UNIVERSIDADE ESTADUAL DO PIAUÍ – UESPI

CAMPUS PROF. ALEXANDRE ALVES DE OLIVEIRA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS RESENDE DE SOUSA AMARAL

PREBOT: UM *FRAMEWORK* PARA A ETAPA DE PRÉ-PROCESSAMENTO DE *CHATTERBOTS*

PARNAÍBA

2018

LUCAS RESENDE DE SOUSA AMARAL

PREBOT: UM FRAMEWORK PARA A ETAPA DE PRÉ-PROCESSAMENTO DE CHATTERBOT

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Ciência da Computação da Universidade Estadual do Piauí, campus Prof. Alexandre Alves de Oliveira, Parnaíba, Piauí, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Atila Rabelo Lopes

PARNAÍBA

2018

A485p Amaral, Lucas Resende de Sousa
Prebot : um framework para a etapa de pré-processamento de Chatterbot / Lucas Resende de Sousa Amaral. – 2018.
56 f.

Monografia (graduação) – Universidade Estadual do Piauí – UESPI, Bacharelado em Ciência da Computação, 2018.
"Orientador Prof. MSc. Átila Rabelo Lopes."

1. Chatterbots. 2. Framework. 3. Pré-Processamento. I. Título.

CDD: 005

Ficha elaborada pelo Serviço de Catalogação da Biblioteca Central da UESPI
Nayla Kedma de Carvalho Santos (Bibliotecária) CRB-3ª/1188

LUCAS RESENDE DE SOUSA AMARAL

PREBOT: UM FRAMEWORK PARA A ETAPA DE PRÉ-PROCESSAMENTO DE CHATTERBOT

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Ciência da Computação da Universidade Estadual do Piauí, campus Prof. Alexandre Alves de Oliveira, Parnaíba, Piauí, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Atila Rabelo Lopes.

Monografia Aprovada em: **27 de julho de 2018**.

BANCA EXAMINADORA:

Prof. Me. Átila Rabelo Lopes
Orientador - UESPI

Prof. Dr. Eyder Franco Sousa Rios
Avaliador - UESPI

Prof. Esp. Denylson Melo Pereira
Avaliador - IFPI

Dedico a minha família e a meus amigos que sempre me ajudaram e me apoiaram nessa jornada.

Sobretudo dedico a Deus que me incentivou em momentos espirituais e que sempre esteve comigo.

AGRADECIMENTO

Primeiramente agradeço a Deus por me dar a oportunidade de ter duas mães e dois pais, meus pais biológicos Patrícia Resende de Sousa e Miguel Arcanjo de Carvalho Amaral e meus tios Anatercia Resende de Sousa Fortes e Disraele Forte de Morais Melo. A minha irmã Lais Resende de Sousa Amaral e aos meus primos Pedro Resende Fortes e Filipe Resende Fortes, que considero irmãos.

A todos os meus amigos de turma que me ajudaram ao decorrer do curso nas mais diversas atividades e aos meus demais amigos de vida.

Ao meu orientador Atila Rabelo Lopes e a todos os demais professores que tive ao longo do curso.

“Eu acredito que às vezes são as pessoas que ninguém espera nada que fazem as coisas que ninguém imagina. ”

(Alan Turing)

RESUMO

Chatterbot são softwares que têm a habilidade de conversar com humanos em sua linguagem natural, devido a essa habilidade suas aplicações são as mais diversas possíveis. As principais áreas que se beneficiam com o uso dessas aplicações são sites de e-commerce, sistema de ensino a distância, entre outras aplicações. Muitas empresas investem em serviços de chatterbots, como por exemplo a Microsoft com o LUIS. Até o exato momento desta pesquisa não foi encontrada uma arquitetura padrão, ela varia muito de acordo com a *engine* que o programador escolheu usar em seu chatterbot. Apesar de existirem várias, a etapa de pré-processamento é comum as aplicações, tendo como objetivo preparar os dados para que a engine possa processa-los ou melhorar a performance da mesma. Durante a pesquisa realizada para a execução deste trabalho não foi encontrado um framework focado nessa etapa, cada programador, desenvolve seus métodos de maneira individual, não havendo reuso de código. Um framework pode proporcionar maior qualidade no resultado final da implementação, uma vez que seus métodos todos já foram testados e validados. Para suprir essa falta, o objetivo desta pesquisa é implementar um framework contendo os métodos mais comuns de diversas aplicações. Para alcançar esse objetivo foi realizado uma revisão na literatura em busca de aplicações exemplos, a implementação do framework foi realizada na linguagem *Python*, os métodos implementados são baseados nos exemplos encontrados na literatura, a fim de diminuir a complexidade do uso do framework, foi usado o padrão de projeto Facade. Além disso também foi usado a biblioteca NLTK para auxiliar em alguns métodos. O código desenvolvido está disponível em um repositório público, hospedado no Github, junto com a documentação de todo o projeto. Os resultados dos testes foram satisfatórios, pois foi realizada uma comparação entre os métodos implementados com os métodos que serviram de base, encontrados na literatura.

PALAVRAS-CHAVES: Chatterbots. Framework. Pré-processamento.

ABSTRACT

Chatterbot are softwares that have the ability to talk to humans in their natural language, because of this ability their applications are as diverse as possible. The main areas that benefit from the use of these applications are e-commerce sites, distance education system, among other applications. Many companies invest in chatterbots services, such as Microsoft with LUIS. Until the exact moment of this search, a default architecture was not found, it varies a lot according to the engine that the programmer chose to use in his chatterbot. Although there are several, the preprocessing stage is common applications, aiming to prepare the data so that the engine can process or improve the performance of it. During the research performed for the execution of this work was not found a framework focused on this step, each programmer, develops their methods in an individual way, without reusing code. A framework can provide greater quality in the end result of the implementation, since all its methods have already been tested and validated. To overcome this lack, the objective of this research is to implement a framework containing the most common methods of several applications. In order to achieve this goal, a review was done in the literature in search of applications examples, the implementation of the framework was carried out in the Python language, the implemented methods are based on the examples found in the literature, in order to reduce the complexity of the use of the framework, was used the Facade design pattern. In addition, the NLTK library was also used to aid in some methods. Developed code is available in a public repository, hosted on Github, along with documentation of the entire project. The results of the tests were satisfactory, as a comparison was made between the methods implemented with the basic methods found in the literature.

KEY-WORDS: Chatterbot. Framework. Preprocessing.

LISTA DE FIGURAS

LISTA DE TABLEAS

LISTA DE ABREVIATURAS E SIGLAS

AIML	Artificial Intelligence Markup Language
ALICE	Artificial Linguistic Internet Computer Entity
FAQ	Frequently Asked Questions
IA	Inteligência Artificial
IDE	Integrated Development Environment
LSMT	Long Short Term Memory
NLTK	Natural Language Toolkit
PEP	Python Enhancement Proposals
PLN	Processamento de Linguagem Natural
POS	Part Of Speech
UML	Unified Modeling Language
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Chatterbot são aplicações geralmente encontradas em sistemas *web*, onde tem por objetivo simular uma conversa com um humano em linguagem natural. Essa é uma área que vem crescendo muito nos últimos anos junto com a expansão da internet, redes sociais e o e-commerce. Tanto no meio acadêmico como no meio empresarial os robôs vêm ganhando mais espaço e diversas utilidades.

Para (LAVEN, 2018) um *Chatterbot* é um programa de Inteligência Artificial que tenta simular uma conversa digitada, com o objetivo de pelo menos, enganar temporariamente um ser humano para pensar em conversar com outra pessoa.

Muitas empresas e academias vêm investindo no desenvolvimento de chatterbots, como a IBM com o Watson e seu modulo de conversação, Microsoft com o LUIS, Google com a sua API *cloud natural language*.

Atualmente, os *chatterbots* são mais utilizados nas áreas empresariais, muitas empresas usam chatterbots em suas redes sociais como forma de automação do atendimento. Eles podem atuar em FAQ, por exemplo. Porém suas aplicações não se restringem a esse nicho, também podem ser encontradas aplicações médicas usando *chatterbots*, entre outras aplicações.

O processo de implementação de um *chatterbot* pode variar dependendo de como será a sua *engine*. A *engine* de um *chatterbot* é sua parte mais importante, ela é responsável por transformar linguagem natural em uma ação entendível por máquinas(KAR; HALDAR, 2016). Atualmente, existem 3 gerações de *chatterbots* que são baseadas na forma que suas *engine* são implementadas são elas: *pattern matching* (casamento de padrão) que corresponde a primeira geração de *chatterbots*, PLN está associado a *chatterbots* da segunda geração, e linguagens de marcação com o IAML que é utilizado amplamente na terceira geração.

O processo de desenvolvimento de *chatterbots* é algo muito variável, pois a escolha da *engine* irá influenciar muito o processo. Basicamente, existem duas arquiteturas: o modelo generativo e o modelo baseado em regras (FERRAZ; GARCIA, 2017). A primeira arquitetura é baseada em PLN onde é necessário o uso de pré-processamento como qualquer outra aplicação de PLN. Já para a segunda arquitetura não é obrigatório, porém o uso pode deixar o *chatterbot* muito mais poderoso no reconhecimento de padrões.

Apesar de serem implementações diferentes, a etapa de pré-processamento é comum nos *chatterbots*, pois ela é responsável por tratar a entrada do usuário. Essa etapa é composta por métodos. Esses métodos podem ser simples, apenas para normalizar a entrada do usuário ou complexos como funções de etiquetagem de palavras. Em *chatterbots* que usam o modelo baseado em regras, por exemplo, essa etapa aumenta o poder de reconhecimento de padrões do mesmo, tornando-o mais “inteligente”.

No que diz respeitos à etapa de pré-processamento nos *chatterbots* é que até o exato momento desta pesquisa, os métodos de pré-processamento são desenvolvidos de forma independente e individual por cada desenvolvedor, não havendo um artefato de software reutilizável que os auxilie nessa etapa do desenvolvimento.

Uma das principais técnicas de reuso de software é o *framework*, segundo (SILVA; PRICE, 2000) *frameworks* são estruturas de classes que constituem implementações incompletas que, quando estendidas produzem diversos artefatos de *software*. A grande vantagem desta abordagem é a promoção de reuso de código.

A implementação de um *framework* não é algo trivial, ao se desenvolver um *framework* deve-se atentar a criar um artefato complexo com muitos métodos e funcionalidades, porém ele deve ser simples de se integrar a projetos de terceiros.

A vantagem de se utilizar técnicas de reuso de código, como o *framework*, é que os mesmos já foram implementados, testados e validados. Sendo assim além do ganho de tempo o programador tem uma garantia maior ao que diz respeito ao funcionamento dos componentes reutilizados.

Procurando suprir a aparente falta de um *framework* para as etapas de pré-processamento no desenvolvimento de *chatterbots*, esta pesquisa tem como objetivo implementar um *framework open source* que contenha os métodos mais comuns nas etapas de pré-processamento de diversos *chatterbots* encontrados na literatura, visando cobrir essa lacuna.

Para a realização do objetivo desta pesquisa foram traçados os seguintes objetivos específicos: (i) analisar a área do domínio desta pesquisa, (ii) criar o desgin do framework, (iii) implementar o framework, (iv) construir a documentação e (v) disponibilizar a implementação.

Para realizar a implementação do *framework* foi necessário realizar a análise do domínio do mesmo, para auxiliar esta etapa foi feito uma revisão bibliográfica em busca de

trabalhos de implementação de *chatterbots* que contivessem suas etapas de pré-processamento descritas, uma vez que a abordagem escolhida para o *framework* é a *bottom-up.*

Para auxiliar a implementação foi criado um diagrama de classes e a modelagem da estrutura do projeto. Também se optou por usar o padrão de projeto *Facade*. Foi escolhido a linguagem de programação *Python* para construir o *framework*, a implementação foi baseada nas informações adquiridas nas etapas anteriores a esta. Foi criado um site para a disponibilização da documentação, a qual encontrasse no repositório do *framework*. O projeto está disponibilizado no repositório do Github.

Além do capitulo de introdução que apresenta uma contextualização geral da pesquisa, este documento segue a seguinte estrutura:

No segundo capitulo, “Referencial Teórico”, serão apresentadas as áreas que envolvem essa pesquisa, sendo elas chatterbots e o reuso de código.

No terceiro capitulo, “Materiais e Métodos”, será discorrido como foram as etapas de implementação realizadas nesta pesquisa.

No quarto capitulo, “Testes e Resultados”, serão apresentados os resultados obtidos nos testes deste projeto.

