Universidade Federal de Goiás
Escola de Engenharia Elétrica, Mecânica e de Computação
Curso de Engenharia de Computação

Danillo da Silva Gontijo
Gabriel Coutinho Brêtas Netto

# *Construindo um* Chatterbot *com Rasa: auxiliando no delivery.*

Goiânia
2018

Danillo da Silva Gontijo
Gabriel Coutinho Brêtas Netto

# *Construindo um* Chatterbot *com Rasa: auxiliando no delivery.*

Monografia apresentada ao Curso de Engenharia de Computação da UFG, como sito para a obtenção parcial do grau de BACHAREL em Engenharia de Computação.

**Orientador: Sandrerley Ramos Pires**

**Doutor em Engenharia Elétrica**

Goiânia
2018

da Silva Gontijo, Danillo.
Coutinho Brêtas Netto, Gabriel.

Construindo um *Chatterbot* com Rasa: auxiliando no *delivery*. : / Danillo da Silva Gontijo
Gabriel Coutinho Brêtas Netto. - 2018. 63p. :

Orientador: Sandrerley Ramos Pires.

Projeto de Conclusão de Curso 2 (Graduação) -Universidade Federal de Goiás- Escola de Engenharia Elétrica, Mecânica e de Computação - Engenharia de Computação.

1. Chatterbot. 2. Processamento de Linguagem Natural. 3. Inteligência Artificial. I. Sandrerley Ramos Pires, orient. II.Título.

Danillo da Silva Gontijo

Gabriel Coutinho Brêtas Netto

# *Construindo um* Chatterbot *com Rasa: auxiliando no delivery.*

Monografia apresentada ao Curso de Engenharia de Computação da UFG, como requisito para a obtenção parcial do grau de BACHAREL em Engenharia de Computação.

Aprovado em 26 de Julho de 2018

## BANCA EXAMINADORA

---

Sandrerley Ramos Pires

Doutor em Engenharia Elétrica

---

Weber Martins

Ph.D em Inteligência Artificial

---

Tales Marinho Godois

Bacharel em Engenharia da Computação

# Resumo

Devido a evolução da inteligência computacional nos últimos anos, consequentemente o avanço prático e comercial dos *chatterbots*, vamos propor neste trabalho, a construção de um atendente virtual que irá auxiliar nos pedidos em um restaurante, substituindo portanto, interações humanas nas conversas. Para alcançar esse objetivo, usaremos como ferramenta base de desenvolvimento, um *framework* chamado *Rasa* que irá abstrair algumas complexidades da inteligência computacional.

Palavras-chaves: *Chatterbots*, PLN, NLU, Rasa, Chatito, spaCy.

# Abstract

We propose in this paper due to the evolution of computational intelligence in recent years and consequently practical and commercial improvements of chatterbots, the development of a virtual assistant that will assist people in placing orders in restaurants, trying to substitute the need for human interactions in conversations. To accomplish this objective, we will use a development framework as basis, that framework is Rasa, and it will help abstracting some of the complexities related to computational intelligence.



Keywords: *Chatterbots*, PLN, NLU, Rasa, Chatito, spaCy.

# Agradecimentos

Primeiramente, gostaríamos de agradecer a Deus por ter nos colocado em um ambiente, no qual nos proporcionou toda a oportunidade de poder entrar e estar, neste momento, concluindo este curso de graduação, em uma universidade pública e referência de ensino na região centro-oeste, como é a Universidade Federal de Goiás (UFG). Agradecemos também à família e aos amigos por sempre terem participado da nossa formação social, incentivando e acreditando em nosso potencial.

E do lado da nossa formação profissional, gostaríamos de agradecer a todos os professores por quem tivemos a oportunidade de muito aprender, em especial, ao Prof. Ph.D Weber Martins, quem nos ajudou a despertar todo o interesse pela área de Inteligência Artificial, além também, de ter sido quem nos ensinou, durante alguns períodos como orientador, a usar o lado da ciência como premissa básica de conhecimento e aperfeiçoamento para executar todas as nossas ideias. Agradeço ao Prof. Dr. Sandrerley Pires, por ter nos ajudado a encontrar o caminho em que pudéssemos aplicar parte desse conhecimento adquirido, e assim, poder concretizar o experimento deste projeto. Por fim, agradecemos ao coordenador do curso de engenharia de computação, Prof. Dr. Carlos Galvão, que no rigor de suas atribuições e responsabilidades, mas tratando todos os fatos sempre de maneira justa e séria, ajudou a desenvolver nosso potencial além de um limite que, apesar de toda a nossa dificuldade de conciliar tempo de estudo e trabalho, em alguns momentos, chegamos a duvidar se era possível alcançar.

Sentiremos falta desse contato de aprendizagem em sala de aula. Seremos eternamente gratos por toda a dedicação de vocês, e o que foi ensinado, iremos colocar em prática para ajudar a desenvolver e contribuir com a sociedade.

*"Conhece-te a ti mesmo".*

*Lema grego oriundo da cidade de Delfos e citado por Sócrates (Diálogos de Platão)*

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviaturas e Siglas

| | |
|---|---|
| Abrasel | Associação Brasileira de Bares e Restaurantes |
| AIML | Artificial Intelligence Markup Language |
| ALICE | Artificial Linguistic Internet Computer Entity |
| API | Interface de Programação de Aplicações |
| CLIR | Recuperação Multilíngue e Multilíngue de Informações |
| DFSG | Definição Debian de Software Livre |
| HTTP | Hypertext Transfer Protocol |
| IA | Inteligência Artificial |
| MT | Machine Translation |
| NLU | Natural language understanding |
| NLP | Neuro-linguistic programming |
| OSI | Open Source Initiative |
| SO | Sistema Operacional |
| PLN | Processamento de Linguagem Natural |
| PyPI | Python Package Index |

# 1 INTRODUÇÃO

*Delivery*. Um termo relativamente novo, aqui no Brasil e no mundo, se considerarmos a história humana, mas um conceito bem antigo: a entrega de comida. A opção de receber refeições em casa ou em qualquer outro local existe há muito tempo[1].

Na Roma Antiga, por exemplo, tem-se conhecimento de um local que fornecia comida preparada e pronta para consumo ao seus clientes, conhecidos como *Termopólio* (Kleberg, 1957, p. 24-25). Mesmo não havendo, ainda, o sistema de entregas, este tipo de estabelecimento, um dos precursores do famoso *fast-food*, já criava o hábito do *take out food*, também conhecido como comida "para viagem" , o que já permitia a criação de um modelo de negócio de armazenagem e entrega de comida.

Mas o modelo atual do sistema de *delivery* que conhecemos, iniciou-se no período pós-Segunda Guerra Mundial. No início da década de 50, e com o avanço de tecnologias como a TV e telefone, a burguesia americana descobriu então, as primeiras propagandas oferecendo o serviço de "entrega em casa" (Emelyn Rude, 2016). E em diante, temos uma rápida expansão desse modelo de negócio, chegando ao Brasil em meados da década de 80(Xavier, 2015), com a vinda dos restaurantes de *fast-food*, e que, segundo a Associação Brasileira de Bares e Restaurantes - Abrasel -, em 2017 o *delivery* de comida faturou mais de R$ 10 bilhões[4], colocando o Brasil entre um dos maiores mercado deste segmento, de acordo com o estudo feito pela *EAE Business School*[5].

Diante deste mercado tão grande e que envolve tanto dinheiro, tem-se aproveitado dos avanços tecnológicos para continuar a crescer. Dos anos 50 para cá, criaram-se diversas facilidades tecnológicas que otimizam recursos e agilizam a realização de pedidos pelos clientes. Dentre estas tecnologias está o foco deste projeto: *chatterbot*.

*Chatterbot* nada mais é que um assistente virtual que irá simular conversas humanas. E com o surgimento e a popularização dos aplicativos de mensageria, como, por exemplo, *Facebook Messenger*, *Whatsapp*, *Telegram* e outros, vamos propor a construção de um *chatterbot* para auxiliar em pedidos de *delivery* em um restaurante, esclarecendo sobre o cardápio, os ingredientes e as formas de pagamento. Posteriormente, esse assis-

tente virtual, poderá ser integrado com esses aplicativos, de acordo com a documentação e instruções específicas de cada um.

Para atingirmos esse objetivo, vamos conhecer algumas subáreas da Linguística e da Inteligência Artificial. A partir do conhecimento dessas áreas, será possível criar um assistente virtual suficientemente eficiente para simular uma conversa digitada com um ser humano. Também utilizaremos técnicas e instrumentos da Ciência da Computação para orientar e ajudar durante o decorrer do processo da solução. Um desses instrumentos, é o *framework Rasa*, que irá abstrair complexidades da Inteligência Artificial, facilitando, portanto, o desenvolvimento. Abordaremos mais sobre *framework*, no decorrer deste projeto.

Para medirmos o nível prévio de conhecimento necessário para usar a ferramenta, e também a qualidade de sua documentação, buscaremos manter o máximo da referência sobre a documentação oficial do Rasa. E para ajudar a se familiarizar, devido a universalização dos termos usado na área tecnológica, buscaremos especificá-los, na maioria das vezes, em inglês.

# 2 Linguística e Processamento de Linguagem Natural

O ser humano é separado de outras espécies pela sua capacidade de linguagem (Russel e Norving, p. 990). Mesmo existindo alguns outros animais capazes de se comunicar, demonstrando alguns sinais de vocabulário, somente o ser humano pode se comunicar de forma confiável utilizando um número ilimitado de mensagens qualitativamente diferentes sobre qualquer tema utilizando sinais discretos.

Linguística é a área que estuda a linguagem, ou seja, como nós seres humanos nos comunicamos. Já o Processamento de Linguagem Natural (PLN) busca estudar como um computador tenta compreender essa comunicação. Essas duas áreas de estudo são a premissa para o compreendimento de como é possível funcionar um *chatterbot* na prática. Vamos analisar neste tópico então, definições, classificações, além de exemplos e aplicações destas duas áreas.

## 2.1 Linguagem Natural

Para Kracht (2007), linguagem é um meio de se comunicar, é um sistema semiótico, ou seja, simplesmente um conjunto de sinais. E para Rich e Knight (1994), a linguagem natural compreende a comunicação dos seres humanos sobre o mundo, ocorrendo em sua maior parte através da fala. Comparando com a linguagem escrita, esta última ainda é muito recente. Porém, a linguagem escrita é mais fácil de ser compreendida por um computador. Em outras palavras, requer menos processamento da máquina para ser interpretada.

Para Russel e Norving (2013), uma linguagem pode ser definida como um conjunto de sequências, sendo que nesse meio, temos as Linguagens Formais e as Linguagens Naturais. A primeira tem modelos de linguagem precisamente definidos, citando como exemplo, as linguagens de programação *Java*, *Python*, e dentre outras. Esse tipo de linguagem é considerada uma linguagem definida, pois é especificada por um conjunto de

regras chamado gramática, e também de regras que definem o seu significado ou sua a semântica. Por exemplo, as regras dizem que o "significado" de "2 + 2" é 4, e o significado de "1/0" será sinalizado como erro.

Já a Linguagem Natural não pode ser caracterizada como um conjunto de sentenças definitivas. Mencionando algumas linguagens, por exemplo, o português, o inglês, etc., é melhor tratá-las como uma distribuição de probabilidade sobre sentenças do que por um conjunto definitivo. Ou seja, em vez de perguntar se uma sequência de palavras é ou não válida dentro do conjunto de regras que o define, perguntamos por sua probabilidade a qual uma sentença aleatória seria palavras ? P(S = palavras). A Linguagem Natural também pode ser ambígua. "Ele encontrou o banco" , pode significar tanto que ele encontrou uma peça imobiliária, como também uma instituição financeira. Portanto, não podemos falar de um único significado para a sentença, mas de uma distribuição de probabilidade sobre possíveis significados.

Em conclusão, existe uma grande dificuldade em lidar com as Linguagens Naturais, isso porque, além de pertencerem a um universo infinito de definições, estão em constante mutação. Assim sendo, os modelos de nossa língua são, na melhor das hipóteses, uma aproximação.

## 2.2 Subdivisões da linguística

No estudo da linguística pode se fazer a divisão entre 4 tipos, morfológica, sintática, semântica e pragmática, explicaremos cada uma delas de forma simplificada nesta seção.

### 2.2.1 Análise Morfológica

Morfologia refere-se à estrutura das palavras. Descreve e analisa os processos e regras de formação e de criação de palavras, a sua estrutura interna, a composição e a organização dos seus constituintes. Ou ainda: "morfologia é o estudo da estrutura interna das palavras" (JENSEN apud MONTEIRO, 2002, p. 11). E segundo Nida (1970, p. 1), a morfologia pode ser definida como "o estudo dos morfemas e seus arranjos
na formação das palavras" .

A análise morfológica é responsável por reconhecer palavras e expressões isoladamente em uma sentença, fazendo uso dos espaços e pontuação para auxiliar nessa análise e definir os elementos mórficos. Essas palavras identificadas são classificadas de acordo com seu uso (Oliveira 2002).

Segundo Oliveira (2002), uma palavra em uma sentença gramaticalmente válida pode ser substituída por uma outra de mesmo tipo, mantendo assim a validade da sentença. Assim um verbo seria substituído por outro verbo, um substantivo substituído por outro e assim por diante. A morfologia trata as palavras de acordo com sua classificação, flexão, forma e estrutura.

