IFCE *CAMPUS* TAUÁ
TECNOLOGIA EM TELEMÁTICA

Douglas Rodrigues de Sousa

Desenvolvimento de um *chatbot* para atendimento automático em pizzarias

Tauá – CE

20 de março de 2021

DOUGLAS RODRIGUES DE SOUSA

DESENVOLVIMENTO DE UM *CHATBOT* PARA ATENDIMENTO AUTOMÁTICO EM PIZZARIAS

Trabalho de Conclusão de Curso (TCC) apresentado ao curso Tecnologia em Telemática do Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) – *Campus* Tauá, como requisito parcial para obtenção do Título de Tecnólogo em Telemática.

Área de concentração: Engenharia de Software.

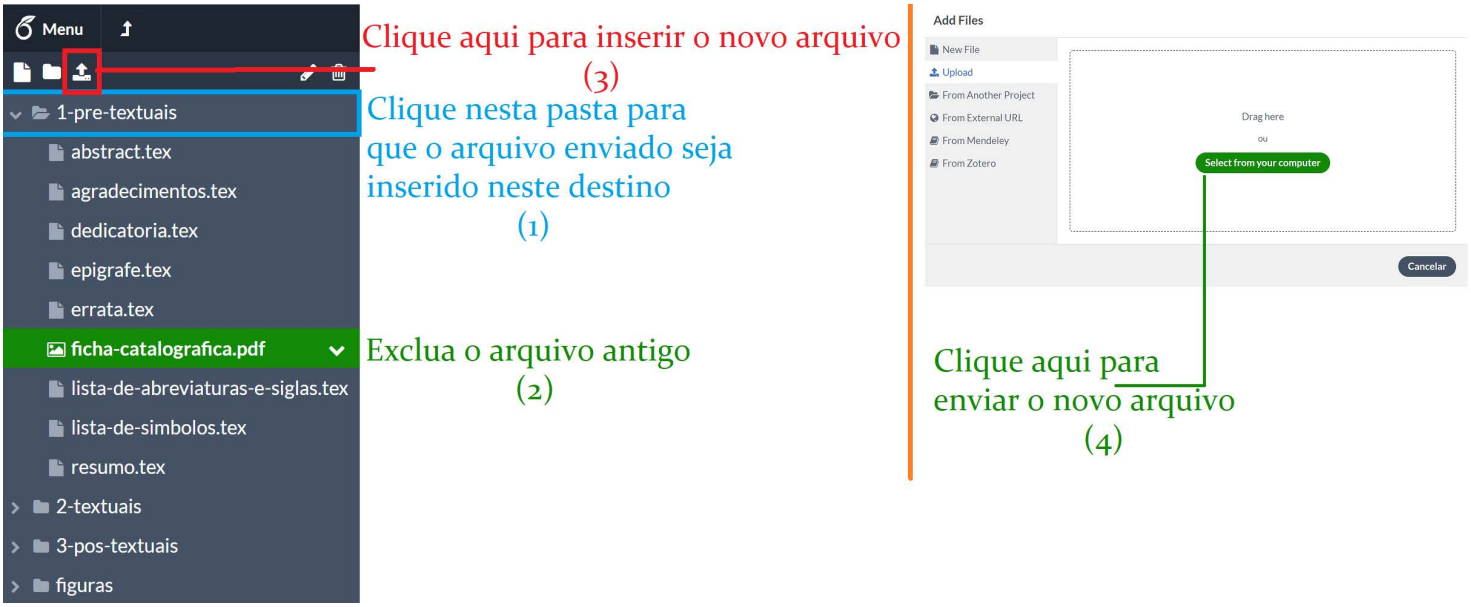
TAUÁ
2020

Para criar sua ficha catalográfica, preencha corretamente o Módulo de Elaboração de Fichas Catalográficas (CATALOG!) disponibilizado no link:

http://fichacatalografica.ufc.br/

Em seguida, deve-se renomear o arquivo gerado como "ficha-catalografica" e adicioná-lo ao *template* na pasta "1-pre-textuais". É necessário, contudo, excluir o antigo arquivo "ficha-catalografica" antes de adicionar o novo.

A figura a seguir mostra os passos enumerados para a inclusão da ficha catalográfica no *Overleaf*.



Douglas Rodrigues de Sousa

Desenvolvimento de um *chatbot* para atendimento automático em pizzarias

Trabalho de Conclusão de Curso (TCC) apresentado ao curso Tecnologia em Telemática do Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) – *Campus* Tauá, como requisito parcial para obtenção do Título de Tecnólogo em Telemática.

Área de concentração: Engenharia de Software.

Aprovado (a) em:

BANCA EXAMINADORA

Orientador: Prof. Me. Saulo Anderson Freitas de Oliveira
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
IFCE *Campus* Tauá

Coorientador: Prof. Me. José Alexandre de Castro Bezerra Filho
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
IFCE *Campus* Tauá

Prof. Me. Adonias Caetano de Oliveira
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
IFCE *Campus* Tianguá

Prof. Me. Lucas Silva de Sousa
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
IFCE *Campus* Umirim

AGRADECIMENTOS

Agradeço aos meus pais, familiares e amigos pelo amor, carinho e incentivo que sempre me deram para a minha formação acadêmica. Ao professor Mestre Saulo Anderson por aceitar ser meu orientador e pelo seu tempo, competência, paciência e dedicação em me guiar nas etapas para desenvolvimento do projeto. Ao meu co-orientador Mestre José Alexandre que desde o inicio acreditou na ideia do trabalho, deu todo o apoio, seu tempo, competência e paciência para me ajudar neste trabalho. E ao IFCE Tauá e todo seu corpo docente, por todo o conhecimento, e experiencia que tive durante a graduação.

"Não importa o que aconteça, continue a nadar."
(WALTERS, GRAHAM; PROCURANDO NEMO, 2003.)

RESUMO

Este trabalho trata do desenvolvimento de um *chatbot* destinado ao atendimento de clientes de pizzarias utilizando ferramentas de processamento de Linguagem Natural e reconhecimento de padrões, voltados à plataforma Telegram. A ideia do *chatbot* é solucionar o problema de sobrecarga de atendimentos, pois na maioria das vezes o atendimento é feito somente por um funcionário por meio telefônico ou aplicativos de trocas de mensagens, tais como Whatsapp, Telegram, Facebook Messenger, entre outros. Dessa maneira, os clientes esperam muito tempo para serem atendidos, pois dependem da capacidade do estabelecimento de lidar com as solicitações. O projeto implementa técnicas de reconhecimento de padrões para o processamento e classificação das mensagens, visando assim auxiliar pizzarias nos atendimentos e questões referentes a dúvidas dos clientes. Os resultados do projeto foram considerados satisfatórios haja vista que a maioria das perguntas feitas ao sistema foram respondidas de forma correta.

Palavras Chave: Atendimento de clientes. Chatbot. Processamento de Linguagem Natural. Naïve Bayes. spaCe. Telegram.

ABSTRACT

This work deals with developing a chatbot aimed at serving pizza shop customers using Natural language processing tools and pattern recognition, aimed at the Telegram platform. The main idea is to build a chatbot to deal with overload problem from calls. Most of the time, the service is done only by one employee by telephone or message exchange applications, such as Whatsapp, Telegram, and Facebook Messenger. In this way, customers end up waiting a long time to be served because they depend on the establishment's ability to deal with inquiries. The project implements pattern recognition techniques for the processing and classifying messages, thus assisting pizzerias in attending and questions related to customer queries. Finally, this work presents the results obtained by implementing and future works. The project results were considered satisfactory, given that most of the questions asked to the system were answered correctly.

Keywords: Customer service. Chatbot. Natural Language Processing. Naive Bayes. spaCy. Telegram.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
PLN	Processamento de Linguagem Natural
SSML	*Speech Synthesis Markup Language*
SEI	Sistema Eletrônico de informações
REST	*Representational State Transfer*
NLTK	*Natural Language Toolkit*
AIML	*Intelligence Markup Language*
IBM	*International Business Machines Corporation*
PHP	PHP: Hypertext Preprocessor
MTP	*Mobile Transport Protocol*
CBOW	*Continuous bag-of-words*
POS	*Part of Speech*
SDB	*Sentence Boundary Detection*
NER	*Named Entity Recognition*
EL	*Entity Linking*
PER	*Person*
RF	Requisito Funcional
RNF	Requisito Não-Funcional
HTTP	*HyperText Transfer Protocol*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

A crescente evolução dos meios de comunicações e com o avanço do Aprendizagem de Máquina (do inglês, *Machine Learning*) e a democratização da Inteligência Artificial (IA), os *bots* começaram a ganhar mais habilidades e passaram a resolver problemas mais complexos do que simplesmente responder perguntas contribuindo assim para o desenvolvimento de novas aplicações que facilitam na comunicação entre usuários e empresas. Uma IA que dispõe de características humanizadas em diálogos processados em linguagem natural é algo muito buscado pelos desenvolvedores. Na década de 50, Alan Turing propôs um desafio para determinar se uma sistema era ou não inteligente com testes feitos com duas pessoas e um computador. Caso não fosse capaz de distinguir um humano de uma maquina por trocas de mensagens, o sistema era considerado inteligente (FOSSATTI; RABELLO; MARCHI, 2011).

Bots[1] estão cada vez mais comuns no nosso dia a dia, e o fato dos aplicativos de mensagens ou redes sociais serem usados como plataformas principais para comunicações utilizando a internet, considera-se cada vez mais vantajoso a construção de *chatbots* (*Bots* de conversa) para este domínio.

Um grande diferencial do uso de *chatbots*, seja para atendimentos, vendas, ou, consultas, é a disponibilidade 24 horas. Pois os usuários estão passando a ser mais exigentes à rapidez nos atendimentos independentemente de horários (Edison Figueira, 2020). Além disso, os *bots* oferecem diferentes funcionalidades de implementação, desde apoio ao cliente, informações, até vendas, pagamentos e transferências. Eles ainda garantem solução de problemas, aberturas de protocolos, coleta de informações e dentre outras funções de maneira rápida, fácil e realizado durante 24 horas por dia. Muitas vezes o atendimento humano é indispensável e a implementação de *chatbots* para tratamento de tarefas repetitivas que antes eram feitas por humanos, podem ser realizadas de forma automática, reduzindo custos e tempo no atendimento.

Segundo Take (2019), o *chatbot* simplifica o modo de marketing e vendas das empresas, tornando assim uma gestão da comunicação mais ágil diminuindo gastos com aquisições de clientes. Adicionalmente, proporciona grande automação para o atendimento de clientes, tais como a solução de dúvidas, em que *chatbots* solucionam dúvidas dos clientes acerca de produtos ou serviços prestados pela empresa. Aponta-se que há uma previsão que em 2021 cerca de 50% das empresas gastarão mais em desenvolvimentos de *chatbots* do que em desenvolvimentos de aplicações móveis (PLUMMER et al., 2017).

Na área alimentícia, o investimento de *chatbots* para atendimento ao cliente ainda é algo muito novo, pois muitos restaurantes, redes *fast food*, pizzarias, dentre outros, fazem o uso de

[1] Diminutivo de *robot*, também conhecido como Internet *bot* ou *web robot*, é uma aplicação de software concebido para simular ações humanas repetidas vezes de maneira padrão, da mesma forma como faria um robô. Fonte: https://pt.wikipedia.org/wiki/Bot. Acessado em: 11 de outubro de 2020.

aplicativos de comunicação via mensagem como WhatsApp ou Telegram, além da comunicação por telefone com os clientes. Dependendo da empresa, esses métodos podem gerar sobrecarga de pedidos, ao mesmo tempo que fazem com que se gere filas de espera, aumentando o tempo de espera para cada pedido. O contexto da pandemia de COVID-19 também trouxe bastante protagonismo aos *chatbots*, haja visto que clientes dos mais diversos segmentos tendem a procurar cada vez mais atendimentos online. Assim, as empresas estão se adaptando perante esta grande demanda.

Atualmente, existem aplicativos e sites que auxiliam na venda de comidas online, por exemplo o iFood que é um serviço de *delivery* onde este disponibiliza cardápios, preços e formas de pagamentos bastante práticos e de forma facilitada, permitindo mapeamento dos restaurantes mais próximos ao endereço do cliente. Para pizzarias ou outros estabelecimentos que oferecem o serviço de *delivery* a utilização dessa tecnologia tem como objetivo principal o atendimento sem limite, já que o atendimento telefônico em maioria das vezes consegue responder apenas um pedido de cada vez, o *chatbot* é capaz de atender vários pedidos ao mesmo tempo (Chatbot Maker, 2018)

Exposto isso, acompanhando este momento de inovação tecnológica, segurança na saúde em tempos de pandemia, e o aumento real de procura de atendimento mais rápidos e onlines, o presente trabalho deve-se ao desenvolvimento de um *chatbot* que opera em Telegram, com servidor que entende e anota os pedidos dos clientes, agilizando e diminuindo o tempo de atendimento dos clientes.

1.1 Objetivos

A seguir, são elencados os objetivos deste trabalho de forma geral e específica.

Objetivo geral

O presente trabalho tem como objetivo desenvolver um sistema de atendimento automático em pizzarias por meio da plataforma de comunicação Telegram e processamento de linguagem natural para agilizar e facilitar o atendimento de clientes.

Objetivos específicos

Os objetivos específicos do presente trabalho seguem os pontos abaixo:

- Realizar uma pesquisa bibliográfica sobre *chatbots* e integração com sistemas;
- Definir o ambiente mais adequado para o desenvolvimento do sistema;
- Realizar de analises na plataforma escolhida que auxilia na criação de *chatbots*;
- Avaliar os resultados gerados pelo sistema em diálogos corriqueiros.