No quinto capitulo, “Considerações Finais”, mostra as conclusões e trabalhos futuros proposto para esta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capitulo irá abordar as temáticas envolvidas nesse projeto, os quais fundamentam esta pesquisa. São demonstradas diversas áreas que se beneficiam com a utilização de chatterbots, as gerações existentes, arquiteturas de chatterbots encontrado na literatura e algumas etapas de pré-processamento. Abordagem sobre reuso de software e as técnicas de implementação de *frameworks*.

2.1 *CHATTERBOT*: O QUE HÁ POR TRÁS DESSES ROBÔS DE CONVERSA

O PLN é uma subárea da IA, a qual tem por objetivo processar texto em linguagem natural como por exemplo o Português (FERRAZ; GARCIA, 2017). Muitas aplicações são fruto dessa subárea como *text mining*, *sentiment analysis*, consulta em bancos de dados usando linguagem natural, *chatterbots* e entre outras. Chatterbots são aplicações decorrente do PLN, apesar de nem todas usarem técnicas avançadas de PLN.

A definição de *chatterbot* é algo bem simples, são sistemas que têm como principal objetivo a interação com humanos em linguagem natural (PILASTRI; BREGA, 2009). Podendo também serem chamados de assistentes virtuais, *chatbot*, agentes conversacionais ou os mais sofisticados sistemas de diálogos que utilização a voz ao invés de somente texto (FERRAZ; GARCIA, 2017).

O principal objetivo de ambas as aplicações apresentadas acima é ser transparente ao usuário, dando-lhe a falsa impressão de que está conversando com outro ser humano. Essa transparência a princípio possibilita que os chatterbots sejam utilizados em qualquer área que possa introduzir um diálogo para a resolução de um problema (FERRAZ; GARCIA, 2017).

Os estudos em chatterbot iniciaram-se em 1950 com Alan Turing, onde ele fez o questionamento "*Can machines think*? " (TURING, 1950). Após anos de evolução na tecnologia, em 1966, surgiu o primeiro *chatterbot* conversável, o ELIZA (WEIZENBAUM, 1966). O principal objetivo do chatterbot era estimular o usuário a falar sobre si, o chatterbot respondia a pergunta com outra pergunta, às vezes usando pedaços da frase dita pelo usuário (JUNIOR; BARROS, 2008).

O ELIZA utilizava a técnica de casamento de padrões para poder gerar as respostas para uma determinada entrada. Essa técnica consiste em casar um conjunto de palavras-chaves a um grupo de respostas, as quais respeitam a ordem e a prioridade da chave (CASTANHO; WAZLAWICK, 2002). Como por exemplo, a palavra futebol tem uma resposta relacionada ao grupo esporte.

Um truque usado pelo autor era buscar responder à pergunta com outra pergunta, afim de deixar o usuário mais atentado a conversa (JUNIOR; BARROS, 2008). A figura 1 representa um exemplo de conversa entre um usuário e o chatterbot ELIZA.

Figura 1 - Exemplo de conversa com o chaterbot ELIZA.

Fonte: ABBAS (2017).

Este *chatterbot* utiliza técnicas da primeira geração, sendo ele o principal representante. Atualmente, existem três geração de chatterbots (ROTHERMEL; DOMINGUES, 2007).

A segunda geração de chatterbots são representados por chatterbots que utilizam técnicas de PLN (ROTHERMEL; DOMINGUES, 2007). Na literatura é comum encontrar trabalhos utilizando IA e *machine learning*, esses trabalhos abordam diferentes algoritmos de classificação. Um chatterbot que pode representar bem essa geração é o JULIA (MAULDIN, 1994).

JULIA tinha como objetivo ajudar outros usuários em um ambiente virtual, conhecido como TinyMUD (Multi-User Dungeons). Esse ambiente é composto por vários outros usuários que controlam seus personagens, que são incluídos numa rede de terminais. JULIA vive nesse ambiente ajudando os usuários mapeando cavernas e enviando mensagens (FILHO; DIAS, 2009).

Nas primeiras implantações o JULIA utilizava técnicas simples de programação, como o uso de estruturas de controle como “if - if else – else”. Posteriormente foi implementado um algoritmo de rede neural, com fins de aumentar a o desempenho do *chatterbot*. Dentro de cada nó consiste: um conjunto de padrão, respostas simples, uma lista de nós ativos e nós inativos. O usuário informa uma entrada, essa percorre os nós e o que tiver maior nível é retornado como resposta (FILHO; DIAS, 2009). Após a utilização da rede neural o *chatterbot* pode ser considerado como de segunda geração.

A terceira geração de chatterbots são os que utilizam linguagens de marcação para a construção da sua base de conhecimento (ROTHERMEL; DOMINGUES, 2007). Uma das linguagens mais citadas na literatura é o AIML. Segundo PRADANA, SING e KUMAR (2017) a maioria dos chatterbot implementados atualmente utilizam como sua base de conhecimento o AIML.

```
<aiml>
    <category>
        <pattern>Ola mundo</pattern>
        <template><srai>SAUDACAO</srai></template>
    </category>
    <category>
        <pattern>SAUDACAO</pattern>
        <template><random>
            <li>Ola estranho</li>
            <li>Ola</li>
            <li>Oin</li>
        </random>
    </category>
</aiml>
```

Figura 2 – Exemplo de código AIML.

Fonte: O Autor (2018).

O AIML é uma linguagem de marcação derivada do XML, com uma estrutura própia. É usada na contrução da base de conhecimento para *chatterbot* (PRADANA; SING; KUMAR, 2017). A linguagem possui um conjunto de tags específicas para poder funcionar corretamente. Cada tag tem uma funcionalidade especifica, as principais tags são:

a) <category> - unidade básica da linguagem e é composta por um conjunto de pattern e templates.

b) <pattern> - é a entrada do usuário, ou seja uma questão realizada ao chatterbot.

c) <template> - é a resposta associada a um <pattern>.

d) <srai> - é utilizado quando dois ou mais pattern têm as mesmas respostas, evitando criar <templates> redundantes.

A estrutura do AIML é bastante simples, por esse motivo se tornou bastante popular, outro fator que contribuiu muito para a sua disseminação, foi o fato de haver implementações do seu kernel nas mais diferentes linguagens de programações, como por exemplo: JAVA, PHP, Python, Ruby, C e C++ e entre outras linguagens (FILHO; DIAS, 2009).

O chatterbot que representa bem essa geração é o ALICE criado pelo próprio inventor do AIML. A figura 3 mostra uma breve ilustração do chatterbot ALICE:

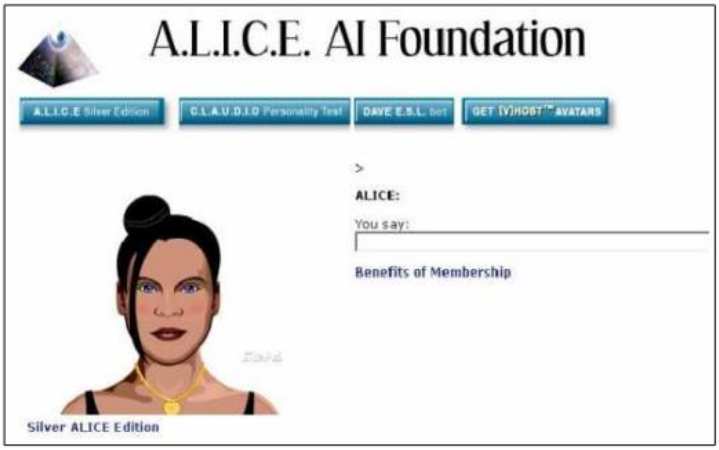

Figura 3- Ilustração do chatterbot ALICE.

Fonte: JUNIOR e BARROS (2008).

Este chatterbot se diferencia das outras gerações pois possui toda sua base de conhecimento armazenada em um arquivo AIML e também pode armazenar informações sobre a conversa podendo assim manter um diálogo mais natural (FILHO; DIAS, 2009).

Apesar de haver três gerações de chatterbots ainda não se tem uma arquitetura de referência. Segundo FERRAZ e GARCIA (2017) não há uma arquitetura definida para a criação de *chatterbots*, cada desenvolvedor escolhe o modelo que achar melhor para a sua aplicação.

A arquitetura de um *chatterbot* é baseada no domínio da conversa e no modelo de resposta. O domínio da conversa pode ser:

a) domínio aberto: uma conversa de domínio aberto o usuário pode fazer qualquer pergunta sobre qualquer tópico e esperar que o *chatterbot*

responda algo relativamente importante, é muito complexo a implementação de *chatterbots* com domínio de conversa aberto;

b) domínio fechado: uma conversa de domínio fechado em contrapartida ao de domínio aberto só é capaz de responder um conjunto de tópicos, sendo assim limitado. Mais simples de se implementar.

Já para o modelo de reposta, existem diversos tipos, mas os dois principais são: o modelo generativo e o modelo baseado em regra. A figura 4 exemplifica a relação entre modelo de resposta e domínio da conversa.

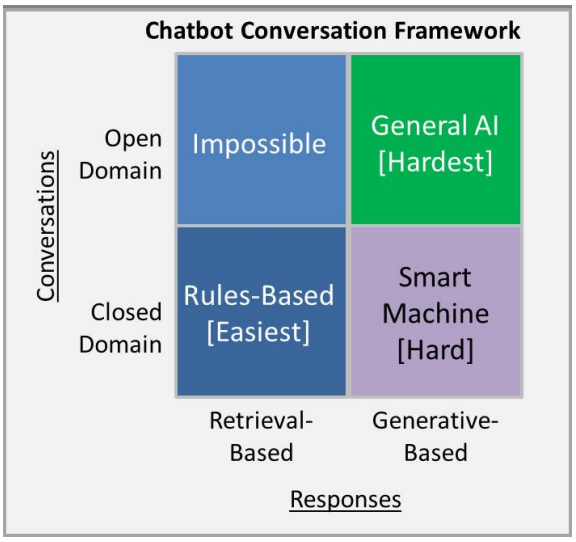

Figura 4 – Relação entre domínio da conversa e modelo de resposta.

Fonte: CLARK (2016).

O modelo generativo tem por objetivo gerar a resposta baseado no histórico da conversa e na última mensagem recebida, usando algoritmos de rede neurais recorrentes. Esse modelo requer uma boa base de treinamento, caso contrário estará sujeito a erros. Esse é um dos motivos que o torna mais complexo (FERRAZ; GARCIA, 2017).

O estado da arte no modelo generativo é a Seq2Seq ou também chamadas de encoder-decorder, são redes neurais recorrentes que utilizam duas camadas de LSTM a qual gera uma saída baseada na entrada (FERRAZ; GARCIA, 2017).

Já o modelo baseado em regras não necessita de algoritmos de inteligência artificial. É utilizado em domínios fechados e não comete erros gramaticais pois suas perguntas e respostas já são pré-definidas em um arquivo de template (FERRAZ; GARCIA, 2017). Um

exemplo de template para armazenamento de conhecimento é o AIML, citado e explicado anteriormente.

Uma das maiores problemáticas desse modelo é a construção de todos os possíveis padrões que possam ser utilizados caso contrário o chatterbot pode ter um comprometimento no seu desempenho (FERRAZ; GARCIA, 2017).

Em ambos os modelos a etapa de pré-processamento é fundamental. No modelo generativo essa etapa é responsável por preparar os dados, para que o classificador do PLN possa processar a entrada e gerar a resposta. Já nos modelos baseados em regras a etapa de pré-processamento não é obrigatória, mas ela ajuda na diminuição de padrões pois ela pode filtrar a entrada do usuário ou corrigir alguns possíveis erros que o usuário possa cometer, tornando o *chatterbot* mais poderoso. Segundo ANDRADE (2012) a utilização de módulos de pré-processamento aumenta a fluidez e coerência na conversa entre o humano e o chatterbot.

Diversos métodos de pré-processamento são encontrados na literatura, mas todos os trabalhos são implementados de forma individual não havendo reuso de código. Alguns métodos são simples de implementar, já outros são mais complexos como o POStagger que envolvem classificação gramatical das palavras. Alguns dos principais métodos são apresentados abaixo:

a) POS tagging – Consiste em etiquetar cada palavra de uma sentença após passar pelo tokenizador de palavras (FERRAZ; GARCIA, 2017). Essa etiquetagem consiste em informar a classe gramatical da palavra como por exemplo: artigo, preposição ou verbo;

b) tokenizador – Método responsável por separar todas as palavras de uma sentença em tokens;

c) spell checker – Utilizado para realizar pequenas correções gramaticais, aumentando o poder do *chatterbot*;

d) steamming – As palavras podem sofrer diversos tipos de flexão como por exemplo em número, grau e entre outros. Esse método consiste em reduzir a palavra ao seu radical;

e) normalização de texto – Conjunto de funções que tem por objetivo normalizar a entrada do usuário como por exemplo: a correção de espaços errados;

f) stopwords – Consiste em remover palavras comuns que não tem significado semântico como por exemplo: artigos (a,o,aos,os).

