Um Chatbot para o Centro de Informática

por

Victor Fontes Seara Ferraz

Trabalho de Graduação

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CIN - CENTRO DE INFORMÁTICA

GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

www.cin.ufpe.br

RECIFE, JULHO DE 2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CIN - CENTRO DE INFORMÁTICA

GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Victor Fontes Seara Ferraz

Um Chatbot para o Centro de Informática

Projeto de Graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Vinícius Cardoso Garcia

RECIFE, JULHO DE 2017

“We no longer think of chairs as technology; we just think of them as chairs. But there was a time when we hadn’t worked out how many legs chairs should have, how tall they should be, and they would often ’crash’ when we tried to use them”.

Douglas Adams

Agradecimentos

Agradeço aos meus amigos e família por todo o incentivo e apoio durante a minha graduação.

Agradeço ao meu orientador Vinícius Garcia pela sua disponibilidade e por todo o apoio fornecido.

Agradeço a todos os professores do CIn por todo o conhecimento e dedicação fornecidos durante o curso.

Resumo

Chatbots existem a mais tempo do que aparentam, desde o surgimento de Chatbots baseados em regras e padrões, até a criação de arquiteturas mais complexas, utilizando múltiplos modelos de Inteligência Artificial. O Processamento de Linguagem Natural e o Aprendizado de Máquina são uma grande parte do desenvolvimento de Chatbots mais modernos, dado que possibilitam a criação de sistemas mais inteligentes e capazes de conversar utilizando linguagem natural. Diante do crescente interesse em aplicativos de conversação, Chatbots possuem um grande potencial de aceitação, ao entregar uma forma de interação mais humana ao usuário. Tendo como base o que foi afirmado, esta pesquisa propõe a criação de um modelo de Chatbot para o Centro de Informática que seja capaz de utilizar das técnicas de Aprendizado de Máquina e Processamento de Linguagem Natural para integrar as informações dentro do contexto do CIn e oferecer ao usuário uma interface de conversação inteligente.

Palavras-chave: *Chatbot*, *Chatterbot*, *Assistentes Virtuais Inteligentes*, *Inteligência Artificial*, *Processamento de Linguagem Natural*, *Aprendizado de Máquina*.

Abstract

Chatbots have been around for longer than they appear to be. Since its beginnings, with the emergence of rules and patterns based Chatbots, up until the creation of more complex architectures, like the ones using multiples Artificial Intelligence models. Natural Language Processing and Machine Learning are a big part of the development of more modern Chatbots, considering that they enable the creation of more intelligent systems that are capable of talking using natural language. Faced with the growing interest in conversational applications, Chatbots have a great acceptance potential, since they deliver a form of human-like interaction to the end user. Based on what has been stated, this research proposes the creation of a Chatbot model for the Centro de Informática of UFPE that's capable of using Machine Learning and Natural Language Processing techniques to integrate the information within the Centro de Informática context and offer to the end user a intelligent conversational interface.

Keywords: *Chatbot*, *Chatterbot*, *Virtual Assistants*, *Artificial Intelligence*, *Natural Language Processing*, *Machine Learning*.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Será apresentada, neste capítulo, a motivação para a realização deste trabalho, os seus objetivos e sua estrutura, a fim de facilitar o entendimento do leitor.

1.1 Motivação

A *Inteligência Artificial (IA)* é uma área que procura desenvolver tecnologias capazes de resolver problemas de modo inteligente. A sua utilização vem viabilizando a criação de diversos sistemas capazes de resolver casos complexos e que antes nunca seriam resolvidos pela computação. Com isso, vem surgindo diversas aplicações de *IA* em várias áreas, cada qual com um propósito diferente, desde técnicas de aprendizado de máquina para identificação de padrões e predições de valores até a tradução e geração de textos.

O *Processamento de Linguagem Natural (PLN)* é uma ramificação da *Inteligência Artificial* voltada para o processamento de textos a fim de facilitar a interação entre o computador e o humano, de forma natural. Podendo ou não trabalhar em conjunto com o *Aprendizado de Máquina*, o *PLN* pode ser usado como auxílio na criação de modelos conversacionais para *Chatbots*, no intuito de simular uma conversação informal entre o usuário e a máquina.

Segundo AbuShawar e Atwell [6], o primeiro protótipo de *Chatbot* foi criado em 1966, no *MIT*, por *Joseph Weizenbaum*. Nomeado de ELIZA, com a intenção de simular uma

psicoterapeuta, ele utiliza apenas de correspondências de palavras-chave para devolver respostas prontas, mas sempre incentivando a conversa, respondendo com perguntas reflexivas e que redirecionam a atenção para os problemas do usuário. Mas ao longo do avanço da interface gráfica na computação e com o surgimento da Inteligência Artificial, diversas arquiteturas para *Chatbots* foram desenvolvidas, dentre elas, destacam-se:

- A *AIML (Artificial Intelligence Markup Language)*, uma linguagem de marcação desenvolvida para a criação de *ALICE (Artificial Linguistic Internet Computer Entity)*, o qual utiliza do reconhecimento de padrões conversacionais e se inspira no formato *XML (Extensible Markup Language)* para a estruturação deles [6];
- A combinação de *NER (Named Entity Recognition)* com algoritmos de *Aprendizado de Máquina*, o qual é o foco deste trabalho, utiliza do reconhecimento de entidades em partes do texto, em conjunto com a classificação da intenções para processar e produzir uma resposta ao usuário;
- A utilização de *Redes Neurais Recorrentes*, como a *Sequence to Sequence*, utilizada para a geração de respostas baseando-se em sequências anteriores.

Diante do crescente interesse em aplicativos de conversação, a ponto de superarem as redes sociais quanto a quantidade de usuários ativos por mês em 2015 [25], *Chatbots* possuem um grande potencial de aceitação, ao entregar uma forma de interação mais humana ao usuário, como é exemplificado em [27]:

> ”There’s one difference between a real person and an AI, though. A chatbot is selfless and dedicated to you, it is always there for you and it always has time for you”.

Para o desenvolvimento deste trabalho, foi necessária a utilização da biblioteca *NLTK (Natural Language Toolkit)* para realizar as tarefas relacionadas ao *Processamento de Linguagem Natural* com auxílio de Aprendizagem de Máquina, desenvolvida para ser utilizada com a linguagem de programação Python; visto que Python é uma das melhores linguagens para desenvolver *Inteligências Artificiais* [1] e a 4ª linguagem de programação mais popular do mundo, segundo a TIOBE em junho de 2017 [48]. Como exemplo, ela é utilizada pela Google em seu framework próprio para *Aprendizado de Máquina*, o TensorFlow [4].

1.2 Objetivos

1.2.1 Objetivos Gerais

O objetivo desta pesquisa é criar um *Chatbot* para o *Centro de Informática* que seja capaz de utilizar das técnicas de *Inteligência Artificial* (mais precisamente, *Aprendizado de Máquina* e *Processamento de Linguagem Natural*) para integrar as informações dentro do contexto do CIn e oferecer ao usuário uma interface de conversação inteligente, auxiliando em questões gerais e administrativas, como localização/horário de salas de aula e professores, notícias recentes, anúncios, eventos e processos importantes, focando em dúvidas que os estudantes mais novos (mas também sendo útil para os que estão a mais tempo no centro) podem chegar a ter.

1.2.2 Objetivos Específicos

- Fazer um estudo acerca do uso de *Inteligência Artificial (IA)* no desenvolvimento de Chatbots;
- Analisar as técnicas de desenvolvimento de *Chatbots* que envolvem *IA* e suas restrições;
- Demonstrar o funcionamento de uma arquitetura de Chatbot que utilize *Inteligência Artificial*;
- Investigar e propor uma arquitetura de *Chatbot* capaz de entender e processar informações dentro do contexto do *Centro de Informática da Universidade Federal de Pernambuco*.

1.3 Organização do Trabalho

O presente trabalho está organizado em sete capítulos, dos quais o primeiro é a introdução e os próximos seis capítulos estão descritos abaixo:

- No Capítulo 2 é apresentado um conjunto de definições relevantes para o entendimento do trabalho;
- No Capítulo 3 é apresentado o conceito de *Chatbots*, em conjunto das suas técnicas de implementação e as aplicações da *Inteligência Artificial* no seu desenvolvimento;
- No Capítulo 4 são apresentados trabalhos relacionados ao tema desta pesquisa;
- No Capítulo 5 é apresentada toda a implementação deste trabalho, com exemplos do seu funcionamento;
- No Capítulo 6 são apresentadas as considerações finais do trabalho assim como as propostas para trabalhos futuros;

Capítulo 2

Referencial Teórico

Este capítulo apresenta os conceitos necessários para o melhor entendimento deste trabalho, trazendo as suas definições e uma visão geral sobre os seguintes temas: Inteligência Artificial, Processamento de Linguagem Natural, Aprendizado de Máquina e NLTK, a biblioteca utilizada para aplicar esses conceitos. O Processamento de Linguagem Natural é uma ramificação da Inteligência Artificial que foca no processamento e entendimento de textos e que pode contar com o auxílio de algoritmos de Aprendizado de Máquina. Utilizando técnicas de Processamento de Linguagem Natural, este trabalho propõe um modelo de reconhecimento de entidades e classificação de intenções que serão utilizados para o entendimento da entrada do usuário pelo Chatbot. Ao fim deste capítulo, o leitor será capaz de entender melhor o conteúdo do trabalho e a linha de raciocínio adotada no mesmo.

2.1 Inteligência Artificial

O termo *Inteligência Artificial (IA)* foi usado pela primeira vez em 1956 por John McCarthy, dando início à área de mesmo nome, na conferência de Inteligência Artificial de Dartmouth [44]. McCarthy define Inteligência Artificial como [5]:

> ”The science and engineering of making intelligent machines, especially intelligent computer programs”.

A inteligência, como palavra, pode ser definida como sendo a *"faculdade de aprender, compreender e adaptar-se"* [30]. Porém, a mesma palavra pode ter significados distintos em diferentes acepções, segundo as definições citadas por Pereira [34]:

- Na filosofia, a inteligência é o "princípio abstrato que é a fonte de toda a intelectualidade";
- Na teologia, é o "dom divino que nos torna semelhantes ao criador";
- E na psicologia, é a "capacidade de resolver problemas novos com rapidez e êxito".

Como não há um consenso sobre o que é a *inteligência* em si, não se pode facilmente definir o real significado de *Inteligência Artificial*. Mas, para Russel e Norvig [39], existem oito importantes definições que surgiram durante toda a história de IA, as quais seguem na Tabela 2.1.

Autor	**Ano**	**Definição**
Haugeland	1985	The exciting new effort to make computers think... machines with minds, in the full and literal sense
Bellman	1978	[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning...
Charniak e McDermott	1985	The study of mental faculties through the use of computational models.
Wiston	1992	The study of the computations that make it possible to perceive, reason, and act.
Kurzweil	1990	The art of creating machines that perform functions that require intelligence when performed by people.
Rich e Knight	1991	The study of how to make computers do things at which, at the moment, people are better.
Schalkoff	1990	A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes.
Luger e Stubblefield	1993	The branch of computer science that is concerned with the automation of intelligent behavior.

Tabela 2.1: Definições de *Inteligência Artificial*. Adaptada de [39]

Com o que foi descrito na Tabela 2.1, a definição de Inteligência Artificial pode ser dividida em quatro categorias [39]:

- Para *Haugeland* e *Bellman*, a IA é definida como *sistemas que pensam como humanos*;
- Para *Charniak, McDermott* e *Wiston*, são *sistemas que pensam de modo racional*;
- Para *Kurzweil*, *Rich e Knight*, são *sistemas que agem como humanos*;
- E para *Schalkof*, *Luger e Stubblefield*, são *sistemas que agem racionalmente*.

O Chatbot apresentado neste trabalho melhor se encaixa na definição de Kurzweil, Rich e Knight, como um *sistema que age como humano*.

2.1.1 O Teste de Turing

O *Teste de Turing* é uma meta de longo prazo da Inteligência Artificial. Definida por *Allan Turing*, ele procura examinar a capacidade de uma máquina de demonstrar um comportamento inteligente e que seja equivalente ao ser humano, a fim de responder a pergunta *"If a computer could think, how could we tell?"*[29]. A máquina passará no teste caso o seu comportamento seja indistinguível do de uma pessoa. O teste pode ser retratado de diversas formas, mas todos são voltados para a seguinte ideia: diante de um juiz, é realizada uma conversa entre ele, uma pessoa e uma máquina (sem que se possa vê-los); o juiz deve ser capaz de identificar, sem dúvidas, quem dos dois é uma máquina. A máquina passa no teste caso o juiz falhe em descobri-la, e passa a ser considerada uma verdadeira Inteligência Artificial [44]. Veja o exemplo na Imagem 2.1.

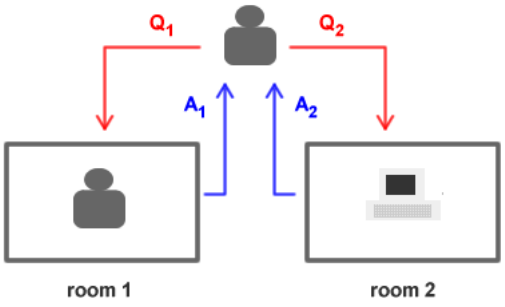

Figura 2.1: Teste de Turing

Em 1990, *Hugh Loebner*, em conjunto com o *Cambridge Center for Behavioral Studies*, criou a *competição anual de Inteligência Artificial*, desenvolvida especificamente para a execução do *Teste de Turing* em todas as Inteligências Artificiais inscritas [29].