1.2 Organização

O restante deste trabalho é dividido em cinco capítulos. No Capítulo 2, é condensado alguns trabalhos relacionados recentes à temática de *chatbots* para atendimento. O referencial teórico está descrito no Capítulo 3, definindo conceitos de Inteligência Artificial, *Chatbots*, Plataformas de mensagens que oferecem suporte a *chatbots* e alguns conceitos de Processamento de Linguagem Natural que foram usadas no projeto, técnicas de representação de textos, as capacidades linguísticas computacionais e por fim o processo de classificação de textos utilizando o Naïve Bayes. Em seguida, no Capítulo 4, são apresentados a utilização das ferramentas e técnicas apresentadas no Capítulo 3 e os resultados. Além disso, nesse capítulo, abordamos organização do sistema, os requisitos funcionais e não-funcionais, a arquitetura e o modelo de implementação aplicadas no desenvolvimento do *chatbot* e suas funcionalidades. Os resultados obtidos são apresentados no Capítulo 5, incluindo exemplos de simulação de conversas entre o usuário e o *bot*. E por fim, no Capítulo 6, são apresentadas as considerações finais e alterações futuras do trabalho.

2 TRABALHOS RELACIONADOS

Neste capítulo são abordados trabalhos relacionados ao desenvolvimento de *chatbots*.

Cunha, Silva e Moura (2019) apresentam o desenvolvimento de um *chatbot* para atendimento a clientes de farmácias. O *chatbot* desse trabalho tem como objetivo retornar informações requisitadas por clientes, tais como o preço de um medicamento, a quantidade de comprimidos que vem na embalagem, a quantidade de miligramas ou mililitros, patologia de doenças, dentre outras informações. A análise do texto feita pelo sistema é dividida em três etapas através de consultas em listas de palavras ou um banco de dados. Na primeira etapa, são analisados textos referentes a saudações. Quando o texto recebido pelo usuário não é uma saudação é passado para a segunda etapa que utiliza etiquetagem do texto com técnica de PLN para identificar qual o verbo e o padrão linguístico da frase. Caso não encontre uma resposta adequada passando pelas duas etapas anteriores, o texto é passado para a terceira e ultima etapa. Na última etapa, é feito o mesmo processo da segunda etapa, mas utilizando uma lista de palavras referente ao ramo farmacêutico.

Em Dias et al. (2019), os autores desenvolveram um assistente virtual chamado SUSi que visa auxiliar as ferramentas de comunicações do Ministério da Saúde para atendimento de usuários. O SUSi utiliza ferramentas como API Dialogflow usada para desenvolvimento de fluxos de diálogos, além da implementação de conversão de texto para voz com o *Speech Synthesis Markup Language* (SSML) para tornar a voz de SUSi mais humanizada. Além disso, o *chatbot* utiliza o Sistema Eletrônico de informações (SEI)[1], visando a substituição de documentos físicos gerados por suportes por suas respectivas versões digitais. O estudo se mostra viável possibilidade de utilização no Ministério de Saúde, podendo ser ampliado e melhorado em relação a funções apresentadas no projeto.

Oliveira et al. (2019) propuseram um protótipo de um *chatbot*, chamado HelpCare, com mecanismos IA. A solução foi desenvolvida com a ferramenta IBM Watson, processando textos com PLN, para auxiliar pacientes que sofrem com doenças crônicas, tais como diabetes, colesterol alto e hipertensão arterial.

Em Leite et al. (2019), os autores projetaram e implementaram um assistente virtual para atendimento de clientes de uma empresa imobiliária Brognoli para processamento de informações sobre pós-venda. O sistema foi projetado a fim de responder perguntas frequentes dos clientes de maneira correta e rápida. A plataforma para criação do assistente virtual foi a blip.ia, que é um ambiente de desenvolvimento, integração e monitoramento de *chatbots*, possuindo estruturas de PLN e fluxo de blocos de conversação utilizando menus com perguntas e respostas. Além disso, o sistema faz uso do provedor de IA, o IBM Watson, que direciona a conversa com o

[1] O SEI é um sistema de produção e gestão de documentos e processos eletrônicos desenvolvido pelo Tribunal Regional Federal da 4ª Região (TRF4) e cedido gratuitamente à administração pública.

usuário para um fluxo especifico com menus. O projeto teve ótimos resultados e permanece ativo e funcionando na empresa escolhida para realização dos testes.

Eyno (2019) desenvolveu um assistente virtual acessível a diversas interfaces tais como terminais, aplicativos com suporte a bate-papo, e solicitações REST para um servidor. O assistente virtual é capaz de executar *scripts* que são mapeados de acordo com o padrão encontrado em palavras analisadas pelo sistema, podendo procurar respostas para perguntas dos usuários e informações pessoais de uma pessoa solicitada de sua base de dados. O projeto usa como ferramentas o Scikit-learn que é uma biblioteca de aprendizado de máquina para Python, Flask, uma microestrutura de desenvolvimento web do Python, e o Telegram bot API, uma interface para criação e disponibilização de *bots* para a plataforma Telegram. Os testes tiveram bons resultados no processamento de perguntas e solicitações de informações pessoais. Para o autor, o projeto pode ser melhorado para que funcione com execuções de tarefas de outros programas em um servidor ou busca de informações.

Valmorbida e Hart (2020) desenvolveram uma API para *chatbots* com foco em vendas e gestão de seguros, utilizando técnicas de PLN e de IA, ao qual permite comunicações com diversas aplicações de comunicações disponíveis, tais como Whatsapp, Telegram, Facebook Messenger e outros. O projeto utiliza a biblioteca *Natural Language Toolkit* (NLTK) em Python para processar mensagens recebidas de usuários e respondê-las de maneira adequada. A API processa as mensagens recebidas e as envia para o servidor utilizando chaves de autenticação para garantia de segurança. Após o envio da informação pelo usuário, é feito o processamento e a classificação da informação com base em um medidor de confiança para se processar somente informações que a API considere de alto grau de confiança. Para definir esse grau de confiança, é utilizado um valor de probabilidade (entre 0 e 1) associado à pontuação atribuída ao processamento do texto. Quanto mais próximo de 1, mais provável é que aquela entrada do usuário corresponda a uma ação mapeada pelo sistema. O sistema teve bons resultados, agilizando processos e melhorando a qualidade de atendimento para empresas do ramo. Ademais, destaca-se que a plataforma de mensagens utilizada nesse trabalho foi a do Facebook Messenger.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Inteligência artificial

Inteligencia artificial (IA) é uma área da conhecimento, bastante estudada da computação, usada para reproduzir comportamentos tais como, raciocínio lógico-matemático, aprendizagem, reconhecimento de padrões e solução de diversos tipos de problemas. Atualmente, a IA tem diversos sub-campos e está presente em diversas aplicações, tais como processamento de textos, diagnósticos de doenças, teoremas matemáticos, xadrez, criação de poesias e entre outros projetos. Também está relacionada a áreas fora da computação, tais como Psicologia, Filosofia e Linguística (NORVIG; RUSSELL, 2011). Nos dias atuais, existem acesso a diversas aplicações que fazem uso de IA, por exemplo os assistentes virtuais que facilitam a execução de tarefas corriqueiras, processando diversos tipos de informações, além de diversas outras funções (Stoodi, 2020).

3.2 Chatbots

Chatbots, ou como também chamados de assistentes virtuais, ou *bots*, são sistemas capazes de processar textos de um ou mais usuários e respondê-los de forma automática e dentro do contexto compreendido. São desenvolvidos para vários propósitos. Os *chatbots* mais antigos eram criados com personificações e diálogos que simulavam conversas, seja elas para prover informações científicas ou diálogos humorísticos[1].

Nos dias de hoje, os *bots* são usados por diversas empresas com foco maior no atendimento ou venda de produtos aos seus clientes. A seguir são apresentadas segundo NEVES e BARROS (2005) as três gerações de *chatbots*:

a) **Técnica com base em padrões gramaticais e regras de processamento de textos**: funcionando com extração de palavras-chaves contidas no texto;

b) **Técnica com base em IA**: funcionando com regras de processamento de texto e redes neurais;

c) **Técnica com base em linguagens de marcação**: método mais atual para extração de informações de textos utilizando linguagem de marcação baseado em *Artificial Intelligence Markup Language* (AIML)[2].

O primeiro chatbot da história foi a ELIZA (WEIZENBAUM, 1966), projeto desenvolvido entre 1964 e 1966 pelo Joseph Weizenbaum. ELIZA tinha o foco de simular um psicotera-

[1] Como o Akinator. Disponível em: https://pt.akinator.com/. Acessado em: 13 de outubro de 2020.

[2] AIML é uma linguagem baseada em XML, uma linguagem de marcação que foi desenvolvida para criar diálogos semelhante a linguagem natural por meio de softwares, simulando assim inteligência humana. Disponível em: http://www.aiml.foundation/. Acessado em 16 de outubro de 2020.

peuta com processamento de diálogos que buscavam padrões dentro de textos e palavras-chaves enviados pelos usuários, respondendo aos usuários com frases mais próximas com aquele padrão. Um projeto que foi capaz de convencer muitos usuários que interagiam com ela de que não poderia ser uma máquina conversando com eles.

Outro programa considerado uma evolução do ELIZA é ALICE (WALLACE, 2009), desenvolvido por Richard Wallace, o sistema utiliza AIML. ALICE possui seu código aberto e isso trouxe muitos desenvolvedores a contribuírem para o projeto. Outra informação interessante, quando comparada com a sua antecessora ELIZA, é que ALICE possui mais de 40.000 categorias de conhecimento, enquanto ELIZA tinha cerca de 200. ALICE ganhou o prêmio Loebner[3], em um teste de Turing criado por Alan Turing em 1950, anualmente ocorridos nos anos 2000 e 2001. Na época, o *bot* foi considerado o "computador mais humano" pelos juízes.

Ao longo do tempo foram surgindo novos projetos, como IBM Watson[4], que foi desenvolvido com foco em perguntas e respostas, Siri[5], Alexa[6], Cortana[7], que são assistentes virtuais, além de *bots* para plataformas de mensagens, tais como o Facebook Messenger, Telegram e Whatsapp, que permitem interações com contatos e entre outros.

3.2.1 Plataforma de Mensagens

Nesta seção serão apresentadas algumas plataformas de trocas de mensagens.

3.2.1.1 WhatsApp

O WhatsApp[8] também é um aplicativo de troca de mensagens, multiplataforma, disponível para dispositivos móveis, web e *desktop*. Foi desenvolvido em 2009 pelo ucraniano chamado Jan Koum nos Estados Unidos. Em 2014, o aplicativo foi comprado pelo Facebook, mas continua operando como um aplicativo independente.

3.2.1.2 Facebook Messenger

O Facebook Messenger é um aplicativo de troca de mensagens que também oferece suporte a chamadas de vídeo. Em 2015, a empresa anunciou que o messenger seria agora uma plataforma de aplicativos oferecendo suporte para integração de aplicações como, chatbots, jogos e diversas outros projetos aos desenvolvedores para tornar conversas entre usuários mais praticas.

[3] O Prêmio Loebner é uma competição anual de inteligência artificial que premia os programas de computador considerados pelos jurados como os mais semelhantes aos humanos. Disponível em: https://aisb.org.uk/aisb-events/. Acessado em 16 de outubro de 2020

[4] IMB Watson. Disponível em: https://www.ibm.com/watson/br-pt/. Acessado em 16 de outubro de 2020.

[5] Siri - Apple. Disponível em: https://www.apple.com/br/siri/. Acessado em 16 de outubro de 2020.

[6] Amazon Alexa Official Site: What is Alexa?. Disponível em: https://developer.amazon.com/pt-BR/alexa. Acessado em 16 de outubro de 2020.

[7] O que é a Cortana? Disponível em: https://support.microsoft.com/pt-br/help/17214/cortana-what-is. Acessado em: 16 de outubro de 2020.

[8] Disponível em: https://www.whatsapp.com/?lang=pt_br. Acessado em: 15 de outubro de 2020.

Dentre as plataformas descritas acima, a plataforma Telegram foi escolhida para o desenvolvimento do presente projeto por conta de suas várias funções disponíveis e do fácil desenvolvimento por meio de sua API que é disponível em diversas linguagens de programação, tais como, PHP, Node.js, Rust, Python, Ruby, Swift, Kotlin, Java, Go, C#, Elixir, C++, Dart, Lua, OCami, Haskell, Scala e Perl. Outro adicional que guiou esta escolha, foi o suporte a teclados personalizados que a plataforma oferece.

3.2.1.3 Telegram

O Telegram (Telegram, 2020b), é um aplicativo de troca de mensagens multiplataforma, ou seja disponível para dispositivos móveis, *web* e *desktop*, com foco em velocidade e segurança. Além disso, a plataforma armazena todos os dados de mensagens em nuvem, disponíveis para acesso em tempo real de qualquer dispositivo com baixo consumo de armazenamento.

Foi desenvolvido pelo Nikolai e Pavel Durov, na Rússia. Para tornar o Telegram possível, Nikolai desenvolveu um protocolo MTProto exclusivo[9], que é aberto, seguro e otimizado para o trabalho com vários *data centers*. Um grande diferencial da plataforma é o suporte a API[10] de fácil acesso para criação de *bots*, em diversas linguagens de programação.

3.2.2 Processamento de comandos em bots

Nesta seção foi realizado uma revisão bibliográfica sobre as principais funções, características e arquiteturas das ferramentas de processamento de textos utilizadas no projeto que foram: vetorização, word2vec, spaCy, e classificação de ações com classificador Baysiano.

3.2.2.1 Expressões regulares

Uma expressão regular permite formalmente especificar padrões de textos, com uso de meta-caracteres que são símbolos com funções especiais (JARGAS, 2009). Comumente apelidado de Regex (do inglês, *Regular Expression*), a sua origem se deu pelos estudos feitos pelos neurologistas Warren McCulloch e Walter Pitts que descreveram uma tese sobre o funcionamento dos neurônios, em que mais tarde, Stephen Kleene passou o estudo para notação matemática. Esses estudos foram aprofundados por muitos matemáticos da época por cerca de vinte anos, até que em 1968 surgiu o primeiro algoritmo de busca utilizado em um editor de textos chamado `qed` que depois deu origem ao `grep`, um aplicativo desenvolvido para sistemas baseados em Unix que aceitava representações regulares e comandos (JARGAS, 2009).