Essas técnicas de pré-processamento combinadas com a *engine* do *chatterbot* são responsáveis por prover o poder de "dialogar" a um software. Essa capacidade possibilita que os *chatterbots* tragam benefícios a diversas áreas, como por exemplo áreas voltadas à educação conforme apresentado no trabalho "ELEKTRA: Um Chatterbot para Uso em Ambiente Educacional" onde os autores trabalharam em um *chatterbot* focado na educação a distância. O uso de robôs de conversação na educação pode ser muito vantajoso, pelo fato de imitarem a realidade humana. Isso os tornam mais fácil de serem manuseados e compreendidos. Outra vantagem seria o fato de que os chatterbots podem redirecionar os interlocutores a bons endereços com informações relevantes e validadas pelo programador da base do conhecimento (LEONHARDT et al., 2003).

Aplicações na área da medicina, como por exemplo o chatterbot Buti que é usado na prevenção e tratamento de problemas cardiovasculares. O principal objetivo do chatterbot é acompanhar o tratamento de pessoas com problemas cardiovasculares, e incentivar crianças e adolescentes a refletirem sobre seu comportamento alimentar e sedentarismo, incentivando-os a uma vida nova mais saudável (JUNIOR; BARROS, 2008). O chatterbot está disponível no site do projeto onde usuários podem interagir com ele.

Também são bastantes utilizados na área empresarial, desde simples *chatterbots* focados em FAQ até aplicações mais complexas como auxiliadores de compras ou até mesmo *helpdesk*. Uma enorme vantagem para empresas é que os *chatterbots* podem oferecer seus serviços 24 horas por dia e 7 dias por semana, sem haver a necessidade de um humano. Um exemplo dessa aplicação encontra-se no trabalho "BANK CHAT BOT – An Intelligent Assistant System Using NLP and Machine Learning", segundo os autores KULKARNI et al. (2017) desenvolver um *chatterbot* fornecerá algumas vantagens como: informações quando for necessário, melhoria no atendimento e aumento do número de clientes.

2.2 REUSO DE *SOFTWARE*

A engenharia de *software* é uma área da engenharia que se preocupa com todos os aspectos de produção de *software* (SOMMERVILLE, 2011). Com o advento da computação as

aplicações se tornaram cada vez mais robustas e complexas, tornando assim mais difícil a sua implementação. A engenharia de *software* surgiu visando essa problemática que é gerenciar, melhorar e manter um *software* focando na qualidade.

O reuso de *software* é uma subárea da engenharia de *software* que segundo FERREIRA e FERREIRA (2010) consiste na reutilização de partes de sistemas diferentes, com ou sem alterações, com o objetivo de auxiliar o desenvolvimento de novas aplicações.

A técnica de reuso de *software* não é apensar cortar pedaços de software e colar em outro, o reuso exige que os componentes que o formam sejam implementados de forma genérica (OLIVEIRA; LOIOLA; SILVEIRA, 2011).

Apesar de não ser uma tarefa trivial o reuso tem seus benefícios, ao reusar *softwares* existentes, você pode desenvolver novos sistemas de maneira mais eficaz, com menos riscos de desenvolvimento e custos. Como o *software* reusado foi testado em outras aplicações, deve ser mais confiável que o novo *software* (SOMMERVILLE, 2011).

De acordo com a literaturas a figura 5 mostra um panorama nas técnicas de reuso de software:

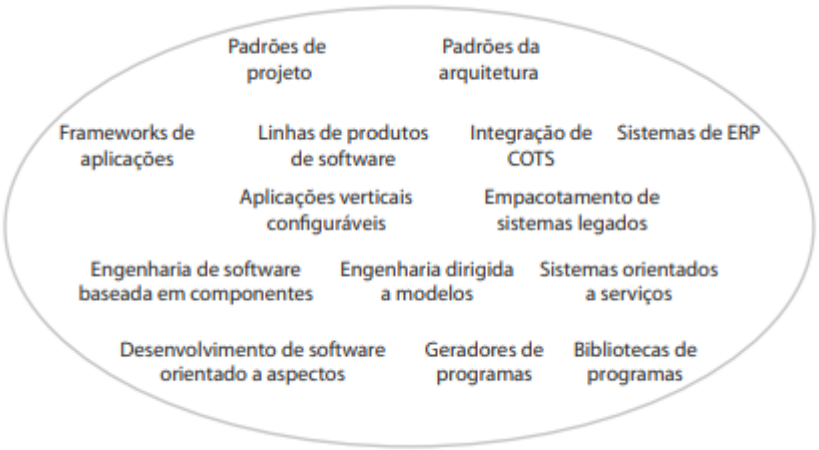

Figura 5 – Panorama nas técnicas de reuso de *software*.

Fonte: SOMMERVILLE (2011).

As três principais técnicas de reuso de código são bibliotecas de classes, componentes de *software* e *framework* (OLIVEIRA; LOIOLA; SILVEIRA, 2011). O último será mais abordado que os outros pois é o foco dessa pesquisa.

a) biblioteca de classe: É um grupo de classes que oferece um conjunto de funcionalidades, que podem estar ou não estão necessariamente

relacionadas (OLIVEIRA; LOIOLA; SILVEIRA, 2011). Como por exemplo, a linguagem JAVA possui várias bibliotecas de classe, uma delas é a java.net que é responsável por fornecer aplicações que manipulam a rede;

b) componente de *software*: são unidades derivadas da decomposição de um *software* funcional. Diferente das bibliotecas de classe, os componentes de *software* possuem funcionalidade e dependência entre as classes. Elas interagem para um propósito, abstraindo a necessidade do conhecimento interno do componente (OLIVEIRA; LOIOLA; SILVEIRA, 2011);

c) framework: é uma arquitetura desenvolvida com o objetivo de atingir o máximo possível de reutilização de *software*, representado como um conjunto de classes abstratas e concretas, que representam uma família de problemas e podem ser especializados (MATTSSON, 1996).

Segundo MALDONATO et al. (2001) citado por OLIVEIRA, LOIOLA e SILVEIRA (2011) existem dois tipos de *framework.* São eles o de caixa branca e o de caixa preta. Nos *frameworks* de caixa branca o reuso de código é a partir da herança, ou seja, o programador deve criar subclasses que herdam as classes abstratas para criar aplicações especificas. Já nos *frameworks* de caixa preta o reuso vem da composição, isto é, é necessário somente combinar classes concretas para a criação de aplicações específicas.

Já para NAKAGAWA (2013) além dos dois tipos citados anteriormente existe um terceiro, os *frameworks* de caixa cinza. Esse tipo consiste em juntar os conceitos de caixa branca com caixa preta, ou seja, o reuso é a partir de combinação e herança.

As duas principais problemáticas nos tipos de *frameworks* são: o desenvolvimento e a facilidade no uso. De acordo com o tipo do *framework* essas complexidades podem variar. É mais fácil por exemplo projetar um *framework* de caixa branca por não haver a obrigação de incluir todas as alternativas de implementação, mas para um desenvolvedor poder usar se torna mais difícil, pois não há garantia de que todas as possibilidades foram implementadas.

Essas problemáticas são inversamente proporcionais, a figura 6 exemplifica claramente a relação entre complexidade de implementação e a facilidade no uso do *framework* de acordo com o seu tipo.

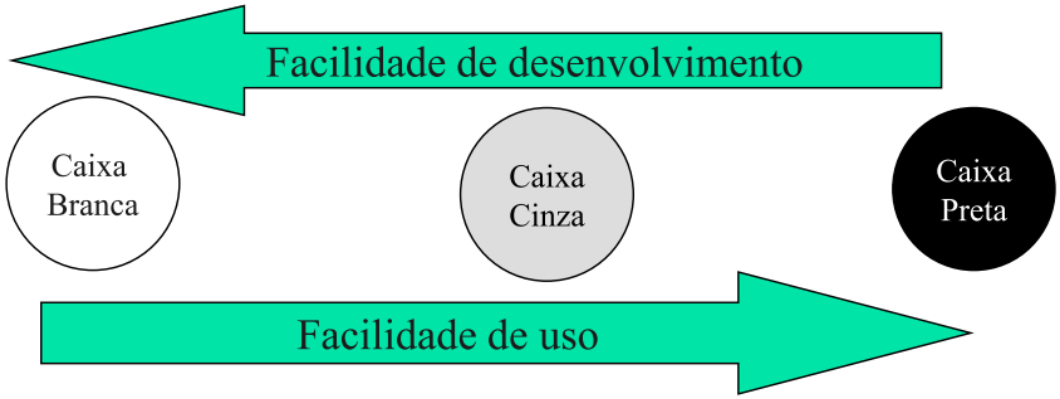

Figura 6- Relação entre tipo e complexidade

Fonte: NAKAGAWA (2013).

Apesar de haver conceitos diferente, geralmente os *frameworks* são formados por pontos de extensão que podem ser heranças ou por composição. Ao longo do processo de amadurecimento de um *framework* eles tendem a se tornarem de caixa preta. Pois a disponibilização de componentes para o *framework* com um domínio maduro tende a ser maior (OLIVEIRA; LOIOLA; SILVEIRA, 2011).

Independentemente do tipo de framework o processo de implementação do *framework* é basicamente o mesmo. Na literatura são encontradas duas abordagens possíveis. Segundo FILHO (2002) citado por OLIVEIRA, LOIOLA e SILVEIRA (2011) existem duas abordagens utilizadas no projeto de *frameworks* são elas a *top-down* e *bottom-up*. Em ambas abordagens o processo de desenvolvimento de um *framework* possui algumas etapas sistemáticas. São elas a análise do domínio, design, implementação e criação da documentação. (OLIVEIRA; LOIOLA; SILVEIRA, 2011).

Para FILHO (2002) citado por OLIVEIRA, LOIOLA, SILVEIRA (2011) a abordagem *top-down* consiste em analisar o domínio e esclarecer as partes comuns a toda uma família de aplicação. Já a abordagem *bottom-up* consiste em analisar o maior número possível de exemplos de aplicações pertencentes à mesma família.

Após a implementação do *framework* o desenvolvedor terá que implementar a documentação do mesmo, com o intuito de ensinar o público a utilizar o seu *framework.* A documentação de um framework deve atingir diferentes públicos. Basicamente são três grupos identificados:

a) o primeiro grupo é o de desenvolvedores que precisam de um framework e ainda não decidiram qual usar. A documentação deve focar em explicar as principais características e funcionalidades do *framework* (OLIVEIRA; LOIOLA; SILVEIRA, 2011);

b) o segundo grupo são os desenvolvedores que já usam o *framework*, onde é importante a documentação descrever como será utilizado a estrutura do projeto (OLIVEIRA; LOIOLA; SILVEIRA, 2011);

c) e o terceiro grupo consiste nos programadores que não apenas usam, mas também desejam implementar novas funcionalidades ao framework. A documentação deve ser rica e conter detalhes do funcionamento interno e a arquitetura do projeto (OLIVEIRA; LOIOLA; SILVEIRA, 2011).

Existem diversas técnicas de documentar informações. Algumas são de mais fácil entendimento e outras são mais especificas e técnicas. Algumas técnicas descritas por FERREIRA e FERREIRA (2010) citado por OLIVEIRA, LOIOLA e SILVEIRA (2011) são elas: manual de referência é uma descrição de cada arquivo e suas variáveis globais, constantes e tipos. Essa descrição contém informações sobre o objetivo, responsabilidade e o papel das estruturas de dados presentes nos métodos e funções. As aplicações exemplos são geradas em conjunto com o desenvolvimento do *framework* e são aplicações completas que servem para exemplificar o *framework.* O *coockbook* é um conjunto de instruções contendo o passo-a-passo sobre o desenvolvimento de aplicações, em uma linguagem mais informal e natural.

3 MATERIAIS E MÉTODOS

Este trabalho tem como objetivo a implementação de um *framework* focado na etapa de pré-processamento em *chatterbots*, contendo os métodos mais comuns. Os métodos estão divididos em 4 etapas, a saber: a etapa de análise do domínio, que consiste no levantamento de informações sobre o domínio do *framework*. O desgin do *framework* onde é levantado os requisitos do *framework*. A implementação do *framework*, será abordado a metodologia e ferramentas usadas. E por último a criação da documentação.