2.1.2 Aplicações da Inteligência Artificial

Em geral, a Inteligência Artificial possui um grande número de possíveis aplicações, cada qual com seus respectivos problemas a serem solucionados, como o desenvolvimento de simples sistemas capazes de provarem teoremas matemáticos, jogarem partidas de xadrez ou até realizarem a difícil tarefa de reconhecimento de entidades em imagens. Basicamente, foca-se em um objetivo central de criar sistemas que sejam vistos como inteligentes [5]. Ela está presente no dia a dia de várias pessoas há décadas, mas sua presença acaba passando por despercebida, pois a visão que se tem de Inteligência Artificial, na maioria das vezes, é voltada para a ideia de robôs e máquinas complexas com um grau elevado de inteligência. Porém, ao longo dos anos, a IA tem crescido em áreas como a Análise Estatística e Marketing [44] (incluindo as recomendações e anúncios do Facebook e Google), dando suportes às decisões de diversos negócios.

Há algumas áreas específicas que são de extrema importância para o entendimento da IA, como é exemplificado em [18]. São elas: *Sistemas Especialistas*, *Robótica*, *Sistemas Visuais*, *Processamento de Linguagem Natural* e *Aprendizado de Máquina*. As duas últimas são o foco deste trabalho e serão apresentadas nas seções a seguir.

2.2 Processamento de Linguagem Natural

O *Processamento de Linguagem Natural (PLN)* é uma ramificação da Inteligência Artificial que foca no processamento e entendimento de textos. Suas primeiras aplicações surgiram com o que era chamado de *Máquinas de Tradução* (do inglês, *Machine Translation*), e trabalhavam, como o nome sugere, com a tradução automática de textos entre linguagens diferentes. As Máquinas de Tradução surgiram na década de 1940, na tentativa de traduzir textos do russo para o inglês. Com o avanço da tecnologia, novas aplicações foram dadas ao PLN. Na década de 1950, a capacidade de processar, en-

tender e gerar textos foi considerada como uma das qualidades necessárias de uma Inteligência Artificial, as quais foram citadas por *Allan Turing* em seu teste, o *Teste de Turing*.

Segundo Müller [31], o Processamento de Linguagem Natural pode ser dividido em quatro componentes, como visto na Tabela 2.2.

Componente	**Descrição**
Análise Morfológica	Diz respeito à construção das palavras, seus radicais e afixos, focando na sua parte estática.
Análise Sintática	Trabalha com as relações formais entre palavras e é uma das partes mais importantes das análises.
Análise Semântica	Visa representar o significado de cada palavra, ou o significado de um conjunto de palavras (sentenças), baseando-se nas suas construções sintáticas.
Análise Pragmática	Diz respeito ao contexto em que as sentenças se aplicam, baseando-se nos sentidos extraídos anteriormente na análise semântica.

Tabela 2.2: Componentes do *Processamento de Linguagem Natural*. Textos adaptados de [31].

2.2.1 Tokenização

Tokenização é o processo de dividir um texto em vários *tokens*, seja a tokenização de sentenças, dividindo o texto em várias sentenças diferentes, ou de palavras, dividindo cada sentença em várias palavras. Tal técnica é utilizada como uma etapa de pré-processamento do texto, antes da execução de outros procedimentos como a *Etiquetagem*. Na *Análise Morfológica*, a tokenização de sentenças pode ser utilizada antes de executar os procedimentos de *lematização* e *stemização* [36].

2.2.2 Lematização e Stemização

A *lematização* executa uma análise morfológica de cada palavra em uma sentença, substituindo-as pelas suas formas presentes no vocabulário, com o objetivo de simplificar as variáveis presentes em uma sentença. Como exemplo, ”projetado”é substituído

por ”projetar”. Já a *stemização* é menos precisa e mais simples, pois ela identifica os radicais e afixos da palavra e os remove, ou seja, é considerada a parte da palavra cujas letras não variam [36]. Exemplo: ”projetado”, no caso da stemização, é substituído por ”projet”.

2.2.3 Etiquetagem

A Etiquetagem é uma técnica utilizada na *Análise Sintática*. Ela tem como objetivo etiquetar cada palavra de uma sentença após passar por um tokenizador de palavras, com a sua classe gramatical, baseando-se na sua formação sintática. Como exemplo, a frase *”O rato roeu a roupa do rei de Roma.”* pode ser etiquetada como (ver a Tabela 2.3):

> *”O/**ART** rato/**N** roeu/**V** a/**ART** roupa/**N** do/**PREP+ART** rei/**N** de/**PREP** Roma/**NPROP** ./**PU**”*.

Etiqueta	**Classe Gramatical**	**Palavras Relacionadas**
ART	Artigo	O, a
N	Substantivo	rato, roupa, rei
NPROP	Substantivo Próprio	Roma
PREP	Preposição	de
PREP+ART	Preposição + Artigo	do
V	Verbo	roeu
PU	Pontuação	.

Tabela 2.3: Tipos de Etiquetas, suas classes gramaticais e palavras relacionadas.

A escolha de etiquetas e as regras as quais se aplicam a elas dependem muito do contexto. Neste caso, é importante entender cada palavra com a sua verdadeira classe gramatical, da forma mais simplificada possível. Ao etiquetar uma sentença, podem ser usadas duas abordagens: a *lexicalista*, a *probabilística* ou uma forma híbrida, contendo as duas [31]; para aplicações que envolvem *Aprendizado de Máquina*, a abordagem probabilística é a melhor escolha. A primeira regra, a *lexicalista*, procura verificar a consistência da linguagem, seguindo regras gramaticais e do dicionário; já a segunda, a *probabilística*, utiliza de *n-gramas* para decidir a etiqueta de cada palavra.

N-grama é um procedimento que se baseia no conjunto de N palavras em uma

sequência, a fim de dividi-la em vários pedaços sequenciais. Por exemplo, a sentença *"O rato roeu a roupa do rei de Roma"*, quando **N = 3**, pode ser dividida em: *"START O rato"*, *"O rato roeu"*, *"rato roeu a"*, *"roeu a roupa"*, *"a roupa do"*, *"roupa do rei"*, *"do rei de"*, *"rei de Roma"*, *"de Roma END"*. Nota-se que há nove conjuntos com três palavras cadas (incluindo os tokens *START* e *END*, indicando o início e fim da sentença). Para cada conjunto é classificada a palavra central, baseando-se nas palavras vistas ao seu redor; ou seja, a palavra *"a"*, no caso *"roeu a roupa"* é classificada como um *artigo*, pois antes dela há um *verbo (roeu)* e depois, um *substantivo (roupa)*. Os valores de N podem variar entre 1 e 3 (chamados de *Unigram, Bigram e Trigram*), dado que utilizar $\mathbf{N} > \mathbf{3}$ não acarreta em um acréscimo significativo do poder de classificação da técnica [2].

2.2.4 Extração da Informação

Extração da Informação (EI) é uma área do *Processamento de Linguagem Natural* voltada para o reconhecimento de *fatos* (nomes, números significantes ou qualquer outro texto capaz de entregar um mínimo de informação relevante) em textos em linguagem natural [35]. O processo de extração de informação é responsável por transformar textos desestruturados em objetos estruturados, que possam ser representados em notações como *JSON* ou *XML*. O *Reconhecimento de Entidades Nomeadas* é um exemplo de execução dos procedimentos da *Extração da Informação* e é utilizado neste trabalho.

A função do *Reconhecimento de Entidades Nomeadas*, do inglês, *Named Entity Recognition (NER)*, é a identificação de entidades em várias partes do texto, ignorando informações irrelevantes e considerando somente aquilo o qual foi treinado para entender [35]. Como exemplo, considerando a sentença fictícia a seguir como entrada de um sistema de NER, que foi anteriormente treinado no contexto do Centro de Informática da UFPE: *"A sala E122 estará ocupada hoje, às 15h, para a aula de reposição de Engenharia de Software do curso de Sistemas de Informação"*. Após executar o reconhecimento de entidades, o sistema retornará, de forma estruturada, as entidades presentes no texto (Tabela 2.4).

Entidade	**Texto**
LOCAL	E122
DATA	hoje
HORARIO	15h

DISCIPLINA	Engenharia de Software
CURSO	Sistemas de Informação

Tabela 2.4: Exemplo do resultado de um sistema de reconhecimento de entidades.

A abordagem utilizada no Processamento de Linguagem Natural para executar o NER é conhecida como *IOB Tagger (Inside, Outside, Begin)*. Com o IOB, as entidades são reconhecidas através da etiquetagem inicial (Begin), seguida das demais palavras que pertençam a entidade (Inside) e termina quando é encontrada uma etiqueta *O* (Outside). Como exemplo, a frase utilizada anteriormente pode ser representada como:

> *"A/**O** sala/**O** E122/**B-LOCAL** estará/**O** ocupada/**O** hoje/**B-DATA**, às/**O** 15h/**B-HORARIO**, para/**O** a/**O** aula/**O** de/**O** reposição/**O** de/**O** Engenharia/**B-DISCIPLINA** de/**I-DISCIPLINA** Software/**I-DISCIPLINA** do/**O** curso/**O** de/**O** Sistemas/**B-CURSO** de/**I-CURSO** Informação/**I-CURSO**"*.

2.3 Aprendizado de Máquina

O *Aprendizado de Máquina*, do inglês, *Machine Learning (ML)*, é uma ramificação da Inteligência Artificial que utiliza de algoritmos fortemente baseados em fórmulas estatísticas, os quais trabalham com uma grande quantidade de dados para construir sistemas capazes de *aprender* e *otimizar* a si mesmos. A ML procura entender, na prática, como seres humanos e demais animais são capazes de adquirir experiência e conhecimento [32].

A palavra *"aprendizado"*, assim como inteligência, possui uma definição abrangente [32]; o Dicionário da Língua Portuguesa define *aprendizado* como o *"ato ou efeito de aprender um ofício, uma arte ou ciência"* ou, como é definido pela Psicologia, *"Denominação comum a mudanças permanentes de comportamento em decorrência de treino ou experiência anterior"* [30]. Nessa subseção serão tratados os seguintes algoritmos de Aprendizado de Máquina, utilizados no desenvolver deste trabalho: *Support Vector Machines, Conditional Random Fields e Maximum Entropy*.

2.3.1 Aprendizado Supervisionado

Aprendizado Supervisionado é uma parte do Aprendizado de Máquina que executa o treinamento de dados conhecidos e previamente categorizados com o objetivo de categorizar dados desconhecidos. Ou seja, o algoritmo deve ser capaz de aprender com exemplos [21]. Para que isso ocorra, ele é alimentado com dois conjuntos de dados: *as características (features, denominadas por X) e as categorias (labels, denominadas por Y)* em que, para cada feature X_n, deve haver uma label Y_n correspondente. Desses dados, é separado um conjunto para executar o treinamento do algoritmo e outro para testar sua precisão, ambos contendo suas features e labels. O conjunto de treinamento corresponde com a maior parte dos dados (geralmente entre 60% e 80% do total), enquanto o de testes utiliza uma pequena quantidade (entre 40% e 20%) para que se possa calcular a precisão do algoritmo ao prever a label de uma feature desconhecida. Apesar de não ser obrigatório, é recomendado que o tamanho do conjunto de treinamento seja maior do que a de testes, visto que o algoritmo poderá cobrir e aprender mais casos desta maneira.

2.3.2 Term Frequency - Inverse Document Frequency

Para que dados textuais possam ser interpretados por qualquer algoritmo de Aprendizado de Máquina, eles devem passar por um pré-processamento. O *TF-IDF*, do inglês, *Term Frequency - Inverse Document Frequency*, é um algoritmo que determina a relevância de cada palavra, utilizando a frequência relativa da palavra em um documento, considerando a *frequência inversa* delas em relação com todos os documentos utilizados no treinamento [37]. Matematicamente, dado um conjunto de documentos D, uma palavra p e um documento $d \in D$, o valor do peso $tfidf_{p,d}$ pode ser representado como [37]:

$$tfidf_{p,d} = f_{p,d} * log(\frac{|D|}{f_{p,D}})$$

Em que $f_{p,d}$ é a frequência da palavra p no documento d, $|D|$ é a quantidade de documentos existentes e $f_{p,D}$ é a frequência da palavra p em relação a todos os documentos

em D. Após a execução do TF-IDF, os dados estão prontos para serem inseridos em um algoritmo de Aprendizado de Máquina.

2.3.3 Conditional Random Fields

Conditional Random Fields (CRF) é um modelo de Aprendizado de Máquina categorizado como um modelo discriminativo (ou condicional), treinado para *maximizar uma probabilidade condicional* [42], ou seja, construir e identificar relacionamentos entre diversas observações e utilizá-las para prever valores consistentes. Modelos discriminativos são aqueles cujo objetivo é calcular *probabilidades condicionais*, ou seja, $p(X|Y)$, em que procura-se encontrar a dependência de uma variável Y em relação a uma variável X. Também pertencem a este grupo os modelos *Support Vector Machine* e *Regressão Linear*. CRFs possuem excelente performance em realizar tarefas como o *Part-Of-Speech Tagging* [47], um procedimento do *Processamento de Linguagem Natural* responsável por etiquetar cada palavra em uma sentença com a sua classe gramatical, e executar técnicas como a *Extração de Informação* [33].