### 2.2.2 Análise Sintática

Recordando que gramática é um conjunto de regras definidos em uma sentença, análise sintática é o processo de analisar uma cadeia de palavras para descobrir a sua estrutura frasal, de acordo com as regras de uma gramática (Russel e Norving, 2013, p. 1028).

O analisador sintático trabalha ao nível de agrupamento de palavras, definindo a estrutura de uma frase, diferente do analisador léxico-morfológico que lida com as estruturas e classificação das palavras.

A análise sintática ou *parsing* é de acordo com Gonzales e Lima (2003) o nome dado ao procedimento de avaliar os modos de combinação das regras gramaticais, com a finalidade de formar uma estrutura sintática em forma de árvore da frase analisada. Em sentenças ambíguas, o *parser* deve obter todas as estruturas possíveis representadas por essa sentença.

### 2.2.3 Análise Semântica

A análise semântica se refere ao significado das sentenças e não só de palavras isoladas, dando significado às estruturas criadas na análise sintática.

Para Rezende (2003, p. 342),

> A análise semântica dos textos é feita para tentar identificar a importância das palavras dentro da estrutura da oração. Quando se utiliza um único texto, algumas funções podem ser identificadas e pela

> função identifica um grau de importância. Por exemplo, um nome tem grande chance de representar informação relevante na estrutura da sentença. Para tarefas como categorização, o interessante seria analisar um documento comparando-o a Bases de Conhecimento de diferentes assuntos, para descobrir a qual categoria ele pertence. Um conjunto bem-selecionado de textos sobre um assunto compõe uma base de Conhecimento.

A primeira etapa desse processo é procurar palavras em um dicionário e extrair seus significados. Por causa da polissemia(palavras com vários significados) pode não ser possível identificar o significado correto olhando isoladamente cada palavra. Por exemplo pode ser usar a palavra vela, que pode significar a vela de um de um barco, uma vela de cera que se usa para iluminação, ou mesmo a conjugação do verbo velar. Nestes casos se faz necessária a uma análise de toda a frase. Sendo a frase "A vela queimava iluminando a sala" , o correto seria identificar como a vela feita de cera. Esse tipo de situação acontece para muitas palavras dificultando a análise. Isso faz com que no marcadores semânticos sejam criadas entradas diferentes para todos os significados da palavra vela, o que facilita o analisador léxico. Esse processo de marcação de palavras e conhecido como Desambiguação Léxica (RICH, KNIGHT, 1994).

Ao final da análise semântica são construídas as chamadas Gramáticas Semânticas, a criação dessas gramáticas apesar de opcionais facilitam a análise pois de acordo com Rich e Knight(1994), com as gramáticas construídas é possível fazer uso direto na aplicações de PLN, evitando-se também as ambiguidades léxicas, pois com os marcadores as palavras estão com seus significados melhor definidos.

### 2.2.4 Pragmática

A analise pragmatica leva em consideração o contexto em que está inserida a sentença sendo avaliada, pois uma frase anterior ou posterior pode dar significado diferente a aquela sentença. Isoladamente a frase "Ele está muito frio" , dá a entender que uma pessoa está fisicamente fria, mas se a frase anterior for "Algo errado aconteceu com Fábio, ele não fala comigo a semanas" , mudaria o sentido da frase para uma reação emocional e não frio físico da pessoa. (RICH, KNIGHT, 1994).

Em uma interação com um *chatbot,* o usuário diria "Quero pagar com cartão de crédito" , o *bot* deve de entender que não só pagamento será efetuado como cartão de crédito, e que também deve fechar o pedido e calcular o valor total.

## 2.3 Processamento de Linguagem Natural

Processamento de Linguagem Natural é a subárea da Inteligência Artificial (IA) que estuda a capacidade e as limitações de uma máquina em entender a linguagem dos seres humanos(Hirschberg, 1988). E segundo Ann Copestake (2004), PLN pode ser definido como o processamento automático (ou semi-automático) da linguagem humana.

Para Elizabeth D. Liddy(2001) Processamento de Linguagem Natural são técnicas para análise e representação de textos com ocorrência natural em 1 ou mais níveis de análise linguística com o propósito de atingir processamento de linguagem semelhante ao de um humano para uma variedades de aplicações.

Processamento de Linguagem Natural é uma área de pesquisa e aplicação que explora como os computadores podem ser usados para entender e manipular texto ou fala em linguagem natural, e assim, executar tarefas úteis. Pesquisadores e estudiosos de PLN buscam reunir conhecimentos sobre como os seres humanos entendem e usam a linguagem, possibilitando que ferramentas e técnicas apropriadas possam ser desenvolvidas, fazendo com que os sistemas de computador entendam e manipulem linguagens naturais para realizar as tarefas. As bases do estudo sobre PLN estão em várias disciplinas, como: ciências da computação, linguística, matemática, engenharia elétrica e eletrônica, inteligência artificial e psicologia (Chowdhury, 2005).

O objetivo do PLN é fornecer aos computadores a capacidade de entender e compor textos. "Entender" um texto significa reconhecer o contexto, fazer análise sintática, semântica, léxica e morfológica, criar resumos, extrair informação, interpretar os sentidos, analisar sentimentos e até aprender conceitos com os textos processados.

### 2.3.1 Breve Histórico

PLN vem sendo estudada e aplicada em casos que é necessário utilizar comunicação com Linguagem Natural.

Desde os anos 40, técnicas estão sendo criadas na tentativa de tornar a comunicação entre seres humanos e computadores mais fluida. *Machine Translation* (MT) foi a primeira aplicação computacional relacionado a linguagem natural. É amplamente aceito que foi Weaver em um memorando em 1949 que sugeriu a idéia do MT, essa idéia acabou consistia em um utilizar técnicas de criptografia e teoria da informação para para

fazer tradução de línguas. A idéia original era simplista e consistia em fazer o uso de um dicionário fazendo a correspondência entre palavras e levava em consideração somente as diferenças na ordenação possível das palavras. (Liddy, 2001)

Nos anos 50 e começo dos anos 60, ideias sobre gramáticas formais começaram a surgir em linguísticas e algoritmos para interpretar linguagem natural começaram a ser desenvolvidos ao mesmo tempo que algoritmos estavam sendo desenvolvidos para interpretar linguagens de programação, porém a maiorias dos linguistas não se interessava por PLN e por isso somente a abordagem de Chomsky foi desenvolvida e acabou sendo utilizada de forma indireta no desenvolvimento do PLN.(Copestake, 2004)

Na década de 50 os avanços na área de teoria sintática deixou as pessoas entusiasmadas e acreditava-se que em alguns anos existiriam sistemas automáticos de tradução que seriam indistinguíveis de uma tradução feita por um ser humano. Isso se provou não só irreal com o conhecimento linguístico da época.(Liddy, 2001)

Trabalhos teóricos na década de 60 e 70 eram focados em como representar significado e em desenvolver soluções palpáveis de rastreamento que as soluções existentes em teorias de gramáticas não eram capazes de contemplar. Devido as teorias de Chomsky e no trabalho de outros várias teorias foram criadas tentando explicar as anomalias sintáticas e prover representações semânticas.

Nas década de 70 e primeira metade da década de 80 PLN era predominantemente baseada num paradigma que uma grande parte da linguística e conhecimento real era codificado na mão, existia certa controvérsia sobre quantidade de conhecimento linguístico necessário para o processamento, com alguns pesquisadores diminuindo a importância da sintaxe e dando mais importância para o conhecimento real do mundo. Nos anos 80, vários de paradigmas linguísticos e lógicos foram firmados no PNL. Infelizmente isso não levou a grandes avanços devido a vários problemas, entre eles o problema de desambiguação, pois esse problemas eram vistos como sendo responsabilidade de outras áreas e também em parte pelo tipo de problema que tentavam resolver na época, que não visava aplicações diretas voltadas ao usuário. Logo ficou aparente que a aquisição léxica era um grande gargalo no desenvolvimento de sistemas de PLN.(Copestake, 2004)

Na década de 90 análise estatística passou ao paradigma mais utilizado na comunidade acadêmica, em que simples técnicas de estatística funcionavam desde que o material de treinamento inicial fosse grande o suficiente. Isso foi possível graças ao

aumento da quantidade de disponível de textos *online*, ao aumento da capacidade computacional e de memória dos computadores, e também da popularização da internet. Métodos estatísticos obtiveram sucesso lidando com muitos problemas genéricos da computação linguística como identificação de fala, desambiguação de palavras e acabaram sendo o padrão utilizado em PNL.(Liddy, 2001)

De maneira mais recente já vem sendo utilizados aprendizado de máquina em conjunto com análises estatísticas e assim gerando resultados melhores.(Copestake, 2004)

## 2.4 Inteligência Artificial

Segundo Rich e Knight (1994), numa definição geral, Inteligência artificial (Al) é o estudo de como fazer os computadores fazerem coisas que, no momento, as pessoas fazem melhor. A IA busca analogias com a própria inteligência natural para funcionar, ou pelo menos, entender sobre IA vai ajudar também a compreender a inteligência natural.

Como mencionado, buscamos, a partir da IA, otimizar soluções e/ou problemas usando o computador. Exemplificando um desses problemas, seria a compreensão e interpretação da linguagem escrita. De acordo com a abordagem de Alan Turing (1950) no artigo chamado "Computing Machinery and Intelligence" , é citado a capacidade que as máquinas têm de pensar e de serem inteligentes. E partindo da imaginação de que um computador digital poderia imitar um ser humano em uma conversa, tal sendo isso possível, e o ser humano com quem estivesse conversando não mais conseguisse distinguir se estava dialogando com uma máquina ou com outro ser humano, seria um indicativo de que o sistema de IA é inteligente o bastante, e teria passado no Teste de Turing.

O teste proposto por Turing motivou vários cientistas a desenvolverem programas na tentativa de passar no teste, esse foi um dos incentivos por trás de ELIZA. Em 1990 Hugh Loebner lançou um concurso com prêmio de $ 100.000(cem mil dólares) para o programa capaz de passar no teste (Mauldin, 1994).

## 2.5 Dificuldades no Processamento de Linguagem

Como mencionado anteriormente, no tópico que diferencia Linguagem Natural da Linguagem formal, e os motivos de trabalharmos não com sequências definitivas, mas sim com

um modelo probabilístico, essa mesma justificativa demonstra as dificuldades que encontraremos no processamento computacional da linguagem. Teremos então, a ambiguidade, a mesma frase ou frases muito parecidas, mas com significados diferentes, ou até mesmo, frases diferentes que podem ter o mesmo significado.

## 2.6 Exemplos de aplicações de PLN

As aplicações da PLN incluem vários campos de estudo, como tradução automática, processamento e resumo de texto em linguagem natural, interfaces de usuário, recuperação multilíngue e multilíngue de informações (CLIR), reconhecimento de fala, inteligência artificial e sistemas especialistas (Chowdhury, 2005).

O que distingue os aplicativos de processamento de linguagem de outros sistemas de processamento de dados é o uso do conhecimento da linguagem (Jurafsky, 2008). O uso de PLN está contido em diversas aplicações.

Algumas áreas em que se faz o uso de PLN:

- verificação de ortografia e gramática
  PLN é utilizado na correção de gramatical e ortográfica de textos, hoje está presente em todos os editores de texto modernos.

- reconhecimento óptico de caracteres (OCR)
  Trata-se do uso de PLN para auxiliar no reconhecimento de caracteres através de imagens.

- sistemas de tradução linguística
  Consiste no uso de PLN para traduzir textos em linguagem natural para outros idiomas. Ainda hoje a tradução automática representa um grande desafio pois as características específicas de cada idioma variam muito.

- classificação de documentos
  Trata-se da tentativa de identificar e classificar documentos, definindo pelo texto o assunto abordado entre eles.

- clusterização de documentos
  A clusterização de documentos resume-se ao agrupamento de documentos baseado

na semelhança de seu conteúdo ou assunto abordado.

- sumarização

  Consiste na identificação das partes mais importantes de um texto e utilizar essas partes para compor uma tentativa de um resumo daquele texto.

- interface de linguagens natural dos bancos de dados

  Consiste em obter informações armazenadas em um banco de dados fazendo o uso de linguagem natural para fazer essa busca.

- compreensão de e-mails

  Consiste em identificar e entender e-mails podendo assim definir seu conteúdo e podendo até fazer sugestão de respostas.

- sistemas de diálogos (*chatterbots)*

  O problema abordado neste trabalho que consiste em fazer o uso de PLN para simular uma conversa com um interlocutor humano.

As aplicações estão ordenadas de acordo com a sua complexidade. Ou seja, da lista acima, as aplicações do topo podem ser classificadas como simples ajuda para os seus usuários e as do final são aplicação que, para seu melhor funcionamento, exigem a utilização de recursos de Inteligência Artificial. Percebendo que o de maior complexidade é o de tema desse projeto.