3.1 ANÁLISE DO DOMÍNIO

Esta pesquisa surgiu da detecção de uma problemática na área dos *chatterbots*, os trabalhos encontrados na literatura que se tratam de implementação de *chatterbots* até esse momento não usam códigos reutilizáveis quando se trata da etapa de pré-processamento. Existem diversas técnicas de reutilização de código, dentre todas a escolhida para esse projeto é o *framework*, pois a aplicação contém classes bem definidas, arquitetura e irá precisar de projeto de terceiros.

O primeiro passo para a criação de um *framework* é o domínio da área em que o mesmo será implementado. O prebot é um *framework* com abordagem *bottom-up*, que consiste em analisar exemplos de aplicações.

A fim de auxiliar a procura por exemplos foi utilizada uma base de conhecimento, de onde foi retirado artigos, trabalhos de conclusão de cursos e teses focados em implementações de *chatterbots*. Existem diversas bases de conhecimento na internet, algumas de domínio aberto ao público e outras bases são privadas. Alguns exemplos de bases de pesquisa são: Scielo e Google Acadêmico como bases de conhecimento abertas. Já o IEEE Xplore e o Scorpus são fechadas. Optou-se por usar a base de conhecimento Google Acadêmico.

O Google Acadêmico é uma ferramenta do Google específica para pesquisa acadêmica, reunindo diversas fontes em um só lugar. Permitindo a localização de trabalhos acadêmicos se eles estiverem disponíveis na web (CIRIACO, 2015).

Como em qualquer outra base de conhecimento é necessária uma *string* de busca para realizar a pesquisa. Elas são responsáveis para informar o motor de busca que o usuário deseja pesquisar. Foram utilizadas as seguintes *strings* de busca: “pré-processamento” AND “chatterbot”, “preprocessing” AND “chatbot”, “pré-processamento” AND “PLN” AND “chatterbot”.

Os critérios de exclusão usados foram os seguintes: trabalhos os quais não informaram etapas de pré-processamento e trabalhos fora do escopo da área da pesquisa. A tabela 1 representa os trabalhos selecionados depois da aplicação dos critérios de exclusão.

Tabela 1 – Artigos usados como base para a pesquisa

Trabalho	Autor
BANK CHAT BOT – An Intelligent Assistant System Using NLP and Machine Learning	KULKARNI et al. (2017)
A Conversational Agent Based on a Conceptual Interpretation of a Data Driven Semantic Space	AGOSTARO et al. (2005)
Implementação do chatterbot ELIZA na linguagem multiparadigma Oz	MITTMANN e MARIANI (2006)
Chatbot Using A Knowledge in Database	SETIAJI e WIBOWO (2016)
A Review of Technologies for Conversational Systems	MASCHE e LE (2017)
Um Chatbot para o Centro de Informática	FERRAZ e GARCIA (2017)
A Chatbot Using LSTM-based Multi-Layer Embedding for Elderly Care	SU et al. (2017)
Question Answer System for Online Feedable New Born Chatbot	ABDUL-KADER e WOODS (2017)

Fonte: O Autor (2018).

Todos os trabalhos acima tratam de implementações de *chatterbots* com etapas de pré-processamento informadas no trabalho. Os exemplos acima estão implementados nas mais diversas *engines* como por exemplo o trabalho “*Chatbot Using a Knowledge in Database: Human-to-Machine Conversation Modeling*” de SETIAJI e WIBOWO (2016) que utiliza casamento de padrão ou o artigo “*A Chatbot Using LSTM-based Multi-Layer Embedding for Elderly Care*” de SU et al. (2017) que utiliza rede neurais recorrentes como engine. Essa diversidade torna o *framework* mais generalista.

Após o levantamento dos artigos base foi realizada a análise dos artigos. Um compilado dos seus métodos de pré-processamento a fim de obter os mais relevantes e mais usados na literatura, geralmente mais de uma etapa de pré-processamento é realizada. A tabela 2 contém os métodos encontrados e suas porcentagens em relação aos trabalhos usados como base.

Tabela 2 – Métodos de pré-processamento encontrados.

Método	Porcentagem
Tokenization	62.5%
Lematização	37.5%
Remoção de caracteres especiais e pontuação	25%
Remoção de *Stop Words*	25%
Split in Sentence	25%
Correção de espaços	25%
POS tagging	25%
Corretor ortográfico	12.5%
Ngram	12.5%
Upper Case	12.5%
Bag of words	12.5%
Named entity recognition	12.5%
Semantic role labeling	12.5%
Dependency parsing	12.5%

Fonte: O autor (2018).

3.2 DESIGN DO *FRAMEWORK*

Ao término da etapa anterior iniciou-se o design do *framework*, as informações colhidas na etapa de Análise do domínio serviram como base para o desenvolvimento do design do framework.

Essa etapa consiste em documentar todas as especificações, funcionalidades, características, arquitetura do *framework* e modelagem do repositório. O objetivo dessa etapa é auxiliar a implementação e a construção da documentação do *framework*.

O prebot é um *framework* de caixa preta, ou seja, o seu reuso de código é através de componentes de *software*. Pelo fato de ser um *framework* de caixa preta o programador deve instanciar as componentes de *software* implementadas nele. Essas instanciações gera uma certa complexidade na utilização do *framework*.

Visando essa problemática foi utilizado um padrão de projeto, existem diversos padrões de projeto na literatura. Mas para o prebot foi escolhido usar o *Facade*, pois sua filosofia é muito simples, consiste em criar uma classe de "fachada" para o *framework*, e a mesma chamar os componentes de *software*, sendo assim o programador só precisa instanciar a "fachada" do *framework*.

Para auxiliar o design do framework optou-se por utilizar a UML, mais especificamente o diagrama de classe. Este diagrama monstra as informações das classes e suas relações (RIBEIRO, 2011). A figura 8 mostra a arquitetura do projeto, as classes e sua relação com a "fachada" do *framework*.

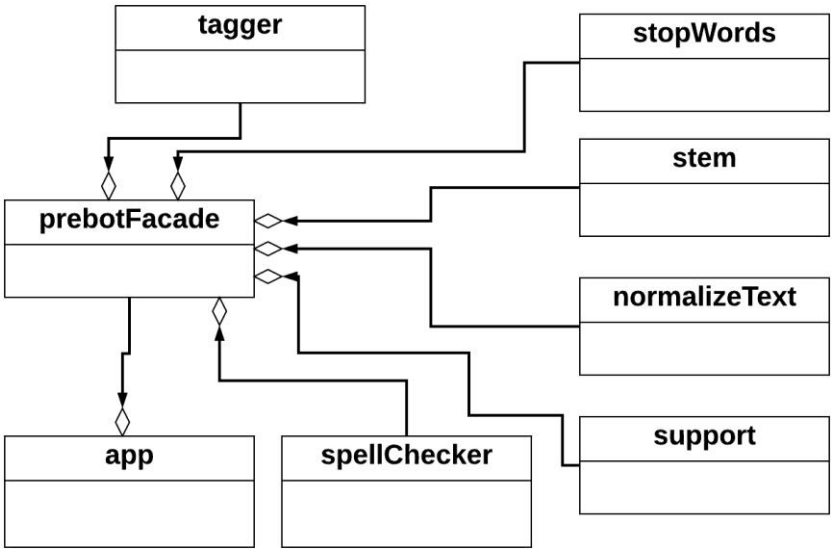

Figura 7 – Diagrama de classes

Fonte: O Autor (2018).

O framework é composto por 8 classes e suas características são descritas a seguir:

a) prebotFacade é a classe que corresponde a "fachada" do *framework*, essa classe importa todas as outras classes, sendo assim o programador quando quiser usar o *framework* só precisará importar está classe;

b) spellChecker é a classe responsável por correções ortográficas;

c) app é a classe à qual recomendasse que o programador implemente sua engine, cabe ao desenvolvedor escolher e implementar a engine que mais lhe parecer conveniente;

d) stopWords é a classe responsável pela eliminação de *stop words*, palavras comuns e que não alteram o significado semântico da frase. Como por exemplo artigos ou preposições;

e) normalizeText esta classe é responsável por métodos de normalização de texto, como por exemplo: funções de correção de espaços, funções de remoção de pontuação e caracteres especiais;

f) support é a classe que contém métodos auxiliares como funções de tokenização ou modelos de vetorização de textos;

g) tagger esta classe é responsável por métodos de POSTagger, que são amplamente utilizados em *chatterbots* que utilizam inteligência artificial;

h) stem esta classe contém métodos relacionados a redução de uma palavra ao seu radical, também é amplamente usado em *chatterbots* que utilizam inteligência artificial.

Além do diagrama de classe foi modelada também a estrutura do repositório, com o objetivo de organizar as relações entre módulos e auxiliar a escolha de um sistema de controle de versão. Decidiu-se por usar o Github, pois além de usar todas as funcionalidades do git para controle de versão, é o maior repositório git da comunidade

Até o momento desta pesquisa, não foi encontrada uma PEP que regularizasse a criação de repositórios para *frameworks* em P*ython*. Optou-se por usar um modelo próprio, baseado no modelo do REITZ (2016).

A modelagem do repositório consiste em cada pasta do repositório que contenha arquivos .py, tenha também um arquivo vazio __init__.py, isso indica para o *Python* que aquela pasta é um módulo. Já para os arquivos que utilizam classe de terceiros, ficaram em módulos separados, no caso o stem.py e tagger.py. As classes support e normalizeText por desempenhar funções semelhantes ficaram no mesmo módulo, e os demais arquivos ficaram na pasta raiz do projeto.

O diretório lang é onde localiza-se os arquivos de texto, que servem de base para os métods implementados. Esse diretório foi pensado em escalabilidade, portanto todos os arquivos TXT ficam numa pasta que se refere a sua língua. Já o diretório docs é responsável por guardar os arquivos da documentação do *framework*.

Além dos arquivos .py e dos diretórios, existem alguns arquivos que são metadados, o arquivo README.md consiste numa breve introdução ao projeto, o LICENSE é o arquivo de licença do *framework*, o mesmo está sobre a licença MIT e o arquivo requeriments.txt contém os requisitos do framework. A figura 8 representa de forma resumida e hierárquica a estrutura do repositório.

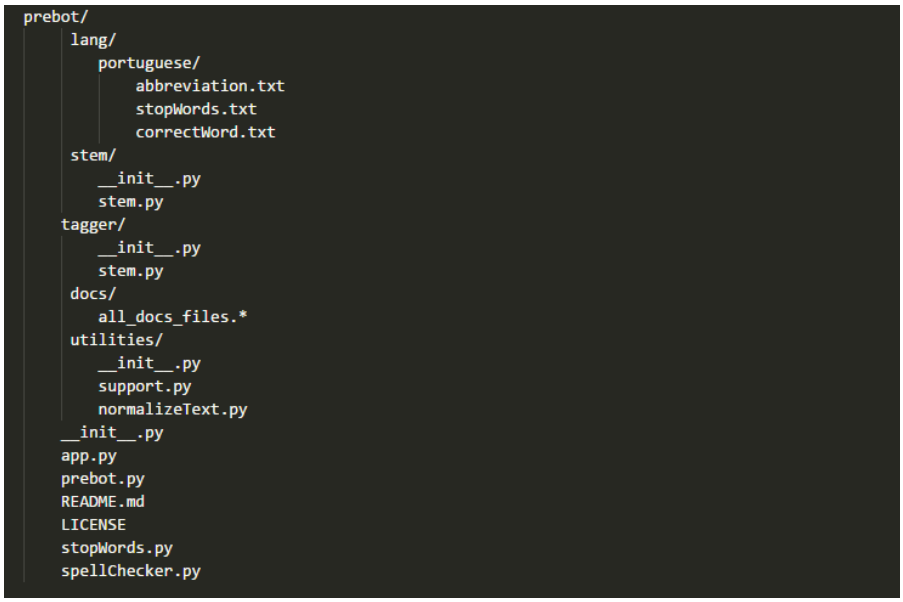

Figura 8 – Representação a nível hierárquico do repositório do framework.

Fonte: O Autor (2018).

3.3 IMPLEMENTAÇÃO DO *FRAMEWORK*

Após o termino das etapas anteriores, todos os pré-requisitos para a implementação foram alcançados. Para a construção do framework foi escolhida a linguagem *Python*, por ser uma linguagem bastante difundida na comunidade de computação. Para auxiliar o desenvolvimento optou-se por usar a IDE PyCharm na versão gratuita voltada para a comunidade.