A utilização de CRF para tarefas voltadas ao Processamento de Linguagem Natural exerce uma vantagem em relação a *modelos generativos* como o *Hidden Markov Models (HMM)* e *Naive Bayes* e seus variantes [47, 42]. Modelos generativos são aqueles cujo objetivo é prever valores quando a informação que se tem não está completa ou possui parâmetros ocultos. A Figura 2.2 ilustra o relacionamento entre diversos modelos, representados por grafos, até chegar ao *Conditional Random Fields* em questão. Os modelos apresentados por cima são modelos generativos, enquanto os localizados embaixo são discriminativos.

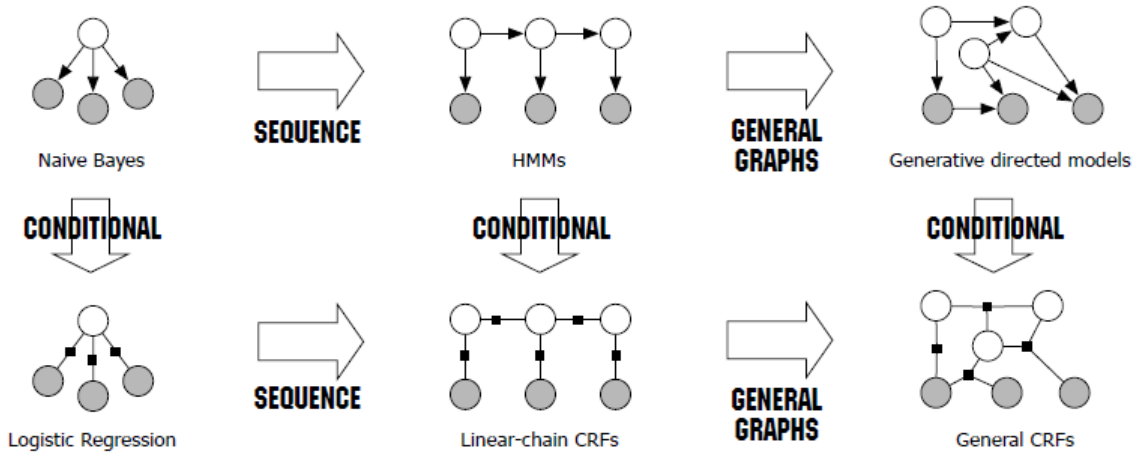

Figura 2.2: Relacionamento entre Naive Bayes, Regressão Logística, HMM, GDM e CRFs [47].

Seguindo o exemplo dado anteriormente, *"O rato roeu a roupa do rei de Roma"* e suas respectivas etiquetas, *"ART, N, V, ART, N, PREP+ART, N, PREP e NPROP"*, considerando a letra **a** em *"roeu **a** roupa"*, um modelo CRF treinado para etiquetar sentenças na língua portuguesa deve ser capaz de utilizar o relacionamento existente entre as palavras ao redor de **a** para definir a sua classe gramatical correta. Segundo Lafferty, McCallum e Pereira, considerando as variáveis aleatórias X sobre as sentenças a serem categorizadas, as variáveis aleatórias Y sobre as possíveis categorias existentes (no caso, as possíveis classes gramaticais) e o grafo $G = (V, E)$ tal que $Y = (Y_v)_v \in V$ e que Y é indexado pelas vértices de G. Logo, (X, Y) é um CRF caso as variáveis aleatórias Y_v obedeçam à *propriedade de Markov* com relação ao grafo $p(Y_v|X, Y, w \neq v) = p(Y_v|X, Y_w, w \sim v)$ quando condicionadas a X, em que $w \sim v$ indica que w e v são vizinhos no grafo G [26]. A *propriedade de Markov* se refere à perda de memória em um *processo estocástico*, ou seja, a perda de memória na evolução de um sistema de valores de acordo com o tempo.

2.3.4 Support Vector Machines

Considerado um dos melhores classificadores de Aprendizado de Máquina, *Support Vector Machine (SVM)* é um modelo desenvolvido para reconhecimento de padrões que trabalha de modo *discriminativo*, possuindo uma ótima precisão na regressão e

classificação linear (conhecida como *Linear SVM*), e na não-linear (conhecida como *Kernel SVM*) [50]. Em geral, o trabalho da SVM é maximizar a distância mínima do hiperplano em relação ao exemplo mais próximo, e assim, definir a área pertencente a cada label. Um hiperplano é um espaço vetorial, o qual em um modelo tridimensional, é considerado um plano, em um bidimensional, uma reta e em um unidimensional, um ponto. As imagens 2.3 e 2.4 são exemplos de como funciona, graficamente, um modelo treinado com *Linear SVM* e um treinado com *Kernel SVM*, respectivamente.

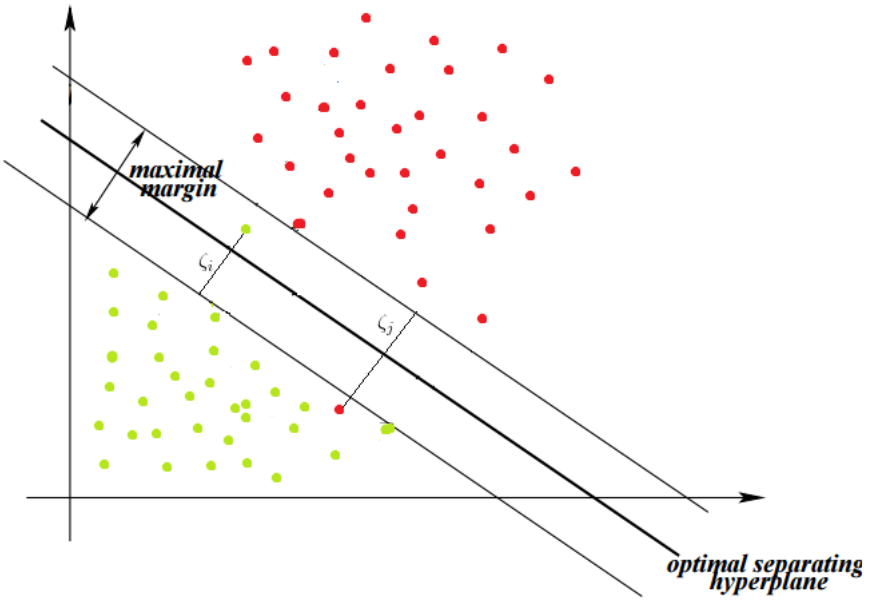

Figura 2.3: Linear SVM [3]

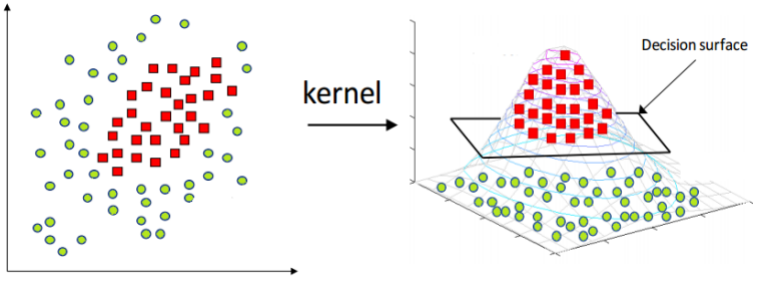

Figura 2.4: Kernel SVM [3]

Como pode ser visto na Figura 2.3, o Linear SVM divide as labels utilizando três retas, as quais foram otimizadas para se posicionarem baseando-se nos dados de treinamento oferecidos. Essas linhas são chamadas de *vetores de suporte (Support Vectors)*, em que a central é o *hiperplano*, e os suportes são as *margens*, que definem as distâncias

mínimas entre o exemplo mais próximo e o hiperplano. No caso da Figura 2.4, é considerado um espaço tridimensional, dado que a distribuição dos dados não é linear. O número de hiperplanos, em ambos os casos, varia dependendo da quantidade de labels utilizadas. Nos exemplos foram considerados problemas binários, ou seja, que possuem somente duas labels cada.

Geralmente, Linear SVM possui uma performance inferior ao do Kernel SVM, porém o último necessita de mais processamento e mais tempo para alcançar um nível de precisão satisfatória [22]. Em alguns casos, o Linear SVM pode ser considerado uma melhor opção quando o conjunto de dados utilizado se encaixa melhor nele [22]. Neste trabalho, foi utilizado o *Linear SVM*, dado que a precisão deste sobre os dados apresentados foi mais favorável quanto ao *Kernel SVM*.

Por definição, dada a sequência de dados de treinamento $(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), ..., (\overrightarrow{x_n}, y_n)$, em que $\overrightarrow{x_n}$ representa o vetor *p-dimensional* de números reais $\overrightarrow{x}$ na amostra n e y_n corresponde à label do ponto $\overrightarrow{x_n}$, procura-se encontrar o *hiperplano de margem máxima* que divida os pontos de cada vetor $\overrightarrow{x_n}$ de acordo com a sua label y_n, seguindo a fórmula $\overrightarrow{w}.\overrightarrow{x} - b = 0$, onde $\overrightarrow{w}$ é o vetor normal do hiperplano. Tal representação pode ser visualizada na Figura 2.3.

2.4 Natural Language Toolkit (NLTK)

NLTK é uma plataforma de código aberto (sob a licença *Apache 2.0*), feita para a criação de programas voltados ao *Processamento de Linguagem Natural* em *Python* e foi originalmente desenvolvida por *Steven Bird, Edward Loper e Ewan Klein*, com a sua primeira versão lançada em 2001. Seu principal objetivo é abstrair o fluxo de desenvolvimento de ferramentas de linguagem natural, oferecendo métodos que facilitem o trabalho com dados textuais, como as analises gramaticais, e auxiliam no processo de *tokenização, etiquetagem e na extração de informação* . A biblioteca oferece uma série de corpus em várias linguagens e dentro de diversos contextos. Para a língua portuguesa, estão presentes três corpus: *Machado*, contendo várias sentenças de Machado de Assis; *Floresta*, com o conteúdo de frases analisadas morfologicamente pela *Linguateca* [28]; e *Mac-Morpho*, contendo uma série de sentenças já etiquetadas [19].

A plataforma dispõe de um livro como tutorial, chamado *NLTK Book* [2], o qual explica, passo a passo, como realizar cada tipo de tarefa do Processamento de Linguagem Natural utilizando as ferramentas oferecidas. O NLTK foi escolhido para ser utilizado neste trabalho por possuir suporte a língua portuguesa, ferramentas que facilitam o desenvolvimento de modelos de *etiquetagem (POS Tagging)*, de *Reconhecimento de Entidades Nomeadas* e suporte ao treinamento de features textuais com *Conditional Random Fields* (com a integração do *python-crfsuite*), além da fácil integração com a biblioteca *scikit-learn*, que possui a implementação de modelos de *Support Vector Machines*, utilizados para realizar a classificação de intenções.

2.5 Considerações Finais

Este capítulo apresentou os conceitos básicos que serão utilizados no trabalho. Inicialmente, foram apresentadas noções de *Inteligência Artificial, Processamento de Linguagem Natural e Aprendizado de Máquina*. Seguindo a linha de reconhecimento de entidades e classificação de intenções em textos, é proposto o desenvolvimento de um Chatbot utilizando a *biblioteca NLTK* como ferramenta para executar, na prática, os conceitos de Processamento de Linguagem Natural e Aprendizado de Máquina.

O próximo capítulo conta com uma definição de Chatbots, apresenta os modelos e técnicas de implementação e as aplicações da *Inteligência Artificial* no seu desenvolvimento, a fim de sintetizar os conhecimentos existentes na área.

Capítulo 3

Chatbots

Uma aplicação pode ser considerada como um Chatbot (também conhecido como Chatterbot, Artificial Conversational Entity ou Assistente Virtual) caso ela funcione por meio de interações através de uma interface de conversação. Com o avanço dos smartphones e das mensagens de texto como um meio de comunicação, grandes e pequenos negócios vem procurando desenvolver chatbots para facilitar a interação dos usuários com seus produtos [8]. A princípio, chatbots podem ser aplicados em qualquer área em que haja a possibilidade de introduzir, de alguma forma, um diálogo para a resolução dos problemas os quais procura-se explorar, como em áreas voltadas para a educação, exemplificado em [51], *"We can imagine Chatterbots acting as talking books for children, Chatterbots for foreign language instruction, and teaching Chatterbots in general"*. A interface de conversação colabora para a sua aceitação, visto que, assim como pessoas normalmente se comunicam utilizando linguagem natural, elas procuram poder fazer o mesmo com os computadores. Utilizar Chatbots é a melhor maneira para facilitar a *interação humano-máquina*, permitindo que os usuários se expressem naturalmente, como se estivessem conversando com um indivíduo qualquer [43].

