



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

MARCOS FLÁVIO SILVEIRA ALVES

LAURA: UM CHATTERBOT PARA RESPONDER PERGUNTAS SOBRE JAVA

QUIXADÁ

2017

MARCOS FLÁVIO SILVEIRA ALVES

LAURA: UM CHATTERBOT PARA RESPONDER PERGUNTAS SOBRE JAVA

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Orientador: Prof. Dr. Marcos Antonio de Oliveira

Coorientadora: Prof^a Dra. Ticiane Linhares Coelho da Silva

QUIXADÁ

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A48l Alves, Marcos Flávio Silveira.

Laura: um chatterbot para responder perguntas sobre Java / Marcos Flávio Silveira Alves. – 2017.
56 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá,
Curso de Engenharia de Software, Quixadá, 2017.

Orientação: Prof. Dr. Marcos Antonio De Oliveira.

Coorientação: Profa. Dra. Ticiania Linhares Coelho da Silva.

1. Chatterbot. 2. Multiagent systems. 3. Mineração de dados (Computação). I. Título.

CDD 005.1

MARCOS FLÁVIO SILVEIRA ALVES

LAURA: UM CHATTERBOT PARA RESPONDER PERGUNTAS SOBRE JAVA

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Aprovado em: __/__/____

BANCA EXAMINADORA

Prof. Dr. Marcos Antonio de Oliveira (Orientador)
Universidade Federal do Ceará – UFC

Prof^a Dra. Ticiane Linhares Coelho da Silva (Coorientadora)
Universidade Federal do Ceará - UFC

Prof. Me. Regis Pires Magalhães
Universidade Federal do Ceará - UFC

Aos meus pais.

AGRADECIMENTOS

Agradeço imensamente à minha mãe, Sandra Maria, e ao meu pai, Júnior, por todos os sacrifícios realizados, por todo apoio, carinho e pelos conselhos durante essa difícil caminhada da minha vida. Obrigado por terem acreditado em mim. Estes sacrifícios certamente jamais serão esquecidos.

Agradeço à Jéssica Lopes, por todo carinho, amor, companherismo, por ter me escutado, pelos conselhos, por ter sempre me oferecido um ombro para descansar e por ter me dado força para continuar nesta caminhada.

Agradeço ao Professor Marcos Antonio de Oliveira pelos anos de trabalho como bolsista e orientando. Obrigado por todos os conselhos, ensinamentos, oportunidades, por ter acreditado em mim e pela ótima orientação neste e em outros trabalhos. Este aprendizado foi muito importante para a minha vida acadêmica.

Agradeço à Professora Ticiania Linhares por ter me orientado e me acompanhado neste trabalho, por todas as ajudas, dicas, críticas, pelos conselhos, ensinamentos e por toda sua gentileza.

Agradeço ao Professor Regis Pires Magalhães pela disponibilidade em participar da banca examinadora. Por todas as dicas, críticas e pela sua atenção para com o meu trabalho.

Agradeço aos meus caros amigos, Thiago Souza e Mário Júnior, pelo companherismo, pelas lutas, pela amizade e por estarem ao meu lado durante os momentos difíceis e felizes desta caminhada.

Agradeço aos meus amigos Eduardo da Silva Lopes, Gustavo Aires, Francisco Wanderson, Caio Melo, Júlio Martins e Lucas Vieira pela amizade, companherismo e por todo apoio durante esta caminhada.

Agradeço aos meus amigos e colegas da Trixlog. Em especial, ao Daniel Alves, por toda atenção, por ter acreditado em mim e por ter me dado a oportunidade de poder trabalhar com flexibilidade durante o desenvolvimento deste trabalho.

Agradeço imensamente à toda comunidade acadêmica da UFC Quixadá que me proporcionou um ambiente de aprendizagem incrível, no qual pude obter experiências, conhecimentos e lições que estarão comigo para sempre.

“Yes, there were times. I’m sure you knew. When I bit off, more than I could chew. But through it all, when there was doubt. I ate it up and spit it out. I faced it all and I stood tall and did it my way.”

(Frank Sinatra)

RESUMO

Um dos fatores de maior impacto para a saída de alunos de cursos superiores em Tecnologia da Informação é a dificuldade em aprender a programar. Dificuldade esta que pode estar associada a diversos motivos, sendo a ausência de interações síncronas um dos principais. O presente trabalho tem como objetivo apresentar uma possível solução para o problema de aprendizado de programação por meio de interações síncronas. O Chatterbot Laura foi construído para responder perguntas sobre a linguagem de programação Java. Foram realizadas técnicas de mineração de dados para a extração de conhecimento a partir do fórum de discussão online *StackOverflow* para servir como base de conhecimento da Laura. Laura foi desenvolvida com uma arquitetura Multiagente, mantendo o sistema distribuído e autônomo.