Em 1986, foi criado o primeiro pacote na linguagem C chamado `regex` que tratava de expressões regulares e disponíveis para importação de forma gratuita a qualquer projeto que o

[9] MTProto Mobile Protocol . Disponível em: https://core.telegram.org/mtproto. Acessado em: 19 de outubro de 2020.

[10] Interface de Programação de Aplicação é um conjunto de rotinas e padrões de programação para acesso a um aplicativo ou plataforma baseado na Web.

necessitasse. Com o passar do tempo, o `regex` foi sendo cada vez mais utilizado em programas e adaptado a diversas linguagens de programação (JARGAS, 2009).

Em Regex, padrões simples podem ser construídos de forma simples, considerando que deseja-se encontrar uma correspondência direta, por exemplo, utilizando o padrão `/abc/`, é encontrado combinações de caracteres em sequências somente quando os caracteres `"abc"` forem encontrados juntos e na ordem especificada. Um exemplo de um texto como `"Meu nome é abcel"` é encontrado. No entanto, no texto `"Meu nome é abrelc"` a expressão não é encontrada, pois o padrão não está na ordem especificada (MDN, 2020).

Também existem padrões que visam buscar expressões com mais de uma ocorrência no texto. Por exemplo, para se encontrar uma correspondência do caractere `"a"` seguido de espaço ou um ou mais `"b"` seguido de `"c"`, a expressão a ser utilizada para corresponder este padrão seria `"/ab*c/"`. Nela, o símbolo `"*"` seleciona zero ou mais ocorrências da parte à esquerda da expressão (nesse caso, só o `"b"`). Aplicando a expressão tanto no texto `"abbbc"`, quanto no texto `"a c"`, o padrão também seria encontrado (MDN, 2020).

Além desses meta-caracteres, existem outros também, como o de ocorrência, alternativa e definição de grupos e conjunto. Os principais estão descritos na Tabela 1.

3.3 Processamento de Linguagem Natural

Processamento de linguagem natural (PLN) (JURAFSKY, 2000) é uma subárea de IA que utiliza um conjunto de técnicas da computação com processamento de dados em alguma linguagem referente a comunicação do ser humano. Sendo o mais comum, textos como artigos, documentos ou qualquer outro tipo de informação em formato de texto. Extraindo informações que sejam consideradas importantes da fonte.

O trabalho em processamento de linguagem natural tende a visualizar o processo de análise de linguagem sendo decomposta para compreender padrões linguísticos de palavras como, substantivo, adjetivo, verbo e outros, de forma hierárquica para realização de análise lexical, sintática, semântica e pragmática. (INDURKHYA; DAMERAU, 2010). Com isso PLN é muito utilizado em diversas aplicações que dependem da extração de grandes quantidades de informações de textos. As diversas técnicas de PLN incluem extração de informações, correção ortográfica, recuperação de informações, reconhecimento de fala, tradução de palavras, entre outros.

Com o avanço da tecnologia tornou-se possível implementar boas ideias de projetos que ajudaram ao PLN ser o que é hoje. Os trabalhos relacionados ao PLN surgiram no final da década de 1940, com o foco em tradução automática (*machine translatio*n). Uma das primeiras pesquisas no campo foi um trabalho de tradução automática do Russo para o Inglês, em um experimento rudimentar e limitado, o qual foi exibido na demonstração IBM-Georgetown em 1954 (AGUIAR, 2016).

Tabela 1 – Principais Meta-caracteres empregados em Regex.

Meta-caracteres	Descrição	Exemplos
.	Coringa de um caracter	vi.a
[ ]	Coincide com qualquer um dos caracteres listados	[gpr]ato
[^]	Coincide com qualquer um dos caracteres, exceto os listados	os [^mf]ato
?	O caracter anterior pode aparecer ou não	meios?
*	O caracter anterior pode aparecer em qualquer quantidade	go*gle
+	O caracter anterior deve aparecer no mínimo uma vez	go+gle
{}	O caracter anterior deve aparecer na quantidade indicada	go{1,5}gle
^	Coincide com o começo da linha	^rio
$	Coincide com o fim da linha	mente$
\b	Limita uma palavra (letras, números e sublinhado)	\b(meu)
\	Torna os metacaracteres caracteres comuns	sério\?
|	Atua como operador "ou".	(colfu)mo
( )	Faz com que vários caracteres sejam vistos como um só	(sai)?rei
.*	Qualquer caracter, em qualquer quantidade.	eu.*você
*?	Semelhante ao asterisco	
+?	Semelhante ao sinal de mais.	
{}?	Semelhante às chaves simples.	

Fonte – Expressões Regulares. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Regular_Expressions. Acessado em: 07 de agosto de 2020.

As subseções que seguem, descrevem os principais conceitos utilizados em um projeto que envolve o PLN, a saber, representação de texto, reconhecimento de entidades e classificação de intenção. O primeiro conceito está relacionado a uma técnica para ajudar a chegar a essa abstração de texto para análise, enquanto os dois últimos são formas de abstrair o texto para análise propriamente dito.

3.3.1 Representação de texto

Vetorização

A vetorização é um processo muito comum em PLN, o qual consiste em avaliar matematicamente cada palavra de um texto. Quando um texto passa pelo processo de vetorização, é aplicada alguma técnica de processamento de linguagem em que cada palavra do texto passa por uma transformação e o texto todo é transformado em vetor de tamanho de acordo com processo a qual posteriormente esse texto será submetido.

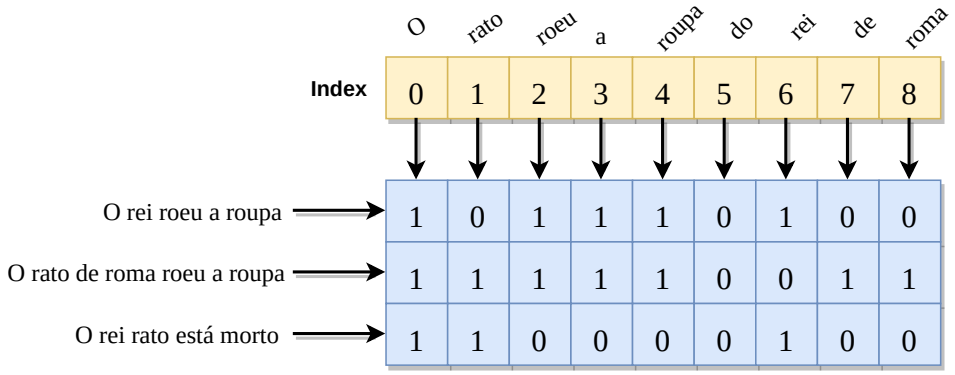
Para realizar essa transformação, alguns algoritmos utilizam diversas características

diferentes das palavras para realizar a vetorização. Uma delas é a codificação *one-hot* que consiste em codificar cada palavra com um número único, ou *embeddings* de palavras. o uso destes algorítimos se torna variável de acordo com o problema em questão. Os algoritmos classificam as palavras processadas em vetores de diversos tamanhos e em várias línguas, incluindo o português (JÚNIOR, 2018).

Codificação One-Hot

O algoritmo de codificação *One-Hot* codifica cada palavra com 0 e 1. Por exemplo a frase `"O rato roeu a roupa do rei de Roma"`, tem o seguinte vocabulário: `(O, rato, roeu, a, roupa, do, rei, de, Roma)`. Para representar cada palavra, é criado um vetor preenchido com 0 e comprimento igual ao do vocabulário em que a cada vocábulo é adicionado 1 no índice que corresponde a palavra (TensorFlow, 2020). Um exemplo dessa codificação é apresentado na Figura 1.

Figura 1 – Codificação *One-Hot*.



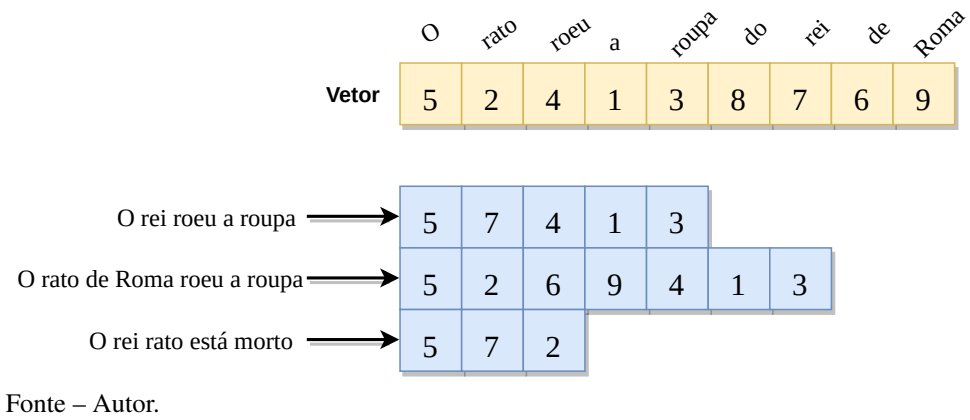
Fonte – Autor.

Essa abordagem, apesar de simples, é pouco utilizada, pois supondo que um vocabulário tenha 10.000 palavras, para codificar 1 a cada palavra é criado um vetor onde $99,99\%$ dos elementos são 0.

Codificar cada palavra com um número único

Uma segunda abordagem é codificar cada palavra usando um número único. Continuando com o exemplo acima, poderíamos atribuir 1 à palavra `"a"`, 2 à `"rato"` e assim por diante, como ilustrado na Figura 2.

Figura 2 – Codificar cada palavra a um único número.



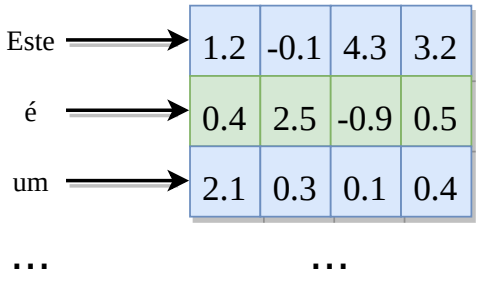
Fonte – Autor.

Além disso, pode-se então codificar a frase `"O rato roeu a roupa do rei de Roma"` como um vetor denso como $[5, 2, 4, 1, 3, 8, 7, 6, 9]$. Essa abordagem é eficiente, pois em vez de um vetor esparso, agora tem-se um vetor denso (em que todas as posições estão preenchidas com valores $\neq 0$) (TensorFlow, 2020).

Embeddings de palavras

Os *embeddings* de palavras fornecem uma maneira de usar uma representação eficiente e densa em que palavras semelhantes têm uma codificação semelhante. Um *embedding* é um vetor denso de valores de ponto flutuante ilustrado na Figura 3 em que se pode observar um vetor quadridimensional com pesos aprendidos. Depois que esses pesos são aprendidos, pode-se codificar cada palavra procurando o vetor denso que a corresponde na tabela.

Figura 3 – Exemplo de *embedding* quadridimensional.



Fonte – Adaptado de TensorFlow (2020).

Essa representação é mais atraente para grande quantidades de informações, pois em vez de especificar os valores para a incorporação manual, eles são parâmetros treináveis, aprendidos. É comum ver *embeddings* de palavras que são 8 dimensionais (para pequenos conjuntos de dados) ou até 1.024 dimensões, quando se trabalha com grandes conjuntos de dados. Uma incorporação

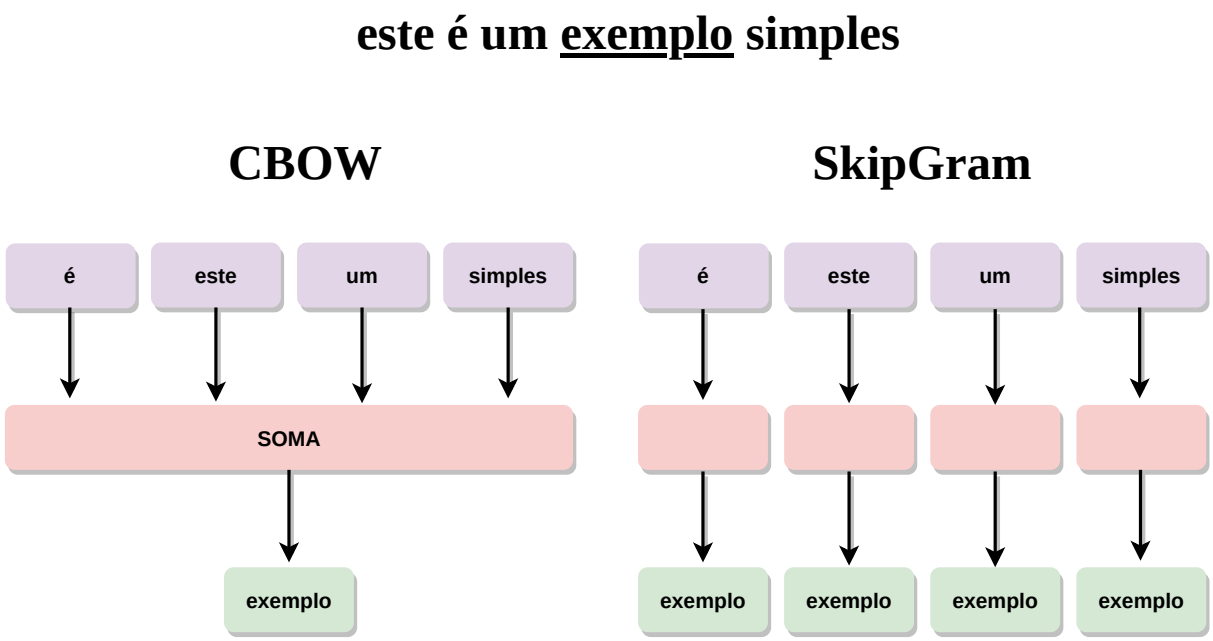
de dimensão maiores pode capturar relacionamentos refinados entre as palavras, mas é preciso fornecer um grande numero de dados para aprendizagem (TensorFlow, 2020).