3.1 Histórico

Chatbots existem há mais tempo do que aparentam. *ELIZA*, o primeiro Chatbot a surgir, criado por *Joseph Weizenbaum* em 1966, assumia a personalidade de um psicoterapeuta

que, apesar de sua simplicidade, era capaz de entregar uma experiência interessante de conversação para a época [6]. Em 1972, o psiquiatra *Kenneth Colby* desenvolveu o *PARRY*, o qual possuía um comportamento mais avançado que seu antecessor, o que o fez ser conhecido como ”ELIZA com atitude”. PARRY foi criado para simular a personalidade de um indivíduo com esquizofrenia paranoide e com um comportamento rude, o qual chegou a ter um resultado interessante no *Teste de Turing*, em que foi identificado como uma máquina somente por 48% dos psiquiatras selecionados como juízes [40] e foi o primeiro Chatbot a passar no teste.

Já o *Jabberwacky*, criado em 1988 por *Rollo Carpenter*, foi projetado para conversar naturalmente com seus usuários, simulando interações humanas de forma bem humorada [10]. Ele chegou a ocupar o terceiro (2003), segundo (2004) e primeiro lugar (2005, passando-se por uma pessoa chamada George e 2006, passando-se por Joan) do *Prêmio de Loebner* [11, 12, 13, 38]. Em 2008, uma variação do Jabberwacky, o *Cleverbot*, foi criada pelo mesmo autor. A grande diferença entre ele e o ELIZA, PARRY e até o *A.L.I.C.E.*, é que o Jabberwacky é capaz de aprender com suas conversas, ou seja, tudo que é dito a ele é armazenado, o qual é utilizado posteriormente a fim de encontrar a resposta mais apropriada para o usuário, utilizando técnicas que dependem de padrões existentes no contexto da conversa [14].

A.L.I.C.E., do inglês, *Artificial Linguistic Internet Computer Entity*, é um Chatbot criado em 1995 por *Richard Wallace* e foi o primeiro a ser desenvolvido com *AIML (Artificial Intelligence Markup Language)*, linguagem de marcação criada por Wallace especificamente para o Chatbot [9]. Também conhecido como *Alicebot*, ou *Alice*, esse Chatbot foi a inspiração da criação de diversos outros, os quais estão presentes na *A.L.I.C.E Artificial Intelligence Foundation*, uma fundação criada para promover a adoção do Alicebot e da AIML, e inspirou a criação de um Chatbot voltado para ajudar novos estudantes universitários [9]. Alice ganhou o prêmio de Loebner em 2000, 2001 e 2004 [15, 20].

Em 2011, a *IBM Research* divulgou sua nova Inteligência Artificial capaz de processar uma grande quantidade de dados desestruturados e responder perguntas utilizando linguagem natural, chamado Watson, em um programa de TV de perguntas e respostas, *”Jeopardy!”*, onde competiu contra os seus dois melhores campeões. Segundo a *IBM*, a meta de longo prazo do Watson é [24]:

> ”[...] to create a new generation of technology that can find answers in unstructured data more effectively than standard search technology”.

Desde sua divulgação, o *IBM Watson* é considerado como um dos melhores, senão o melhor, sistema de processamento de perguntas e respostas existentes. Sua arquitetura DeepQA, criada especificamente para ele, é capaz de assimilar diversas interpretações para uma questão e escolher qual delas é a melhor de ser utilizada [24]. Embora não sendo necessariamente um Chatbot, o Watson segue como uma ótima opção para abstrair o trabalho envolvendo a criação de uma Inteligência Artificial, em que a própria IBM disponibiliza um passo a passo para a criação de um Chatbot através da API do Watson [23].

Em 2016, com as integrações para Chatbots lançadas por aplicativos de conversação como o *Telegram*, *Slack* e o *Facebook Messenger*, ambos utilizados por um crescente número de usuários, surgiu uma grande quantidade de novos bots, em que o número de Chatbots desenvolvidos pela plataforma do *Facebook* passou da faixa dos 10.000 [16].

3.2 Arquiteturas de Chatbots

Não há uma arquitetura definida para a criação de um Chatbot, e nem uma melhor escolha. Cabe ao desenvolvedor escolher o modelo que o melhor agrada ou lhe pareça mais coerente quanto a sua situação.

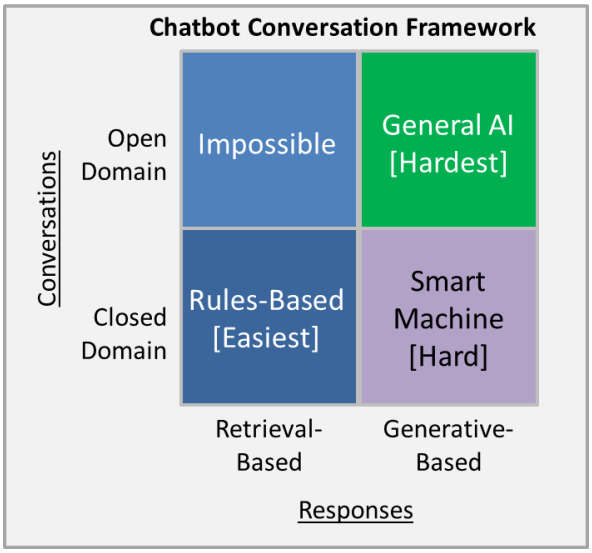

Figura 3.1: Modelos de Chatbots [17]

No geral, existem dois principais modelos de Chatbots, o modelo *generativo (Generative Model)* e o *baseado em regras (Retrieval-Based Model)*, em que cada um pode ser de domínio aberto ou fechado, como é demonstrado na Figura 3.1. Este trabalho foi desenvolvido a fim de aderir ao modelo baseado em regras com o domínio fechado.

3.2.1 Modelo Generativo

O *Modelo Generativo* é uma arquitetura de Chatbot cujo objetivo é gerar respostas baseando-se na última mensagem recebida e no histórico da conversa com o usuário, utilizando algoritmos de *Redes Neurais*, mais precisamente, uma Rede Neural específica conhecida como *Rede Neural Recorrente*. Este modelo necessita de uma grande quantidade de dados para funcionar como esperado (cerca de milhões de dados de treinamento) e quanto menor a sua base de treinamento, maiores são as chances de erros gramaticais em suas respostas, visto que todo o trabalho de geração de resposta é feita somente pelo o algoritmo. Por conta da sua presente complexidade, costumam ser utilizados mais a fins de pesquisas e testes do que em implementações sérias [45].

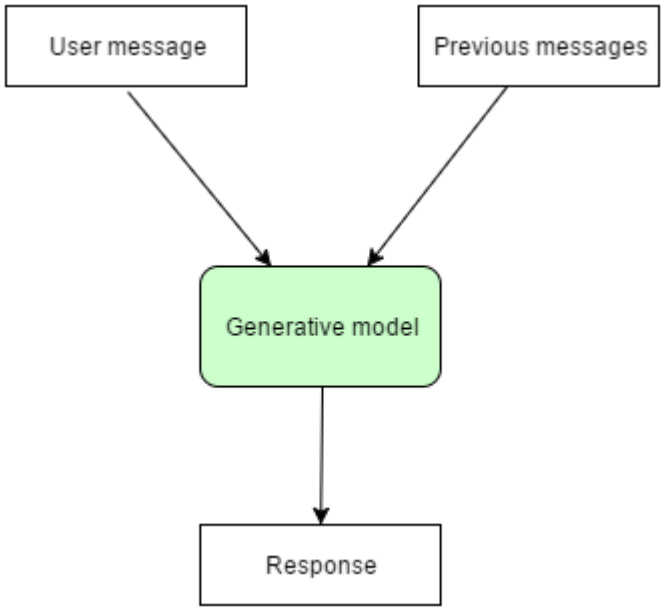

Figura 3.2: Modelo Generativo [45]

O atual Modelo Generativo no *estado da arte* é o *Sequence to Sequence (Seq2Seq)*, o qual utiliza de duas camadas com *LSTM (Long Short Term Memory)*, um tipo de *Rede Neural Recorrente*, para gerar uma saída baseando-se no conteúdo das entradas [46].

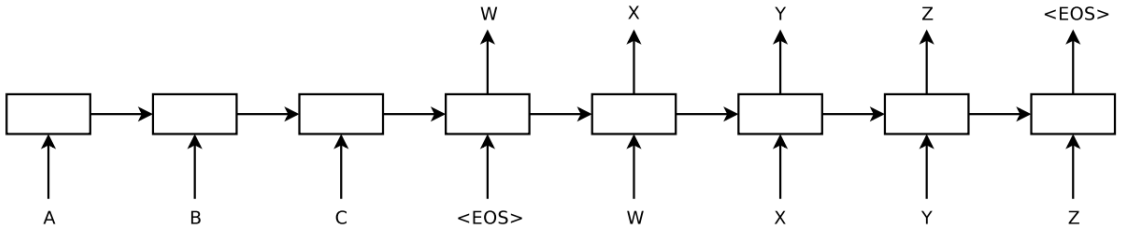

Figura 3.3: Exemplo do Sequence to Sequence [46]

Como pode é demonstrado na Figura 3.3, cada retângulo corresponde à um *neurônio* do LSTM, em que os quatro primeiros pertencem à primeira camada, responsável por processar a entrada *(A, B e C)*, e os quatro últimos pertencem à segunda camada, responsável por gerar a resposta *(X, Y e Z)*.

3.2.2 Modelo Baseado em Regras

Modelos baseados em regras são menos complexos e não necessariamente necessitam de Inteligência Artificial para serem desenvolvidos (como o ELIZA). Chatbots criados com este modelo constam com uma base de dados de templates de respostas pré-definidas, que podem ou não ter variações dependendo das entradas do usuário. Esses Chatbots são utilizados para casos de domínio fechado, em que há uma razão e objetivo específico para se iniciar uma conversa, como em atendimentos na área de vendas ou em um central de ajudas. Eles não costumam cometer erros gramaticais, mas podem ser facilmente reconhecidos como robôs.

As regras utilizadas podem variar de padrões pré-definidos até a classificação de intenções e reconhecimento de entidades. O A.L.I.C.E. é um exemplo de Chatbot que utiliza uma linguagem própria para o reconhecimento de padrões, o *AIML (Artificial Intelligence Markup Language)*, o qual utiliza um formato baseado no *XML (Extensible Markup Language)* para registrar padrões e as respostas referentes a eles [45]. O maior impasse encontrado nesses casos é a necessidade de descrever todos os possíveis padrões que possam ser utilizados, incluindo suas variações, o que acaba por comprometer a precisão e o desempenho do Chatbot.

```
<aiml version="1.0.1">
    <category>
        <pattern>Sala da disciplina:</pattern>
        <template>
            <think>
                <set name="DISCIPLINA">
                    <star/>
                </set>
                <script>:QUERY</script>
            </think>
        </template>
    </category>
</aiml>
```

Figura 3.4: Exemplo da estrutura de um arquivo AIML

A Figura 3.4 mostra um exemplo fictício de um arquivo escrito em AIML, em que ao encontrar o padrão *"Sala da disciplina:"*, o sistema deve extrair o código da disciplina o qual será utilizado para fazer uma *query* na base de dados, retornando o local esperado. Então, se a entrada for *"Sala da disciplina: IF1000"*, o valor *IF1000* será salvo na variável

com nome *DISCIPLINA* e utilizada como argumento da *query* para retornar a sala. No caso do AIML, é possível aplicar Aprendizado de Máquina para a geração automatizada de padrões, diminuindo a carga de trabalho a ser feita.

A aplicação do Aprendizado de Máquina e do Processamento de Linguagem Natural em um modelo baseado em regras geralmente é feita com o objetivo de realizar tarefas como a *classificação de intenções* e o *Reconhecimento de Entidades Nomeadas* em uma etapa de pré-processamento, facilitando o trabalho de processamento da entrada do usuário e de escolha da resposta, pois a classificação substitui o grande número de padrões a serem testados, enquanto o reconhecimento de entidades separa as palavras-chave presentes no texto (Figura 3.5).