## 2.7 *Natural Language Understanding* (NLU)

No centro de qualquer tarefa de PNL está a questão importante do entendimento da linguagem natural. O processo de construção de programas de computador que entendem a linguagem natural envolve três grandes problemas: o primeiro diz respeito aos processos de pensamento, o segundo à representação e ao significado do input linguístico e o terceiro ao conhecimento do mundo. Assim, um sistema de PNL pode começar no nível da palavra para determinar a estrutura morfológica e a natureza (como parte da fala ou significado) da palavra; e, em seguida, pode passar para o nível da sentença para determinar a ordem das palavras, a gramática e o significado da sentença inteira; e depois para o contexto e o ambiente ou domínio geral. Uma dada palavra ou sentença pode ter

um significado ou conotação específica em um dado contexto ou domínio, e pode estar relacionada a muitas outras palavras e / ou sentenças no contexto dado.

NLU é uma área do Processamento de Linguagem Natural que visa a compreender o texto - envolvendo análise semântica e pragmática (NAVIGLI et al., 2018). Por estar envolvendo essas duas análises, trabalhar com NLU se torna particularmente desafiador. Isso, como vimos anteriormente, é devido à ambiguidade difusa da linguagem e às percepções sutilmente diferentes que os humanos têm da palavra e significados sentenciais. O NLU procura entender a linguagem, permitindo que os computadores leiam e compreendam o texto. Podemos, então, a partir dessa interpretação, concluir o porquê do termo "Understanding".

Um sistema de NLU verdadeiro e completo seria capaz de através de uma entrada de texto, traduzi-lo para outro idioma, responder perguntas relacionadas ao conteúdo do texto, e fazer inferências sobre o texto.(LIDDY et al., 2001)

# 3 *Trabalhos Correlatos)*

Como a maioria das invenções os *chatterbots* foram criados para facilitar tarefas que normalmente exigiria uma pessoa para serem realizadas.

O termo *chatterbot* em si foi criado por Michael Mauldin(Deryugina, 2010) criador do primeiro Verbot(Verbal-Robot), Julia.

## 3.1 ELIZA

ELIZA foi o primeiro *chatterbot* que foi idealizada em 1966 por Joseph Weizenbaum, professor no *Massachussets Institute of Technology*, como um Processador de linguagem natural (PLN)[Weizenbaum, 1966],. Foi desenvolvido utilizando a idéia de casamento de padrões. Seu potencial foi demonstrado através de entrevistas em que ELIZA imitava respostas de um terapeuta utilizando uma abordagem centrada na pessoa. O ELIZA respondia a cada frase do usuário com uma pergunta, fazendo assim com que o usuário guiasse a conversa.

*Chatterbots* como ELIZA, fazem o uso de técnicas de casamento de padrões, efetuando uma busca sequencial utilizando palavras-chave a procura de respostas pré definidas.

A arquitetura do ELIZA era baseada numa lista de de regras em que as frases a serem analisadas se encontravam, e um programa simples capturava as entradas do usuário. A entrada fornecida pelo usuário era lida e se buscava uma palavra chave, se encontrada, a frase era transformada de acordo com a regra a que pertencia aquela palavra chave.

### 3.1.1 Funcionamento ELIZA

ELIZA começa identificando as palavras mais importantes na sentença, aplica uma regra de modificação que contextualiza as palavras, por exemplo “você” seguindo de “é” se classifica como afirmação. Em casos que a sentença não se encontra em nenhuma

regra é retornada uma resposta já utilizada ou uma resposta completamente livre de contexto

## 3.2 Julia

JULIA[Mauldin, 1994] foi o *chatterbot* criado por Michael Mauldin na Carnegie Mellon University e tinha a função de auxiliar usuários a participar de jogos feitos para terminais somente com interface de texto.

JULIA atuava como um personagem cujo propósito era o de auxiliar outros usuários em um ambiente virtual, conhecido como TinyMUD (Multi-User Dungeons). JULIA ajudava usuários nesse mundo virtual mapeando cavernas e enviando mensagens.

Sua primeira iteração possuia um mecanismo de memória simples fazendo uso de "*if*" que alteravam o comportamento, posteriormente passou-se a utilizar redes neurais para melhorar suas respostas aos comandos dos jogadores. Na rede os nós são conjuntos de padrões, uma resposta, e os nós ativos e os inibidos. Quando um padrão é acionado, os nós que tem aquele padrão são estimulados sendo escolhido o de maior nível.

JULIA possui uma curiosidade, ele foi implementado para simular levemente alteração de humor, e possui alguns objetivos definidos para si, o que faz com que seja capaz de avaliar se esta sendo útil naquele ambiente que está inserido.

## 3.3 ALICE

ALICE(*Artificial Linguistic Internet Computer Entity*)[Wallace, 2009], é um dos *chatterbots* que veio em conjunto com a criação da AIML(*Artificial Intelligence Markup Language*), que é uma linguagem de marcação baseado em xml.

O comportamento de ALICE está relacionado a categorías definidas por padrões, pergunta-resposta. Cada categoria possui um padrão que é o estímulo e uma resposta *template*. A cada frase é feita uma pesquisa buscando a categoria, e assim uma resposta correspondente é determinada pelo estímulo da entrada. ALICE faz o uso de aprendizado supervisionado em que um responsável pode inserir novos padrões, já que não há um padrão específico.

O grande avanço que ALICE nos trouxe foi o *AIML* que teve seus interpretadores implementados em diversas linguagens, além de ter gerado linguagens derivadas que utilizando-se de princípios semelhantes expandiram e melhoraram seu desempenho. Um exemplo dessa linguagem é o iAIML, que consiste em um mecanismo desenvolvido para tratamento de intenções em *Chatterbots* baseado na Teoria de Análises da Conversação propondo uma análise tanto no nível local quanto no global.[Neves e Barros, 2005]

## 3.4 *Chatterbot* iFood

O iFood empresa de *Delivery Online* possui um *chatterbot* chamado de iFood Guru feito em cima do *Messenger* do *Facebook*. Com esse *bot* é possível fazer pedidos de pizzas e bebidas. O iFood Guru um *chatterbot* muito simples que baseado no endereço informado, mostra os sabores de pizza disponíveis. Escolhido o sabor da pizza são mostrados os restaurantes abertos que atendem o endereço e que possuem aquele sabor. Nessa tela são mostradas as logomarcas e notas de avaliação do restaurante. Escolhido o restaurante o valor total com taxa de entrega é informado ao cliente que pode confirmar ou não aquele pedido. [Lima, 20017] Toda essa interação é feita utilizando diálogos textual e menus interativos que dão opções para o usuário escolher o que facilita tanto no desenvolvimento do *bot* quando no entendimento do usuário.

# 4 Abordagem

Neste tópico, explicaremos o planejamento e a arquitetura da solução. Abordando também conceitos de tecnologias e ferramentas utilizadas para o desenvolvimento do *chatterbot* proposto neste projeto. Sendo que uma das principais ferramentas que iremos utilizar, será o *framework* Rasa. Mas, antes de aprofundarmos mais sobre essa ferramenta, que será usada para modelar e desenvolver nosso *chatterbot*, vamos abordar um pouco sobre o que é um *framework*.

## 4.1 O que é um *Framework*?

*Frameworks* são técnicas de reutilização orientada a objetos. Compartilhando com diversas características com técnicas de reutilização em geral, e com técnicas de reutilização orientadas a objeto em particular. (Ralph E. Johnson. Framework, Components, Patterns. 1997)

E segundo Jacques Sauvé (*Frameworks* e Componentes. 2000), um *framework* captura a funcionalidade comum a várias aplicações, mais especificamente, provendo uma solução para uma família de problemas semelhantes, ao usar um conjunto de classes e interfaces que mostra como decompor esses problemas, e como objetos dessas classes colaboram para cumprir suas responsabilidades. Continuando na ideia de Jacques Sauvé, o conjunto de classes deve ser flexível e extensível para permitir a construção de várias aplicações com pouco esforço, especificando apenas as particularidades de cada aplicação.

Resumimos então, que um *framework* abstraí algumas complexidades e códigos que são comuns a um determinado problema, com a principal característica de conter técnicas de reutilização. Facilitando, portanto, o desenvolvimento de uma solução que se encaixa dentro do domínio de atuação do *framework*.

Como mencionado, as aplicações devem ter algo em comum, mas não apenas uma pequena fração, é necessário ser razoavelmente grande, pertencendo a um mesmo domínio de problema. Vejamos na imagem abaixo quando, para a solução de determinado problema, é possível criar um *framework*. ompreendendo o conceito de *framework*, as

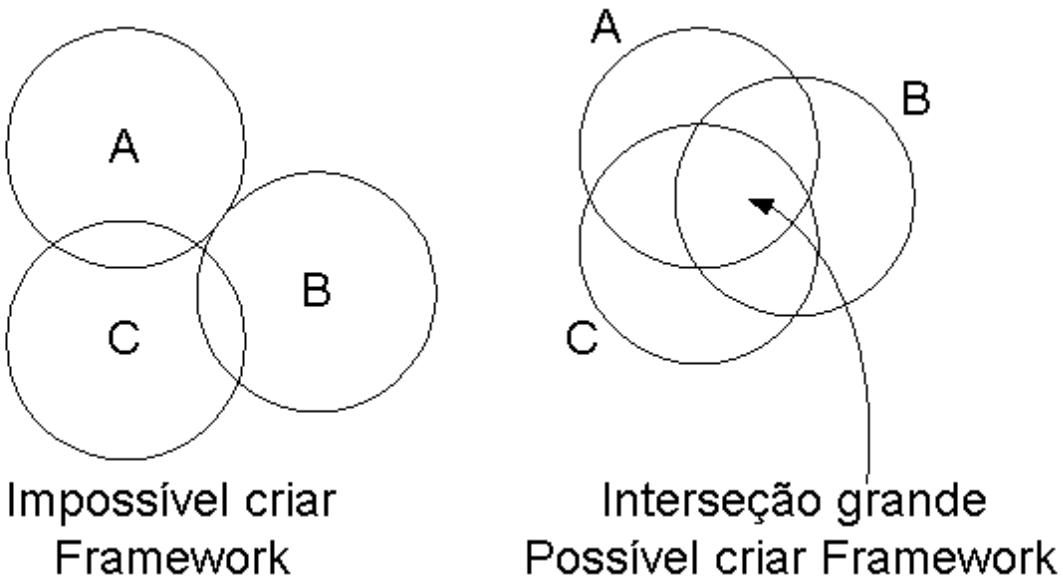



Figura 4.1: Quando é possível criar um *Framework*, segundo Jacques Sauvé (Frameworks e Componentes. 2000).

vantagens em utilizá-lo, portanto, ficam bem claras: redução de tempo e custos, a partir da maximização de seu re-uso. Detalhando mais das suas vantagens, quem está usando o *framework*, se concentra em adicionar valor em vez de "reinventar a roda" . Já as suas desvantagens, estaria basicamente na sua construção. Desenvolver um *framework* é bastante complicado, o que requer um bom planejamento. Demandando, portanto, muito tempo e esforço. Como não é o objetivo deste projeto desenvolver um *framework*, mas sim apenas utilizar um pronto. Vamos ficar, então, somente com as suas vantagens.

Voltando ao desafio desse projeto, o desenvolvimento de um assistente virtual para auxiliar nos pedido de *delivery*, temos, então, um problema em domínio específico: o *chatterbot*. E, portanto, por todas as vantagens explicadas, esse é o motivo de escolhermos trabalhar no projeto usando um *framework*. A escolha do Rasa em específico, se baseou primeiramente, em buscar uma ferramenta *open source*, ou seja, de código aberto. Analisamos também, sobre quais tecnologias foram utilizadas para construir a ferramenta, e a linguagem de programação utilizada para se trabalhar com ela. Explicaremos mais a seguir.

## 4.2 O que é *Open Source?*

A definição de *Open Source*, ou código aberto, em tradução livre, é um termo baseado na Definição Debian de *Software* Livre (DFSG) [24]. Foi esboçado primeiramente por Bruce Perens, sendo editado e aprovado como política formal pela comunidade de desenvolvedores Debian em 1997. Em fevereiro de 1998, foi revisado pela DFSG e removido as referências específicas do Debian, ganhando a denominação de "*The Open Source Definition*" , e adotado pela *Open Source Initiative* (OSI) durante o seu lançamento nesta mesma data [25].

O "*The Open Source Definition*" ou "A Definição de Código Aberto" determina se uma licença pode ou não ser considerada *open source*. Pois ser *open source* não implica apenas em ter acesso ao código-fonte, contendo as instruções, em uma das linguagens de programação, que descrevem o funcionamento de uma aplicação ou *software*.

Vamos então conhecer as regras ou princípios que definem se um *software* é ou não *open source* [26] [27]. Vejamos a seguir:

- Distribuição livre. A licença não deve restringir ninguém de dar ou vender o *software* bem como distribuí-lo numa distribuição de *software* de várias fontes.
- O código fonte tem que estar incluído ou ter acesso gratuito.
- Trabalhos derivados. As modificações e redistribuições do *software* têm de ser permitidas bem como a possível redistribuição do mesmo sobre a mesma licença
- Integridade do autor do código fonte. A licença pode restringir o código de ser distribuído modificado apenas se as mesmas licenças permitirem que as modificações sejam redistribuídas como *patches*.
- Sem discriminação contra pessoas ou grupos. Ninguém pode ser bloqueado ao uso do *software*.
- Sem discriminações contra grupos de trabalho. A sua utilização não pode ser vedada a nenhum tipo de fim como por exemplo o seu uso num negócio ou numa pesquisa genética.

- Distribuição da licença. A licença que vai com o *software* tem que se referir ao programa todo sem a necessidade de licenças adicionais por outros grupos.