Word2Vec

Um dos algoritmos mais comum é o Word2Vec (MIKOLOV et al., 2013), uma ferramenta *open-source* utilizada para calcular palavras provenientes de um texto como vetores e classificar similaridades. Em um dos seus exemplos, é possível relacionar semelhanças entre as palavras vetorizadas em seu exemplo, o `vetor("Rei") - vetor("Homem") + vetor("Mulher")` resulta em um vetor que está mais próximo da representação vetorial da palavra `vetor("Rainha")`, mas isso não quer dizer que o vetor resultado seja igual aos vetores da operação como `"Rei"`, `"Homem"` e `"Mulher"`, mas sim a palavra `Rainha` é a mais próxima do vetor encontrado. Esta similaridade corresponde a `"Rei"` está para `"Homem"`, da mesma forma que `"Mulher"` está para `"Rainha"`. Estas similaridades estão presentes em qualquer que seja o idioma, não limitando-se somente ao inglês.

Neste contexto, são destacadas duas arquiteturas para realizar essas representações, a saber, *continuous bag-of-words* (CBOW) e SkipGram. Na arquitetura SkipGram, tem-se uma representação mais eficiente de palavras quando a quantidade de dados é menor. Já na arquiterura CBOW, além de ser mais rápida de se obter uma representação, é aplicável em grande quantidade de dados. Na Figura 4 é ilustrado o funcionamento de ambas arquiteturas.

Figura 4 – Diferença entre as arquiteturas de treinamento SkipGram e CBOW.



Fonte – Adaptado de Kavita Ganesan (2020).

Na Figura 4, a mesma frase `"este é um exemplo simples"` é adicionada ao fluxo das arquiteturas do Word2Vec. É possível perceber que o modelo CBOW aprende a prever uma

palavra-alvo com base em seu contexto. Os vetores do contexto são somados para prever a palavra-alvo. Já o modelo SkipGram, ele aprende a prever uma palavra com base em uma palavra vizinha. As palavras vizinhas são levadas em consideração e, a partir daí, são determinadas por um tamanho de janela predefinido ao redor da palavra alvo. Um tamanho de janela típico pode ser 5, significando 5 palavras anteriores e 5 palavras posteriores totalizando 10 palavras.

3.3.2 Capacidades linguísticas computacionais

Processar texto bruto de forma inteligente é difícil: a maioria das palavras são raras, e é comum que palavras que parecem completamente diferentes signifiquem quase a mesma coisa. As mesmas palavras em uma ordem diferente podem significar algo completamente diferente. Até mesmo dividir o texto em unidades semelhantes a palavras úteis pode ser difícil em muitos idiomas. Embora seja possível resolver alguns problemas começando apenas com os caracteres brutos, geralmente é melhor usar o conhecimento linguístico para adicionar informações úteis (Explosion AI, 2020).

Dentre as capacidades relacionadas ao conhecimento linguístico, destacam-se:

a) **Tokenização (*Tokenization*)**: Segmenta o texto em palavras, sinais de pontuação, dentre outros;

b) **Marcação de partes do texto (*Part-of-speech tagging*, POS)**: Atribui tipos gramaticais a *tokens*, tais como verbo ou substantivo;

c) **Análise de dependência (*Dependency Parsing*)**: Atribuição de rótulos de dependência sintática, descrevendo as relações entre *tokens* individuais, tais como sujeito ou objeto;

d) **Lematização (*Lemmatization*)**: Atribui formas básicas às palavras. Por exemplo, o lema de `"era"` é `"ser"` enquanto que o lema de `"ratos"` é `"rato"`;

e) **Detecção de limite de sentença (*Sentence Boundary Detection*, SBD)**: Encontra e segmenta frases individualmente;

f) **Reconhecimento de entidades (*Named Entity Recognition*, NER)**: Rotula objetos nomeados do `"mundo real"`, tais como pessoas, empresas ou locais;

g) **Associação de Entidades (*Entity Linking*, EL)**: Atua como elemento de desambiguação em entidades textuais para as identificador de forma única em uma Base de Conhecimento;

h) **Similaridade**: Capacidade de se comparar palavras, extensões de texto e documentos e indicar o grau de semelhança entre si;

i) **Classificação de textos:** Atribuição de categorias ou rótulos a um documento inteiro ou partes de um documento;

j) **Correspondências com base regras (*Rule-based Matching*)**: Encontra sequências de *tokens* com base em seus textos e anotações linguísticas, semelhantes a expressões regulares;

As subseções que seguem vão descrever as capacidades utilizadas mais a frente neste Trabalho de Conclusão de Curso.

3.3.2.1 Tokenização

A tokenização consiste em dividir um texto em segmentos, chamados de *tokens*. Esses segmentos podem ser palavras, pontuação, números ou outros caracteres especiais, a depender do idioma utilizado. Em Srinivasa-Desikan (2018), é mostrado um exemplo de como a tokenização da frase `"Let us go to the park"` acontece. No entanto, para fins didáticos, consideremos um caso de tokenização em português da seguinte frase: `"Jesus é o jardineiro e as árvores somos nós."`, cujo resultado está exemplificado na Figura 5.

Figura 5 – Exemplo de tokenização.

	0	1	2	3	4	5	6	7	8	9
Token	Jesus	é	o	jardineiro	e	as	árvores	somos	nós	.

Fonte – Autor.

3.3.2.2 Marcação de partes do texto

Nesta etapa, uma marcação de cada *token* da frase com sua parte gramatical apropriada é definida, como substantivo, verbo e assim por diante.

Figura 6 – Exemplo de marcação dos *tokens* com classes gramaticais.

	0	1	2	3	4	5	6	7	8	9
Token	Jesus	é	o	jardineiro	e	as	árvores	somos	nós	.
Marca	SUBST.	VERBO	ARTIGO	ADJETIVO	CONJ.	ARTIGO	SUBST.	VERBO	PRON.	PONT.

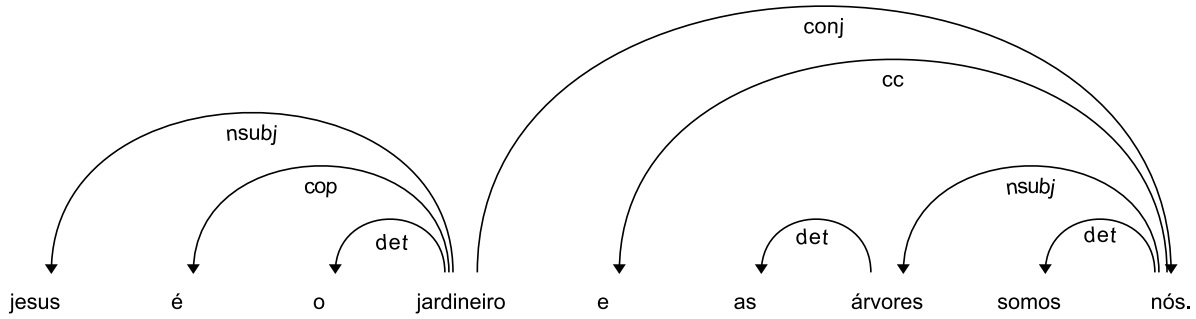
Fonte – Autor.

3.3.2.3 Análise de dependência

Nesta etapa, a aplicação de um analisador relaciona a dependência entre os *tokens*, buscando a compreensão das dependências, como ilustrado na Figura 7.

A compreensão das dependências ajuda a identificação de relações em textos. Por exemplo, pode-se automatizar a busca e análise de biografias profissionais através da extração de nomes de pessoas e de empresas por meio da análise de dependências. Buscando-se o termo

Figura 7 – Exemplo de análise de dependência.



Considerando os relacionamentos:

- `nsubj`: Sujeito nominal.
- `cop`: Verbo de ligação.
- `det`: Determinante.
- `cc`: Conjunção coordenativa.
- `conj`: Conjunção.

Fonte – Autor.

`"trabalho"`, independentemente do tempo verbal, e em seguida, verificar se os nomes das empresas estão associados ao termo e se existe algum termo indicando sujeito que indique quem é a pessoa (spaCy, 2020).

3.3.2.4 Reconhecimento de entidade nomeada

O reconhecimento de entidade nomeada consiste em dar nomes específicos ao contexto da aplicação aos *tokens* processados e classificados no texto.

Figura 8 – Exemplo do reconhecimento de entidade nomeada.

Fonte – Autor.

Na Figura 8, pode-se ver o reconhecimento de `Jesus` como uma entidade do tipo pessoa (com marcação `PER`). Este tipo de reconhecimento de entidade é muito utilizado para detectar pessoas, nacionalidades, construções, monumentos, aeroportos, companhias, objetos, veículos, músicas, moedas, dentre outros.

3.3.3 Classificação de texto

Apesar de muito úteis, as capacidades linguísticas são somente a ponta do *iceberg*. Ainda é preciso reconhecer a ação que aquela frase estará associada. Como as ações do sistema de

chatbot estão definidas *a priori*, esta tarefa de reconhecimento se torna uma classificação de ação em uma dada categoria. A tarefa de classificação consiste em separar algo (textos, imagens, áudios) em categorias distintas. Atualmente, as técnicas de aprendizagem de máquina, que serão introduzidas aqui, são vastamente utilizados para isso. De modo geral, a ideia é prever o significado de uma frase ou a intenção do usuário ao enviar uma palavra ou frase.

Para realizar a classificação do texto do usuário, utiliza-se uma base de conhecimento ou base de treinamento contendo possíveis formas de comunicação que o usuário pode fazer. Quanto mais informações essa bases tiver, maior será a capacidade em prever uma ação ao texto de entrada. Na Tabela 2 são apresentados exemplos de como textos poderiam ser classificados e sua respectiva ação associada.

Tabela 2 – Exemplos de textos de entrada e reconhecimento da ação.

TEXTO	**AÇÃO**
“Gostaria de uma maçã”	`PEDIDO`
“Boa Noite”	`SAUDAÇÃO`
“Fazem entrega?”	`INFORMAÇÃO`
“Qual o cardápio?”	`CARDÁPIO`

Fonte – Autor.

O processo de classificação é de extrema relevância para um *chatbot*, sendo este responsável por fornecer a resposta mais adequada dada entrada de texto do usuário. Nas subseções que seguem, será introduzido o conceito do Teorema de Bayes e por conseguinte, o classificador que deriva desse teorema. Uma explicação mais detalhada de como o esse classificador será utilizado para classificar textos (ações para o *chatbot*) foi apresentado no Capítulo 4.

3.3.3.1 Teorema de Bayes

O teorema de Bayes foi desenvolvido pelo Thomas Bayes, um inglês, ministro e matemático do século XVIII, para determinar probabilidades de eventos incorporando informações sobre eventos subsequentes (TRIOLA, 2010). Do conceito de Probabilidade Condicional, oriundo da Estatística elementar, nota-se que a probabilidade condicional de um evento é uma probabilidade obtida com a informação adicional de algum outro evento já ocorrido. A equação pode ser utilizada para encontrar $P(B|A)$ ou seja, a probabilidade de B dada ocorrência de A:

$$P(B \mid A) = \frac{P(A \mid B) \times P(B)}{P(A)}, \tag{3.1}$$

em que $P(A)$ e $P(B)$ são probabilidades *a priori*, ou seja, são valores de probabilidade inicial originalmente obtido antes de qualquer informação adicional serem obtidas de A e B, ambas $P(B \mid A)$ e $P(A \mid B)$, probabilidades condicionais, correspondem a probabilidade *a posteriori* e verossimilhança, respectivamente.

Na teoria da probabilidade, a regra de Bayes mostra a relação entre uma probabilidade condicional e sua inversa, ou seja, a probabilidade de um efeito dada a observação de uma causa e a probabilidade da causa dada pelo efeito (NORVIG; RUSSELL, 2011). Por exemplo, um médico sabe que a meningite faz o paciente ter uma rigidez no pescoço, 70% do tempo. O médico também conhece alguns fatos incondicionais como: a probabilidade *a priori* de um paciente ter meningite é de $\frac{1}{50.000} = 0,00002$, e a probabilidade *a priori* de qualquer paciente ter rigidez no pescoço é de 1%. Sendo S a proposição de que o paciente tem rigidez no pescoço e M a proposição de que o paciente tem meningite, então tem-se:

$$P\left(M \mid S\right) = \frac{P\left(S \mid M\right) P\left(M\right)}{P\left(S\right)} = \frac{0,7 \times 0,00002}{0,01} = 0,0014. \tag{3.2}$$

Logo, a probabilidade de o paciente que tem rigidez no pescoço ter meningite é de $0,14\%$.

3.3.3.2 Classificador ingênuo de Bayes (Naïve Bayes)

Classificador Naïve Bayes e construído com base no Teorema de Bayes. Ele possui este naïve (ingênuo) pela premissa de independência entre variáveis a serem classificadas. O classificar é muito utilizado em diversos tipos de aplicações, se adaptando muito bem até mesmo à escala de problemas muito grandes (NORVIG; RUSSELL, 2011).