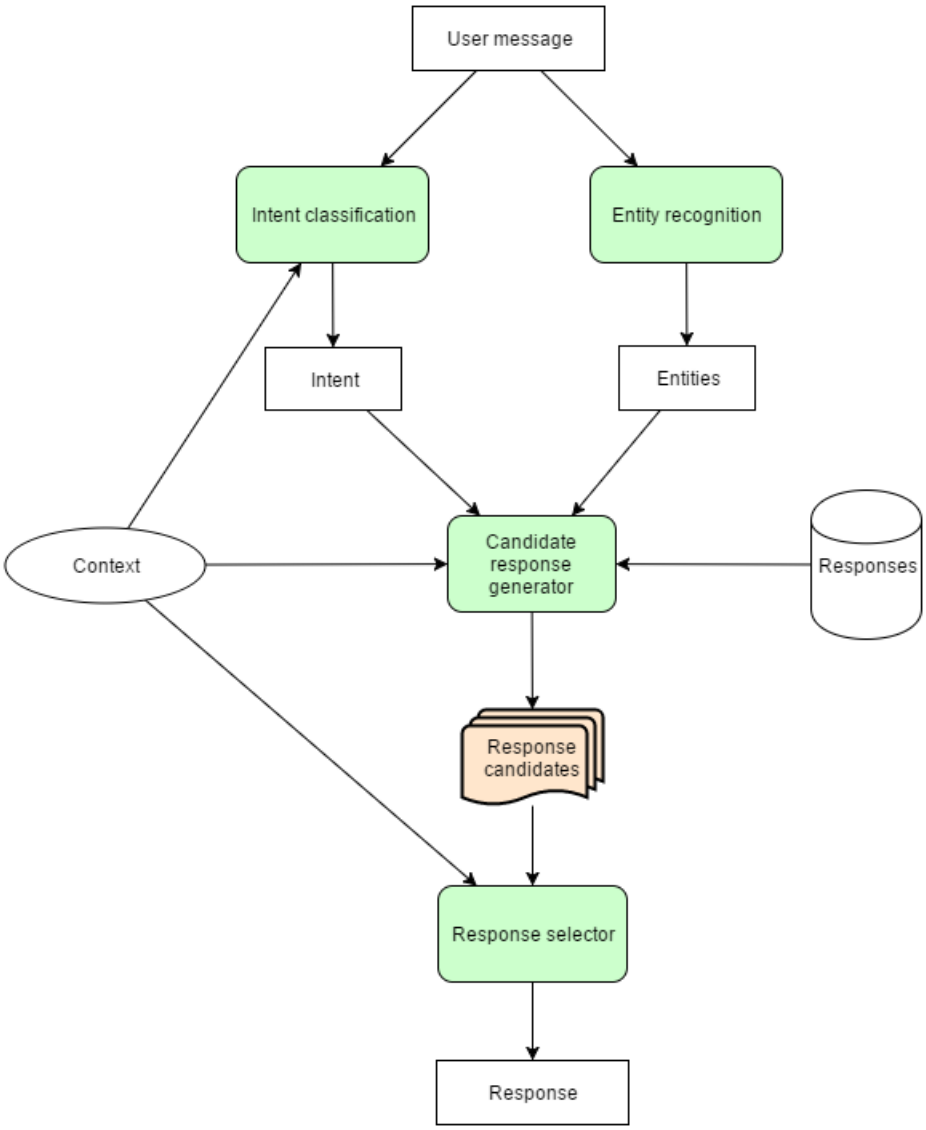

Figura 3.5: Exemplo de uma arquitetura de Chatbot com o uso da Classificação de Intenções e do Reconhecimento de Entidades [45]

A *Classificação de Intenções* é treinada utilizando algoritmos de Aprendizado de Máquina (como o *SVM ou Naive Bayes*) com todas as possíveis intenções e as entradas correspondentes a cada uma, mapeando todas as possibilidades para que o algoritmo consiga identificar, com a melhor precisão possível, em que intenção uma determinada frase desconhecida se encaixa. Dessa forma, o Chatbot consegue identificar frases distintas, mas que possuem a mesma intenção, como exemplo, *"Qual a sala da cadeira IF1000?", "Onde será a aula da disciplina IF1000?", "Qual será o local da aula da disciplina IF1000?"*, são perguntas que possuem a mesma intenção, *saber qual é sala de aula de uma determinada disciplina*, cujas sentenças possuem diferentes estruturas.

O *Reconhecimento de Entidades Nomeadas* também é treinado utilizando algoritmos de Aprendizado de Máquina (nesse caso, *CRF ou Maximum Entropy*) com as possíveis sentenças que se pretende trabalhar, categorizando e destacando nelas todas as entidades relevantes para o Chatbot, para que o algoritmo possa identificá-las posteriormente em entradas desconhecidas. Assim, o Chatbot consegue categorizar por si só todas as entidades presentes em um questionamento de usuário, como exemplo, *"Qual a sala da cadeira IF1000?"*, seria categorizada a entidade *DISCIPLINA*, cujo valor é *IF1000*. Este procedimento, junto da classificação de intenções, é utilizado para processar a resposta que será dada ao usuário, visto que, com as classificações feitas, tem-se os dados pré-processados, *localização da aula*, como a intenção do usuário e *(DISCIPLINA, IF1000)* como a disciplina cuja sala deve ser localizada.

3.3 Considerações Finais

Este capítulo apresentou a definição de *Chatbots* e suas possíveis arquiteturas. Foram comparados a utilização de modelos generativos com modelos baseados em regra e a estrutura de reconhecimento de padrões com a estrutura de *Classificação de Intenções e Reconhecimento de Entidades Nomeadas*, evidenciando a vantagem existente na utilização de *Aprendizado de Máquina* e *Processamento de Linguagem Natural* no processo de desenvolvimento de um *Chatbot*. Por fim, foi mostrado como funciona a arquitetura de um modelo baseado em regras utilizando a classificação de intenções e reconhecimento de entidades. Considerando o conteúdo deste capítulo e o referencial teórico, o próximo capítulo procura mostrar alguns casos que utilizaram a arquitetura em questão.

Capítulo 4

Trabalhos Relacionados

Este capítulo tem como objetivo apresentar trabalhos relacionados ao tema desta pesquisa e que foram utilizados como inspiração para a implementação da solução proposta.

4.1 Processamento de Linguagem Natural

No serviço de Processamento de Linguagem Natural, *wit.ai* [52], é trabalhada a abordagem de *classificação de intenções* e *reconhecimento de entidades* em conteúdos de texto fornecidos por desenvolvedores, o qual também utiliza de Aprendizado de Máquina e cujo sistema oferece uma *API (Application Programming Interface)* de acesso e uma área voltada para treinamento de modelos próprios. Tal procedimento é utilizado neste trabalho para o mesmo fim (mas não necessariamente possuindo a mesma arquitetura e funcionalidades), que é *classificar as intenções dos usuários e identificar entidades relevantes para o Chatbot, suportando o modelo baseada em regras*.

4.1.1 Dados Estruturados

A estruturação dos dados de entrada feita tanto pelo *wit.ai* quanto pelo *api.ai* [7] é baseada em *JSON (JavaScript Object Notation)*, uma linguagem de notação para representação de objetos. Ao enviar uma sentença para o sistema, as tarefas necessárias são executadas sobre ela (classificação de intenções, reconhecimento de en-

tidades) e é retornado ao desenvolvedor um JSON com os dados estruturados, representando o que foi identificado no texto. Desta forma, a sentença é transformada em um objeto cujo sistema é capaz de compreender. Neste trabalho, é feita a estruturação dos dados utilizando a mesma abordagem (representar a sentença em um *JSON*). Como exemplo, caso seja enviada a frase *"Onde será a aula da disciplina IF1000?"*, o retorno seria:

```
[{
    "action": "get_class_location",
    "entities": [{
        "name": "DISCIPLINA",
        "value": "IF1000"
    }]
}]
```

Figura 4.1: Exemplo de uma sentença representada em *JSON*, destacando suas entidades e intenções

4.2 Conditional Random Fields

Segundo *Lafferty, McCallum e Pereira* [26], *Conditional Random Fields (CRF)* oferece uma vantagem quanto aos modelos de Markov, *Hidden Markov Models (HMM)* e *Maximum Entropy Markov Models (MEMM)*, na categorização de sequências, técnica utilizada no Processamento de Linguagem Natural para executar tarefas como a *etiquetagem de classes gramaticais (POS Tagging)* e o *Reconhecimento de Entidades Nomeadas (NER)*. Apesar do *NLTK* recomendar o uso de HMMs ou MEMMs [2], neste trabalho foi utilizado CRF para realizar ambas as tarefas, seguindo as recomendações em [26].

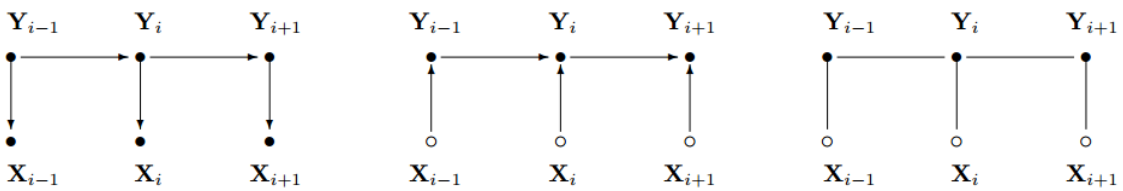

Figura 4.2: Diferenças entre os grafos de HMM, MEMM e CRF, respectivamente [26]

Capítulo 5

Implementação da Inteligência Artificial do Chatbot

O Chatbot[1] proposto neste trabalho tem como objetivo auxiliar os estudantes quanto ao que está acontecendo no *Centro de Informática da Universidade Federal de Pernambuco*, junto de dúvidas relacionadas a horários, localizações, processos e eventos, sendo capaz de entendê-las e respondê-las de forma natural. Para isso, foram desenvolvidos modelos para a *classificação de intenções* e para o *reconhecimento de entidades* em textos, utilizando *Aprendizado de Máquina* e *Processamento de Linguagem Natural*. Seguem as principais funcionalidades implementadas:

- *Part-Of-Speech Tagger (POS)*, para a classificação gramatical de palavras, treinado com *CRF*;
- *Inside-Outside-Begin Tagger (IOB)*, para executar o reconhecimento de entidades, treinado com *CRF*;
- Um *Classificador de Intenções*, treinado com *SVM*;
- Uma interface para *classificar intenções*, *identificar entidades* e retornar a entrada de modo estruturado;

Estas funcionalidades serão detalhas nas próximas seções, junto dos procedimentos utilizados para tratar os dados e treinar os modelos.

[1] Link para o repositório do Chatbot: https://github.com/victorfsf/eva

Para o desenvolvimento do projeto, foram utilizadas diversas bibliotecas de código aberto para facilitar o processo de implementação, em que todas estão disponíveis para instalação através do gerenciador de pacotes do Python, o *PIP*. Seguem as especificações dessas bibliotecas, em conjunto com as razões para serem utilizadas:

- *nltk (3.2.4)*, a biblioteca principal, utilizada para facilitar tarefas de processamento de linguagem natural;
- *boltons (17.1.0)*, uma biblioteca contendo diversas utilidades que deveriam ser nativas do Python. No projeto, foi utilizada sua funcionalidade de *caching*, para evitar carregar os modelos treinados mais de uma vez;
- *numpy (1.12.1)*, biblioteca para operações matemáticas complexas, desenvolvida em C (para ser rápida), utilizada pelo *NLTK*, *SciPy* e *scikit-learn* em operações de vetores e matrizes;
- *python-crfsuite (0.9.2)*, responsável pela integração do classificador de *Conditional Random Fields* com o *NLTK*;
- *regex (2017.5.26)*, implementação mais rápida e melhorada do módulo de regex do Python, utilizado no *IOBReader*;
- *requests (2.18.1)*, biblioteca que facilita solicitações em HTTP, utilizada na função de download dos modelos pré-treinados;
- *scikit-learn (0.18.1)*, oferece diversas implementações de modelos de *Aprendizado de Máquina*. Neste trabalho, foi utilizado para treinar o modelo de *Support Vector Machines*;
- *scipy (0.19.0)*, outra biblioteca para operações matemáticas complexas que pode ser ainda mais rápida que o *numpy*. Utilizada pelo *scikit-learn* para diminuir o tempo de treinamento dos seus modelos.

5.1 Arquitetura

Diante das arquiteturas apresentadas anteriormente, o foco deste trabalho se encontra na Figura 5.1, pertencente ao *processamento de linguagem natural* de uma arquitetura

baseada em regras.

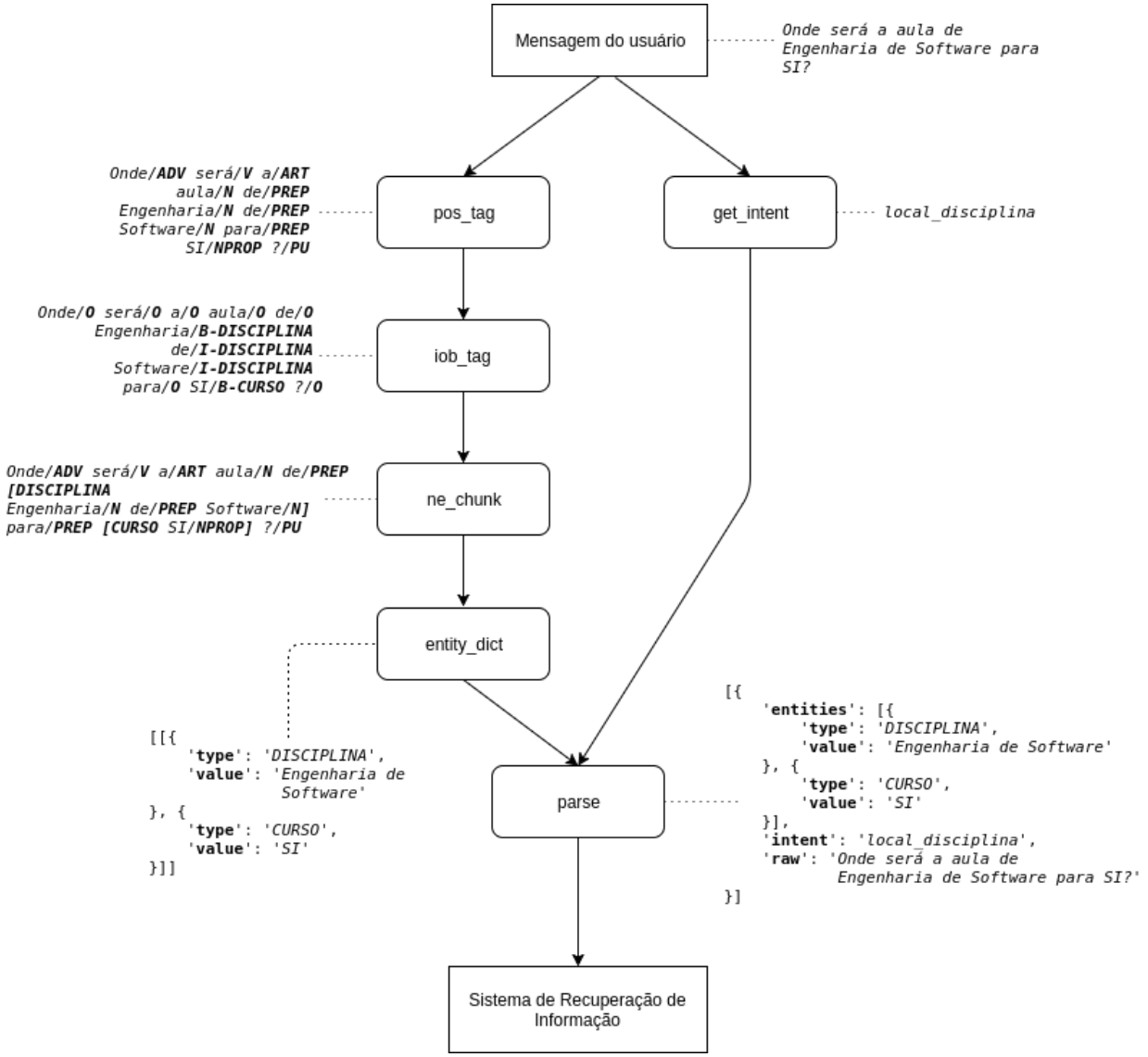

Figura 5.1: Arquitetura do projeto

5.2 Part-Of-Speech Tagger

O *POS Tagger* é responsável pela etiquetagem gramatical de cada palavra de um texto e é uma etapa de pré-processamento da sentença, antes da execução do *IOB Tagger*. A Tabela 5.1 mostra a relação das etiquetas que o algoritmo pode reconhecer.