Palavras-chave: Chatterbot. Sistemas Multiagente. Clusterização.

ABSTRACT

One of the factors that have the greatest impact on the exit of higher education students in Information Technology is the difficulty in learning to program. This difficulty can be associated to several reasons, and the absence of synchronous interactions is one of the main ones. This work aims to present a possible solution to the problem of programming learning through synchronous interactions. The Laura Chatterbot was built to answer questions about Java programming. Data mining techniques were used to extract knowledge from the online discussion forum StackOverflow to serve as Laura's knowledge base. Laura was developed with a Multiagent architecture, keeping the system distributed and autonomous.

Keywords: Chatterbot. Multiagent Systems. Clustering

LISTA DE FIGURAS

Figura 1 – Exemplo de clusterização	18
Figura 2 – Um agente em seu ambiente	23
Figura 3 – Representação de um típico sistema multiagente	24
Figura 4 – Etapas do processo em cascata do <i>framework</i> proposto por Huang, Zhou e Yang (2007)	26
Figura 5 – Visão em alto nível da Arquitetura Multiagente	37
Figura 6 – Análise de Desempenho - <i>Levenshtein Distance</i>	41
Figura 7 – Análise de Desempenho - <i>Jaro-Winkler Distance</i>	41
Figura 8 – Análise de Desempenho - <i>Cosine Similarity</i>	42

LISTA DE TABELAS

Tabela 1 – Tabelas da base de dados do <i>StackOverflow</i> com suas respectivas quantidades de registros.	33
Tabela 2 – Tempo de execução dos algoritmos de clusterização.	39
Tabela 3 – <i>Clusters</i> criados - Parte 1	43
Tabela 4 – <i>Clusters</i> criados - Parte 2	44
Tabela 5 – <i>Clusters</i> criados - Parte 3	45

LISTA DE QUADROS

Quadro 1 – Comparativo entre os Trabalhos	29
---	----

SUMÁRIO

1	INTRODUÇÃO	13
2	FUNDAMENTAÇÃO TEÓRICA	16
2.1	Chatterbots	16
2.2	Clusterização	17
2.2.1	<i>K-medoids</i>	19
2.2.2	<i>Clusterização Hierárquica Aglomerativa</i>	19
2.3	Medidas de Distância	20
2.3.1	<i>Levenshtein Distance</i>	20
2.3.2	<i>Jaro–Winkler Distance</i>	21
2.3.3	<i>Cosine Similarity</i>	21
2.3.4	<i>Elbow method</i>	22
2.4	Representação de Documentos	22
2.4.1	<i>Term frequency–inverse document frequency</i>	22
2.5	Arquitetura Multiagente	23
3	TRABALHOS RELACIONADOS	26
3.1	Extracting Chatbot Knowledge from Online Discussion Forums	26
3.2	Multi-Agent Systems in a Computational Environment of Education: A Chatterbot Case Study	27
3.3	Desenvolvimento de um Chatterbot para a Página Web de um Curso de Nível Superior	28
3.4	Comparativo entre os Trabalhos	29
4	PROCEDIMENTOS METODOLÓGICOS	30
4.1	Desenvolvimento de uma Arquitetura Multiagente	30
4.2	Coleta dos Dados	30
4.3	Pré-processamento dos Dados	31
4.4	Análise de Desempenho dos Algoritmos K-Medoids x Clusterização Hierárquica Aglomerativa	31
4.5	Análise de Desempenho das Medidas de Distância	32
4.6	Clusterização das Perguntas extraídas do <i>StackOverflow</i>	32
4.7	Desenvolvimento de uma Métrica para a criação de um Ranking das Respostas	32

5	RESULTADOS E DISCUSSÕES	33
5.1	Resultados da Coleta dos Dados	33
5.2	Resultados do Pré-processamento	34
5.3	Arquitetura Multiagente Desenvolvida	36
5.3.1	<i>Agente Interface</i>	36
5.3.2	<i>Agente Analista</i>	36
5.3.3	<i>Agente Social</i>	38
5.3.4	<i>Agente Domínio</i>	38
5.3.5	<i>Agente Tensor</i>	39
5.3.6	<i>Agente Processador</i>	39
5.4	Resultados da Análise de Desempenho dos Algoritmos K-Medoids x Clusterização Hierárquica Aglomerativa	39
5.5	Resultados da Análise de Desempenho das Medidas de Distância	40
5.6	Resultados da Clusterização das Perguntas	42
5.7	Resultados da Criação e Aplicação da Métrica para o Ranking das Respostas	45
5.8	Experimentos com Laura	48
5.8.1	<i>Experimento 1</i>	48
5.8.2	<i>Experimento 2</i>	49
5.8.3	<i>Experimento 3</i>	49
5.8.4	<i>Experimento 4</i>	50
5.8.5	<i>Experimento 5</i>	51
5.8.6	<i>Experimento 6</i>	51
5.8.7	<i>Experimento 7</i>	51
6	CONSIDERAÇÕES FINAIS	53
	REFERÊNCIAS	55

1 INTRODUÇÃO

Muitos alunos, ingressantes dos cursos superiores de Tecnologia da Informação, passam por diversas dificuldades durante o processo de aprendizagem de programação (CHARLTON; LUCKIN, 2012). Segundo Ponzanelli et al. (2014), o mito do programador solitário é ainda persistente, em contraste com a realidade: desenvolvimento de software não deve ser uma atividade solitária, sendo "uma mão amiga" muitas vezes necessária. Essa "mão amiga" pode ser a ajuda de um professor, de um monitor ou até mesmo de um colega de turma.