- Licença não deve ser específica de um produto. A licença do programa não pode ser dependente do facto do mesmo fazer parte de uma distribuição particular de *software*.

- Licença não pode ser restritiva a outro *software*. Esta não pode obrigar que outro *software* com o qual é incluída, sejam *open source* por exemplo.

- Licença deve ser tecnologicamente neutra. Não devem ser obrigatórias formas específicas de aceitar a licença.

Mencionado sobre linguagem de programação, e como o *Rasa* utiliza sua abstração de suas funções em *Python*, essa linguagem foi também uma das características a ser analisada para a escolha deste *framework.* Vejamos a seguir.

## 4.3 Linguagem de programação *Python*

Como citado, anteriormente, no tópico 3.3 - Linguagem Natural -, segundo Russel e Norving, temos dois tipos de linguagens: Linguagens Formais e as Linguagens Naturais. Dentro de um modelo de linguagem precisamente definido, temos então, o *Python*.

Bem, de maneira geral, a definição que *Python* é um tipo de Linguagem Formal ainda é uma definição muito ampla. Vamos então, partir da seguinte pergunta: para que serve uma linguagem de programação? Para responder isso, faremos uma simples pesquisa no Wikipedia para descobrir o que é programação de computador. E encontramos que a programação de computadores é:

"... um processo que leva de uma formulação original de um problema de computação a programas de computador executáveis. A programação envolve atividades como análise, desenvolvimento, geração de algoritmos, verificação de requisitos de algoritmos, incluindo correção e consumo de recursos e implementação (geralmente referido como codificação) de algoritmos em uma determinada linguagem de programação."

Em resumo, a codificação está dizendo a um computador para fazer algo usando

uma linguagem que ele compreende. A partir desse conhecimento, já entendemos a importância de uma linguagem de programação dentro da solução proposta deste trabalho, que no final, um *chatterbot* será um programa de computador.

A escolha de uma linguagem de programação parte de cada um, de acordo com a sua familiaridade e nível de entendimento. Porém, para grande maioria dos *frameworks,* eles foram desenvolvidos em uma determinada linguagem, para ser usado especificamente com essa linguagem. Existindo, porém, maneiras integrar os serviços dessas ferramentas ou dos *softwares*, que foram construídos sobre diferentes linguagens de programação. Adiantando um pouco, essa integração pode ser feita a partir de um protocolo comum de comunicação, chamados "Interface de Programação de Aplicações" , comumente conhecido como API. Veremos mais adiante.

Voltando ao *Python* e o que nos motivou a escolha do *framework* Rasa, além, lógico, de ser uma linguagem que já estamos um pouco familiarizados, ainda tem as seguintes qualidades (Romano, 2015):

- **Portabilidade**. *Python* pode ser executado em praticamente todos os Sistemas Operacionais. Por exemplo, um programa desenvolvido em *Python* do Linux, pode ser executado também no Windows ou no Mac, dependendo apenas de uma questão de corrigir caminhos e configurações.

- **Coerência**. *Python* é muito lógico e coerente. Na maioria das vezes você pode adivinhar como um método é chamado, se você não souber. Talvez, não perceba o quanto isso é importante agora, especialmente se estiver no começo, mas essa é uma característica, que no decorrer do desenvolvimento, irá lhe trazer menos confusão e menos informações para ser pesquisado através de documentação.

- **Produtividade**. De acordo com Mark Lutz (*Learning Python*, 5ª edição, O'Reilly Media), um programa em *Python* tem tipicamente um quinto a um terço do tamanho do código *Java* ou *C ++* equivalente. Isso significa que o mesmo programa pode ser feito de modo mais rápido. Outro aspecto importante é que é executado sem a necessidade de etapas um pouco demoradas, como por exemplo, a de compilação, ou seja, não precisa esperar para ver os resultados do seu trabalho. Basta codificar e executar.

A partir dessas características como qualidades da linguagem de programação

*Python*, já é o suficiente para justificarmos o seu peso dentro da nossa escolha pelo *framework* Rasa.

## 4.4 Processando a linguagem natural com spaCy

Bem, já falamos sobre *Framework, Open Source, Python*, mas falta ainda tratar do núcleo de funcionamento do assistente virtual, de modo mais geral, como vamos transformar uma linguagem natural em algo que o computador compreenda, ou seja, como fazer esse processamento. Inicia-se aqui a "inteligência" do *chatterbot*.

Recapitulando, conforme mencionado no tópico "Linguística e Processamento de Linguagem Natural" , *chatterbot* é uma das formas de aplicar PLN, que é uma subárea da Inteligência Artificial. Portanto, *chatterbot* está contido no domínio de IA.

Foi mencionado no título deste tópico, o termo spaCY. Mas então o que é spaCy? Uma pesquisa rápida dentro de sua documentação, disponível no seu próprio *website*, obtivemos a seguinte resposta, em tradução livre:

"*spaCy é uma biblioteca gratuita e de código aberto, para Processamento de Linguagem Natural (PLN) avançado em Python.* (Brennan, 2017)"

Indo por partes, temos nessa definição um novo termo: biblioteca. Separaremos em um novo subtópico, a fim de tratar um pouco mais sobre esse assunto.

### 4.4.1 Biblioteca vs. *Framework*

Lembrando que o objetivo deste projeto não é aprofundar demais em tais conceitos, mas sim, buscar explicar de maneira que seja possível atingirmos o seu principal foco, que é o desenvolvimento do *chatterbot* usando o *framework* Rasa. Por isso, vamos fazer uma pesquisa rápida no *Wikipedia,* para buscarmos o significado do termo Biblioteca (em computação):

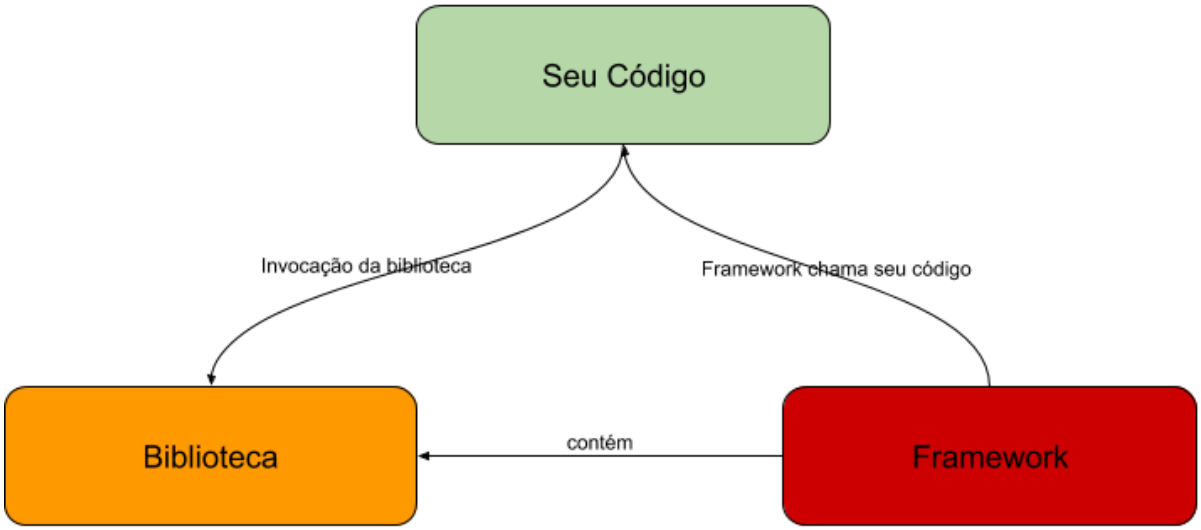
"*...biblioteca é uma coleção de subprogramas utilizados no desenvolvimento de software. Bibliotecas contém código e dados auxiliares, que provém serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular. Alguns executáveis são tanto programas independentes quanto bibliotecas, mas a maioria das bibliotecas não são executáveis. Executáveis e bibliotecas fazem referências*

*mútuas conhecidas como ligações, tarefa tipicamente realizada por um ligador. [30]*"

Se compararmos biblioteca e *framework*, talvez possa gerar um pouco de confusão entre os dois termos. E a diferença entre esses dois pode, algumas vezes, se tornar uma fonte de debate (Brennan, 2017). Para evitarmos essa confusão, vamos retornar a definição de *framework* que concluímos no tópico anterior: um *framework* abstraí complexidades e códigos que são comuns a um determinado problema, com a principal característica de conter técnicas de reutilização. Bem, se analisarmos agora a definição de Biblioteca que pesquisamos no *Wikipedia*, resumi-la em um conjunto de código e dados que auxiliam, e que também podem ser compartilhados em programas independentes, percebemos claramente a proximidade entre esses dois termos. Ou seja, todos eles abstraem complexidades e também compartilham características de reuso.

Mas Jacques Sauvé, detalha um ponto importante: no *framework* o conjunto de classes deve ser flexível e extensível para permitir a construção de várias aplicações, especificando apenas as particularidades de cada aplicação. Ao usar esse o termo "particularidades" , Jacques Sauvé consegue diferenciar esses dois pontos. Ou seja, um *framework* contém apenas abstrações de funcionalidades e nas Bibliotecas já temos as implementações das funcionalidades.

Para concluir, e deixar mais claro essa diferença, vejamos a seguir o seguinte gráfico que também pode definir os dois termos (Brennan, 2017).



### 4.4.2 Características do spaCy

Agora que já compreendemos a definição de Biblioteca, continuaremos a falar sobre o spaCy. Vamos seguir, basicamente, a sua documentação [31]. Partindo de início para a suas características, pois já sabemos que é uma biblioteca que irá tratar das funcionalidades de PLN.

Algumas das características do spaCy, referem-se a conceitos linguísticos, enquanto outras estão relacionados a funcionalidades mais gerais de aprendizado de máquina. Vamos conhecê-las.

- **Modelos estatísticos**. Embora alguns dos recursos do *spaCy* funcionem de forma independente, outros exigem que sejam carregados modelos estatísticos, permitindo que o *spaCy* faça previsão de anotações linguísticas. Por exemplo, se uma palavra é um verbo ou um substantivo. Atualmente, o *spaCy* oferece modelos estatísticos para 8 idiomas, que podem ser instalados como módulos individuais do *Python*.

- **Anotações linguísticas**. O *spaCy* fornece uma variedade de anotações linguísticas, a fim de fornecer informações sobre a estrutura gramatical de um texto. Incluindo o tipo de palavras e como estão relacionadas uma com a outra. Por exemplo, ao analisar um texto, faz uma enorme diferença saber se um substantivo é o assunto de uma sentença ou o objeto - ou se "andar"é usado como verbo, ou se refere a uma característica da maneira de se caminhar, nesse caso como substantivo.

- **Tokenização**. Tokenizar, segundo a própria documentação do *spaCy*, é descrito como o ato de "segmentar texto em palavras, pontuações, etc." . Esse processo de tokenização tem como objetivo separar palavras ou sentenças em unidades. Permitindo-nos fazer análises linguísticas, de acordo o que já foi visto em Linguística e Processamento de Linguagem Natural.

- **Marcação de parte da fala e Análise de Dependência**. Com a tokenização, o spaCy consegue atribuir tipos de palavras a cada *token* gerado, como verbo ou substantivo, ou seja, é possível fazer uma análise sintática e descrever as relações entre os tokens individuais. Veja figura 2.

- **Entidade nomeada**. Para o spaCy, uma entidade nomeada é um objeto o qual é atribuído um nome do mundo real - por exemplo, um país, uma pessoa, etc. O

spaCy pode reconhecer vários tipos de entidades. Mas, por ser um universo muito grande para definir todas as entidades, dependendo do seu tipo de caso, possa vir a precisar de algum ajuste.

- **Similaridade vetorial de palavras**. spaCy é capaz de comparar dois objetos e fazer uma previsão de quão semelhantes eles são . A previsão de similaridade é útil para criar sistemas de recomendação ou para indicar informação duplicada.

- **Treinamento**. Os modelos do SpaCy são estatísticos e cada "decisão"que eles fazem - por exemplo, marcação de parte da fala e análise de Dependência, ou decidir se uma palavra é uma uma entidade nomeada - é uma previsão. E essa previsão é baseada nos exemplos que o modelo viu durante o treinamento.

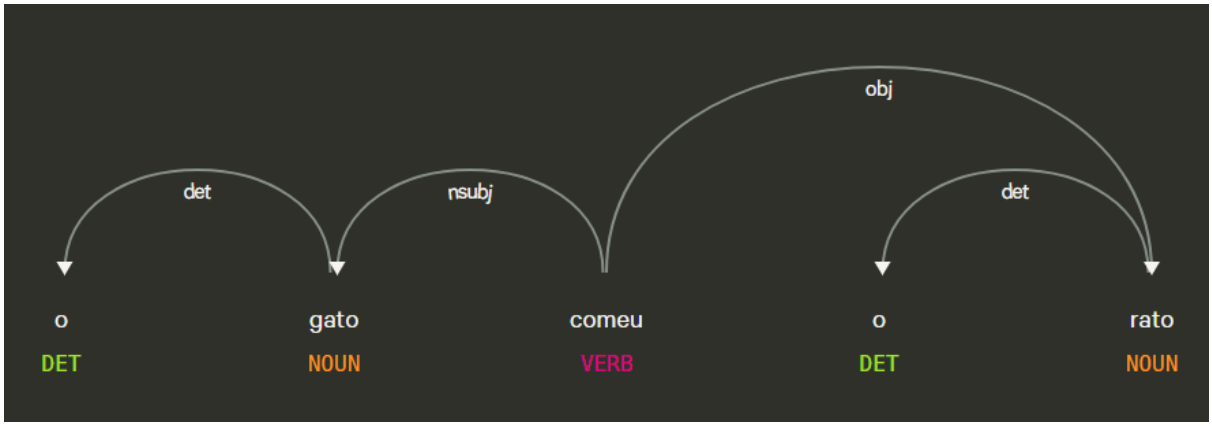



Figura 4.2: A marcação (tag) dos tokens gerados de uma sentença, juntamente com a sua análise de dependência.