Etiqueta	**Significado**
PRO-KS	Conjunção, fazendo o papel de Pronome

NPROP	Substantivo Próprio
PU	Pontuação
PROADJ	Pronome Adjetivo
NUM	Numeral
PREP+PROADJ	Preposição + Pronome Adjetivo
ADV	Advérbio
ADV-KS	Conjunção, fazendo o papel de Advérbio
PREP+PROPESS	Preposição + Pronome Pessoal
PROSUB	Pronome Substantivo
CUR	Dinheiro
PROPESS	Pronome Pessoal
PREP+ADV	Preposição + Advérbio
PREP+PROSUB ou PREP+PRO-KS	Preposição + Pronome Substantivo
N	Substantivo Comum
V	Verbo
ADJ	Adjetivo
KC ou KS	Conjunções
ART	Artigo
PREP+ART	Preposição + Artigo
PREP	Preposição
PDEN	Advérbio, geralmente relacionado aos advérbios que demostrem sentido de inclusão ou exclusão (também, até, somente...)
PCP	Verbo substantivado
IN	Interjeição

Tabela 5.1: Relação de etiquetas utilizadas no POS Tagger.

Para utilizar as funcionalidades do *POS Tagger*, foi implementada a função *pos_tag*, a qual recebe como entrada uma sentença (ou lista de sentenças) e as retorna devidamente etiquetadas, como é demonstrado na Figura 5.2.

```
11  def __cached_tagger(model_file, cache_key, tagger_model):
12      if cache_key not in cache:
13          tagger = tagger_model()
14          tagger.set_model_file(
15              os.path.join(
16                  os.getcwd(), model_file
17              )
18          )
19          cache.update({
20              cache_key: tagger
21          })
22      return cache[cache_key]
23
24
25  def pos_tag(*sents, **kwargs):
26      tagger = __cached_tagger(
27          kwargs.pop('model', 'models/pos.model'),
28          'pos_tag', CRFTagger
29      )
30      return [
31          tagger.tag(word_tokenize(sent))
32          for sent in sents
33      ]
34
```

Figura 5.2: Implementação da função *pos_tag*

A função *__cached_tagger* é responsável por manter um *cache* dos modelos carregados, dado que esses carregamentos tendem a gastar um tempo consideravelmente longo na execução da função. Desta forma, o modelo é carregado somente uma vez, e as próximas chamadas da função não vão precisar carregá-lo novamente.

Antes de etiquetar cada palavra, estas devem ser separadas da sentença. O *NLTK* possui a função *word_tokenize*, responsável pelo reconhecimento e tokenização de palavras em uma sentença e é utilizada durante a execução do *POS Tagger*, para que ele possa passar por cada *token* gerado. A Figura 5.3 trás um exemplo da função.

```
In [1]: pos_tag('Onde será a aula de Engenharia de Software do curso de SI?')
Out[1]:
[[('Onde', 'ADV'),
  ('será', 'V'),
  ('a', 'ART'),
  ('aula', 'N'),
  ('de', 'PREP'),
  ('Engenharia', 'N'),
  ('de', 'PREP'),
  ('Software', 'N'),
  ('do', 'PREP+ART'),
  ('curso', 'N'),
  ('de', 'PREP'),
  ('SI', 'NPROP'),
  ('?', 'PU')]]
```

Figura 5.3: Exemplo de uso da função *pos_tag*

5.2.1 Coleta e Tratamento dos Dados

Para o treinamento do *POS Tagger* na língua portuguesa, foram utilizadas as sentenças do corpus *Mac-Morpho*, alterando algumas etiquetas (as etiquetas finais são as presentes na Tabela 5.1).

O Corpus *Mac-Morpho* é disponibilizado pelo *NLTK* [2] e foi utilizado pelo *NLPNet*[2] para a implementação de um *POS Tagger* em português com *Redes Neurais*, utilizando *Python 2*. Para o tratamento dos dados, foram coletadas amostras do *NLPNet*, as quais foram utilizadas como base de treinamento.

5.2.2 Pré-processamento

O treinamento do *POS Tagger* foi realizado com o auxilio do *NLTK* e sua integração com o *pycrfsuite*[3] para trabalhar com *Conditional Random Fields (CRF)*. O CRF necessita de uma lista de features pré-definidas para que possa gerar o modelo desejado. Nesse caso, foram utilizadas as features presentes na Tabela 5.2, utilizadas por padrão pelo NLTK.

Feature	**Significado**
CAPITALIZATION	Caso a palavra tenha a primeira letra maiúscula, esta feature é adicionada

[2]Link para o repositório do NLPNet: https://github.com/erickrf/nlpnet
[3]Link para o repositório do python-crfsuite: https://github.com/scrapinghub/python-crfsuite

PUNCTUATION	Caso a *palavra* seja uma pontuação, esta feature é adicionada
SUF	Sufixo da palavra atual
WORD	A palavra atual
HAS_NUM	Caso a palavra tenha a um número, esta feature é adicionada

Tabela 5.2: Features utilizadas para treinar o POS Tagger

5.2.3 Treinamento

Após o pré-processamento das features, é realizado o treinamento do modelo de *Conditional Random Fields*. Para cada palavra em cada sentença, são selecionadas as features X_n e a label Y_n, em que n é a posição da palavra na sentença X e Y_n corresponde a classificação da entidade das features em X_n. Logo, tem-se $fl_s = [(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)]$, sendo fl a lista de sentenças pré-processadas e fl_s a representação de uma sentença s. A Tabela 5.3 mostra os resultados obtidos no treinamento do modelo.

Base	**Precisão**
Treinamento	97%
Teste	95%

Tabela 5.3: Resultados da precisão da classificação gramatical

A *base de teste* corresponde a 20% da base total e a *base de treinamento* corresponde a 80%. O resultado é calculado ao utilizar a função *crf.evaluate(base)* a qual retorna o percentual de precisão do modelo, em que *crf* é o objeto do modelo treinado e a *base* pode ser de treinamento ou de teste.

5.2.4 Relatórios

O *CRFTagger* oferece dois relatórios pós-treinamento, o *relatório de transições* (Tabela 5.4), o qual mostra as transições mais comuns encontradas, e o *relatório de estado das features* (Tabela 5.5), o qual mostra as features mais relevantes do modelo.

Transição	**Relevância**
ART → NPROP	7.595956
ART → N	6.725485
NPROP → NPROP	6.526405
PREP+ART → NPROP	5.705348
N → ADJ	5.240425
NUM → NPROP	4.727889
PREP+PRO-KS → PRO-KS	4.720989
PDEN → PDEN	4.587779
NUM → N	4.585131
N → NPROP	4.491869
KS → KS	4.444063
PREP → NPROP	4.337675
ART → ADJ	4.124396
ADV → NPROP	4.122848
PRO-KS → PRO-KS	4.115734
PREP+ART → N	4.107100
V → NPROP	4.013984
PREP+PROADJ → N	3.936370
PROADJ → N	3.934893
N → PU	3.882811

Tabela 5.4: As 20 transições mais relevantes do *POS Tagger*

Feature	**Label**	**Relevância**
WORD_e	KC	11.461304
WORD_quais	PRO-KS	10.473347
WORD_de	PREP	10.297778
WORD_entanto	KC	9.784026
WORD_da	PREP+ART	9.776174
WORD_quando	KS	9.556150
WORD_para	PREP	9.512720
WORD_-se	PROPESS	9.502324
WORD_mas	KC	9.361279
WORD_nada	PROSUB	9.174009
WORD_é	V	9.050204
WORD_das	PREP+ART	8.925653
WORD_o	ART	8.914779
WORD_na	PREP+ART	8.819466
WORD_dos	PREP+ART	8.790487

WORD_do	PREP+ART	8.721774
WORD_em	PREP	8.654769
WORD_quando	ADV-KS	8.568677
WORD_cada	PROADJ	8.425719
WORD_E	KC	8.418804

Tabela 5.5: As 20 features mais relevantes do *POS Tagger*

5.3 Inside-Outside-Begin Tagger

O *IOB Tagger* é responsável pela identificação e etiquetagem de entidades em textos. Ele utiliza do resultado do *POS Tagger* para executar o reconhecimento de entidades baseando-se em um grupo de *features* pré-definidas e que foram selecionadas de acordo com experimentos sobre a precisão do modelo, os quais estão descritos nas próximas subseções. A Tabela 5.6 mostra as entidades que foram trabalhadas.

Entidade	**Exemplos**
DISCIPLINA	Engenharia de Software, Banco de Dados, Gestão de Processos de Negócio...
CURSO	Sistemas de Informação, SI, Ciência da Computação, CC, Engenharia da Computação, EC...
PESSOA	Victor Ferraz, João, Maria...
DATA	1º de agosto, 14/07, 20/06/2017...
CENTRO	Centro de Informática, CIn, Centro de Ciências Exatas e da Natureza, CCEN...
ORG	Universidade, UFPE...

Tabela 5.6: Relação de entidades utilizadas no IOB Tagger.

Para utilizar o Reconhecimento de Entidades, foram implementadas as funções *iob_tag* e *ne_chunk*. A função *iob_tag* recebe uma ou mais sentenças e as retorna com todas as entidades reconhecidas marcadas, enquanto a função *ne_chunk* retorna uma representação das entidades em forma de *árvores*, as quais podem ser representadas graficamente. A Figura 5.4 mostra a implementação das duas funções e as Figuras 5.5 e 5.6 trazem seus exemplos.

```
36  def iob_tag(*sents, **kwargs):
37      tagger = __cached_tagger(
38          kwargs.pop('model', 'models/iob.model'),
39          'iob_tag', IOBTagger
40      )
41      return [
42          [(w, p, i) for (w, p), i in tagger.tag(sent)]
43          for sent in pos_tag(*sents)
44      ]
45
46
47  def ne_chunk(*sents):
48      return [
49          conlltags2tree(i) for i in iob_tag(*sents)
50      ]
51
```

Figura 5.4: Implementação das funções *iob_tag* e *ne_chunk*

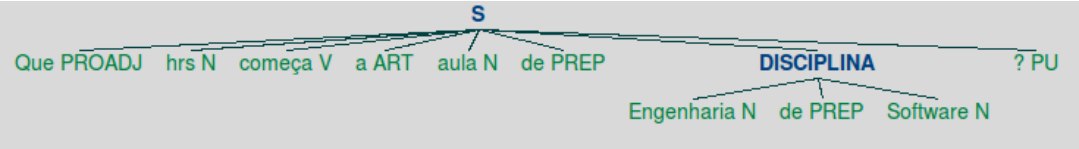

Figura 5.5: Exemplo da função *ne_chunk*

```
In [1]: iob_tag('Onde será a aula de Engenharia de Software do curso de SI?')
Out[1]:
[[('Onde', 'ADV', 'O'),
  ('será', 'V', 'O'),
  ('a', 'ART', 'O'),
  ('aula', 'N', 'O'),
  ('de', 'PREP', 'O'),
  ('Engenharia', 'N', 'B-DISCIPLINA'),
  ('de', 'PREP', 'I-DISCIPLINA'),
  ('Software', 'N', 'I-DISCIPLINA'),
  ('do', 'PREP+ART', 'O'),
  ('curso', 'N', 'O'),
  ('de', 'PREP', 'O'),
  ('SI', 'NPROP', 'B-CURSO'),
  ('?', 'PU', 'O')]]
```

Figura 5.6: Exemplo de uso da função *iob_tag*

5.3.1 Coleta e Tratamento dos Dados

Os dados utilizados no treinamento do *IOB Tagger* foram gerados à mão, procurando representar ao máximo as possíveis entradas de um usuário em uma conversa com um Chatbot. Os dados foram salvos num formato criado neste trabalho e lidos por uma

ferramenta chamada *IOB Reader*, a fim de facilitar a digitação e leitura dos dados. Segue o exemplo:

> *"Onde será a aula de **[DISCIPLINA Engenharia de Software]** do curso de **[CURSO Sistemas de Informação]**?"*

As entidades são delimitadas por colchetes, cuja primeira palavra é a sua identificação em maiúsculo e o resto é o seu valor, como pode ser visto em *"[CURSO Sistemas de Informação]"*, a entidade é *"CURSO"* e o valor é *"Sistemas de Informação"*. Ao serem lidos pelo *IOB Reader*, estes dados serão processados e cada sentença passará pelo *IOB Tagger*, resultando no exemplo da Figura 5.6. O *IOB Reader* também é utilizado para salvar as *features e labels de treinamento e de teste*, utilizadas na *classificação de intenções*.