Naïve Bayes é uma técnica simples para construção de classificadores, ou seja modelos que atribuem rótulos de classe a instancias do problema, representadas como vetores de características. Um vetor de características pode ser representado por $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ com n representando o número de características de uma das variáveis a ser classificada. Para se classificar uma instancia $\mathbf{x}$ cuja classe é desconhecida, supondo a existência de m classes, $C_1, C_2, \ldots, C_m$, o classificador atribui a classe que possui maior probabilidade *a posteriori* $P\left(C_k \mid \mathbf{x}\right)$. Fazendo com que, $\mathbf{x}$ seja associado a classe C_k, somente se:

$$P\left(C_k \mid \mathbf{x}\right) > P\left(C_j \mid \mathbf{x}\right) \;\; \forall j, 1 \leq j \leq m, j \neq k, \tag{3.3}$$

assim, para se descobrir qual classe será atribuída a instancia $\mathbf{x}$, basta encontrar a classe C_k que possui a maior probabilidade de $P(C_k \mid \mathbf{x})$ (MERSCHMANN, 2007).

Ademais, como expresso na Equação 3.1, o cálculo da probabilidade *a posteriori* pode ser reescrito, negligenciando $P(\mathbf{x})$, resultando em $P\left(C_1 \mid \mathbf{x}\right) \propto P\left(\mathbf{x} \mid C_1\right) \times P\left(C_1\right)$. As probabilidades *a priori* de cada classe $P(C_i)$ é obtida da base de treino:

$$P\left(C_i\right) = \frac{\mid C_i \mid}{n}, \tag{3.4}$$

em que $\mid C_i \mid$ é o numero de instâncias pertencentes à classe C_i e n é o total de todas as instâncias de treino. Além disso, como o classificador assume independência condicional entre os atributos, a função de verossimilhança $P\left(\mathbf{x} \mid C_i\right)$ é expressa como:

$$P\left(\mathbf{x} \mid C_i\right) = \prod_{j=1}^{n} P\left(x_j \mid C_i\right). \tag{3.5}$$

Aplicando em um exemplo simples, como o da Base de Dados apresentado na Tabela 3 em que existem os atributos A_1 e A_2, aos quais descrevem cinco instâncias pertencentes as classes C_1 ou C_2. Para predizer qual classe será definida para $\mathbf{x} = [x_1, x_2] = [1, 1]$, a predição comparará as probabilidades de $P(C_1 \mid \mathbf{x})$ com $P(C_2 \mid \mathbf{x})$, atribuindo sendo a que possui maior probabilidade.

Tabela 3 – Exemplo de Base de dados.

Atributo #1 (x_1)	Atributo #2 (x_2)	Classe
1	0	C_1
0	0	C_1
2	1	C_2
1	2	C_2
0	1	C_1

Fonte – Adaptado de Merschmann (2007).

As probabilidades *a priori* das classes são obtidas pela Equação 3.4, resultando em $P(C_1) = \frac{3}{5} = 0,6$ e $P(C_2) = \frac{2}{5} = 0,4$. Assim, conforme dados da Tabela 3, pode-se calcular as funções de verossimilhança de cada classe (Equação 3.5) da seguinte maneira:

$$P(\mathbf{x} \mid C_1) = P(x_1 = 1 \mid C_1) \times P(x_2 = 1 \mid C_1) = \frac{1}{3} \times \frac{1}{3} = \frac{1}{9} = 0,11. \tag{3.6}$$

$$P(\mathbf{x} \mid C_2) = P(x_1 = 1 \mid C_2) \times P(x_2 = 1 \mid C_2) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} = 0,25. \tag{3.7}$$

Finalmente, em posse de ambos os valores de probabilidade *a priori* e verossimilhança, podemos calcular as probabilidades *a posteriori* de cada classe e, assim, identificar a classe com maior probabilidade:

$$P(C_1 \mid \mathbf{x}) \propto 0,11 \times 0,6 = 0,066 \quad \text{e} \quad P(C_2 \mid \mathbf{x}) \propto 0,25 \times 0,4 = 0,1. \tag{3.8}$$

Como, neste caso, $P(C_2 \mid \mathbf{x})$ possui maior probabilidade, o classificador atribui à $\mathbf{x}$ a classe C_2.

4 PROJETO E IMPLEMENTAÇÃO

O presente trabalho consiste no desenvolvimento de um *chatbot* para pizzarias, Para que o usuário faça interação com o sistema é necessário ter acesso à plataforma por meio de um *smartphone*, computador ou *tablet*, no qual será feito o processamento das mensagens pela API do Telegram.

O sistema proposto busca solucionar um problema referente à sobrecarga no atendimento de clientes. Isso acontece porque, atualmente, grande parte dos estabelecimentos que oferecem formas de contato para atendimento é realizado por meio telefônico, ou aplicativos de mensagens, tais como Telegram e Whatsapp, gerenciados por um número pequeno de pessoas. Geralmente, essas pessoas conseguem responder apenas um pedido por vez e, dependendo do dia, horário e porte de solicitações, podem gerar uma sobrecarga fazendo com que clientes acabem esperando muito mais tempo para serem atendidos.

Em um mundo cada vez mais conectado, os usuários tendem a exigir que suas solicitações de informações ou dúvidas sejam solucionadas o mais rápido o possível. Com isso, empresas de vários setores tendem a investir cada vez mais em *chatbots* seja para auxiliar funcionários destinando as dúvidas e informações simples para um *bot* e informações que precisam ser tratadas pelo funcionário como também para vendas de produtos.

Com este cenário atual de pandemia de COVID-19, a demanda por pedidos aumentou consideravelmente fazendo com que a quantidade de solicitações também acompanhasse esse crescimento.

Desta forma, o *chatbot* foi pensado para que os clientes por meio de um *smartphone*, computador ou *tablet*, e acesso a plataforma Telegram, possa fazer um pedido, ou solicitações de informações ou dúvidas de forma rápida e fácil. E também disponibilizar as informações de cada pedido e endereço de cada usuário que solicitou um pedido de forma confiável para os estabelecimentos.

4.1 Projeto

Nesta seção, são descritos os atores do sistema, os requisitos funcionais e não-funcionais que foram analisados com base nos trabalhos relacionados, a implementação do sistema e as ferramentas utilizadas no projeto. Além disso, será explicado como é feito o processamento das mensagens dos usuários com as ferramentas utilizadas. Foi escolhido utilizar o nível de descrição classificando como requisitos de usuário, isto é, "declarações, em uma linguagem natural como diagramas, de quais serviços o sistema deverá fornecer a seus usuários e as restrições com as quais este deve operar". (SOMMERVILLE, 2011)

4.1.1 Analise de requisitos

Nesta seção são especificados os requisitos da aplicação proposta em duas partes: requisitos funcionais e não funcionais.

4.1.1.1 Requisitos funcionais

a) **RF01 – Iniciar conversa:** Toda vez que um usuário que iniciar uma conversa com o sistema pela primeira vez, ele receberá uma saudação, apresentando as funcionalidades e todos os comandos que podem ser interpretados ali pelo sistema;

b) **RF02 – Processar entrada:** Ao iniciar uma conversa, os usuários podem optar por definir um comando especifico para o que desejam, tais como `/pedido` ou `/help`, por exemplo, ou simplesmente fazer o pedido de forma direta, tal como `"Quero uma pizza de queijo"`. Neste caso, o usuário pode fornecer um texto gramaticalmente correto ou com pequenos erros de digitação e, ainda assim, o sistema deverá vincular alguma ação à entrada do usuário e retornar uma resposta referente a esta;

c) **RF03 – Leitura de respostas rápidas:** O sistema deve reconhecer se entrada do usuário é uma ação correspondente a uma resposta simples como `"SIM"` ou `"NÃO"`;

d) **RF04 – Frases não entendidas:** O sistema deve informar todos os comandos que o sistema consegue entender, caso não consiga encontrar uma resposta a ação que foi atrelada a entrada do usuário;

e) **RF05 – Finalizar conversa por *timeout*:** Usuários que iniciarem uma conversa e deixarem de responder por cinco (05) minutos terão a conversa encerrada pelo sistema. Dessa forma, o estado da conversa deve voltar ao início por conta deste encerramento;

f) **RF06 – Preencher dados faltantes do pedido:** O sistema deve ser capaz de identificar informações faltantes na entrada do usuário, tais como tamanho, sabor, quantidade e produto;

g) **RF07 – Solicitar endereço:** Ao concluir um pedido, o sistema deve requisitar informações de endereço do cliente, tais como rua, bairro, número e contato, e confirmar se dados passados pelos usuários estão corretos;

h) **RF08 – Armazenamento de histórico:** O sistema deve ser capaz de armazenar o histórico da conversa do usuário toda vez que um pedido for concluído.

4.1.1.2 Requisitos não-funcionais

a) **RNF01 – Idioma:** O sistema deve apresentar interação com o usuário na língua portuguesa;

b) **RNF02 – Linguagem de programação:** O sistema deve ser desenvolvido na linguagem de programação Python;

c) **RNF03 – Disponível a Plataforma Telegram:** O sistema deve estar disponível via plataforma de comunicação Telegram;

d) **RNF04 – Servidor Python:** O sistema deve operar através de um servidor python que se comunica com a API da plataforma Telegram;

e) **RNF05 – Base de conhecimento:** O sistema deve ser capaz de realizar o processamento de um arquivo com exemplos de palavras processado para gerar a base de conhecimento via ferramenta Scikit-learn;

f) **RNF06 – Desempenho:** O sistema deve ser capaz de obter respostas para o usuário o mais rápido o possível;

g) **RNF07 – Confiabilidade:** O sistema deve um taxa de acerto elevada, de forma a retornar a resposta mais adequada a solicitação do usuário;

h) **RNF08 – Fácil usabilidade:** Facilidade na acessibilidade ao *bot*, apresentando exemplos de usabilidade ao usuário;

4.1.2 Identificação dos atores do sistema

O sistema possui dois (02) atores primários: usuários que são clientes do estabelecimento e o secundário que é o estabelecimento. O ator **Cliente** fornece informações sobre os produtos que deseja. Ele é responsável por iniciar a conversa e inserir os dados requisitados pelo sistema para se completar um pedido. Já o ator **Estabelecimento**, recebe todo o histórico de conversas dos clientes ao se encerrar um pedido, assim como é o responsável pelo envio dos produtos solicitados pelos clientes.

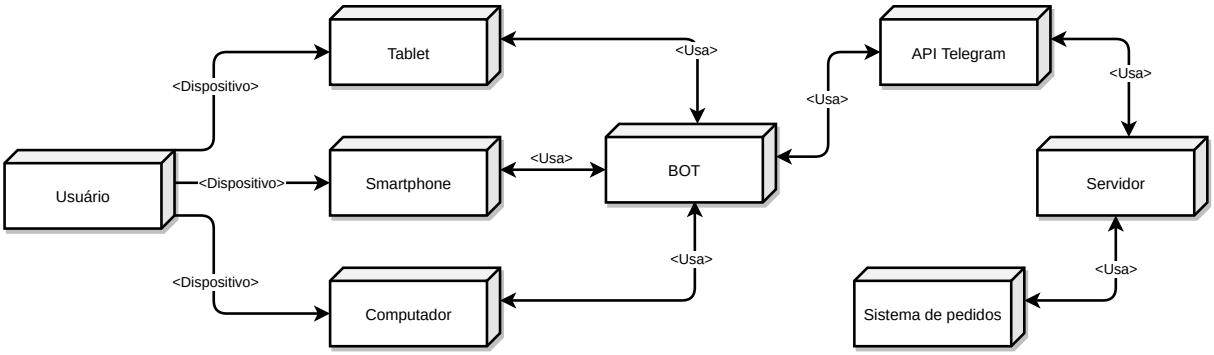
4.2 Implementação

Nesta seção é apresentada um modelo de arquitetura utilizado para construção do projeto, definindo o funcionamento de cada módulo, assim como as ferramentas e como foram utilizadas para realizar o processamento de mensagens.

4.2.1 Arquitetura

A Figura 9 mostra uma visão geral da arquitetura do sistema.

Figura 9 – Modelo de implementação.



Fonte – Autor.

Para que a comunicação com o usuário seja feita, o sistema deve ser executada em um servidor Python, podendo ser de forma local em um computador com internet, como também via nuvem. Existem diversas plataforma que oferecem serviços de computação em nuvem, como Amazon Web Services (AWS)[1], Google Cloud[2] e alguns outros serviços de hospedagem que também oferecem planos gratuitos e vitalícios, mas com pouco desempenho de processamento como PythonAnywhere[3].

O servidor é responsável pelo processamento da frase enviada pelo usuário, utilizando as ferramentas de processamento de texto e classificador Naïve Bayes que serão apresentadas com mais detalhes nas próximas seções. Ele também é responsável pela comunicação com a API de desenvolvimento de *chatbots* do Telegram para recebimento de mensagens passadas pelos usuários e envio das respostas mais adequadas com base no resultado de seu processamento.

Para iniciar uma conversa, o usuário por meio de um dispositivo conectado a internet e via aplicativo de mensagens instantâneas Telegram, envia uma mensagem podendo ser referente a um pedido, ou solicitação de informações ao bot. A mensagem é identificada pela API e passada ao servidor para ser feito o processamento e classificação da frase. Por fim o sistema retorna a resposta referente a solicitação do usuário.

Quando a solicitação do usuário é um pedido, é passado ao sistema de pedidos todas as informações referente ao pedido e os dados como: nome, endereço e contato para entrega de seu produto.

[1] Amazon Web Services (AWS) - Serviços de computação em nuvem. Disponível em: https://aws.amazon.com/pt/. Acessado em: 19 de outubro de 2020

[2] Serviços de computação em nuvem | Google Cloud. Disponível em: https://cloud.google.com/?hl=pt-br. Acessado em 19 de outubro de 2020

[3] Host, run, and code Python in the cloud: PythonAnywhere. Disponível em: https://www.pythonanywhere.com/. Acessado em 19 de outubro de 2020

4.2.2 Ferramentas

Foram utilizadas algumas ferramentas para o desenvolvimento deste trabalho. Os critérios de escolha para utilização foram feitos através de pesquisas e avaliações realizadas. Abaixo serão listadas e introduzidas as principais ferramentas usadas no projeto: linguagem de programação, softwares e bibliotecas.