O Rasa também permite trabalhar com outras bibliotecas de NLP - por exemplo, Mitie ou TensorFlow [32]. A escolha do spaCy partiu basicamente de uma escolha pessoal, pela simplicidade que a documentação transmitiu, por ser *open source* e usar a linguagem *Python*, ser também uma biblioteca de uso profissional - como consta em sua página principal na internet: "spaCy foi projetado para ajudar a realizar trabalhos reais - para construir produtos reais [33]" . E por fim, o suporte de modelos estatístico em português.

## 4.5 O *Framework* Rasa

Depois, de abordar todo o processo de decisões e escolha e conceitos prévios, enfim, chegamos na ferramenta que será responsável por nos ajudar a desenvolver o nosso *chatterbot*: o *Framework* Rasa. Nesse tópico, abordaremos o Rasa em duas partes: Rasa NLU e Rasa Core. Por fim, demonstraremos na arquitetura a integração de ambos.

Toda a nossa referência vai partir especificamente da documentação da ferramenta. Isso vai nos permitir, extrair análises não somente da ferramenta como um *framework* para desenvolver *chatterbots*, mas também da qualidade de sua documentação, que impactará no nível de dificuldade para alcançar o objetivo proposto do projeto.

### 4.5.1 Rasa NLU

Segundo a documentação, Rasa NLU é uma ferramenta de código aberto para classificação de intenção e extração de entidade [34]. E partindo da conclusão anterior, de que NLU procura entender a linguagem, permitindo que os computadores leiam e compreendam o texto, nos dá por base a função do Rasa NLU.

A intenção, mencionada na definição do Rasa NLU, é basicamente a compreensão do texto. E entidades são os nomes do mundo real, as quais falamos nas características do spaCy, os nomes de pessoas, países, produtos, etc. E por aí, já temos a ideia de onde a biblioteca spaCy será usada. Faltando, apenas dizer, como será usada. Mas vou deixar essa parte para o tópico que mencionaremos a implementação completa do *chatterbot*.

### 4.5.2 Rasa Core

Com o Rasa NLU vamos extrair as intenções das sentenças de entrada - por exemplo, se a pessoa quem está conversando com o assistente virtual, está querendo saber sobre um item específico do cardápio ou se ele está apenas cumprimentando - e também, as entidades. Mas isso só, não é suficiente para funcionar um *chatterbot*. Precisamos de algo que trate e controle o fluxo da conversa, respondendo e interagindo diante dessas sentenças de entrada. Quando alguém cumprimentar o *chatterbot*, que ele responda de volta um cumprimento ou boas-vindas, e se for perguntado sobre um item do cardápio,

que retorne também a uma resposta que esteja de acordo. É para isso então, que temos o Rasa Core. Mas não é somente isso que a ferramenta faz não, temos outras funcionalidade. Citaremos a seguir.

#### 4.5.2.1 Rasa Core sem *Python*

Foi brevemente mencionado no tópico sobre *Python*, o termo "Interface de Programação de Aplicações" , conhecido como API. Vamos novamente, pesquisar de maneira rápida, com o objetivo apenas de alcançarmos um nível superficial de compreensão do termo, e que de uma maneira simples e geral, nos dará o entendimento de sua função. Para também não fugirmos do objetivo, e não complicar o experimento em si. Segundo, portanto o Wikipedia, API é:

"...um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do *software*, mas apenas usar seus serviços.[35]"

Resumindo, ao invés de codificarmos, implementando as funcionalidades em uma determinada linguagem de programação, uma API vai expor serviços específicos, abstraindo tanto as implementações quando o tipo de linguagem usada.

No Rasa Core, existe uma API HTTP*(Hypertext Transfer Protocol)* que irá realizar exatamente esse função de abstração [36]. E o usuário desta ferramenta, caso não saiba codificar em *Python*, isso não será será um obstáculo para a desenvolver um *chatterbot*. A menção desta API foi apenas para mostrar uma funcionalidade de desenvolvimento do *framework*, mas não será utilizada no experimento deste projeto.

#### 4.5.2.2 Integração com plataformas de mensagens

Outra funcionalidade do Rasa Core, é de estar preparado para integrar com diferentes tipos de plataformas de mensagens - por exemplo, *Facebook Messenger*, *Slack*, *Telegram*, e outros [36].

Para tornar o uso do *chatterbot* mais amigável ao usuário, precisamos de uma interface gráfica, que irá permitir o seu uso, expondo-o ao mundo exterior. Aproveitamos que hoje já temos diversas plataformas oferecendo serviços de troca de mensagens, e que essas plataformas permitem ser acessadas através de seus serviços de API, o que o Rasa

então irá fazer, é conectar o *chatterbot* criado com ela, a essas plataformas. Ou seja, todo o fluxo de conversa, entrada e respostas, pode ser feito em qualquer plataforma que o Rasa tem um conector. No tópico de implementação, vamos detalhar a integração com o *Facebook Messenger*.

## 4.6 Conhecendo o problema

Abordamos, nos tópicos anteriores, noções e características do *framework* Rasa. Mas antes de entrarmos na implementação da solução, vamos conhecer mais sobre o problema tema deste projeto.

O objetivo é desenvolver um *chatterbot* que auxilie nos pedidos de *delivery* de um restaurante. Para detalhar melhor esse papel de ajuda que será oferecido através do nosso assistente virtual, partiremos da premissa que, antes de um cliente realizar o seu pedido, usando uma plataforma ou *interface* de comunicação por mensagens - por exemplo, o *Facebook Messenger* -, ele vai buscar conhecer os itens ou opções presentes no cardápio deste restaurante, para então, fazer a sua escolha. Com isso, primeiramente precisamos conhecer é o próprio cardápio. Para o restaurante, denominaremos como *DG Food*. Vejamos o seu cardápio na Tab. 4.1:

| Cardápio de Sanduíches - *DG Food* | |
|---|---|
| Nome do sanduíche | Descrição |
| Brigitte | Pão com gergelim, hambúrguer artesanal de frango de 135g, queijo cheddar, alface, tomate, cebola roxa e molho verde. Preço: R$ 16,40. |
| Madonna | Pão com gergelim, hambúrguer artesanal de carne bovina de 135g, queijo cheddar, alface, tomate, cebola roxa e molho rosé. Preço: R$ 17,40. |
| Elvis | Pão com gergelim, hambúrguer artesanal de carne bovina de 135g, bacon, queijo cheddar, alface, tomate, cebola roxa e molho especial da casa. Preço: R$ 19,40. |
| Tim Maia | Pão com gergelim, hambúrguer artesanal de carne bovina duplo de 135g cada, bacon, queijo cheddar, alface, tomate, cebola roxa e molho especial da casa. Preço: R$ 24,00. |

Tabela 4.1: Cardápio de sanduíches do restaurante *DG Food*.

Como o problema está relacionado ao auxílio de pedido, ou seja, uma solicitação de compra, logicamente, o cliente também vai querer conhecer as formas de pagamento que ele poderá utilizar. Então, vejamos a tabela logo abaixo:

| Formas de pagamento | |
|---|---|
| Método | Descrição |
| Cartões de crédito | Bandeiras MasterCard e Visa. |
| Cartões de débito | Bandeiras MasterCard e Visa. |
| Vale Alimentação | Bandeiras Sodexo e Elo. |
| À vista | Dinheiro |

Tabela 4.2: Formas de pagamento aceitas pelo restaurante *DG Food*.

Comentando um pouco, sobre essas duas tabelas, temos que:

1) Na tabela de cardápio(Tab. 4.1) demos um nome para cada sanduíche - esses nomes serão usados para trabalharmos com entidades nomeadas -, uma descrição de ingredientes e o valor de cada um. Na descrição, temos 3 (três) opções de hambúrgueres: frango, bovino simples e bovino duplo.

2) Na tabela de formas de pagamentos, temos o método e na descrição nomes relacionados a esses métodos - por exemplo, nomes de bandeiras (empresas) do método cartão de crédito.

A partir dessas duas tabelas, definimos o domínio do problema em que o *chatterbot* irá atuar. Resumindo, o cliente que está buscando fazer seu pedido de *delivery*, poderá solicitar ao assistente virtual informações como, a descrição dos itens presente no cardápio ou perguntar também sobre métodos de pagamento aceito pelo restaurante *DG Food*.

# 5 Testes e Resultados

Trataremos aqui, a implementação da solução em si. Abordando, pontos necessários para compreensão do processo de desenvolvimento. Vimos as principais definições presente no *framework* Rasa, mas detalhes não menos importante, será necessário para uma boa compreensão.

## 5.1 Documentação oficial

Vamos falar um pouco sobre a documentação oficial do *framework* Rasa, que foi a nossa fonte principal de referência para trabalhar com a ferramenta.

Dentro dessa documentação existem 4 (quatro) tutoriais que ensinam, a partir de exemplos já definidos, como usar a ferramenta na construção de um *chatterbot*, são eles: *Building a Simple Bot [37], Supervised Learning Tutorial [38], Interactive Learning [39] e Rasa Core without Python [40].* Buscando familiarização com o *framework*, parti para o primeiro tutorial da lista, que é a construção de um *bot* simples. Vejamos a definição desse tutorial, segundo a sua documentação:

"...criar seu primeiro *bot*, adicionando todas as partes de um aplicativo Rasa, que verifique o nosso estado de humor atual e tente nos motivar quando estivermos tristes. Ele consultará nosso humor e, com base em nossa resposta, responderá com uma imagem engraçada ou uma mensagem."

Seguindo a documentação, foi possível estruturar e conhecer melhor todas as partes de desenvolvimento do *chatterbot* usadas pelo Rasa, incluindo: a sua instalação e configuração, definição de domínio, definição do interpretador NLU, definição das histórias, a junção de todas as partes anteriores, e por fim, um item bônus de como integrar a solução com o *Facebook Messenger*.

Concluindo o tutorial básico, mantendo a ordem, iniciei o tutorial de aprendizado supervisionado. Vamos então, conhecer o objetivo deste tutorial:

"...criaremos um *bot* de pesquisa de restaurantes, treinando uma rede neural

a partir de exemplos de conversas. Um usuário pode perguntar algum tipo restaurante - por exemplo, um restaurante mexicano -, e o *bot* perguntará mais detalhes até que esteja pronto para lhe sugerir um restaurante."

A diferença deste exemplo com o primeiro, está na complexidade do fluxo da conversa. Necessitando ao segundo, uma melhor definição de histórias, que será a base de treinamento para o que o *bot* possa aprender. E é em cima dessa dificuldade de se definir as histórias, no caso de não se ter pronto uma base de treinamento, que trata o terceiro tutorial. Nele é possível conhecer uma forma mais simples de gerar essas histórias, que é através de um "treinamento *online*" . A partir do conhecimento em cima desses três exemplos, temos informações suficiente para atingirmos o objetivo deste projeto.

## 5.2 Instalação

Destacamos que, todo o ambiente de desenvolvimento foi instalado e configurado, utilizando o Sistema Operacional (SO) Linux/Ubuntu versão 16.04. Mas, como a linguagem *Python*, como mencionamos, é uma linguagem de programação que pode ser executado em praticamente todos os Sistemas Operacionais. Então, tudo aqui que foi feito, também pode ser usado em qualquer outro SO de sua preferência. E o que irá mudar, será basicamente a maneira de instalarmos a linguagem *Python*. E por ter adiantado o assunto, iremos, primeira instalar a versão 3.6 do *Python*.

Seguindo as orientações da documentação oficial do Rasa, pede-se que instalemos uma distribuição específica da linguagem Python, chamada: *Anaconda*. Que segundo o seu *site* oficial, destaca que:

"*...a Distribuição Anaconda é open source, e também, a maneira mais fácil e rápida, usada para fazer ciência de dados e aprendizado de máquina em Python e R, tanto no Linux, Windows ou Mac OS X. É o padrão da indústria para desenvolvimento, teste e treinamento em única máquina [41].*"

Ao citar ciência de dados e aprendizado de máquina, concluímos que, a *Distribuição Anaconda* ou, simplesmente, *Anaconda*, irá nos ajudar a trabalhar com a área da Inteligência Artificial, usando as linguagens de programação *Python* ou *R*. Porém, em nosso projeto, utilizaremos apenas a primeira.