5.3.2 Pré-processamento

O treinamento do *IOB Tagger* foi realizado de modo semelhante ao do *POS Tagger*, mas utilizando features diferentes. Nesse caso, foram utilizadas as features presentes na Tabela 5.7.

Feature	**Significado**
SUF	Sufixo da palavra atual
PREVSUF	Sufixo da palavra anterior
NEXTSUF	Sufixo da próxima palavra
PREVSUF+SUF	Sufixo da palavra anterior + sufixo da palavra atual
SUF+NEXTSUF	Sufixo da palavra atual + sufixo da próxima palavra
PREVSUF+SUF+NEXTSUF	Sufixo da palavra anterior + sufixo da palavra atual + sufixo da próxima palavra
CAPITALIZATION	Caso a palavra tenha a primeira letra maiúscula, esta feature é adicionada
ARTIGO	Se a POS da palavra for ART ou PREP+ART
VERBO	Se a POS da palavra for V
NOME-PROPRIO	Se a POS da palavra for NPROP
PONTUACAO	Se a POS da palavra for PU
PREPOSICAO	Se a POS da palavra for PREP+
IS-NUMBER	Se a *palavra* for um número, esta feature é adicionada

HAS-NUMBER	Se a palavra conter um número, esta feature é adicionada
TAGS-SINCE-ART	Todas as POS desde o último artigo registrado na sentença atual
MOST-USED-WORD	Palavra mais utilizada na sentença
MOST-USED-POS	POS mais utilizada na sentença
WORD	Palavra atual
POS	Classe gramatical da palavra atual
WORD+POS	Palavra atual + sua classe gramatical
PREVPOS	Classe gramatical da palavra anterior
NEXTPOS	Classe gramatical da próxima palavra
PREVPOS+POS+NEXTPOS	Classe gramatical da palavra anterior + classe gramatical da palavra atual + classe gramatical da próxima palavra
POS+NEXTPOS	Classe gramatical da palavra atual + classe gramatical da próxima palavra
PREVPOS+POS	Classe gramatical da palavra anterior + classe gramatical da palavra atual
PREVWORD+WORD+NEXTWORD	Palavra anterior + palavra atual + próxima palavra
WORD+NEXTWORD	Palavra atual + próxima palavra
PREVWORD+WORD	Palavra anterior + palavra atual

Tabela 5.7: Features utilizadas para treinar o IOB Tagger

As features utilizadas foram selecionadas de acordo com recomendações da própria implementação do *CRFTagger* do NLTK e com as recomendações de *Tkachenko e Simanovsky* [49], os quais demonstram os resultados da concatenação de features para formar uma nova feature, desde que estejam relacionadas entre si.

5.3.3 Treinamento

Após o pré-processamento das features, é realizado o treinamento do modelo de *Conditional Random Fields*, exatamente do mesmo modo utilizado no treinamento do *POS Tagger*, substituindo as POS por entidades e as palavras por tuplas, ou seja, para cada sentença s, tem-se a lista de tuplas $[((w_1, p_1), e_1), ((w_2, p_2), e_2), ..., ((w_n, p_n), e_n)]$, em que w_n corresponde à palavra na posição n da sentença s, p_n a sua classe gramatical e e_n a entidade atribuída a ela. A Tabela 5.8 mostra os resultados obtidos no

treinamento do modelo.

Base	Precisão
Treinamento	99.96%
Teste	97.64%

Tabela 5.8: Resultados da precisão do reconhecimento de entidades

5.3.4 Relatórios

Assim como no *POS Tagger*, o *CRFTagger* oferece dois relatórios pós-treinamento, o *relatório de transições* (Tabela 5.4) e o *relatório de estado das features* (Tabela 5.5).

Transição	Relevância
I-DISCIPLINA → I-DISCIPLINA	3.019540
O → O	2.812277
B-DISCIPLINA → I-DISCIPLINA	2.677819
B-PESSOA → I-PESSOA	2.421836
O → B-DATA	2.020265
I-DATA → I-DATA	2.005833
O → B-CURSO	1.864552
O → B-DISCIPLINA	1.838447
I-CURSO → I-CURSO	1.711786
O → B-PESSOA	1.681188
I-PESSOA → I-PESSOA	1.611307
B-CURSO → I-CURSO	1.558444
B-DATA → I-DATA	1.341504
I-CENTRO → I-CENTRO	0.977289
I-DISCIPLINA → O	0.912742
B-CENTRO → I-CENTRO	0.779442
O → B-ORG	0.703629
O → B-CENTRO	0.696861
I-DATA → O	0.272966
B-ORG → B-DATA	0.270192

Tabela 5.9: As 20 transições mais relevantes do *IOB Tagger*

Feature	Label	Relevância
PREVPOS_<START>	O	2.437604

HAS_NUMBER	B-DISCIPLINA	1.465399
PONTUACAO	O	1.271919
POS_PU	O	1.271919
NEXTPOS_N	O	1.244457
HAS_DASH	B-DATA	1.175233
POS+NEXTPOS_PU+<END>	O	0.962380
PREVSUF_e	B-DISCIPLINA	0.946450
SUF_ro	B-DATA	0.922460
PREVPOS_PREP+ART	B-CENTRO	0.918072
ARTIGO	O	0.879643
NEXTPOS_NPROP	O	0.867666
SUF_ção	I-CURSO	0.864518
PREVSUF_ina	B-DISCIPLINA	0.861234
SUF_n	B-CENTRO	0.841538
PREVSUF+SUF_o+n	B-CENTRO	0.841538
HAS_NUMBER	B-DATA	0.838111
PREVPOS_PREP+ART	B-ORG	0.835548
WORD_?	O	0.830830
WORD+POS_?+PU	O	0.830830

Tabela 5.10: As 20 features mais relevantes do *IOB Tagger*

5.4 Classificador de Intenções

A *classificação de intenções* é responsável por identificar as intenções presentes nas sentenças que serão interpretadas pelo Chatbot. O classificador utiliza dos mesmo dados que o *IOB Tagger*, apesar de ter uma interpretação diferenciada. A Tabela 5.11 lista todas as intenções mapeadas neste trabalho.

Intenção	**Exemplo**
local_disciplina	”Onde será a aula de Engenharia de Software?”, ”Qual a sala de Programação I?”
horario_disciplina	”Que horas é a aula de Introdução a Programação para CC?”, ”Qual o horário da aula de Banco de dados?”
professor_disciplina	”Quem dá as aulas de Lógica?”, ”Quem ensina na disciplina de Matemática Discreta?”

aulas_professor	”Quais as aulas que o professor João ensina?”, ”Disciplinas da professora Carla”
noticias	”O que há de novo no CIn?”, ”O que aconteceu na UFPE ontem?”
introducao	”Meu nome é Victor”, ”Eu sou Eva”
cumprimento	”Oi”, ”Olá”

Tabela 5.11: Relação de intenções reconhecidas pelo classificador

Para executar o classificador, foi implementada a classe *IntentClassifier* para facilitar a entrada de dados que será utilizada no treinamento do modelo SVM. A Figura 5.7 trás a sua implementação e a 5.8 trás um exemplo da execução do modelo.

```python
class IntentClassifier(LinearSVC):

    def fit(self, path='data/', **kwargs):
        reader = IOBReader(
            path,
            test_size=kwargs.pop('test_size', 0.2),
            random_state=kwargs.pop('random_state', 42)
        )
        self.x_features, self.x_labels = zip(*reader.train_set)
        self.y_features, self.y_labels = zip(*reader.test_set)
        # TF-IDF
        self.tfidf = TfidfVectorizer()
        x_tfidf = self.tfidf.fit_transform(
            self.stem_features(self.x_features)  # STEMMING
        )
        return super().fit(x_tfidf, self.x_labels)

    def stem_features(self, features):
        if not hasattr(self, 'stemmer'):
            self.stemmer = SnowballStemmer(language='portuguese')
            self.stemmer.stopwords = set(stopwords.words('portuguese'))
        return [
            ' '.join(
                self.stemmer.stem(x) for x in word_tokenize(f)
                if x not in self.stemmer.stopwords
            ) for f in features
        ]

    def predict(self, features):
        if not hasattr(self, 'tfidf'):
            raise AttributeError(
                'The model must be trained with fit() first.'
            )
        features = self.tfidf.transform(
            self.stem_features(features)
        )
        return super().predict(features)
```

Figura 5.7: Implementação da classe *IntentClassifier*

```
In [1]: svm = IntentClassifier()

In [2]: svm.fit(path='data/', test_size=0.2)
Out[2]: IntentClassifier(accuracy=0.97452, features=1693, labels=10)

In [3]: svm.predict(['Que horas começa a aula de DB para SI?'])
Out[3]:
array(['horario_disciplina'],
      dtype='<U18')
```

Figura 5.8: Exemplo de execução da classe *IntentClassifier*

5.4.1 Coleta e Tratamento dos Dados

Os dados selecionados para o treinamento de intenções foram os mesmos utilizados no *IOB Tagger*, em que cada intenção mapeada é separada em um arquivo diferente, o qual cada sentença dentro de cada arquivo possui a mesma intenção. Como exemplo, o arquivo *local_disciplina.iob* descreve a intenção *local_disciplina* e cada sentença nele pertence a essa classificação.

Antes de serem inseridas no algoritmo, as sentenças devem ser separadas em labels e features e cada feature deve ser tratada através do processo de *stemização* e do cálculo do *TF-IDF*. Com o auxílio do NLTK, foi utilizado o *SnowballStemmer* para a língua portuguesa, filtrando as *stopwords*; quanto ao cálculo do TF-IDF, foi utilizada a classe *Tfidf-Vectorizer*, do *scikit-learn*, treinado sobre os dados de todas as intenções existentes. A implementação de ambos pode ser visualizada na Figura 5.7.

5.4.2 Treinamento

Após a coleta e tratamento dos dados, é realizado o treinamento utilizando *Support Vector Machines (SVM)*, inserindo as features e labels correspondentes, representadas pela lista de tuplas $[(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)]$, em que X_n é a matriz representando os pesos de cada token na sentença (resultado da *stemização* e do *TF-IDF*), e Y_n corresponde à intenção da feature na posição n.

Base	**Precisão**
Treinamento	99.73%
Teste	97.94%

Tabela 5.12: Resultados da precisão da classificação de intenções

A *base de treinamento* corresponde a 80% da base total e a *base de teste* corresponde a 20%. O resultado é calculado ao utilizar a função *svm.accuracy(base)* a qual retorna o percentual de precisão do modelo, em que *svm* é o objeto do modelo treinado.

5.4.3 Relatórios

O *scikit-learn* oferece um conjunto de relatórios possíveis que podem ser utilizados para testar a eficiência dos modelos treinados utilizando a biblioteca. Dentre eles, foram utilizados a *Matriz de Confusão* e o *Relatório de Classificação*. A *Matriz de Confusão* mostra as quantidades de acertos e erros presentes em cada label treinada, como pode ser visto na Figura 5.9.

```
In [1]: svm.confusion_matrix(train_set)
Out[1]:
array([[43,  0,  0,  0,  0,  0,  0],
       [ 0, 28,  0,  0,  0,  0,  0],
       [ 0,  0, 57,  0,  0,  0,  0],
       [ 1,  0,  0, 42,  0,  0,  0],
       [ 0,  0,  0,  0, 68,  0,  0],
       [ 0,  0,  0,  0,  0, 98,  0],
       [ 0,  0,  0,  0,  0,  0, 36]])

In [2]: svm.confusion_matrix(test_set)
Out[2]:
array([[11,  0,  0,  0,  0,  0,  0],
       [ 0,  7,  0,  0,  0,  1,  0],
       [ 0,  0, 15,  0,  0,  0,  0],
       [ 0,  0,  0, 11,  0,  0,  0],
       [ 0,  0,  0,  0, 18,  0,  0],
       [ 0,  0,  0,  0,  0, 25,  0],
       [ 0,  0,  0,  0,  1,  0,  8]])
```

Figura 5.9: Valores da *Matriz de Confusão* do *IntentClassifier*

Em que as linhas são as labels existentes e as colunas as labels que foram atribuídas pelo classificador para cada sentença usada. Os valores na diagonal correspondem aos acertos, e os que estão fora, aos erros. Por definição, dada uma matriz de confusão C, o valor na posição C_ij equivale ao número de features que pertencem à label i, mas foram classificadas como pertencentes à label j [41].