4.2.2.1 Python

Python[4] é uma linguagem de programação de código aberto e de programação de alto nível, interpretada, de *script*, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. que permite trabalhar mais rapidamente e integrar seus sistemas de forma mais eficaz.

A escolha dessa linguagem foi motivada pelo fato de necessitar de pouco código e por oferecer suporte a uma boa quantidade de bibliotecas que são necessárias para o desenvolvimento do trabalho.

4.2.2.2 Scikit-learn

Scikit-learn[5] é uma biblioteca de aprendizado de máquina disponível em Python. O objetivo principal é a analise de dados. Também é construído nas bibliotecas NumPy, SciPy e matplotlib, todas em Python. A biblioteca contém algoritmos que fazem classificação, regressão, agrupamento, pré-processamento, além de outras tarefas. Scikit-learn foi escolhida por ser uma biblioteca que cumpre com os objetivos necessários para construção do trabalho. Foi dela que utilizamos o classificado Naïve Bayes.

4.2.2.3 Python Telegram Bot API

Neste trabalho, foi utilizado o projeto `Python-Telegram-bot`[6] que é uma interface baseada em HTTP criada para desenvolvedores interessados em desenvolver *bots* para o Telegram. Esta API permite a conexão de um *bot* ao sistema do Telegram. Os *bots* construídos são contas especiais que não necessitam ser atreladas a um número de telefone e servem como uma interface para o código em execução em algum lugar no servidor criado pelo desenvolvedor. Para se ter acesso a API, é necessário um *token* de autorização que é fornecido na construção do *bot*. Para se construir um *bot*, basta conversar com o `@BotFather`, um *bot* especial desenvolvido pelo Telegram que auxilia os desenvolvedores na criação de um novo *bot*.

[4] Welcome to python.org. Disponível em: https://www.python.org/, Acessado em 16 de outubro de 2020.
[5] Scikit-learn Machine Learning in Python. Disponível em: https://scikit-learn.org/. Acessado em 16 de outubro de 2020.
[6] Python-telegram-bot. Disponível em: https://github.com/python-telegram-bot/python-telegram-bot. Acessado em: 18 de setembro de 2020.

Cada *bot* criado recebe um *token* de autorização único, que também o identifica. O *token* é semelhante à seguinte sequência: `123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11`. A API necessita desse *token* para dar suporte aos métodos `HTTP GET` e `HTTP POST`. Por exemplo, **`https://api.telegram.org/bot`**`<token>`**`/getMe`**, oferecendo, assim, quatro maneiras de passagem de parâmetros em solicitações do *bot*:

a) String de consulta de URL;

b) `application/x-www-form-urleancoded`;

c) `application/json` (exceto para envio de arquivos);

d) `multipart/form-data` (para envio de arquivos).

Existem diversos exemplos de projetos possíveis para criação de *bots* no Telegram e sua API possui diversas funções que auxiliam no desenvolvimento de *chatbots*. A seguir, serão apresentadas algumas das funções disponíveis na API:

a) **Envio de notificações personalizadas**: Capacidade do *bot* enviar notificações sobre noticias, dicas ou avisos aos usuários;

b) **Interação com diversos serviços**: Prover conteúdos provenientes de serviços externos, tais como Gmail *bot*, Youtube *bot*, Wiki *bot*, Github *bot*, entre outros;

c) **Meios de pagamentos**: Um *bot* pode promover serviços pagos ou funcionar como uma loja virtual, oferecendo opções de pagamentos pela própria aplicação;

d) **Ferramentas personalizadas**: Diversas ferramentas podem ser desenvolvidas, tais como alertas, previsões do tempo, traduções, calculadora, ou outros serviços;

e) **Jogos**: A API também oferece suporte a criação de jogos em HTML5, podendo ser desenvolvidos jogos desde fliperamas e quebra-cabeças e até jogos mais complexos com gráficos em 3D ou em tempo real com um ou vários jogadores.

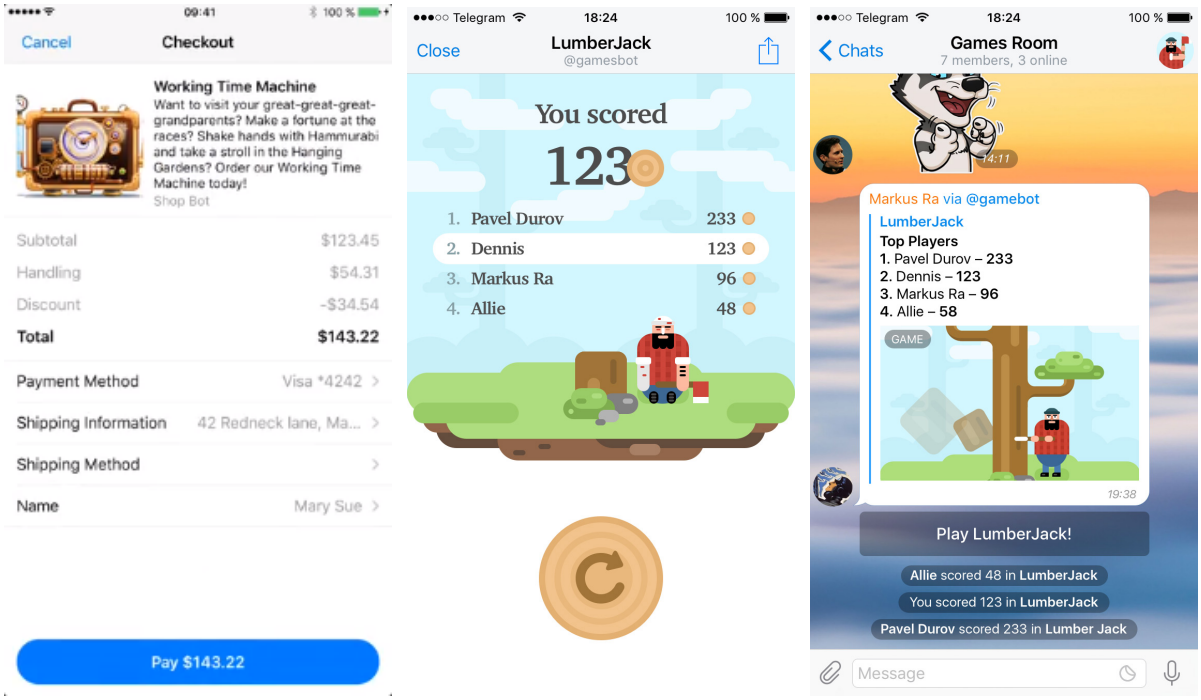
Para ilustrar alguns dos exemplos supracitados, a Figura 10 ilustra opções de pagamento (a) e jogos (b) e (c) dentro da plataforma Telegram.

4.2.2.4 spaCy

A spaCy (Explosion AI, 2020) é uma biblioteca de código aberto e gratuita para PLN avançado em Python. Foi projetada especificamente para criar aplicativos que processam e "entendem" grandes quantidades de texto, sendo muito utilizada em sistemas de extração de informações ou de compreensão de linguagem natural, ou para pré-processar texto para aprendizado profundo.

O código da Figura 11 exemplifica um pouco o nível de abstração de como é feito a nomeação de entidades pelo spaCy usando um idioma próprio de Pizzaria. Nele, após a inclusão da biblioteca via diretiva `import spacy`, seu funcionamento consiste em criar uma instância

Figura 10 – Exemplo algumas das funções disponíveis na API da plataforma Telegram.



(a) Captura de tela da revisão de um pagamento.

(b) Captura de tela de pontuação do jogo LumberJack.

(c) Captura de tela de uma sala de jogos via `@gamesbot`.

Fonte – Telegram (2020a).

Figura 11 – Exemplo de nomeação de entidades pelo spaCy.

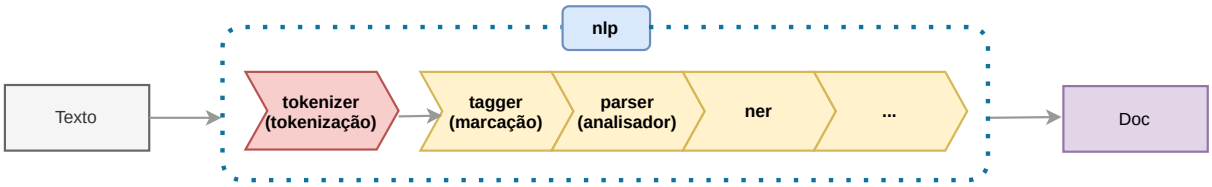
```
import spacy

nlp = spacy.load("Pizzaria_TCC")
doc = nlp("Olá gostaria de uma pizza sabor queijo por favor")

for ent in Doc.ents:
    print(ent.text, ent.label_)

>>>  uma QUANTIDADE
>>>  pizza PIZZA
>>>  queijo SABOR
```

Fonte – Autor.

da um objeto base `nlp` que corresponde a um objeto do tipo Linguagem que contém todos os componentes e dados necessários para se processar texto.

Em posse de um objeto da classe `nlp`, pode-se passar uma sequência codificada no padrão unicode para que então o spaCy possa anotar cada elemento do texto, produzindo assim um objeto do tipo `doc`, conforme o pipeline ilustrado na Figura 12. Por meio do objeto *doc* é acessado os resultados dos processamentos das capacidades linguísticas.

Figura 12 – Pipeline de processamento utilizada no SpaCy.



Fonte – Adaptado de Srinivasa-Desikan (2018).

Das capacidades do spaCy relacionadas à linguística, foram utilizadas no projeto a tokenização e o reconhecimento de entidades para fazer a análise de textos da conversação. A decisão por trás do uso dessas capacidades se deu pela necessidade de se dar "nomes" a objetos reais dos textos recebidos pelos usuários e classificar com base no treinamento específico ao contexto de Pizzarias.

4.2.3 Criação do idioma para o contexto de pizzarias

Quando o texto analisado é identificado como pedido, é necessário identificar as entidades presentes no texto para se derivar os itens do pedido. As entidades detectadas pelo spaCy originalmente identificam elementos comuns, a saber, pessoas, lugares, objetos, dentre outros. Assim, faz-se necessário adicionar entidades específicas ao spaCy relacionadas ao contexto de pedidos. Para tal, foi utilizado um arquivo contendo diversos exemplos de nomes de entidades. Na Figura 13 é possível visualizar parte do arquivo que foi desenvolvido para treino do *bot* com as marcações das entidades.

Figura 13 – Estrutura do arquivo utilizado para criação do idioma para pizzarias.

FRASE	TOKEN	INICIO	FIM	ENTIDADE
Vou querer `uma` pizza `grande` de `queijo` .	"uma"	11	14	QUANTIDADE
	"grande"	21	27	TAMANHO
	"queijo"	31	37	SABOR
Boa noite, eu gostaria de `duas` pizzas `grandes` de `frango` .	"duas"	26	30	QUANTIDADE
	"grandes"	38	45	TAMANHO
	"frango"	49	55	SABOR

Fonte – Autor.

4.2.4 Treinamento do classificador de texto

Para a base de treino do classificador Naïve Bayes foi criado um arquivo de treino com vários exemplos de possíveis frases que possam ser enviadas pelos usuários. Cada frase é

referenciada a uma ação, como observado em Tabela 2).

Em todas as frases o texto é analisado pelo spaCy para filtrar os lemas. Os verbos conjugados vão para o infinitivo e substantivos no plural vão para o singular, dentre outros. Após essa etapa, o texto é vetorizado utilizando a codificação *One-hot* (veja a Figura 2). Esse processo é realizado para que seja possível identificar a verossimilhança do padrão da frase do usuário com os dados treinados.

O texto passado pelo usuário, também passa pelo mesmo processo, pois para usar o texto no Naïve Bayes para classificação, já que o Naïve Bayes trabalha com vetores de características e necessita que os padrões sejam passados como vetores de atributos.

4.2.5 Recebimento de mensagens e classificação de ações do chatbot

Quando o usuário inicia uma conversa com o *chatbot*, o texto é analisado pelo spaCy para se extrair as entidades acerca do domínio de Pizzarias. Assim, o texto é vetorizado utilizando a codificação *One-hot* (veja a Figura 2) para que seja possível usar essa representação de vetor de atributos do texto no classificador.

Quando o *bot* reconhece ou classifica o texto do usuário como pedido é verificado o item do pedido e coletada todas as informações necessárias para o processamento do pedido. Quando a ação encontrada no texto não é um pedido, é analisado o contexto da frase via Expressão Regular para retornar a resposta mais apropriada. Essa resposta pode ser informações sobre o funcionamento do *bot*, dúvidas sobre pagamentos, entregas, dentre outras. Em algumas questões que necessitam respostas do tipo `"sim"` ou `"não"` também é feito o uso de Expressões Regulares, tais como `"faz entrega?"`, ou `"Está correto?"` para a confirmação de telefone ou dados do endereço da entrega.

Este trabalho possui dois módulos para processamento de textos: um módulo referentes a solicitações de informações, cardápio e dúvidas e outro para pedidos. Estes módulos funcionam separadamente. O elemento que indica qual desses módulos processa uma mensagem é um manipulador que a `Python-Telegram-Bot-API` oferece. Na Figura 14 é mostrado todos os estados possíveis que se pode estar.

Figura 14 – Estados do manipulador da Python-Telegram-Bot-API.

```
states = {
    FAZENDO_PEDIDO: [MessageHandler(Filters.text, fazendo_pedido)],
    PIZZA: [MessageHandler(Filters.text, pizza)],
    LANCHE: [MessageHandler(Filters.text, lanche)],
    PEDIR_MAIS: [MessageHandler(Filters.text, pedir_mais)],
    NOME_CLIENTE: [MessageHandler(Filters.text, nome_cliente)],
    ENVIANDO_ENDERECO: [MessageHandler(Filters.text, enviando_endereco)],
    BAIRRO: [MessageHandler(Filters.text, bairro)],
    NUMERO: [MessageHandler(Filters.text, numero)],
    CONTATO: [MessageHandler(Filters.text, contato)],
    CONFIRMAR_ENDERECO: [MessageHandler(Filters.text, confirmar_endereco)],

    ConversationHandler.TIMEOUT: \
        [MessageHandler(Filters.text | Filters.command, timeout)]
}
```

Fonte – Autor.