As classes implementadas foram baseadas na análise realizada nas etapas anteriores. Como esse trabalho se baseia numa abordagem *bottom-up*, a implementação dos métodos e classes são baseados na engenharia reversa dos exemplos listados na fase de análise do domínio.

A primeira classe que foi implementada, foi a classe support pois os métodos dela servem de base para as outras classes. Alguns métodos são de simples implementação como o tokenizator, que consiste na separação de uma *string* em uma lista de tokens. Já outros são mais complexos como o método splitInPhrase.

O splitInPhrase consiste em separar um texto em sentenças. O início de uma sentença pode ser qualquer caractere, já o seu final é um sinal de pontuação. A implementação consiste em varrer o texto atrás de um delimitador, no caso, um sinal de pontuação. Após encontrar o algoritmo verifica se é um sinal correspondente a uma abreviatura, como por exemplo a palavra Defesa, que tem como abreviação Def., de acordo com a ABNT NBR 6032. Para realizar essa verificação foi criada uma lista com todas as abreviaturas da norma NBR 6032, essa lista pode ser encontrada no arquivo abbreviation.txt, se a palavra associada ao delimitador for uma abreviação o algoritmo ignora, e continua até achar um delimitador válido para o fim de sentença. Após encontrar um delimitador válido, a sentença é adicionada em uma lista, o algoritmo continua fazendo isso até o final do texto e retorna a lista com as sentenças.

A figura 9 representa a estrutura do arquivo abrevviation.txt, cada linha corresponde a uma abreviação.

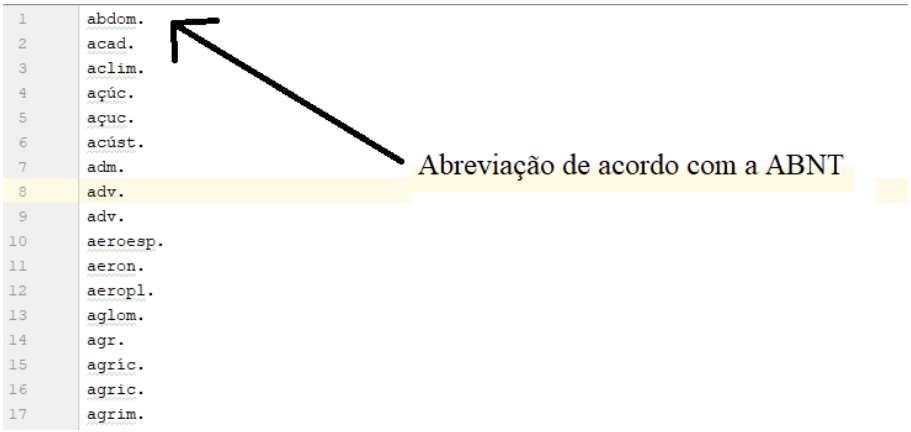

Figura 9 – Estrutura do arquivo abbreviation.txt.

Fonte: O Autor (2018).

A escolha de armazenar a lista em um arquivo TXT se deu por causa da simplicidade para editar, criar ou excluir registros, uma vez que isso é feito de maneira natural, sem dependência de instruções ou aplicações de terceiros como banco de dados.

Além dos métodos anteriores, a classe de support também conta com algoritmos de vetorização de texto sendo eles: os métodos ngram e bagOfWords. Ngram consiste em uma sequência de n itens de um texto. A chamada para o método consiste na passagem de 3 parâmetros: uma string que será processada, o n que representa a quantidade da sequência e o ultimo informa o item que compõem a sequência, podendo ser caractere ou palavras.

O método ngram inicia com uma condição, caso a *string* seja menor que o item a *string* é retornada. Uma vez que a *string* é maior que o item, o algoritmo verifica qual foi o item passado na chamada da função, caractere ou palavra. Logo após ele entra em um loop, esse loop começa no primeiro caractere da string e vai até o índice X que é calculado da seguinte forma:

$$x = len(string) - (N - 1)$$

Esse cálculo é importante para que o loop percorra a string de forma correta, sendo len(string) o tamanho da *string* menos N (informado na chamada do método) menos uma constante 1. Em cada interação do loop o algoritmo fatia a *string* na posição atual (i) até i mais N (informado na chama do método), essa fatia é armazenada numa lista, após o termino do loop o método retorna a lista. No caso de palavra ao invés do loop interar sobre os caracteres da *string* ele irá interar sobre as palavras que compõem a *string*.

Já o algoritmo de bagOfWords é mais simples, pois consiste apenas em contar a quantidade de palavras de uma *string*, o algoritmo implementado recebe uma *string*, realiza a contagem e retorna um dicionário com o resultado da contagem.

A classe tagger é responsável por fazer a etiquetagem das palavras de acordo com a classe gramatical da palavra. O método mais importante dessa classe é o taggerPhrase, a entrada dessa função é uma *string* que passa pelo tokenizador, implementado na classe de support, cada *token* é comparado a um banco de palavras, após ser identificado é adicionado uma tupla numa lista, essa tupla consiste na classe gramatical da palavra e a própria palavra, sequencialmente. Para auxiliar esse método foi escolhido usar um corpus.

Conforme MCENERY e WILSON (1996) citado por ALUÍSIO e ALMEIDA (2006) copus é um conjunto finito de enunciados que servem de análise para uma determinada língua. As características básica de um corpus são quatros sendo elas: quantidade significativa

de amostra sobre a língua alvo, um tamanho finito, formato eletrônico e um formato padrão para a variedade de língua que ele representa (MCENERY; WILSON, 1996).

Optou-se por usar o Floresta Sintá(c)tica, que cobra duas variantes do português, o português do Brasil e o português da Europa. Utilizando dois corpora de textos jornalísticos sendo eles o CETEMPúblico e o CETENFolha (FREITAS; AFONSO, 2008).

A tabela 3 a seguir corresponde as classes gramaticais e sua representação, que compõem o corpus Floresta em seus nós terminais.

Tabela 3 – Categoria gramatical.

Símbolo	Categoria
n	Nome, substantivo
prop	Nome próprio
adj	Adjetivo
n-adj	Flutuação entre substantivo e adjetivo
v-fin	Verbo finito
v-inf	Verbo Infinitivo
v-pcp	Verbo Particípio
v-ger	Verbo Gerúndio
art	Artigo
pron-pers	Pronome pessoal
pron-det	Pronome determinativo
pron-indp	Pronome independente
adv	Advérbio
num	Numeral
prp	Preposição
intj	Interjeição
conj-s	Conjunção subordinativa
conj-c	Conjunção coordenativa

Fonte: FREITAS e AFONSO (2008)

Para auxiliar a manipulação do corpus Floresta foi utilizado a biblioteca NLTK. Esta biblioteca foi escolhida para esta pesquisa pois é implementada em *Python*, mesma linguagem do *framework* proposto , contém métodos de manipulação para corpora e uma série de corpora, inclusive o Floresta.

O NLTK não foi usado só na classe tagger, a classe stem também usou métodos dessa biblioteca. A classe stem representa métodos que reduzem a palavra ao seu radical, uma palavra da língua portuguesa pode ser flexionada por número, gênero, e entre outras, contudo todas tem o mesmo significado. O objetivo de reduzir a palavra ao seu radical é aprimorar o processamento.

O método central dessa classe é o stemWord que consiste em receber uma *string* como entrada, após isso o algoritmo RSLP é usado para reduzir a *string* para o seu radical. O RLSP foi escolhido pois é um *stemming* voltado para a língua portuguesa (ORENGO; HUYCK, 2001). E também pode estar disponível na biblioteca NLTK.

Já a classe spellChecker consiste em tratar de erros gramaticais, gírias e marcas de oralidade. Tendo como método fundamental o fixPhrase, ele recebe uma *string* como entrada, essa *string* passa pelo tokenizador, após isso cada *token* é comparado a uma lista de palavras comumente escritas de maneira errada, essa lista pode ser encontrada no arquivo wordsCorrect.txt. Caso o token seja igual a palavra errada, o algoritmo troca a palavra errada pela correta que também se encontra na mesma lista e depois retorna a string.

A figura 10 representa o arquivo e sua composição em relação a posição da palavra errada e correta. O algoritmo lê linha por linha, cada linha é armazenada numa lista.

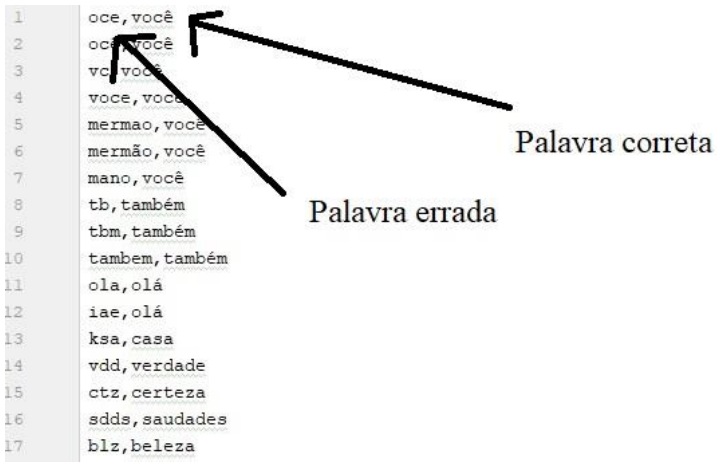

Figura 10 – Arquivo wordsCorrect.txt.

Fonte: O Autor (2018).

Apesar de arquivos TXT não serem os mais recomendados para a persistência de dados, são os mais simples pois a sua manipulação é algo natural não dependendo de instruções

ou aplicações de terceiros como banco de dados, elevando ainda mais a complexidade do *framework.*

Já a classe normalizeText consiste em métodos mais simples de implementação, focado em normalizar textos, como por exemplos funções que corrige os espaços de uma entrada, *upper case* em todas as palavras de uma string, remoção de pontuação e caracteres especiais. Foram utilizadas técnicas de expressão regular em alguns métodos.

Expressão regular consiste em identificar uma cadeia de caracteres de interesse, a partir do padrão inserido. O método removeWrongSpace é um dos que utiliza essa técnica, uma vez que o padrão desejado é \s+, onde o \s é o caractere de escape para o espaço em branco e o símbolo + procura por um ao mais em sequência. Uma vez que o algoritmo encontra esse padrão, ele o substitui por apenas um espaço, no caso \s.

A classe de stopWords consiste em métodos que removem essas palavras de uma *string*. *Stop Word* são palavras que não possuem nenhum valor semântico, apenas são úteis para o entendimento geral do texto (OLIVEIRA, 2010). O método responsável por remover essas *stop words* é o removeStopWord. A entrada desse método passa pelo tokenizador e o algoritmo verifica se o token é uma *stop word,* caso verdade ela é removida da *string.* Para auxiliar esse algoritmo foi utilizada uma *stoplist,* contendo várias *stop words* da língua portuguesa. Mais uma vez essa lista é carregada a partir de um arquivo TXT, devido a sua facilidade de modificação.

A figura 11 exemplifica como é a organização da *stoplist,* cada linha representa uma *stop word.*

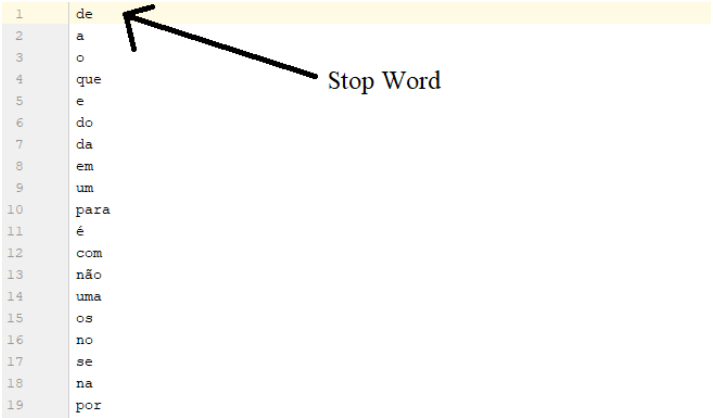

Figura 11 – Arquivo stopWords.txt.

Fonte: O Autor (2018).

A classe prebotFacade corresponde a “fachada” do *framework.* No padrão de projeto *facade*, uma classe importa todas as outras classes do sistema, sendo assim o programador só precisa instanciar a classe correspondente a “fachada”. Apesar de essa classe ser simples, o construtor da classe pode receber como parâmetro uma lista, com as classes que devem ser instanciadas, caso o programador não deseje usar todas as classes.