Continuando, a instalação é feita baixando a última versão presente no seu en-

dereço oficial, em: `https://www.anaconda.com/download/` - o redirecionamento é feito automaticamente para baixar a distribuição correta, a partir do seu SO de origem. Concluindo a instalação, vamos conferir se o Python está funcionando, para isso, abrimos o console terminal [1] e digitamos:

*python -V*

Deverá ver o retorno em tela, algo como:

*Python 3.6.3 :: Anaconda, Inc.*

Confirmamos então, que a Distribuição Anaconda foi instalada corretamente, e que o *Python* também está funcionando. O próximo passo, é instalar o *framework Rasa*. Ainda console terminal, digitamos:

*pip install rasa_core*

Para o Python, existe um repositório de *software* chamado "*Python Package Index* (PyPI)" , em tradução livre: Índice de Pacotes Python. O PyPI ajudará a encontrar e instalar *softwares* desenvolvidos e compartilhados pela comunidade *Python* [42]. E *pip* é um programa de linha de comando [43], utilizado para instalar localmente os pacotes ou *softwares* existentes em seu repositório. No caso do Rasa, como é um *software* que está no repositório *PyPI*, para a sua instalação, basta-se utilizar o comando acima mencionado.

Lembrando que no tópico que abordamos o *framework* Rasa (4.6), fizemos a separação em *Rasa Core* e *Rasa NLU*. Até agora, instalamos somente o *Rasa Core*, precisamos então, adicionar o NLU. Digitamos:

*pip install rasa_nlu[spacy]*

Esse comando por si, é bem explicativo. Lembrando que iremos utilizar a biblioteca spaCy, que será responsável por trabalhar com Processamento de Linguagem Natural, e o NLU que fará a classificação de intenção das sentenças e extração de entidades, que faz parte do NLP, necessita, portanto, chamar o spaCy. E como o NLP será feito sobre a língua portuguesa, instalaremos o modelo de idioma que dará suporte ao spaCy para essa linguagem:

*python -m spacy download pt*

Apenas uma observação, o *Rasa Core* permite usar outros serviços NLU, não

[1] No Ubuntu, basta utilizarmos as teclas de atalho, pressionadas juntas, Ctrl + Alt + T.

necessariamente, o Rasa NLU - por exemplo, *wit.ai*, *dialogflow* ou *LUIS*. E com isso, finalizamos a instalação das principais ferramentas necessárias para desenvolvimento e implementação de um *chatterbot*.

## 5.3 Implementação

Antes, vamos já conhecer toda a estrutura de pastas e arquivos que compõem o *chatterbot* do restaurante *DG Food*. Vejamos a figura abaixo:

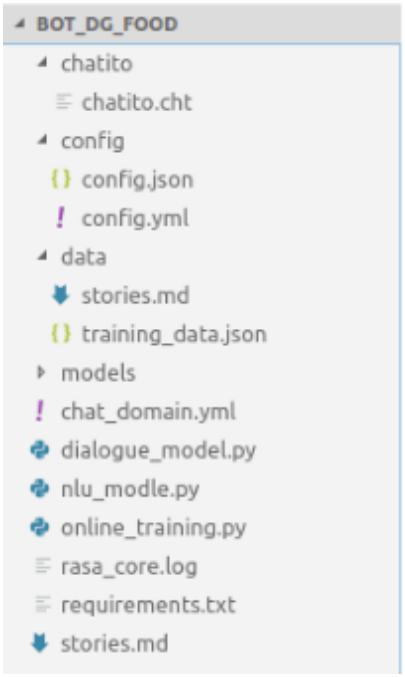



Figura 5.1: Estrutura de pastas e arquivos do projeto *chatterbot*.

Para manter um nível de organização textual, resolvemos incluir nesse projeto, apenas trechos de código, sendo que o trabalho completo se encontra disponível em um repositório público, no seguinte endereço: `https://github.com/danillogontijo/bot_dg_food`.

### 5.3.1 Construindo o domínio

O domínio define o universo em que *chatterbot* vai operar. Nele são listadas as intenções (*intents*), ações (*actions*), modelos (*templates*) de respostas pré definidas, entidades e *slots*.

*Intents* e entidades são as mesmas já definidas no modelo de NLU.

*Actions* são tudo o que o *chatbot* irá responder como texto, não estando limitado somente a respostas textuais, pode-se definir ações mais complexas - por exemplo, uma chamada para uma API externa ou uma busca em um banco de dados. Essas ações mais complexas podem ser usadas para gerar a resposta que o *chatbot* irá dar ao usuário.

*Slots* são atributos que serão mantidos no contexto para serem reutilizadas em novas iterações com o usuário. Essa é uma das maneiras que o *Rasa* mantem o contexto da conversa fazendo com que ela possa ser mais semelhante a conversa de um humano real. Neste projeto não teremos o uso de *slots*.

O Rasa usa as *intents*, as entidades e o estado interno do diálogo para selecionar a próxima ação que deve ser executada. Se for uma ação de texto, um *template* correspondente é selecionado, preenchendo qualquer variável necessária, e enviando de volta esse texto como resposta. Para ações complexas pode se definir através de classes em *Python*, fazendo apenas o referência da chamada do método dentro de *actions* - por exemplo, - *bot.ActionBuscarItensCardapio*.

Exemplo de arquivo de domínio:

```
slots:
  item_escolhido:
    type: text

intents:
    - cumprimento
    - sanduiche_cardapio

entities:
    - cardapio

templates:
    utter_cumprimento:
        - 'Olá, como posso lhe ajudar?'
        - 'Oi, estou aqui para ajudar.'
    utter_cardapio:
        - 'Nosso cardápio é composto por 5 tipos de sanduiches: brigitte, madonna, elvis, raul e tim maia'

actions:
    - utter_cumprimento
    - utter_cardapio
```

Figura 5.2: Modelo de um arquivo de domínio

Em nosso projeto, o arquivo de domínio se encontra na pasta raiz, com o nome de *chat_domain.yml*. No domínio do *chatterbot* deste projeto, foi definido 5 intenções. São elas:

- **cumprimento**: responsável por iniciar a conversa com *bot*, a partir de um olá, bom dia, entre outras formas de cumprimento.

- **sanduiche_cardapio**: intenção que irá tratar possíveis sentenças que solicitam uma ação de envio do cardápio do restaurante *DG Food*.

- **sanduiche_brigitte**: define a intenção de obter informações sobre o sanduíche *brigitte*.

- **sanduiche_bovino**: define a intenção de obter informações sobre sanduíches que tem como ingrediente hambúrguer bovino.

- **forma_pagamento**: define a intenção de obter informações sobre os métodos aceitos como pagamento.

E para determinada sentença que entrada, o *bot* poderá tomar as seguintes 7 tipos diferentes de ações.

- **utter_cumprimento**: a partir da entrada com a intenção de cumprimentar, poderá responder de duas maneiras: "Olá, como posso lhe ajudar?" e "Oi, estou aqui para ajudar" . Essa escolha é aleatória do *bot*.

- **utter_cardapio**: a partir da *intent sanduiche_cardapio*, o *bot* responderá com o texto correspondente ao cardápio.

- **utter_sanduiche**: ao ser solicitado o cardápio, o *bot* irá complementar com a pergunta: "Gostaria de obter mais informação de um sanduiche específico? Digite o nome dele ou tente alguma especificação qualquer" . Essa ação irá orientar o usuário dentro do fluxo da conversa, para que ele possa dar prosseguimento, indicando o que poderá ser perguntado para o *bot*.

- **utter_sanduiche_brigite**: quando o usuário perguntar ao *bot*, informações sobre o sanduíche brigite, este responderá com a descrição desse item em específico.

- **utter_sanduiche_bovino**: já aqui, é a intenção de perguntar sobre um ingrediente específico do sanduíche, o tipo do hambúrguer. E igual a utter_cumprimento, essa ação poderá terá duas respostas: descrever o próprio hambúrguer bovino ou informar quais itens do cardápio contém hambúrguer bovino.

- **utter_algo_mais**: essa *utter* tem a intenção de transmitir a quem está conversando com o *bot*, informado que ele está aguardando uma nova sentença. Fazendo o *bot*

sempre responder no final de uma resposta o seguinte: “Posso lhe ajudar em algo mais? Faça uma nova pergunta” .

- **utter_forma_pagamento**: essa *utter* já é para informar ao usuário as formas de pagamento aceitas pelo restaurante *DG Food*, quando este escrever uma sentença de entrada que tem intenção de obter esse tipo de informação.

### 5.3.2 Definindo as Histórias

O *Rasa* constrói histórias baseadas em exemplos. Estes exemplos são conversas completas que são utilizadas para treinar o *chatterbot*. A partir dessa conversa, um fluxo de decisão do *chatbot* é criado. Tornando assim, capaz de definir qual ação deve ser tomada baseado na entrada do usuário.
Essa opção não é sempre a mais indicada, pois nem sempre existem dados suficientes para realizar o treinamento do *chatbot* dessa forma. Para esses casos, existe o treinamento *online*. Esse treinamento consiste em conversar com o *chatbot*, gerando um fluxo de aprendizado supervisionado. A cada entrada será mostrado a *intent* que foi identificada e a ação que o *bot* acha que deve fazer, sendo possível corrigir tanto a ação quanto a *intent* quando necessário. A cada erro o *bot* será treinado levando em conta o novo caminho que foi informado. Isso faz com que o *chatterbot* possa estar sempre evoluindo, pois através desse aprendizado supervisionado é fácil corrigir comportamentos inesperados e melhorar a confiança das respostas.

Para usar o treinamento *online* e gerar o conjunto de dados de histórias, basta executar o seguinte comando:

*python online_training.py*

Todo o código necessário para executar, se encontra no arquivo *online_training.py*, presente na raiz da estrutura do projeto. E o conjunto de dados de histórias, encontra-se em *data/stories.md*. Segue abaixo, o exemplo de uma história gerada.

Percebe-se na Fig. 5.3 que dentro do contexto de uma história, abaixo de cada *intent*, temos uma ação do *bot*, ou seja, uma resposta utilizando as *utters*.

Criar histórias é simples, porém descrever todas as histórias e combinações geraria um esforço muito grande, e talvez nem existisse memória suficiente para acomodar esses dados. Para fazer essa generalização são utilizadas políticas (*policies)*, ou seja, um

```
## Generated Story 7914822348745348761
* cumprimento
    - utter_cumprimento
* sanduiche_cardapio{"cardapio": "menu"}
    - utter_cardapio
    - utter_algo_mais
* forma_pagamento
    - utter_forma_pagamento
    - utter_algo_mais
* forma_pagamento{"forma_pagamento": "dinheiro"}
    - utter_forma_pagamento
    - utter_algo_mais
    - export
* cumprimento
    - utter_cumprimento
* sanduiche_cardapio
    - utter_cardapio
    - utter_algo_mais
* sanduiche_brigitte
    - utter_sanduiche_brigite
    - utter_algo_mais
* sanduiche_bovino
    - utter_sanduiche_bovino
    - utter_algo_mais
* forma_pagamento
    - utter_forma_pagamento
    - utter_algo_mais
```

Figura 5.3: Exemplo de história de conversa gerada pelo treinamento *online*.

padrão ou regra a ser seguido. A regra usada no projeto deste trabalho é a política *Keras*. *Keras* é uma API de redes neurais de alto nível, desenvolvida para execução de rápida prototipação com suporte para redes convolucionais e redes recorrentes. É possível implementar políticas diferentes ou alterar os modelos utilizados pelo *Keras* dentro do *Rasa*, porém tal exercício foge ao escopo deste trabalho.

Todo o modelo treinado aqui, através do treinamento online, é armazenado na pasta *models/dialogue/*.

### 5.3.3 NLU Model: treinando o *bot*

O grande desafio ao se criar um *chatterbot* é torná-lo inteligente, fazendo com que compreenda as frases do interlocutor, e assim possa responder de forma coerente e concisa. Para isto, é necessário que o *bot* tenha um bom NLU, ou seja, um bom

entendimento da língua.

Um *bot* com um bom NLU será capaz de extrair das frases passadas a ele a intenção do usuário e assim, definir qual a melhor resposta para cada situação. Um *chatterbot* terá esta capacidade se ele tiver um bom treinamento, este treinamento é dividido em três partes pelo *Rasa*: *Common Examples*, *Entity Synonyms* e *Regular Expression Features*. Vejamos cada um a seguir.

#### 5.3.3.1 *Common Examples*

São elencadas frases com as quais o *bot* pode se deparar, mostrando exatamente o que ele deve buscar em cada frase para que entenda como extrair a intenção de cada uma. Os *Common Examples* são divididos em três partes: *text*, *intent* e *entities*. *Text* é a frase, e este termo é obrigatório. A *intent* é opcional e demonstra qual a intenção a que deve ser associada a esta frase. As *entities* também são opcionais e definem as partes da frase que precisam ser identificadas. Abaixo temos um exemplo de um *Common Example* que utilizamos no nosso treinamento:

"*Pode me enviar o seu cardápio?*"

Neste exemplo temos:
*Text*: Pode me enviar o seu cardápio?
*Intent*: sanduiche_cardapio
*Entities*: cardápio

#### 5.3.3.2 *Entity Synonyms*

Como sabemos, na linguagem existem diversas maneiras diferentes de se transmitir a mesma mensagem, assim, nesta parte do treinamento podemos definir outros termos que possam substituir as *entities* a serem identificadas pelo *chatterbot*.