Cada estado é responsável por solicitar informações para que o pedido seja concluído, abaixo serão apresentados o que cada estado representa:

a) `FAZENDO_PEDIDO`: Quando a mensagem do usuário é identificada como pedido;

b) `PIZZA`: Quando a mensagem do usuário é identificada como pedido de uma pizza;

c) `LANCHE`: Quando a mensagem do usuário é identificada como pedido de um lanche;

d) `PEDIR_MAIS`: Ao concluir um pedido, é perguntado ao usuário se ele deseja mais algum produto. Caso a resposta seja sim, o usuário é redirecionado ao estado `FAZENDO_PEDIDO`. Caso contrário, o usuário é redirecionado para o estado `ENVIANDO_ENDERECO`;

e) `ENVIANDO_ENDERECO`: Ao finalizar, é necessário informações do endereço para entrega do produto;

f) `NOME_CLIENTE`: Solicita o nome do usuário;

g) `BAIRRO`: Solicita o bairro do usuário;

h) `NUMERO`: Solicita o número da residencia do usuário;

i) `CONTATO`: Solicita um contato para caso haja algum problema na localização da residencia do usuário;

j) `CONFIRMAR_ENDERECO`: Pergunta ao usuário se os dados de endereço informado estão corretos.

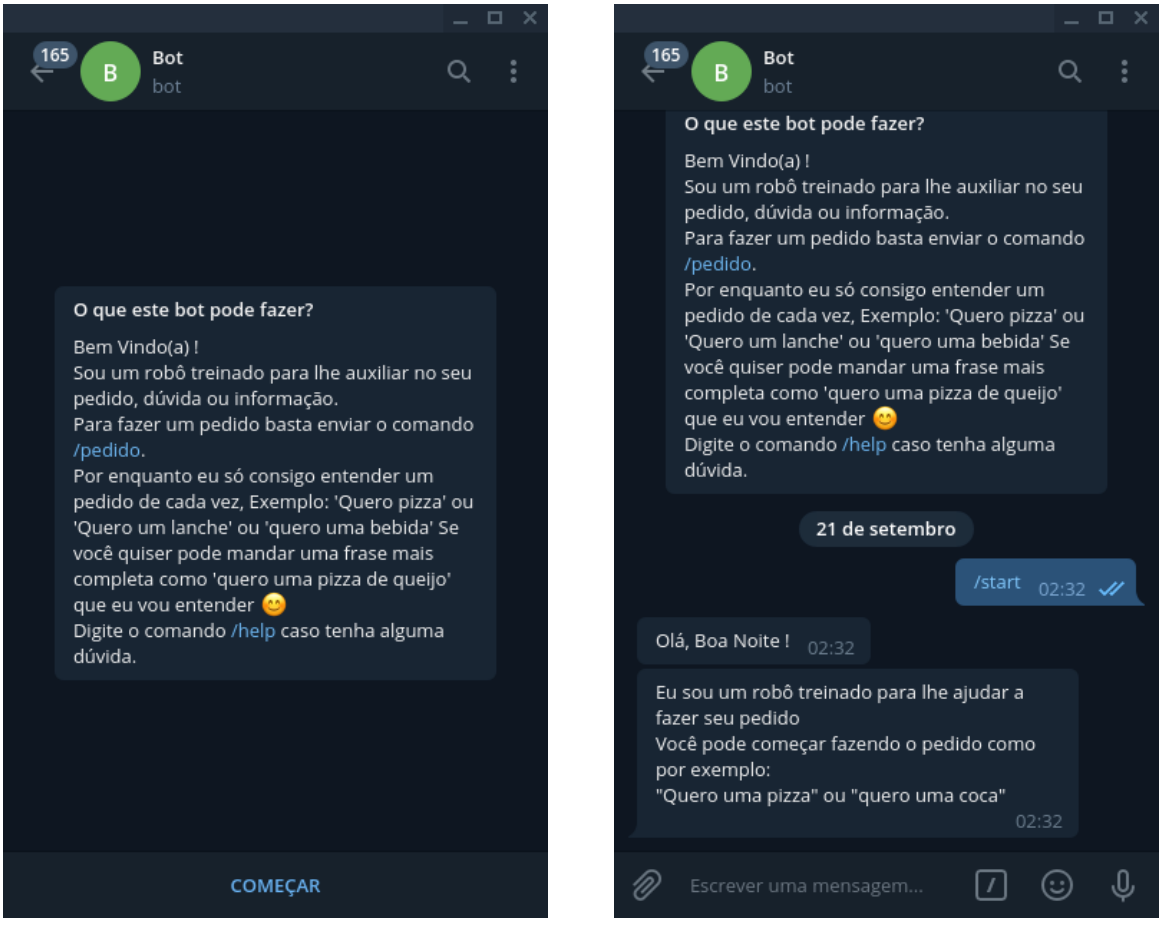
5 RESULTADOS

Neste capítulo, é realizado a análise dos resultados do desenvolvimento do sistema *chatbot*. Serão apresentadas implementação e telas da plataforma com mensagens enviadas e recebidas pelo *bot* e comentários sobre o funcionamento de cada etapa do dialogo do usuário, relacionando-os com os requisitos funcionais do sistema.

5.1 Apresentação e saudação

O construtor `@Botfather` do Telegram, permite editar diversas configurações do *bot*, como, nome, descrição, comandos e foto de perfil. Assim, o *chatbot* foi configurado com uma descrição em que quando o usuário inicia sua sessão no Telegram e interage com o *chatbot* (`RF01`) é exibido exemplos de algumas das funções e comandos entendidos pelo sistema, como mostra a Figura 15 (a).

Figura 15 – Capturas de telas acerca da conversa inicial e saudação do projeto.



(a) Iniciando conversa com o *chatbot*.

(b) Saudando o usuário e exemplificando pedidos.

Fonte – Autor.

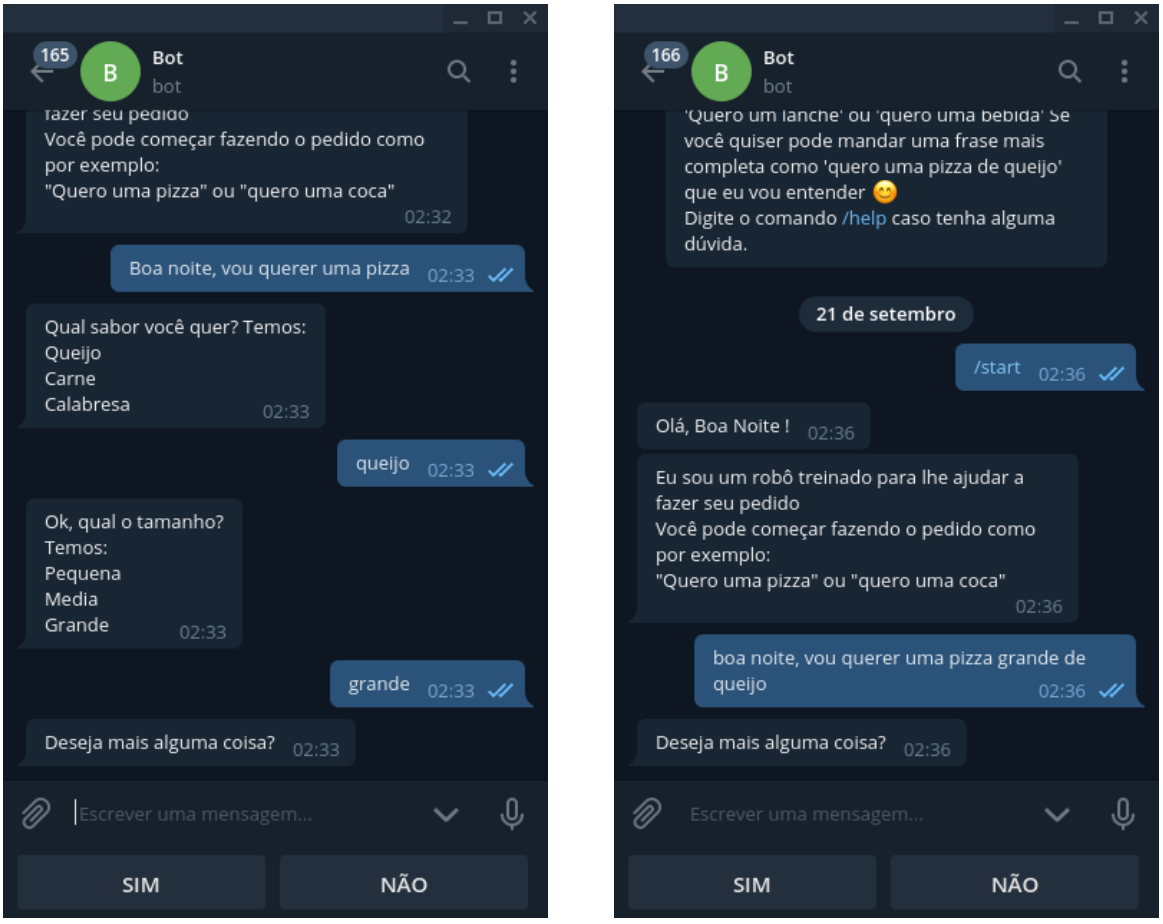
A sessão só começa quando o usuário dá início à conversa na opção `"COMEÇAR"` (veja o botão localizado na parte inferior da Figura 15 (a)). Nesse momento, o usuário não precisa se identificar explicitamente, pois o *chatbot* considera o identificador de usuário no Telegram.

Ao iniciar uma conversa, o *chatbot* responde com uma saudação de acordo com o horário em que a conversa se iniciou. Após essa saudação, uma apresentação do *chatbot* e alguns exemplos de como realizar pedidos é apresentado por ele, conforme ilustrado na Figura 15 (b) em que se pode notar o *chatbot* cumprimentando o usuário com "Olá, boa noite!".

5.2 Processando informações

O processamento de informações é realizado via mensagens de textos (`RF02`), tais como `"Quero uma pizza"`, ou `"quero um suco"`, ou `"gostaria de uma pizza grande de queijo"`, ou também com comandos específicos, a saber, `/pedido`, `/menu` ou `/help`. Após análise pelo sistema, é retornado uma resposta de acordo com o padrão encontrado no texto ou comando. Um exemplo de um pedido de pizza é apresentado na Figura 16 (a).

Figura 16 – Capturas de telas acerca da realização de pedidos.



(a) Realizando pedido incompleto.

(b) Realizando pedido incompleto.

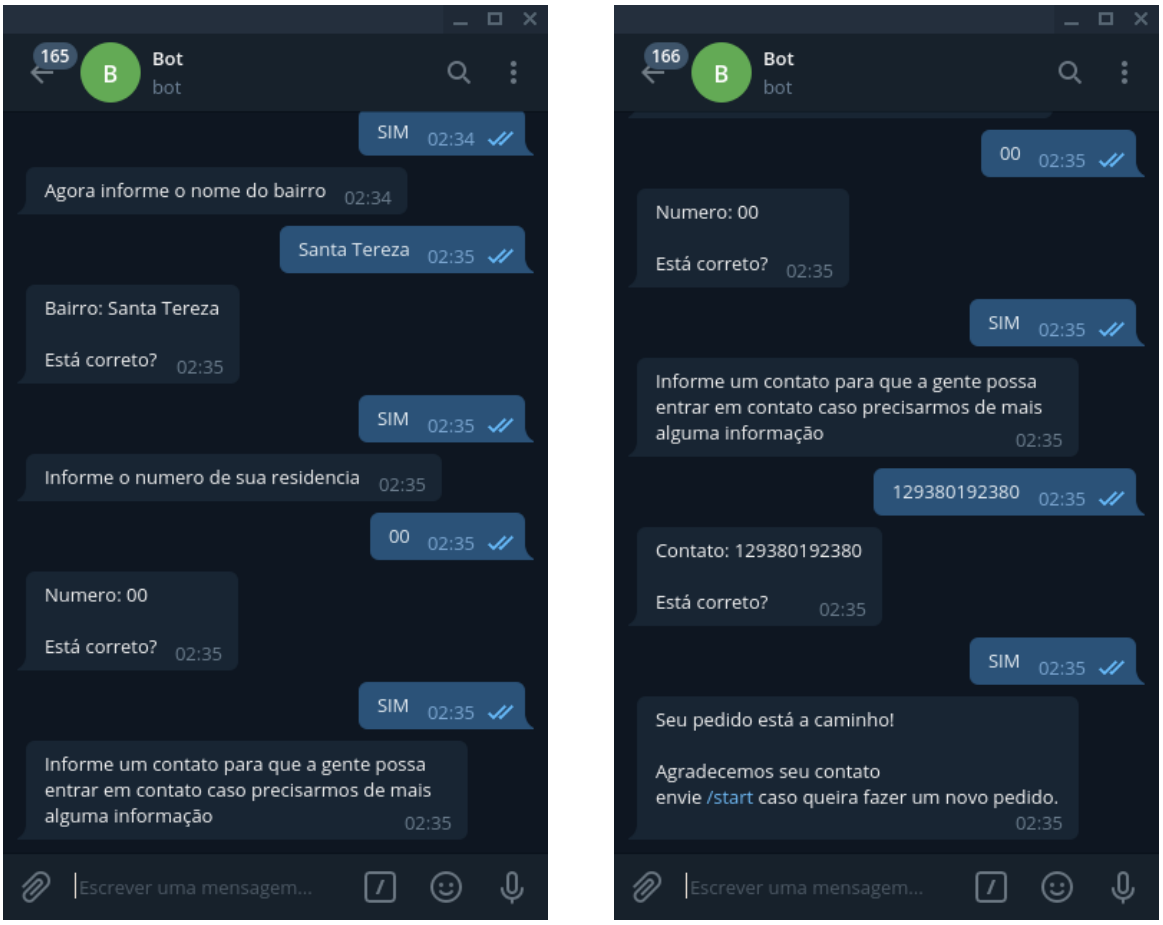
Fonte – Autor.