Por último foi implementada a classe app, essa classe consiste num esqueleto básico para usar o *framework*, recomenda-se que o programador implemente o seu *chatterbot* nela, pois ela já importa a prebotFacade, instancia e tem o código básico para o loop de conversa de *chatterbot*.

3.4 CONSTRUÇÃO DA DOCUMENTAÇÃO

A documentação é extremamente importante para um *framework,* ela deve ser simples e fácil de se encontrar. O projeto está usando o repositório do github, o mesmo contém uma funcionalidade para o gerenciamento de documentação.

O sistema do github procura por uma pasta chamada docs no repositório e cria uma url que redireciona para ela. A escolha de usar esse método se deu por causa do grande alcance que um site pode ter. Em comparação a documentações tradicionais baseadas em pdfs, os sites estão disponíveis a qualquer momento, basta que o programador tenha acesso à internet, além de ser uma mídia de maior interatividade.

Para a construção do site, utilizou-se o editor de códigos da Microsoft, o VS *Code*. O *template* usado na codificação do site foi o cube, disponível na plataforma de *templates* grátis FREEHTML5.CO (2017). O VS *Code* foi usado na adaptação do templete, as imagens e logo marca do framework, foram criadas com o auxílio da ferramenta Photoshop CS6.

O conteúdo da documentação consiste em aplicações exemplos, à medida em que o *framework* era implementado, as aplicações exemplos eram construídas. A vantagem de usar essa abordagem é que a documentação contém passo a passo de aplicações prontas, basta o programador fazer pequenas alterações para adaptar ao seu projeto.

4 TESTES E RESULTADOS

Esse capítulo aborda os resultados desta pesquisa e as discussões a respeito dos mesmos. Os testes realizados neste trabalho foram feitos da seguinte forma: uma lista de entradas, essa lista é composta por sentenças de um corpus real de *chatterbot* e a entrada dos métodos dos artigos base dessa pesquisa. Uma vez que a implementação do *framework*, foi baseada nos métodos dos trabalhos listados na etapa de análise do domínio na metodologia, para os testes estarem válidos os métodos implementados devem conter as mesmas saídas dos métodos que serviram de base para esse trabalho.

O projeto do autor COX (2017), contém corpus de várias línguas. Foi escolhido corpus da língua portuguesa para servir de entrada nos testes do *framework*.

4.1 RESULTADOS DA CLASSE SUPPORT

A classe support implementa os métodos base para as outras classes e também alguns algoritmos de vetorização textual. O método string2Token que está nesta classe, consiste em receber uma *string* e dividir em *tokens* e retorná-los numa lista. A tabela 4 demostra os resultados do método.

Tabela 4 – Teste do método string2Token

Entrada	Saída
Quais as áreas da Linguística?	['Quais', 'as', 'áreas', 'da', 'Linguística?']
Quem foi o trigésimo sétimo presidente dos Estados Unidos?	['Quem', 'foi', 'o', 'trigésimo', 'sétimo', 'presidente', 'dos', 'Estados', 'Unidos?']
Na casa do médico, todos estão doentes	['Na', 'casa', 'do', 'médico,', 'todos', 'estão', 'doentes']
Ganhei o livro O Quinze, mas queria O Sagarana	['Ganhei', 'o', 'livro', 'O', 'Quinze,', 'mas', 'queria', 'O', 'Sagarana']
ARE YOU THERE	['ARE', 'YOU', 'THERE']

Fonte: O autor (2018).

Todos os testes receberam uma string e retornaram uma lista de tokens, mesmo resultado obtido no trabalho “Implementação do chatterbot ELIZA na linguagem multiparadigma O”, o método ao receber a string “ARE YOU THERE”, retorna [“ARE”, “YOU”, “THERE”] (MITTMANN; MARIANI, 2006). O mesmo resultado foi obtido no *framework* proposto, demostrado na última linha da tabela.

O método splitInPhrase consiste em separar uma *string* em uma lista de sentenças, a tabela 5 representa os resultados obtidos nos testes, além das sentenças do corpus e o trabalho base, foram adicionadas algumas *strings* para testar melhor o potencial do método.

Tabela 5 – Teste do método splitInPhrase

Entrada	Saída
Quer bem feito, faça você mesmo!	['Quer bem feito,', 'faça você mesmo!']
E aí beleza?	['e aí beleza?']
Chamando Dr. Hans Chucrute! Chamando Dr. Hans Chucrute!	['chamando dr. hans chucrute!', 'chamando dr. hans chucrute!']
Eu gosto de maça, e você?	['eu gosto de maça,', 'e você?']
Hello, are you there?	['hello,', 'are you there?']

Fonte: O autor (2018).

O método retornou uma lista de string em todos os testes, essa lista contém as sentenças, mesmo resultado obtido no trabalho “Implementação do chatterbot ELIZA na linguagem multiparadigma O”, onde o algoritmo recebeu uma string “HELLO, ARE YOU THERE?” e retornou [‘HELLO’, ‘ARE YOU THERE’] (MITTMANN; MARIANI, 2006). A única diferença é que o autor do trabalho remove os sinais da pontuação e a implementação proposta não. A saída do método é satisfatória para o objetivo da sua implementação.

O algoritmo de ngram tem como objetivo separar uma string em n itens. A tabela 6 demostrar os resultados obtidos com os testes. O N e item representam os parâmetros passados na chamada da função.

Tabela 6 – Teste de vetorização de texto com ngram.

N	Item	Entrada	Saída
3	C	Oi	[‘oi’]

2	C	Ouviu as notícias?	['ou', 'uv', 'vi', 'iu', 'u ', ' a', 'as', 's ', ' n', 'no', 'ot', 'tí', 'íc', 'ci', 'ia', 'as', 's?']
2	W	Eu estou trabalhando em um projeto	[['Eu', 'estou'], ['estou', 'trabalhando'], ['trabalhando', 'em'], ['em', 'um'], ['um', 'projeto']]
3	W	Olá mundo!	['olá', 'mundo!']
2	C	burrungg	['bu', 'ur', 'rr', 'ru', 'un', 'ng', 'gg']

Fonte: Autor (2018).

Em todos os testes a função retornou uma lista de itens baseada no n e o item informado na chamada do método. O método também passou nos testes extremos, a primeira e quarta linhas do teste, eram para falhar pois o tamanho da *string* era menor que a interação, ocasionando um erro de *index out of range*, pois, o algoritmo tentaria acessar um índice que não tem. Porém o algoritmo trata os casos extremos.

A última linha obteve o mesmo resultado do trabalho "*Chatbot Using A Knowledge in Database Human-to-Machine Conversation Modeling*", onde os autores ao inserir a string burrungg no método de ngram tiveram a seguinte saída: {"bu", "ur", "rr", "ru", "un", "ng", "gg"} (SETIAJI; WIBOWO, 2016). Sendo assim os testes passaram.

A função bag of words é mais simples, ela retorna um dicionário contendo todas as palavras de uma *string*, onde a chave é a palavra e o valor é a quantidade de sua ocorrência. A tabela 7 contém os resultados das chamadas para este método.

Tabela 7 – Teste de vetorização de texto com bagOfWord

Entrada	Saída
Qual é o nome da grande galáxia mais próxima da Via Láctea?	{'Qual': 1, 'é': 1, 'o': 1, 'nome': 1, 'da': 2, 'grande': 1, 'galáxia': 1, 'mais': 1, 'próxima': 1, 'Via': 1, 'Láctea': 1}
Amar é sonhar, amar é viver, amar é curtir.	{'amar': 3, 'é': 3, 'sonhar': 1, 'viver': 1, 'curtir': 1}
Qual era o nome do primeiro satélite artificial da Terra?	{'Qual': 1, 'era': 1, 'o': 1, 'nome': 1, 'do': 1, 'primeiro': 1, 'satélite': 1, 'artificial': 1, 'da': 1, 'Terra': 1}

E ai como vai?	{'E': 1, 'ai': 1, 'como': 1, 'vai': 1}

Fonte: O autor (2018).

Os testes foram satisfatórios, pois em todos os casos o método retornou um dicionário onde a chave é a palavra e o valor é a quantidade da mesma na *string* que foi passada na chamada para a função.

4.2 RESULTADOS DA CLASSE NORMALIZETEXT

A classe normalizeText faz parte do módulo de utilitários do *framework.* Seu objetivo é normalizar a entrada do usuário, com métodos de correção de espaços errados, remoção de sinais de pontuação e remoção de caracteres especiais.

A função removeWrongSpaces consiste em normalizar os espaços de uma *string*, a tabela 8 apresenta os resultados obtidos nas chamadas da função.

Tabela 8 – Teste do método removeWrongSpaces

Entrada	Saída
Quem é você?	Quem é você?
O bolo é uma mentira.	O bolo é uma mentira.
Fale-me sobre você .	Fale-me sobre você.
Halo, apa kabar	Halo, apa kabar
Jarak Jauh sekali	Jarak Jauh sekali

Fonte: O autor (2018).

Em todas as chamadas para a função, o método normalizou os espaços da string, alcançando o esperado. No trabalho "*Chatbot Using a Knowledge in Database: Human-to-Machine Conversation Modeling*" o autor obteve os seguintes resultados "*Halo,apa kabar*" e "*Jarak jauh sekali*", para as frases "*Halo, apa kabar*" e "*Jarak Jauh sekali*" (SETIAJI; WIBOWO, 2016). Mesmo resultado obtido nas duas últimas linhas dos testes.

O método removePunctuation recebe uma *string* e remove todas os sinais de pontuação da mesma. A tabela 9 contém os resultados obtidos nos testes da função.

Tabela 9 – Teste do método removePunctuation

Entrada	Saída
Complexo é melhor que complicado.	Complexo é melhor que complicado
Ele não dispensa nada	Ele não dispensa nada
Quem é Marcos Bagno?	Quem é Marcos Bagno
Senin; Selasa; Rabu	Senin Selasa Rabu
Tanda Seru!!	Tanda Seru

Fonte: O autor (2018).

Em todos os testes a função recebeu uma *string* e retornou uma *string* sem os sinais de pontuação. Esse retorno condiz com o objetivo da função. O resultado dos dois últimos teste foram iguais ao dos autores SETIAJI e WIBOWO (2016), onde ao receber as *strings*, "*Senin; Selasa; Rabu*" e "*Tanda Seru*!!", a função retornava "*Senin Selasa Rabu*" e "*Tanda Seru*".

A função removeSpecialCharacter remove os caracteres especiais de uma *string*. A tabela 10 apresenta os resultados obtidos nos testes.

Tabela 10 – Teste da função removeSpecialCharacter

Entrada	Saída
O que e uma metáfora?	O que e uma metafora?
Quem é o autor de Análise de Textos de Comunicação?	Quem e o autor de Analise de Textos de Comunicacao?
O que é a parole	O que e a parole
Quem foi Saussure?	Quem foi Saussure?
Como vai?	Como vai?

Fonte: O autor (2018).

A função se comportou satisfatoriamente nos testes, uma vez que todos os caracteres especiais foram removidos, e nos casos onde não havia nada a ser modificado, a função não fez nenhuma alteração na *string*.

4.3 RESULTADOS DA CLASSE STOPWORDS

A classe de stopWords contém os métodos que estão associados a remoção de stop words. O principal método dessa classe é o removeStopWord, os demais são métodos que auxiliam essa classe, como o getStopWordFromFile, que carrega as *stopwords* a partir de um arquivo TXT.

A tabela 11 demostra os resultados obtidos nos testes da função removeStopWord, além da entrada e da saída, há também uma coluna onde são listadas as *stopwords* retiradas.

Tabela 11 – Resultados do método removeStopWords

Entrada	**Saída**	***Stopwords***
Como faço para tirar pelos da roupa preta?	Como faço tirar pelos roupa preta?	[‘para’, ‘da’]
O zíper da minha bolsa está travando, o que faço?	O zíper bolsa travando, faço?	[‘da’, ‘minha’, ‘está’ , ‘o’, ‘que’]
Golfinhos usam um sentido, semelhante ao sonar, para determinar a localização	Golfinhos usam sentido, semelhante sonar, determinar localização	[‘um’, ‘ao’, ‘para’, ‘a’]
É um prazer te conhecer.	É prazer conhecer.	[‘um’, ‘te’]
o que é a ANPOLL?	ANPOLL?	[‘o’, ‘que’, ‘é’, ‘a’]
Pode explicar o uso dos porquês?	Pode explicar uso porquês?	[‘o’, ‘dos’]
quem é Noam Chomsky?	Noam Chomsky?	[‘quem’, ‘é’]

Fonte: O autor (2018).