Vale ressaltar que duas palavras ou expressões podem não ser sinônimas na nossa língua, e mesmo assim serem definidas como *Entity Synonyms* desde que em um determinado contexto, a mudança da *entity* original pelo *entity synonym* faça sentido. Vejamos o exemplo abaixo, que foi extraído do treinamento utilizado para o nosso *bot*:

"*Que cardápio tem aí?*"

Neste exemplo, para a *intent* sanduiche_cardapio, a *entity* cardápio possui diversos sinônimos, como, por exemplo: menu, tipo de sanduíche, lanche, comida, opções, etc. Percebe-se que ao substituir cardápio por esses termos, a frase não perde o seu sentido, demonstra ainda a intenção do interlocutor de visualizar o cardápio.

#### 5.3.3.3 *Entity Synonyms*

As *Regular Expression Features* são utilizadas para facilitar a classificação de uma *intent* pelo *bot* e, posteriormente, a identificação da *entity*. Para isto, são passadas ao *chatterbot* o formato esperado por uma determinada *entity*.

Por exemplo, sabemos que o CEP deve possuir um formato, composto por números, XXXXX-XXX. Nesse caso, então, podemos utilizar uma expressão regular que representa o CEP. Não houve necessidade de se utilizar *Regular Expression Features* no nosso treinamento.

### 5.3.4 NLU *Dataset*: conjunto de dados para treinar o *bot*

Para que o *bot* possa compreender as intenções do seu interlocutor, necessariamente, precisa aprender a interpretar as suas sentenças de entrada. E para se obter esse conhecimento, necessita-se de um bom conjunto de dados que expressam bem as diversas possibilidades de frases, que fazem parte do domínio de atuação do *chatterbot*. Vejamos um trecho do *dataset* de treinamento que se encontra no arquivo *training_data.json* .

Esse arquivo completo tem exatamente 8.801 linhas. É um arquivo razoavelmente grande, para que possamos definir bem todas as possíveis sentenças de entradas. E para nos ajudar nessa tarefa, usamos uma ferramenta chamada *Chatito*. Vejamos a seguir.

#### 5.3.4.1 Chatito

Para a montagem deste treinamento, seguindo a estrutura entendida pelo *Rasa*, utilizamos, conforme sugerido por sua própria documentação, uma ferramenta denominada *Chatito*. Encontrada em seu *site* oficial - `https://rodrigopivi.github.io/`

```
{
  "rasa_nlu_data": {
    "regex_features": [],
    "entity_synonyms": [],
    "common_examples": [
      {"text": "boa tarde", "intent": "cumprimento", "entities": []},
      {"text": "obrigado", "intent": "cumprimento", "entities": []},
      {"text": "como vai?", "intent": "cumprimento", "entities": []},
      {"text": "boa noite", "intent": "cumprimento", "entities": []},
      {"text": "agradeço a atenção", "intent": "cumprimento", "entities": []},
      {"text": "tudo bem?", "intent": "cumprimento", "entities": []},
      {"text": "olá", "intent": "cumprimento", "entities": []},
      {"text": "bom dia", "intent": "cumprimento", "entities": []},
      {"text": "oi", "intent": "cumprimento", "entities": []},
      {"text": "ola", "intent": "cumprimento", "entities": []},
      {
        "text": "cardapio",
        "intent": "sanduiche_cardapio",
        "entities": [
          {"end": 8, "entity": "cardapio", "start": 0, "value": "cardapio"}
        ]
      },
      {
        "text": "quais os tipos de cardápio que tem",
        "intent": "sanduiche_cardapio",
        "entities": [
          {"end": 26, "entity": "cardapio", "start": 18, "value": "cardápio"}
        ]
      },
```

Figura 5.4: Trecho do arquivo de *dataset training_data.json*

`Chatito/` -, e pode ser utilizada neste mesmo local, sem a necessidade qualquer configuração ou instalação.

Com esta ferramenta, podemos nos preocupar com a criação dos exemplos e com a definição das intenções e entidades para o treinamento do *bot,* sem ser necessário também entender a sintaxe usada pelo Rasa NLU, pois o *Chatito* utiliza uma linguagem fluída e bem simples. Sendo que o *Chatito* fica responsável pela conversão do código escrito para um *dataset* com o formato esperado pelo *Rasa*. Mas o melhor de tudo, é as 8.801 linhas existente no *dataset* de treinamento, podem ser convertidas em algumas poucas dezenas de código. Vejamos um exemplo.

Com essa simples descrição da Fig. 5.5, é possível gerar o arquivo utilizado pelo *Rasa* (abaixo), e nele pode-se ver que ele gerou o *Common Examples* da forma esperada.

Da mesma forma adicionando o sinônimo “menu” para cardápio.

Temos o seguinte código gerado.

```
{
  "rasa_nlu_data": {
    "regex_features": [],
    "entity_synonyms": [],
    "common_examples": [
      {"text": "boa tarde", "intent": "cumprimento", "entities": []},
      {"text": "obrigado", "intent": "cumprimento", "entities": []},
      {"text": "como vai?", "intent": "cumprimento", "entities": []},
      {"text": "boa noite", "intent": "cumprimento", "entities": []},
      {"text": "agradeço a atenção", "intent": "cumprimento", "entities": []},
      {"text": "tudo bem?", "intent": "cumprimento", "entities": []},
      {"text": "olá", "intent": "cumprimento", "entities": []},
      {"text": "bom dia", "intent": "cumprimento", "entities": []},
      {"text": "oi", "intent": "cumprimento", "entities": []},
      {"text": "ola", "intent": "cumprimento", "entities": []},
      {
        "text": "cardapio",
        "intent": "sanduiche_cardapio",
        "entities": [
          {"end": 8, "entity": "cardapio", "start": 0, "value": "cardapio"}
        ]
      },
      {
        "text": "quais os tipos de cardápio que tem",
        "intent": "sanduiche_cardapio",
        "entities": [
          {"end": 26, "entity": "cardapio", "start": 18, "value": "cardápio"}
        ]
      },
```

Figura 5.5: Exemplo de código no *Chatito*

```
{
  "rasa_nlu_data": {
    "regex_features": [],
    "entity_synonyms": [],
    "common_examples": [
      {
        "text": "Pode me enviar o seu cardápio?",
        "intent": "sanduiche_cardapio",
        "entities": [
          {"end": 29, "entity": "cardapio", "start": 21, "value": "cardápio"}
        ]
      }
    ]
  }
}
```

Figura 5.6: Resultado da conversão do código *Chatito* da Fig. 5.5.

O arquivo completo do *chatito* usado para o projeto, encontra-se em *chatito/-chatito.cht*.

### 5.3.5 Executando o treinamento

Com o modelo definido é necessário que ele seja treinado. Esse treinamento consiste em classificar as *intents* e extrair as *entities*. E após esse treinamento, o modelo NLU pode ser utilizado para verificar entradas e definir com qual confiança aquela frase se enquadra em cada *intent*. Vejamos antes, os tipos de treinamentos feito pelo Rasa NLU:

- ***Intent Classification***. A classificação é feita utilizando vetores de palavras, e visa identificar nos dados de treinamento o que é uma *intent*, isso é feito utilizando a similaridade de palavras nos espaços vetorias já conhecidos pelo *Spacy* e pelo *Rasa NLU*.

```
%[sanduiche_cardapio]
    pode me enviar o seu @[cardapio]

~[cardapio]
    cardápio
    menu

@[cardapio]
    ~[cardapio]
```

Figura 5.7: Adicionado o sinônimo cardápio.


```
{
  "rasa_nlu_data": {
    "regex_features": [],
    "entity_synonyms": [],
    "common_examples": [
      {
        "text": "Pode me enviar o seu cardápio?",
        "intent": "sanduiche_cardapio",
        "entities": [
          {"end": 29, "entity": "cardapio", "start": 21, "value": "cardápio"}
        ]
      },
      {
        "text": "Pode me enviar o seu menu?",
        "intent": "sanduiche_cardapio",
        "entities": [
          {"end": 25, "entity": "cardapio", "start": 21, "value": "menu"}
        ]
      }
    ]
  }
}
```


Figura 5.8: Resultado da conversão do código *Chatito* da Fig. 5.7.

- ***Entity extraction***. Essa extração consiste em realizar um modelo probabilístico do que a sua frase deveria ser e como cada palavra tende a se transicionar para outras, retornando a entidade com a maior probabilidade de se encaixar nesse modelo.

Para treinar o modelo propriamente dito, iremos utilizar um *script* auxiliar em *Python* que iniciará o interpretador do *Rasa NLU,* e invocará o treinamento, passando como parâmetros as nossa *intents* e *entities* definidas anteriormente utilizando o *Chatito*. O arquivo contendo o script, encontra-se na raiz do projeto, com o nome de *nlu_modle.py*.

Invocando esse *script, o Rasa NLU* fará o treinamento e retornará a confiança da interpretação das nossa frases de teste.

```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

from rasa_nlu.training_data import load_data

from rasa_nlu.config import RasaNLUModelConfig
from rasa_nlu.model import Trainer, Metadata, Interpreter
from rasa_nlu import config

def train (data, config_file, model_dir):
    training_data = load_data(data)
    configuration = config.load(config_file)
    trainer = Trainer(configuration)
    trainer.train(training_data)
    model_directory = trainer.persist(model_dir, fixed_model_name = 'chat')

def run():
    interpreter = Interpreter.load('./models/nlu/default/chat')
    print(interpreter.parse('Pode me enviar o seu cardápio?'))
    print(interpreter.parse('Me mostra o menu?'))

if __name__ == '__main__':
    train('./data/training_data.json', './config/config.yml', './models/nlu')
    run()
```

Figura 5.9: *Script* de treinamento do NLU codificado em *Python*.

Para nossa frase “Pode me enviar o seu cardápio?” o *Rasa* retorna:

```
{'intent': { 'name': 'sanduiche_cardapio', 'confidence': 0.759540566172552  },
  'entities': [
  { 'start': 21,'end': 29,
    'value': 'cardápio',
    'entity': 'cardapio', 'confidence': 0.8336381424483267,
    'extractor': 'ner_crf' }
  ],
  'intent_ranking': [
    {'name': 'sanduiche_cardapio', 'confidence': 0.759540566172552},
    {'name': 'greet','confidence': 0.24045943382744822}
  ],
  'text': 'Pode me enviar o seu cardápio?'}
```

Figura 5.10: Retorno das frases de teste.

O que isso nos mostra é que o interpretador foi capaz de identificar que nossa *intent* era “sanduiche_cardapio” com uma confiança de 75.95% , e também, que nossa *entity* era “cardapio” , com 83.36% de confiança.

Todo o modelo treinado aqui pelo Rasa NLU, é armazenado na pasta models/nlu/default/chat/.

### 5.3.6 Integrando as partes

Temos, então até aqui, o conjunto de histórias, que define o fluxo de conversa entre o *chatterbot* e o seu usuário, treinado pela ferramenta de treinamento *online*. A classificação de intenções e a extração de entidades, também treinadas pelo Rasa NLU. O que precisamos agora, é integrar todas essas partes, para que possam trabalhar juntas, formando portanto, o nosso *chatterbot*.

Para realizar isso, executaremos o seguinte *script Python*, que está contido no arquivo *dialogue_model.py* de nosso projeto.

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

from rasa_core import utils
from rasa_core.agent import Agent
from rasa_core.policies.keras_policy import KerasPolicy
from rasa_core.policies.memoization import MemoizationPolicy
from rasa_core.featurizers import (MaxHistoryTrackerFeaturizer,
                                   BinarySingleStateFeaturizer)

if __name__ == '__main__':
    utils.configure_colored_logging(loglevel="INFO")

    training_data_file = './data/stories.md'
    model_path = './models/dialogue'

    featurizer = MaxHistoryTrackerFeaturizer(BinarySingleStateFeaturizer(),
                                             max_history=5)

    agent = Agent("chat_domain.yml",
                  policies=[MemoizationPolicy(max_history=5), KerasPolicy(featurizer)])

    training_data = agent.load_data(training_data_file)

    agent.train(
            training_data,
            augmentation_factor=50,
            epochs=500,
            batch_size=10,
            validation_split=0.2
    )

    agent.persist(model_path)
```

Figura 5.11: *Script* de integração *Rasa Core* e *Rasa NLU*.

Por fim, executamos o nosso *chatterbot*.

*python -m rasa_core.run -d models/dialogue/ -u models/nlu/default/chat/*

Esse comando está dizendo para rodar o *bot* usando o modelo treinado de nlu (-u models/nlu/default/chat/), juntamente com o modelo fluxo de conversa (histórias), treinado pelo Rasa Core (-d models/dialogue/).

### 5.3.7 Integrando com o *Facebook Messenger*

Como foi mencionado, o Rasa permite integração com diversas plataformas de mensagens, e uma delas é o *Facebook Messenger*. A integração é feita de forma muito simples bastando passar o conector "facebook" como parâmetro e as credenciais do *Facebook*. Essas credenciais são fornecidas pela API do *Facebook*, sendo necessário criar uma página no *Facebook* e um aplicativo para essa página.

De posse do *secret* do aplicativo e do *token* gerado na página do desenvolvedor do *Facebook* criamos um arquivo .yml que passaremos para o rasa como parâmetro. Com esse arquivo o Rasa se comunicará direto com o API do *Facebook* recebendo as mensagens postados no *Messenger* na página do *Facebook* do nosso aplicativo.