Dados necessários não fornecidos para concluir o pedido são solicitados até que todos sejam preenchidos (RF06), impedindo que o pedido fique incompleto. O usuário também pode acrescentar um novo pedido ao fim de cada pedido escolhendo a opção SIM (botão localizado na parte inferior). Neste caso, o processo acima é repetido. Na Figura 16 (b) é mostrado um exemplo de um texto com todos os dados necessários para concluir um pedido de pizza.

5.3 Solicitando endereço e finalizando pedido

Com o pedido feito, o *chatbot* solicita seu nome, nome da rua, bairro, número, e um número de telefone para contato caso haja alguma dificuldade para realizar a entrega do pedido (RF07). Com o pedido feito e endereço enviado, é retornado ao usuário que seu pedido será entregue, conforme ilustrado na Figura 17. Adicionalmente, o *chatbot* indica que um novo fluxo de pedido pode ser feito a partir dali.

Figura 17 – Capturas de telas acerca da solicitação de endereço e finalização de pedido.



(a) Informando o endereço.

(b) Finalizando pedido.

Fonte – Autor.

5.4 Solicitando o cardápio e outras informações

O cardápio pode ser solicitado via comando `/menu` ou com mensagens, tais como `"gostaria do cardápio por favor"`, `"quais os sabores de pizzas disponíveis?"`, dentre outros. Quando solicitado, um cardápio interativo é apresentado informando os itens disponíveis, sabores e preços (veja Figura 18).

Figura 18 – Capturas de telas acerca da solicitação de endereço e finalização de pedido.

(a) Cardápio interativo.

(b) Solicitação de informações.

Fonte – Autor.

A escolha do cardápio em formato de menu de opções visa facilitar a leitura dos itens, pois estes são divididos em seções especificas para cada tipo, como mostrado na Figura 18 (a). Pode-se perceber que os itens são catalogados em `Pizzas Salgadas`, `Pizzas Doces`, `Lanches`, `Bebidas` e `Preços das Pizzas`. Optou-se por essa forma de apresentação, pois, ao informar todo o cardápio em um único texto, poderia deixar a navegação cansativa e difícil de se organizar (`RNF01` e `RNF08`).

Além disso, também é possível solicitar informações que podem ser entendidas pelo sistema, tais como se está fazendo entregas, quais os meios de pagamento disponíveis dentre

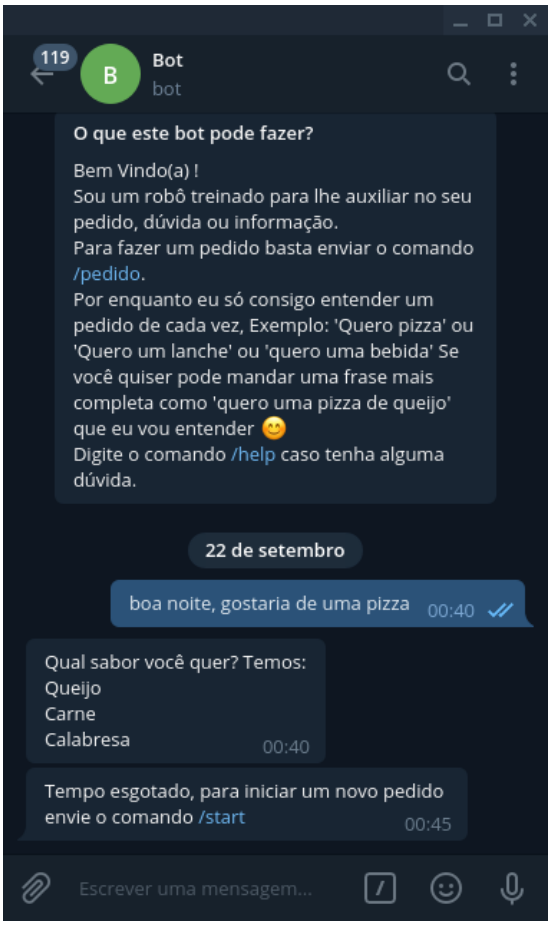
outras, conforme ilustrado na Figura 18 (b).

Ademais, toda vez sistema analisa o texto e verifica se a informação requerida é uma resposta mais simples, ele opta por utilzar um leiaute de teclado na tela com somente os botões referentes a `"sim"` ou `"não"` (RF03).

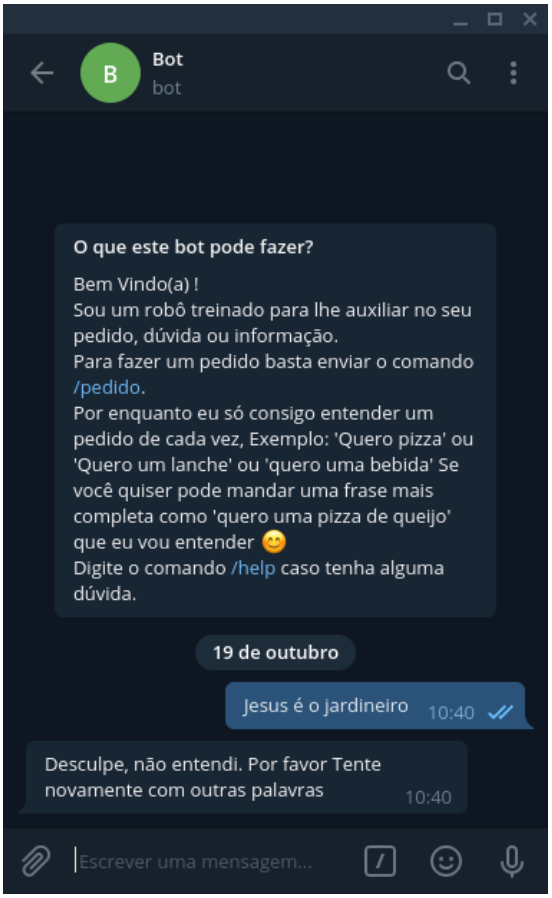
5.5 Timeout e mensagens inteligíveis

O sistema é capaz de finalizar pedidos em que usuários passam mais de cinco (05) minutos sem interagir com o *bot* (RF05). Neste caso, é notificado ao usuário que o tempo foi esgotado e que é necessário refazer o pedido do início, conforme mostrado na Figura 19 (a). Por fim, frases que não seguem nenhum padrão com as informações que são compreendidas pelo sistema levam o *chatbot* a solicitar que o usuário tente novamente com outras palavras (RF04), ver Figura 19 (b).

Figura 19 – Capturas de telas acerca do *timeout* e inteligibilidade da mensagem pelo *chatbot*.



(a) Informando ao usuário em que o tempo foi esgotado.



(b) Informando ao usuário que o texto não pode ser compreendido.

Fonte – Autor.

6 CONCLUSÃO

O presente trabalho teve como objetivo o desenvolvimento de um *chatbot* para atendimentos automáticos de clientes em estabelecimentos de pizzarias ou semelhantes. O projeto foi desenvolvido em Python, utilizando as ferramentas de processamento natural de linguagem e classificação com Naïve Bayes para identificação de ações, contextos das frases via Regex e entidades nomeadas via spaCy nos textos recebidos pelos usuários. Sua base de treino e definições de intenções ainda é feita manualmente alterando o código, sendo possível, futuramente, a implementação de uma estrutura web para adicionar, remover e editar ações e entidades de treino. Foi verificado em testes simulando possíveis solicitações de usuários e há indicações que o *chatbot* funciona corretamente, pois o sistema se mostrou capaz de identificar as ações, as entidades contidas no texto e respondeu corretamente a maioria das mensagens analisadas.

REFERÊNCIAS

AGUIAR, E. M. d. **Aplicação do Word2vec e do Gradiente descendente dstocástico em tradução automática**. Tese (Doutorado), 2016.

Chatbot Maker. **Porque usar um chatbot em seu delivery**. 2018. https://chatbotmaker.io/blog/porque-usar-chatbots-em-seu-delivery/. Acesso em: 7 de outubro de 2020.

CUNHA, D. M.; SILVA, L. O. da; MOURA, R. S. Um chatbot para atendimento a clientes de farmácias. In: SBC. **Anais da VII Escola Regional de Computação do Ceará, Maranhão e Piauí**. [S.l.], 2019. p. 32–39.

DIAS, W. S. et al. Susi: uma proposta de chatbot para o atendimento de usuários do ministério da saúde. Universidade Federal de Minas Gerais, 2019.

Edison Figueira. **Chatbot: o "colaborador" indispensável para transformação digital de sucesso no atendimento ao consumidor 4.0**. 2020. http://noticias.dino.com.br/chatbot-o-colaborador-indispensavel-para-transformacao-digital-de-sucesso-no-atendimento-ao-consumidor
Acesso em: 02 de outubro de 2020.

Explosion AI. **spaCy 101: Everything you need to know**. 2020. https://spacy.io/. Acesso em: 25 de agosto de 2020.

EYNO, P. Natural language processing and chat-bot implementation. 2019.

FOSSATTI, M. C.; RABELLO, R. d. S.; MARCHI, A. C. B. d. Agebot: um chatterbot em aiml voltado para responder questões sobre epilepsia. In: SN. **Anais do XXXI Congresso da Sociedade Brasileira da Computação**. [S.l.], 2011.

INDURKHYA, N.; DAMERAU, F. J. **Handbook of natural language processing**. [S.l.]: CRC Press, 2010. v. 2.

JARGAS, A. M. **Expressões regulares: uma abordagem divertida**. [S.l.]: Novatec Editora, 2009.

JÚNIOR, M. L. B. L. **Desenvolvimento de um Chatbot usando redes de aprendizado profundo**. Dissertação (B.S. thesis) — Universidade Federal do Rio Grande do Norte, 2018.

JURAFSKY, D. **Speech & language processing**. [S.l.]: Pearson Education India, 2000.

Kavita Ganesan. **Word2Vec: A Comparison Between CBOW, SkipGram & SkipGramSI**. 2020. https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/#.X4n5LXVKjDE. Acesso em: 16 de outubro de 2020.

LEITE, M. R. et al. Desenvolvimento de assistente virtual para atendimento imobiliário. Florianópolis, SC., 2019.

MDN. **Expressões Regulares**. 2020. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Regular_Expressions. Acesso em: 7 de outubro de 2020.

MERSCHMANN, L. **Classificação probabilística baseada em análise de padrões**. [S.l.]: Niterói, 2007.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013.

NEVES, A. M. M.; BARROS, F. d. A. **iaiml: Um mecanismo para tratamento de intençao em chatterbots**. 2005.

NORVIG, P.; RUSSELL, S. **Artificial Intelligence: A Modern Approach**. Pearson Education, 2011. ISBN 9780133001983. Disponível em: https://books.google.com.br/books?id=Na8rAAAAQBAJ.

OLIVEIRA, N. et al. Helpcare: Um protótipo de chatbot para o auxílio do tratamento de doenças crônicas. In: SBC. **Anais Principais do XIX Simpósio Brasileiro de Computação Aplicada à Saúde**. [S.l.], 2019. p. 282–287.

PLUMMER, D. et al. Top strategic predictions for 2018 and beyond: pace yourself for sanity's sake. **Gartner. com**, 2017.

SOMMERVILLE, I. **Engenharia de software**. PEARSON BRASIL, 2011. ISBN 9788579361081. Disponível em: https://books.google.com.br/books?id=H4u5ygAACAAJ.

spaCy. **Rule-based matching**. 2020. https://spacy.io/usage/rule-based-matching. Acesso em: 25 de agosto de 2020.

SRINIVASA-DESIKAN, B. **Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras**. Packt Publishing, 2018. ISBN 9781788837033. Disponível em: https://books.google.com.br/books?id=48RiDwAAQBAJ.

Stoodi. **Inteligência artificial: o que é, como funciona e aplicações!** 2020. https://www.stoodi.com.br/blog/atualidades/inteligencia-artificial/. Acesso em: 30 de setembro de 2020.

Take. **Automação de marketing com chatbots: tudo que você precisa saber!** 2019. https://take.net/blog/chatbots/automacao-de-marketing-digital-com-chatbots/. Acesso em: 02 de outubro de 2020.

Telegram. **API do Telegram Bot**. 2020. https://core.telegram.org/bots/api. Acesso em: 17 de setembro de 2020.

Telegram. **Telegram FAQ**. 2020. https://telegram.org/faq. Acesso em: 17 de setembro de 2020.

TensorFlow. **Embeddings de Palavras**. 2020. https://www.tensorflow.org/tutorials/text/word_embeddings. Acesso em: 16 de outubro de 2020.

TRIOLA, M. F. Bayes' theorem. **PDF. Dostupno: http://faculty. washington. edu/tamre/-BayesTheorem. pdf**, 2010.

VALMORBIDA, W.; HART, L. M. Desenvolvimento de uma api para chatbots de vendas e gestão de seguros. **Revista Destaques Acadêmicos**, v. 11, n. 4, 2020.

WALLACE, R. S. The anatomy of alice. In: **Parsing the Turing Test**. [S.l.]: Springer, 2009. p. 181–210.

WEIZENBAUM, J. Eliza—a computer program for the study of natural language communication between man and machine. **Communications of the ACM**, ACM New York, NY, USA, v. 9, n. 1, p. 36–45, 1966.