Stopwords são palavras sem relevância, apenas são uteis para entendimento geral do texto (OLIVEIRA, 2010). Geralmente essas palavras são comuns na língua. Todas as *stopwords* removidas nos testes estão baseados no arquivo stopWords.txt, que contém uma *stoplist.* Nos testes a *string* que foi retorna não cotinha nenhuma *stopwords*, satisfazendo os testes, uma vez que o objetivo desse método é remover *stopwords*.

4.4 RESULTADOS DA CLASSE STEM

A classe stem consiste em métodos responsáveis por reduzir as palavras ao seu radical. O principal método dessa classe é o stemWord, onde a palavra é reduzida ao seu radical através do algoritmo RSLPSteamming.

A tabela 12 contém o teste do método com diversas palavras da língua portuguesa, nas suas mais diversas flexões

Tabela 12 – Teste de stemming de palavra.

Entrada	Saída
Copiar	Copi
Copiando	Copi
Cachorro	Cachorr
Cachorrão	Cachorr
Cachorra	Cachorr
Cachorrinha	Cachorr
Morrendo	Morr
Morrer	Morr
Morreu	Morr
Morremos	Morr
Morra	Morr

Fonte: O autor (2018).

O teste foi realizado com diversas palavras, em alguns casos a mesma palavra está flexionada em diversas formas. Além disso, utilizou-se palavras de classes gramaticais diferente, como verbos e substantivos. A palavra cachorro, que é um substantivo foi flexionada em número, gênero e grau. Em todos os casos a stemming da palavra foi retornada igual, pois ambas têm o mesmo radical.

Já para o verbo morrer, que também foi flexionado, o algoritmo retornou o mesmo radical para ambos os testes, pois a palavra sempre tem o mesmo radical. Os resultados obtidos condizem com o objetivo da implementação do método.

4.5 RESULTADOS DA CLASSE POSTAGGER

Essa classe tem como objetivo implementar métodos responsáveis por fazer a etiquetagem de palavra. Essas etiquetem é dada por uma tupla, onde o primeiro elemento é a palavra e o segundo é a classe gramatical à qual aquela palavra pertence.

Os seus principais métodos são o taggerWord e taggerPhrase. O taggerWord consiste em fazer a etiquetagem de uma única palavra. A tabela 13 contém os resultados dos testes da função taggerWord.

Tabela 13 – Teste da etiquetagem da palavra.

Entrada	Saída
correr	('correr', 'v-inf')
andando	('andando', 'v-ger')
cachorro	None
onde	('onde', 'adv')
o	('o', 'art')
cadeira	('cadeira', 'n')
morrendo	None
pistola	('pistola', 'n')
para	('para', 'prp')
de	('de', 'prp')
dez	('dez', 'num')
pulou	('pulou', 'v-fin')
ela	('ela', 'pron-pers')
aquele	('aquele', 'pron-det')
isto	('isto', 'pron-indp')

Fonte: O autor (2018).

O método recebia uma string e retornava uma tupla contendo a *string* recebida e sua devida classe gramatical. Em dois casos o método retornou None, as palavras cachorro e morrendo não foram etiquetadas. Isso significa que o corpora não contém essas duas últimas palavras. Os demais casos dos testes, a função conseguiu fazer a etiquetagem correta das palavras.

O algoritmo de taggerPhrase também implementado nessa classe, usa o método anterior uma vez que seu objetivo é etiquetar uma frase toda. A tabela 14 apresenta os resultados obtidos nos testes desta função.

Tabela 14 – Teste da etiquetagem de frase.

Entrada	Saída
Qual é o seu livro favorito?	[('qual', 'pron-det'), ('é', 'v-fin'), ('o', 'art'), ('seu', 'pron-det'), ('livro', 'n'), ('favorito', 'n')]
Agora é melhor do que nunca.	[('agora', 'adv'), ('é', 'v-fin'), ('melhor', 'adj'), ('que', 'pron-indp'), ('nunca', 'adv')]
Eu te admiro por seu talento	[('eu', 'pron-pers'), ('te', 'pron-pers'), ('admiro', 'v-fin'), ('por', 'prp'), ('seu', 'pron-det'), ('talento', 'n')]

Fonte: O autor (2018).

Os testes acima receberam uma *string* na chamada da função e retornaram uma lista de tuplas, contendo a *string* e sua classe gramatical. A tabela 15 demostra a comparação do resultado obtido do framework com o trabalho "Um chatbot para o centro de informática" dos autores FERRAZ e GARCIA (2017).

Tabela 15 – Comparação dos resultados do método taggerPhrase

Resultado *framework*	Resultado trabalho
('onde', 'adv')	('onde', 'adv'),
('ser', 'v-inf'')	('ser', 'v')
('a', 'art')	('a', 'art')

(‘aula’, ‘n’)	(‘aula’, ‘n’)
(‘de’, ‘prp’)	(‘de’, ‘prep’)
(‘engenharia’, ‘n’)	(‘engenharia’, ‘n’)
(‘de’, ‘prp’)	(‘de’, ‘prep’)
(‘software’, ‘n’)	(‘software’, ‘n’)
(‘DOS’, ‘prop’)	(‘do’, ‘prep+art’)
(‘cursos’, ‘n’)	(‘cursos’, ‘n’)
(‘de’, ‘prp’)	(‘de’, 'prep’)
(‘SI’, ‘pron-pers’)	(‘SI’, ‘nprop’)

Fonte: O autor (2018).

Como o corpora do trabalho comparado é diferente do corpora usado nesse projeto, em alguns casos a tag retornada era diferente porém tinha o mesmo valor sintático. Como por exemplo, a tag da preposição no corpora do trabalho comparado corresponde como “prep” já no corpora usado no framework é “prp”. Desconsiderando os que tem nomenclatura diferente, mas o mesmo valor semântico. O método obteve 83% de acerto em relação ao comparado.

4.6 RESULTADOS DA CLASSE SPELLCHECKER

A classe spellChecker contém os métodos responsáveis por fazer a correção ortográfica nas palavras de uma *string*. O principal método dessa classe é o fixPhrase, esse método recebe uma *string* e retorna uma *string* com a correção das palavras. A tabela 16 representa os resultados obtidos nos testes.

Tabela 16 – Testes do método fixPhrase.

Entrada	Saída	Correção
Ola	Olá	Ola => Olá
Quem é vc	Quem é você	Vc => Você
Vc tem ksa	Você tem casa	Vc => Você Ksa => casa
Eu tb to com sdds	eu também estou com saudades	Tb => também To => estou Sdds => saudades

Eu não tou nem ai pra voce	Eu não estou nem ai para você	Tou => estou Você => Você

Fonte: O autor (2018).

Em todos os testes a função recebia uma *string* e retornava uma *string* com as palavras corrigidas. Essa correção potencializa chatterbots que trabalham com casamento de padrões, como por exemplo o AIML. Nos testes o algoritmo substituiu as palavras erradas, que na sua maioria eram vícios de linguagem e/ou abreviaturas usadas na linguagem da internet, pela sua escrita correta.

4.7 TESTE COM IMPLEMENTAÇÃO REAL DE UM *CHATTERBOT*

Após todos os testes de comparação, para validar a performance do framework, foram implementados dois chatterbots. Optou-se por implementar em AIML e usar o framework em um chatterbot e o outro não. A base de conhecimento dos chatterbots são iguais e se encontra no apêndice A. E as mesma entradas foram usadas em ambos os chatterbots.

A figura 12 apresenta uma pequena conversa com o chatterbot sem o framework.

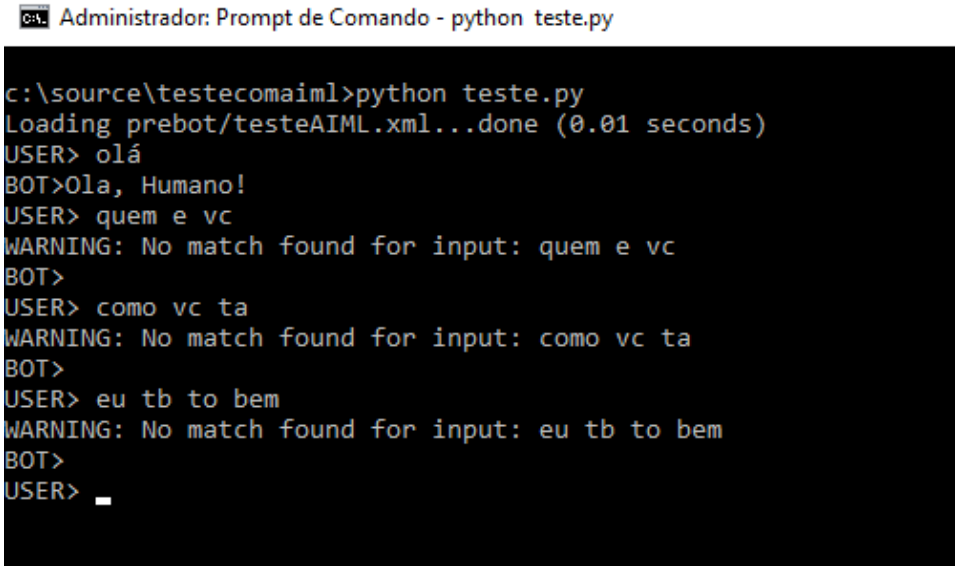

Figura 12 – Chatterbot sem o *framework*.

Fonte: o Autor (2018).

O chatterbot não conseguiu responder a alguns padrões, o AIML usa técnicas de casamento de padrão, caso ele não consiga casar a entrada do usuário com um padrão, que

consiste em um arquivo de conhecimento, ele não consegue responder. Para o chatterbot ser capaz de reconhecer esses padrões, o seu arquivo de conhecimento deve ser mais complexo, englobando todas as possíveis entradas do usuário.

A figura 13 contém o teste com o chatterbot que usa o framework proposto, foi usado o método de fixPhrase, para corrigir a entrada do usuário.

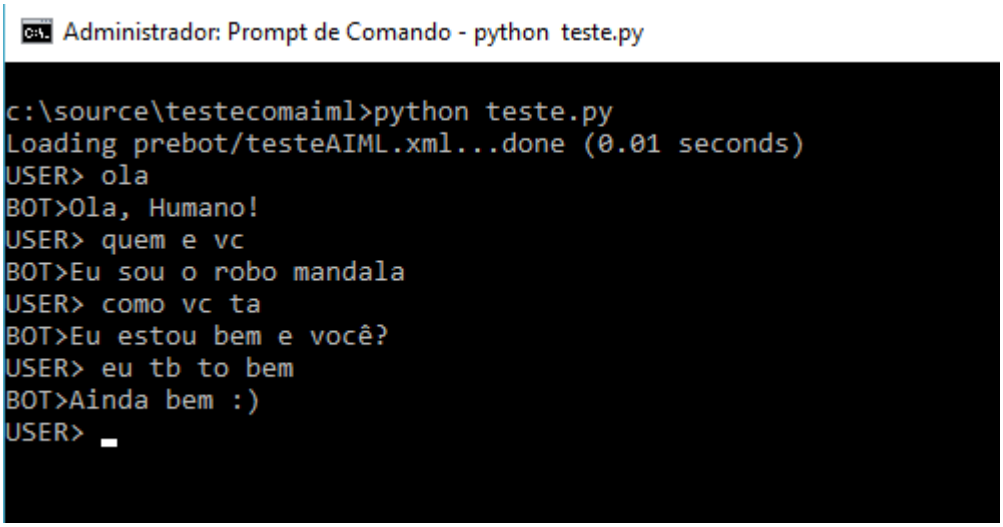

Figura 13 – Chatterbot com o *framework*.

Fonte: O Autor (2018).

Já os testes realizados com o framework, o chatterbot conseguiu responder a todas as perguntas informadas. O método fixPhrase corrigiu a entrada do usuário fazendo com que o chatterbot casasse os padrões corretamente. Uma vez que o método faz a correção da entrada do usuário o desenvolvedor não precisa se preocupar com a criação de mais padrões.

A figura 14 apresenta um *printscreen* da tela principal do site da documentação.

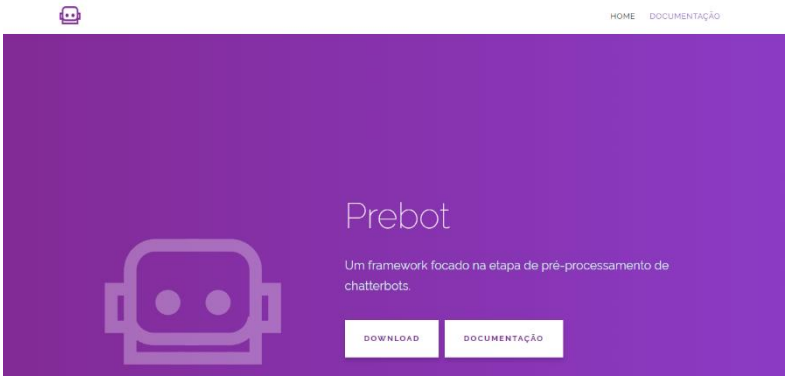

Figure 14 – Documentação do framework

Fonte: O Autor (2018).