O *Relatório de Classificação* trás uma tabela contendo as informações de precisão (precision), da revocação (recall), do *f1-score* e do número de features utilizadas (support) para cada label treinada. Segue a Figura 5.10.

```
In [1]: print(svm.report(r.train_set))
                    precision    recall  f1-score   support

     aulas_professor       0.98      1.00      0.99        43
         cumprimento       1.00      1.00      1.00        28
  horario_disciplina       1.00      1.00      1.00        57
          introducao       1.00      0.98      0.99        43
    local_disciplina       1.00      1.00      1.00        68
             noticias       1.00      1.00      1.00        98
professor_disciplina       1.00      1.00      1.00        36

         avg / total       1.00      1.00      1.00       373


In [2]: print(svm.report(r.test_set))
                    precision    recall  f1-score   support

     aulas_professor       1.00      1.00      1.00        11
         cumprimento       1.00      0.88      0.93         8
  horario_disciplina       1.00      1.00      1.00        15
          introducao       1.00      1.00      1.00        11
    local_disciplina       0.95      1.00      0.97        18
             noticias       0.96      1.00      0.98        25
professor_disciplina       1.00      0.89      0.94         9

         avg / total       0.98      0.98      0.98        97
```

Figura 5.10: Relatório de Classificação do *IntentClassifier*

5.5 Estruturação de Sentenças

A *estruturação de sentenças* é a etapa final do trabalho da *Inteligência Artificial* deste trabalho e é responsável pela combinação da *classificação de intenções* com o *reconhecimento de entidades* para retornar a sentença de forma *estruturada*, destacando a sua intenção e suas entidades. Para isso, foi implementada a função *parse(*sents)* (descrita na Figura 5.11), que recebe uma ou mais sentenças e as retorna no formato *JSON*, como é exemplificado na Figura 5.13. Também foi implementada a função *entity_dict(*sents)*, responsável por converter uma *árvore de entidades* em um *dicionário de entidades*, exemplificada na Figura 5.12.

```
7   def parse(*sents):
8       return [
9           {
10              'entities': entities,
11              'intent': intent
12          } for entities, intent, sent in zip(
13              entity_dict(*sents),
14              get_intent(*sents),
15              sents
16          )
17      ]
18
```

Figura 5.11: Implementação da função *parse*

```
58  def entity_dict(*sents, **kwargs):
59      for tree in ne_chunk(*sents, **kwargs):
60          entities = []
61          for branch in tree:
62              if isinstance(branch, Tree):
63                  entities.append({
64                      'type': branch.label(),
65                      'value': ' '.join([
66                          x for x, _ in branch.leaves()
67                      ])
68                  })
69          yield entities
70
```

Figura 5.12: Implementação da função *entity_dict*

```
In [1]: parse('Qual o professor de Gestão de Processos de Negócios para SI?')
Out[1]:
[{
    'entities': [{
        'type': 'DISCIPLINA',
        'value': 'Gestão de Processos de Negócios'
    }, {
        'type': 'CURSO',
        'value': 'SI'
    }],
    'intent': 'professor_disciplina'
}]

In [2]: parse('Onde será a aula de programação 1?')
Out[2]:
[{
    'entities': [{
        'type': 'DISCIPLINA', 'value': 'programação 1'
    }],
    'intent': 'local_disciplina'
}]

In [3]: parse('Quais as notícias do dia 28 de junho de 2017?')
Out[3]:
[{
    'entities': [{
        'type': 'DATA', 'value': '28 de junho de 2017'
    }],
    'intent': 'noticias'
}]

In [4]: parse('Aulas dadas pelo professor João')
Out[4]:
[{
    'entities': [{
        'type': 'PESSOA',
        'value': 'João'
    }],
    'intent': 'aulas_professor'
}]
```

Figura 5.13: Exemplos da função *parse*

5.6 Considerações Finais

A implementação demonstrada nesta seção é uma representação de como funciona a *Inteligência Artificial* do Chatbot. Nela, foram exibidas os resultados dos treinamentos dos modelos de *Conditional Random Fields* e *Support Vector Machines* junto das funções desenvolvidas para auxiliar na tarefa de classificação de sentenças.

Capítulo 6

Conclusão e Trabalhos Futuros

Com a popularização da Inteligência Artificial e devido ao grande aumento de usuários em aplicações de conversação, incluindo aquelas que oferecem APIs para desenvolvimento de Chatbots (Telegram, Facebook Messenger, Slack), a liberdade e facilidade de acesso a essas ferramentas acarretou na criação de diversos Chatbots de assuntos variados, indo de arquiteturas simples até as mais complexas. Este trabalho mostrou como é desenvolvida a arquitetura de uma Chatbot utilizando *Processamento de Linguagem Natural* e *Aprendizado de Máquina*, demonstrando o quão importante é a *Inteligência Artificial* para o processo.

O principal objetivo desta pesquisa foi propor uma arquitetura para Chatbots utilizando *Inteligência Artificial*, focando na *Classificação de Intenções* e no *Reconhecimento de Entidades*, com o intuito de implementar um Chatbot capaz de *processar a linguagem natural*, sem necessitar de reconhecimento de *padrões estáticos* (padrões que necessitam ter uma relevância de 100% na combinação com a entrada, ou que utilizem de *regex*), e que saiba reconhecer as similaridades entre os padrões conhecidos e a entrada do usuário, utilizando *Aprendizado de Máquina*.

A arquitetura desenvolvida possibilita a expansão do contexto do Chatbot, disponibilizando uma interface com exemplos de como implementar *entidades e intenções* de forma simples, rápida, fácil de serem treinadas e de serem postas em produção.

Trabalhos futuros incluem:

- Adicionar mais entidades e intenções relevantes ao contexto do CIn, como processos e eventos;
- Implementar integrações com o *Telegram*, *Slack* ou *Facebook Messenger*;
- Implementar um sistema de respostas utilizando técnicas de *Recuperação de Informação* (utilizando *gensim*[1], caso seja em python);
- Explorar a implementação do *Reconhecimento de Entidades* utilizando *Redes Neurais Recorrentes*;
- Implementar um sistema de *Natural Language Understanding (NLU)* para melhorar o processamento do Chatbot;
- Implementar uma arquitetura que utilize *Sequence to Sequence* e demais técnicas de *Deep Learning*.

[1] Link para a documentação do gensim: https://radimrehurek.com/gensim/

Referências Bibliográficas

[1] The 5 best programming languages for ai development. http://www.infoworld.com/article/3186599/artificial-intelligence/the-5-best-programming-languages-for-ai-development.html. Accessed: 2017-05-19.

[2] Natural language toolkit book. http://www.nltk.org/book/. Accessed: 2017-06-03.

[3] Simple tutorial on svm and parameter tuning in python and r. http://blog.hackerearth.com/simple-tutorial-svm-parameter-tuning-python-r. Accessed: 2017-06-13.

[4] Tensorflow - google's latest machine learning system, open sourced for everyone. https://research.googleblog.com/2015/11/tensorflow-googles-latest-machine_9.html. Accessed: 2017-05-19.

[5] What is artificial intelligence. http://www.aisb.org.uk/public-engagement/what-is-ai. Accessed: 2017-06-02.

[6] Bayan AbuShawar and Eric Atwell. Automatic extraction of chatbot training data from natural dialogue corpora. *Workshop on Collecting and Generating Resources for Chatbots and Conversational Agents*, 2016.

[7] API.AI. Conversational user experience platform. https://api.ai/, 2017. Accessed: 2017-06-22.

[8] Michael Ask, Julie A.; Facemire and Andrew Hogan. The state of chatbots. *Forrester*, 2016.

[9] Balbir Singh Bani and Ajay Pratap Singh. College enquiry chatbot using a.l.i.c.e. *International Journal of New Technology and Research (IJNTR)*, pages 64–65, 2017.

[10] Rollo Carpenter. About the jabberwacky ai. https://www.technologyreview.com/s/406539/how-to-be-human/. Accessed: 2017-06-18.

[11] Rollo Carpenter. German chatty bot is 'most human'. http://www.jabberwacky.com/j2presspr-J1754-German-chatty-bot-is-most, 2003. Accessed: 2017-06-18.

[12] Rollo Carpenter. Jabberwacky second in the loebner prize 2004. http://www.jabberwacky.com/j2presspr-J1743-Jabberwacky-second-in-the-Loebner, 2004. Accessed: 2017-06-18.

[13] Rollo Carpenter. I, george - jabberwacky character wins loebner prize 2005. http://www.jabberwacky.com/j2presspr-L2215-I-George-Jabberwacky-character-wins, 2005. Accessed: 2017-06-18.

[14] Chatbots.org. Chatbot jabberwacky. https://www.chatbots.org/chatterbot/jabberwacky/. Accessed: 2017-06-18.

[15] Chatbots.org. Loebner prize 2004 #1. https://www.chatbots.org/awards/winner_nomination/a.l.i.c.e6/, 2004. Accessed: 2017-06-18.

[16] Kathleen Chaykowski. More than 11,000 bots are now on facebook messenger. https://www.forbes.com/sites/kathleenchaykowski/2016/07/01/more-than-11000-bots-are-now-on-facebook-messenger/#4ca3c3414fd7, 2016. Accessed: 2017-06-19.

[17] Mark Clark. A chatbot framework. https://www.linkedin.com/pulse/chatbot-framework-mark-clark, 2016. Accessed: 2017-06-18.

[18] Dennis dos Santos Gomes. Inteligência artificial - conceitos e aplicações. *Revista Olhar Científico*, 2010.

[19] Erick Rocha Fonseca and João Lúıs G. Rosa. Mac-morpho revisited: Towards robust part-of-speech tagging.

[20] Alicebot AI Foundation. Second-in-a-row victory for famous chat robot. http://www.alicebot.org/press_releases/2001/alice-loebner-2001.html. Accessed: 2017-06-18.

[21] Erik G. Learned-Miller. Introduction to supervised learning. *University of Massachusetts, Amherst*, 2014.

[22] Hsin-Yuan Huang and Chih-Jen Lin. Linear and kernel classification: When to use which?

[23] IBM. How to build a chatbot. `https://www.ibm.com/watson/how-to-build-a-chatbot/`. Accessed: 2017-06-18.

[24] IBM100. A computer called watson. `http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/watson/`. Accessed: 2017-06-18.

[25] BI Intelligence. Messaging apps are now bigger than social networks. `http://www.businessinsider.com/the-messaging-app-report-2015-11`, 2015. Accessed: 2017-05-19.

[26] Andrew McCallum John Lafferty and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Department of Computer and Information Science, University of Pennsylvania, Philadelphia*.

[27] Letzgro. 9 reasons to build a chatbot now. `http://letzgro.net/blog/9-reasons-to-build-a-chatbot-now/`, 2017. Accessed: 2017-05-19.

[28] Linguateca. Floresta sintática. `http://www.linguateca.pt/Floresta/`. Accessed: 2017-06-15.

[29] Hugh Loebner. Home page of the loebner prize in artificial intelligence. `http://www.loebner.net/Prizef/loebner-prize.html`. Accessed: 2017-06-18.

[30] Michaelis. *Dicionário da Língua Portuguesa*. Cia. Melhoramentos, 1998.

[31] Daniel Nehme Müller. Processamento de linguagem natural. 2003.

[32] Nils J. Nilsson. *Introduction to Machine Learning*. Stanford University, 1998.

[33] F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. *Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*.

[34] Silvio do Lago Pereira. Introdução a inteligência artificial. *USP*.

[35] H.; Piskorski J.; Yangarber R. Poibeau, T.; Saggion. *Multi-source Multilingual Information Extraction and Summarization*. 2013.

[36] Cambridge University Press. Stemming and lemmatization. `https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html`, 2008. Accessed: 2017-06-04.

[37] Juan Ramos. Using tf-idf to determine word relevance in document queries. *Rutgers University*.

[38] MIT Technology Review. How to be human. `https://www.technologyreview.com/s/406539/how-to-be-human/`, 2006. Accessed: 2017-06-18.

[39] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, 1995.

[40] Cicekli Saygin and Akman. Turing test: 50 years later. 2000.

[41] scikit learn. Machine learning in python. `http://scikit-learn.org`, 2017. Accessed: 2017-06-29.

[42] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. *Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*.

[43] Bayan Abu Shawar and Eric Atwell. Chatbots: Are they really useful? 2007.

[44] Brian; Huang Ting Smith, Chris; McGuire and Gary Yang. The history of artificial intelligence. *University of Washington*, 2006.

[45] Pavel Surmenok. Chatbot architecture.

[46] Oriol Sutskever, Ilya; Vinyals and Quoc V. Le. Sequence to sequence learning with neural networks. *Google*.

[47] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning, Vol. 4, No. 4*.

[48] The Quality Software Company TIOBE. Tiobe index for june 2017. `https://www.tiobe.com/tiobe-index/python/`, 2017. Accessed: 2017-06-10.

[49] Maksim Tkachenko and Andrey Simanovsky. Named entity recognition: Exploring features. 2012.

[50] Stanford University. Stanford's machine learning course. http://cs229.stanford.edu/materials.html. Accessed: 2017-06-10.

[51] Hideto Wallace, Richard; Tomabechi and Doubly Aimless. Chatterbots go native: Considerations for an eco-system fostering the development of artificial life forms in a human world. 2003.

[52] Inc. Wit.ai. Natural language for developers. https://wit.ai/, 2017. Accessed: 2017-06-22.