```
verify: "delivery-bot"
secret: "3e34709d01ea89032asdebfe5a74518"
page-access-token: "EAAbHPa7H9rEBAAuFk4Q3gPKbDedQnx4djJJ1JmQ7CAqO4iJKrQcNT0wtD"
```

Figura 5.12: Exemplo de arquivo yml com as credencias do *Facebook*

O *bot* se comunica com o *Facebook* fazendo o uso de *webhooks,* ou seja, ele invoca uma url passando os dados da conversa e espera uma resposta que será devolvida para o usuário. Para isso funcionar o *chatbot* precisa estar acessível pela internet, isso pode ser feito colocando o *bot* para funcionar em um servidor na internet, ou fazendo o uso de uma ferramenta de desenvolvimento chamada *ngrok*. Essa ferramenta chamada *ngrok* é capaz de pegar a porta local exposta pelo rasa e expô-la na internet fazendo um roteamento para um endereço do *ngrok*, esse endereço que o *ngrok* gera é acessível por qualquer pessoa na internet e muito utilizado para desenvolvimento de aplicações e testes.

# 6 Conclusão

Seguindo no fluxo de desenvolvimento do *chatterbot*, iniciaremos citando as dificuldades e/ou problemas encontrados durante o projeto. Na fase de definição do domínio de atuação do *bot*, deparamos com a dificuldade de modelar as intenções e ações das possíveis conversas que o interlocutor poderia ter com o *bot*. Essa dificuldade estava em encontrar uma maneira de não fugir do seu escopo de atuação. Ou seja, planejar e analisar bem todos esses atributos de *intents* e *actions* para aquilo que realmente é desejável no contexto do projeto. Simplificar ao máximo a quantidade de intenções, pois elas estão proporcionalmente relacionadas ao tamanho do domínio. Sendo que ao final, o objetivo é conseguir um *chatterbot* que irá sempre tentar manter o usuário no fluxo da conversa, sem dispersar o foco de seu assunto. E ainda, durante a definição do domínio, surgiram diversos debates, e até mesmo confusões sobre como seria o fluxo da conversa. Chegando a ser alterada diversas vezes a quantidade e o contexto das intenções, e também as ações do bot. Até então, obtermos o modelo que trabalhamos no projeto.

Depois de termos passado um bom tempo para chegar em um modelo de domínio aceitável para atuação do *bot*, partimos para a preparação dos conjuntos de dados, ou seja, os *datasets* de treinamento: NLU e *histories*. Para o primeiro, usamos o *Chatito*, que é uma ferramenta que nos ajudou a evitar um árduo trabalho para gerar um *dataset* no padrão aceitável do Rasa NLU. Porém, é uma ferramenta, que mesmo com poucas *intents* no modelo de domínio, mas se estiver contido um conjunto grande de frases, sinônimos ou entidades, vai gerar um *dataset* para o NLU muito grande. E em alguns dos itens, ao serem analisados mais de perto, podem não fazer sentido de estarem no conjunto de dados, pois, nitidamente, perdem a sua semântica através da lógica de expansão de contexto usada pelo *Chatito*. Mas de toda forma, é uma ferramenta que permitiu economizar bastante tempo, simplificando em muito o trabalho.

Ainda sobre NLU, umas das coisas que percebemos e que altera bastante a previsão de uma intenção dada uma determinada sentença, são as pontuações, principalmente a interrogação. Uma frase com o ponto de interrogação que no *Chatito* foi definida sem, gerou previsões de *intents* totalmente diferente da intenção real da sentença. Isso fez com que tivéssemos que duplicar as mesmas frases das *intents* no *Chatito*, com e sem

a interrogação. Consequentemente, houve um aumento no tamanho do *dataset* de treinamento. Mas a interferência disso é que irá demorar um pouco mais no seu treinamento. E o tempo médio de treinamento, segundo a própria documentação, é de uns 13 minutos para um computador básico com processador Intel Core I7.

Para as *histories*, que é o *dataset* que irá treinar o modelo estatístico do fluxo da conversa, como não tínhamos acesso a dados de conversas reais, podemos dizer então que esse *dataset* usado para este projeto, é um *dataset* fictício. Em outras palavras, é simplesmente um *dataset* simulado, gerado a partir do treinamento *online*. Construir um *dataset* dessa maneira, pode não refletir corretamente uma conversa real, pois, como mencionado, é apenas uma simulação de possíveis conversas entre os interlocutores e o *bot*. Mas para demonstrar como se constrói um *chatterbot*, esse *dataset* atendeu bem o seu objetivo.

A integração com o *Facebook* do *chatterbot* construído nesse trabalho foi feita com o auxílio do framework do *Rasa* que já por já criar os *endpoints* necessário para a integração só exige que o *bot* seja disponibilizado em em servidor com acesso direto a internet. Assim pudemos testar a integração com o *Messenger* e vimos que funcionava da mesma maneira que rodando localmente, ficou bem claro que a integração com qualquer tipo de serviço de mensageria seria simples pois já existem conectores para serviços muito utilizados como: *slack, telegram* e *twilio.*

Sobre a documentação do *Rasa*, buscamos sempre manter a orientação sobre esta referência. E por termos atingido um resultado satisfatório, podemos concluir que é uma documentação de boa qualidade. De fácil compreensão nos seus tutoriais, e também da explicação em si sobre as funcionalidades do *framework*. Só precisamos sair do escopo da referência oficial, quando foi mencionado ferramentas ou bibliotecas externas ao *Rasa*, com por exemplo o *spaCy*, o *Chatito* ou a integração com o *Facebook Messenger*. Agora, com relação ao *Python*, é uma documentação que parte do suposto que o usuário já tem um certo nível de domínio com a linguagem de programação. Mas isso, é se você não usar a API HTTP, que permite você usar o *Rasa* sem precisar tanto conhecer de *Python*. E dentro de uma abordagem mais pessoal, a documentação do *Rasa* ajudou indiretamente a compreender melhor sobre Processamento de Linguagem Natural, devido a um bom detalhamento das partes que compõe o desenvolvimento do *bot* através do *framework*. Suas explicações permitiram que pudéssemos facilmente referenciá-las aos conceitos de PLN e Linguística.

Como próximos passos deste trabalho, poderíamos dar continuidade adicionando mais funções ao *chatterbot*. Hoje o *bot* somente auxilia nos pedidos ao restaurante, mas seria possível que o pedido já fosse enviado diretamente ao restaurante fazendo o uso de alguma API ou serviço disponibilizado pelo restaurante. Também seria um novo desafio fazer com o que o pagamento fosse gerenciado pelo *bot* fazendo o uso de pagamento *online*. Para que isso tudo pudesse funcionar, teríamos que guardar informações de um carrinho de compras com todos os itens selecionados pelo usuário, além de pegarmos e salvarmos seu endereço, com o cuidado de confirmar o endereço antes da finalização do pedido, evitando potenciais transtornos. Poderíamos também melhor a interação com usuário fazendo com que a escolha dos itens fosse baseada em múltiplas escolhas, podendo ser utilizados, inclusive, menus simples que funcionam dentro do contexto do *Messenger* do *Facebook*.

Finalizando, o desenvolvimento deste projeto, enriqueceu-nos e nos motivou ainda mais a compreender e trabalhar com área de Inteligência Artificial. Diante de uma simples solução aos olhos externos, mas que envolvem conceitos e estudos tão ricos quanto complexos, nos faz imaginar a infinitude de aplicações para a área.

# Referências Bibliográficas


[1] *História do Delivery: uma prática antiga com novo visual*, https://blog.sistemavitto.com.br/historia-do-delivery-no-mundo/

[2] Weizenbaum, Joseph. *ELIZA−a computer program for the study of natural language communication between man and machine*. Communications of the ACM 9.1 (1966): 36-45.

[3] Xavier, Maurício. *A moda de pedir pizza pelo telefone*, 1 Julho 2015, https://vejasp.abril.com.br/blog/30-anos/a-moda-de-pedir-pizza-pelo-telefone/

[4] *Mercado de delivery de alimentos fatura mais de R$ 10 bi no Brasil*, 27 Fevereiro 2018, http://www.abrasel.com.br/component/content/article/7-noticias/5905-27022018-mercado-delivery-de-alimentos-fatura-mais-de-r-10-bi-no-brasil.html

[5] *Fast food consumption in Spain will rise by 50% over the next five years*, Lunes, 11 de Enero, 2016, https://www.eae.es/actualidad/noticias/fast-food-consumption-in-spain-will-rise-by-50-over-the-next-five-years

[6] YU, D.; WANG, S.; KARAM, Z.; DENG, L. *Language recognition using deep-structured conditional random fields*. Proc. ICASSP, 2010. p.5030-5033.

[7] A. M. TURING. *I.?COMPUTING MACHINERY AND INTELLIGENCE*, Mind, Volume LIX, Issue 236, 1 October 1950, Pages 433?460, https://doi.org/10.1093/mind/LIX.236.433

[8] GANASCIA, Jean-Gabriel. *Inteligência artificial*. São Paulo: Editora Ática, 1997.

[9] Deryugina, O.V. *Sci. Tech.Inf.* Proc. (2010) 37: 143. https://doi.org/10.3103/S0147688210020097

[10] Wallace R.S. *(2009) The Anatomy of A.L.I.C.E.*. In: Epstein R., Roberts G., Beber G. (eds) Parsing the Turing Test. Springer, Dordrecht

[11] Fryer, L. K., and Rollo Carpenter. *Bots as language learning tools.* Language Learning & Technology (2006).

[12] Mauldin, M. L. (1994). *Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition.*

[13] Neves, A. M. M. e Barros, F. A. (2005). *iaiml: Um mecanismo para tratamento de intenção em chatterbots.*

[14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. *GloVe: Global Vectors for Word Representation.*

[15] Nichol, Alan. *rasa_nlu Documentation.* Mar 23, 2018, `https://media.readthedocs.org/pdf/rasa-nlu/latest/rasa-nlu.pdf`

[16] J. Hirschberg, B. W. Ballard and D. Hindle, "Natural language processing,"in AT&T Technical Journal, vol. 67, no. 1, pp. 41-57, January-February 1988.

[17] Copestake, Ann, *8 Lectures.* 2004 https://www.cl.cam.ac.uk/teaching/2002/NatLangProc/revise

[18] Chollet, François and others *Keras*, 2015, `https://keras.io`

[19] Jurafsky, Daniel; Marin, James H. *Speech and Language Processing*, 2008.

[20] Liddy, Elizabeth D. *Natural Language Processing*, Syracuse University, 2001.

[21] Lima, Mariana, *IFOOD LANÇA CHATBOT PARA PEDIR PIZZA NO MESSENGER, MAS SERÁ SE FUNCIONA*, NoVarejo 2017. `http://www.portalnovarejo.com.br/2017/07/12/ifood-chatbot-messenger/`

[22] Navigli, Roberto. *Natural Language Understanding: Instructions for (Present and Future) Use*, 2018.

[23] Sauvé, Jacques. *Frameworks O que é um framework?.* urlhttp://www.dsc.ufcg.edu.br/j̃acques/cursos/map/html/frame/oque.htm

[24] *Debian Social Contract*, Debian Project. `https://www.debian.org/social_contract#guidelines`

[25] *History of OSI*, Open Source Initiave. `https://opensource.org/history`

[26] Pereira, Fernando; Marinho, Ivo; Oliveira, Nelson. *Open Source software development.*

[27] *Open Source Definition*, Open Source Initiave. https://opensource.org/docs/osd

[28] Romano, Fabrizio. *Learning Python*, Packt Publishing, 2015.

[29] Brennan, Bobby (2017). *Libraries vs. Frameworks.* https://medium.com/datafire-io/libraries-vs-frameworks-626cdde799a7

[30] *BIBLIOTECA (COMPUTAÇÃO).* In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2018. Disponível em: https://pt.wikipedia.org/w/index.php?title=Biblioteca_(computa%C3%A7%C3%A3o)&oldid=51936326.

[31] *spaCy 101: Everything you need to know.* In: spaCy Usage Documentation. https://spacy.io/usage/spacy-101

[32] *Processing Pipeline.* In: Rasa NLU Documentation. https://nlu.rasa.com/pipeline.html

[33] *spaCy – Industrial-Strength Natural Language Processing* https://spacy.io/

[34] *Language Understanding with Rasa NLU.* In: Rasa NLU Documentation. https://nlu.rasa.com/index.html

[35] *INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES.* In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2018. https://pt.wikipedia.org/w/index.php?title=Interface_de_programa%C3%A7%C3%A3o_de_aplica%C3%A7%C3%B5es&oldid=52559842

[36] *Connecting to messaging & voice platforms.* In Rasa Core documentation. https://core.rasa.com/connectors.html

[37] *Building a Simple Bot* In: Rasa Core Documentation. https://core.rasa.com/tutorial_basics.html

[38] *Supervised Learning Tutorial.* In: Rasa Core Documentation. https://core.rasa.com/tutorial_supervised.html

[39] *Interactive Learning.* In: Rasa Core Documentation. https://core.rasa.com/tutorial_interactive_learning.html

[40] *Rasa Core without Python*. In: Rasa Core Documentation. https://core.rasa.com/tutorial_remote.html

[41] *Anaconda. The Most Popular Python Data Science Platform*. https://www.anaconda.com/what-is-anaconda/

[42] *PyPI - The Python Package Index*. https://pypi.org/

[43] *User Guide - pip Documentation*. https://pip.pypa.io/en/stable/user_guide/