4.8 ANÁLISE E DISCUSSÕES

A documentação do framework foi implementada na forma de um site, a qual está hospedada na página principal o repositório.

Mediante os resultados obtidos nos testes realizados, a presente pesquisa atingiu o seu objetivo, que era implementar um framework com os métodos mais comuns da etapa de pré-processamento de *chatterbot*.

A implementação foi baseada nos métodos mencionados em trabalhos da literatura. Para validar essa implementação foram comparados os resultados obtidos nos trabalhos com os métodos implementados nessa pesquisa. Essa comparação foi satisfatória, uma vez quem os testes demostraram que os retornos dos métodos eram iguais aos trabalhos comparados.

Já para testar a capacidade de ser reutilizável e sua funcionalidade foi realizado um teste funcional onde o chatterbot que continha o framework conseguiu responder mais perguntas do que o chatterbot que não tinha.

O projeto desta pesquisa está hospedado no Github, o repositório encontra-se público para que os desenvolvedores possam clonar o projeto. O está sobe a licença MIT, permitindo que terceiros possam baixar e alterar o seu conteúdo. Podendo ser encontrado no seguinte endereço: https://github.com/mandala21/prebot.

5 CONSIDERAÇÕES FINAIS

Esta pesquisa teve como objetivo a implementação de um framework focado na etapa de pré-processamento de *chatterbots*. Para alcançar seu objetivo, foi realizado uma pesquisa na literatura, cujos métodos encontrados nesta pesquisa serviram de base para a implementação. O projeto utiliza o *design pattern Facade*, foi implementado na linguagem de programação *Python* e também se utilizou de biblioteca de terceiros com o NLTK, essa ferramenta gerenciou o uso do corpus e o algoritmo de *stemming*.

O teste funcional provou que o *framework* implementado nessa pesquisa alcançou o seu objetivo, uma vez que os resultados dos métodos do framework foram comparados com os métodos de outros chatterbots encontrados na literatura, e foram iguais. Mostrou se também que é capaz de melhorar o poder de casamento de padrões dos chatterbots, pois a entrada do usuário pode conter alguns erros, que diminuem a acurácia do casamento de padrão, o método de correção de palavras, pode evita-los.

As principais contribuições deste trabalho são: incentivar novas pesquisas na área de chatterbots e difundir o conhecimento, uma vez que o projeto está disponível num repositório público e altamente difundido na comunidade científica, a lista de palavras escritas comumente errada e sua correção, a modelagem do repositório do projeto e entre outras.

Os resultados desta pesquisa poderiam ser melhores se usasse um corpus com mais palavras da língua portuguesa, para as funções de etiquetagem de palavra, a utilização de técnicas de computação paralela para otimizar a mesma função, nos testes realizado o método demostrou uma demora para retornar o resultado.

Como trabalho futuro, pretende-se implementar mais métodos de pré-processamento, a utilização de threads no método de etiquetagem de frase, e a implementação de *engines*, fazendo o *framework* ser capaz de criar chatterbots completamente, não só auxiliar a etapa de pré-processamento de dados.

REFERÊNCIAS

ABBAS, S. **History and future of chatbots**. Disponível em: <https://hashnode.com/post/history-and-future-of-chatbots-cj22u2rx5002z1m532cbx7pk5>. Acesso em: 25 maio. 2018.

ABDUL-KADER, S. A.; WOODS, D. J. Question Answer System for Online Feedable New Born Chatbot. n. September, p. 863–869, 2017.

AGOSTARO, F. et al. A Conversational Agent Based on a Conceptual Interpretation of a Data Driven Semantic Space. v. 3673, n. May, 2005.

ALUÍSIO, S. M.; ALMEIDA, G. M. D. B. O que é e como se constrói um corpus ? Lições aprendidas na compilação de vários corpora para pesquisa lingüística. **Calidoscópio**, v. 4, n. 3, p. 155–177, 2006.

ANDRADE, R. M. Mobile bot: Um chatterbot educacional para dispositivos móveis. **Revista Brasileira de Computação Aplicada**, v. 4, n. 2, p. 83–91, 2012.

BRITO, F. N.; OLIVEIRA, M. A. DE. **Desenvolvimento de um Chatterbot para a página web de um curso de nível superior**. 2017.

CASTANHO, C. L. DE O.; WAZLAWICK, R. S. A AVALIAÇÃO DO USO DE CHATTERBOTS NO ENSINO ATRAVÉS DE UMA FERRAMENTA DE AUTORIA. 2002.

CIRIACO, D. **O que é e como usar o Google Acadêmico**. Disponível em: <https://canaltech.com.br/mercado/o-que-e-e-como-usar-o-google-academico/>. Acesso em: 12 fev. 2018.

CLARK, M. **A Chatbot Framework**. Disponível em: <https://www.linkedin.com/pulse/chatbot-framework-mark-clark>. Acesso em: 29 maio. 2018.

COPPIN, B. **Inteligência Artificial**. Rio de Janeiro: LTC, 2010. v. 1

COX, G. **portuguese conversations**. Disponível em: <https://github.com/gunthercox/chatterbot-corpus/tree/master/chatterbot_corpus/data/portuguese>. Acesso em: 27 jun. 2018.

FERRAZ, V. F. S.; GARCIA, V. C. **Um chatbot para o centro de informática**. Universidade Federal de Pernabuco, 2017.

FERREIRA, L. R.; FERREIRA, F. P. Utilização do Processo de Desenvolvimento de Framework Dirigido por Hot Spot no domínio de gerenciadores de conteúdo. 2010.

FILHO, E. C. P.; DIAS, M. S. O Uso do Processamento de Linguagem Natural na Construção de Chatterbots. p. 46, 2009.

FILHO, I. M. **A documentação e instanciação de frameworks orientado a objetos**. Rio de Janeiro: Universidade Católica do Rio de Janeiro, 2002.

FREEHTML5.CO. **Cube: Free HTML5 Bootstrap Website Template**. Disponível em: <https://freehtml5.co/cube-free-html5-bootstrap-website-template/>. Acesso em: 26 jun. 2018.

FREITAS, C.; AFONSO, S. **Bíblia Florestal: Um manual lingüístico da Floresta Sintá(c)tica.** Disponível em: <https://www.linguateca.pt/Floresta/BibliaFlorestal/>. Acesso em: 17 jun. 2018.

JUNIOR, A. F. L. J.; BARROS, F. DE A. Buti : um Companheiro Virtual baseado em Computação Afetiva para Auxiliar na Antonio Fernando Lavareda Jacob Junior. p. 103, 2008.

KAR, R.; HALDAR, R. Applying Chatbots to the Internet of Things: Opportunities and Architectural Elements. **International Journal of Advanced Computer Science and Applications**, v. 7, n. 11, p. 147–154, 2016.

KULKARNI, C. S. et al. BANK CHAT BOT – An Intelligent Assistant System Using NLP and Machine Learning. 2017.

LAVEN, S. **What is a Chatterbot?** Disponível em: <http://www.simonlaven.com/>. Acesso em: 8 fev. 2018.

LEONHARDT, M. D. et al. ELEKTRA: Um Chatterbot para Uso em Ambiente Educacional Michelle. **RENOTE: Novas Tecnologias na Educação**, v. 1, n. 2, p. 1–11, 2003.

MALDONATO, J. C. et al. **Padrões e Frameworks de Software**. UNIVERSIDADE DE SÃO PAULO - INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO, 2001.

MASCHE, J.; LE, N.-T. A Review of Technologies for Conversational Systems. **Advances in**

Intelligent Systems and Computing, v. 629, n. June, 2017.

MATTSSON, M. **Object-Oriented Frameworks: A Survey of Methodological Issues**. 167. ed. Lund University, 1996.

MAULDIN, M. L. ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition. **Aaai**, p. 16–21, 1994.

MCENERY, T.; WILSON, A. Corpus linguistics. 1996.

MITTMANN, A.; MARIANI, A. C. Implementação do chatterbot ELIZA na linguagem multiparadigma Oz. 2006.

NAKAGAWA, E. Y. SSC-526 – Análise e Projeto Orientados a Objeto. 2013.

NEIGHBORS, J. Draco: A Method for Engineering Reusable Software Systems. In: **Software reusability**. v. 1p. 295–319. 1987.

OLIVEIRA, L. L. DA S.; LOIOLA, E. M.; SILVEIRA, D. S. DA. Um Framework Para Instanciação De Blogs Acessíveis Visando Os Usuários Lorena Lopes Da Silva Oliveira Um Framework Para. 2011.

OLIVEIRA, T. DA V. Adaptação do Algoritmo Levenshtein Distance para o Cálculo de Similaridade entre Frases. p. 20–21, 2010.

ORENGO, V. M.; HUYCK, C. A stemming algorithm for the Portuguese language. **Proceedings - 8th Symposium on String Processing and Information Retrieval, SPIRE 2001**, p. 186–193, 2001.

PILASTRI, A. L.; BREGA, J. R. F. Chatterbot com Interatividade ao Avatar Encapsulado no Ambiente Virtual Second Life usando a base de conhecimento em AIML. **Workshop de Realidade Virtual e Aumentada (WRVA)**, 2009.

PRADANA, A.; SING, G. O.; KUMAR, Y. J. SamBot -Intelligent Conversational Bot for Interactive Marketing with Consumer-centric Approach. **International Journal of Computer Information Systems and Industrial Management Applications**, v. 6, n. 2014, p. 2150–7988, 2017.

REITZ, K. **Structuring Your Project**. Disponível em: <http://docs.python-guide.org/en/latest/writing/structure/>. Acesso em: 25 maio. 2018.

RIBEIRO, G. F. **Os principais diagramas da UML – Resumo rápido**. Disponível em:

<https://www.profissionaisti.com.br/2011/07/os-principais-diagramas-da-uml-resumo-rapido/>. Acesso em: 30 maio. 2018.

ROTHERMEL, A.; DOMINGUES, M. J. C. DE S. MARIA : Um chatterbot desenvolvido para os estudantes da disciplina “ Métodos e Técnicas de Pesquisa em Administração ”. **SEGeT - Simpósio de Excelência em Gestão e Tecnologia**, p. 1–12, 2007.

SETIAJI, B.; WIBOWO, F. W. Chatbot Using a Knowledge in Database: Human-to-Machine Conversation Modeling. **Proceedings - International Conference on Intelligent Systems, Modelling and Simulation, ISMS**, p. 72–77, 2016.

SILVA, R. P.; PRICE, R. T. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000.

SOMMERVILLE, I. **Engenharia de Software**. 9ª edição ed. São Paulo: Addison Wesley, 2011.

SU, M.-H. et al. A chatbot using LSTM-based multi-layer embedding for elderly care. **2017 International Conference on Orange Technologies (ICOT)**, p. 70–74, 2017.

TEIXEIRA, S. et al. Chatterbots em ambientes de aprendizagem – uma proposta para a construção de bases de conhecimento. **Wie**, p. 2806–2814, 2005.

TURING, A. M. Computing machinery and intelligence. **Mind**, v. 59, p. 433–460, 1950.

WEIZENBAUM, J. ELIZA — A Computer Program For the Study of Natural Language Communication Between Man And Machine. **Communications of the ACM**, v. 9, n. 1, p. 36–44, 1966.

APÊNDECE A – BASE DE CONHECIMENTO DO CHATTERBOT

```
<aiml version="1.0">
        <!-- This category works with the Standard AIML Set -->
        <category>
            <pattern>OLÁ</pattern>
           <template>Ola, Humano!</template>
        </category>
        <category>
           <pattern>QUEM E VOCÊ</pattern>
           <template>Eu sou o robo mandala</template>
        </category>
        <category>
           <pattern>EU GOSTO DE VOCÊ</pattern>
           <template>Eu tambem gosto de você</template>
        </category>
        <category>
           <pattern>COMO VOCÊ ESTÁ</pattern>
           <template>Eu estou bem e você?</template>
        </category>
        <category>
           <pattern>EU TAMBÉM ESTOU BEM</pattern>
           <template>Ainda bem :)</template>
        </category>
</aiml>
```