Há algumas limitações com esta dependência, tais como: na maioria das vezes o professor só estará disponível durante o horário das aulas ou em determinados momentos para tirar dúvidas do aluno. O aluno também dependerá dos horários da monitoria para tirar suas dúvidas e da disponibilidade do monitor. Segundo Teixeira et al. (2005), a ausência das interações síncronas, presenciais ou não, é o principal fator de desestímulo dos alunos de diversos cursos.

Chatterbots ou *Chatbots* representam uma possível alternativa para o problema das limitações de disponibilidade na aprendizagem interativa. Um *Chatterbot* é um agente de conversação que interage com usuários. O objetivo principal de um *Chatterbot* é responder perguntas, que na grande maioria são de um assunto específico. Uma característica importante de um *Chatterbot* é sua disponibilidade, visto que o *Chatterbot* estará disponível a qualquer horário para interagir e responder perguntas dos usuários.

Além de responder perguntas, os *Chatterbots* devem ser capazes de interagir com as pessoas de um modo que elas se sintam confortáveis, a um ponto que a interação humano-máquina seja transparente. Para que isso seja possível, os *Chatterbots* devem processar as perguntas recebidas em linguagem natural, e respondê-las na mesma linguagem.

Eliza foi o primeiro *Chatterbot* criado, desenvolvido na década de 60 por Joseph Weizenbaum. Possuía uma pequena e estática base de dados, e tinha como objetivo principal emular uma psicoterapeuta (WEIZENBAUM, 1966). A ideia por trás de Eliza era bastante simples, e se baseava na correspondência de palavras-chave: dada uma entrada pelo usuário, Eliza verificava se esta entrada continha a presença de uma palavra-chave; caso alguma palavra-chave fosse encontrada, era gerada uma resposta associada com aquela palavra-chave (SHAWAR; ATWELL, 2007).

Desde então, muitos *Chatterbots* foram desenvolvidos, para diversas áreas e propósitos, como, por exemplo, no suporte médico, serviço empresarial, comércio eletrônico, e também em ambientes de ensino virtual. No caso de ambientes de ensino virtual, o uso de

Chatterbots simula um diálogo de tutores falando com alunos, atuando como tutores virtuais, visando reduzir o sentimento de isolamento que o aluno possa sentir em tais ambientes (BATISTA et al., 2010). O *Chatterbot* sempre estará disponível para ajudar os alunos com suas dúvidas, o que traz um sentimento de segurança e satisfação, devido ao fato de que o aluno sempre poderá realizar perguntas ao seu tutor virtual e estabelecer um diálogo de forma natural, como se estivesse conversando com algum colega ou professor.

Contudo, há algumas críticas sobre os sistemas *Chatterbot*. Para Tan, Duan e Inamura (2012), um dos maiores problemas é com a inteligência limitada e a falta de raciocínio dos mesmos. Um outro problema encontrado na maioria dos *Chatterbots* é o de recuperação de informação. A maioria dos *Chatterbots* utilizam um mecanismo de recuperação de informação por casamento de padrões, ou seja, dada uma entrada recebida, essa entrada é comparada com as informações presentes na base de conhecimento do *Chatterbot*. Caso exista alguma chave semelhante a esta entrada, é retornado um valor, ou seja, uma resposta. Isso é um problema pois não basta apenas verificar as semelhanças de sintaxe entre a consulta e a resposta a ser dada mas sim, avaliar também a semântica que a entrada possui (NIRANJAN; SAIPREETHY; KUMAR, 2012).

O problema de recuperação de informação pode ser exemplificado da seguinte maneira. Se imaginarmos a seguinte situação: as perguntas "Como percorrer uma string em Java utilizando um laço do tipo For?" e "Como ler os caracteres de uma string em Java usando For?", possuem a mesma semântica, mas não a mesma sintaxe. Para este exemplo, muitos *Chatterbots* irão classificar estas perguntas como distintas, porém, ambas possuem o mesmo significado. Este trabalho visa também contribuir para sanar este problema de classificação das perguntas recebidas.

*StackOverflow*¹ é um dos maiores fóruns do mundo sobre desenvolvimento de software e tecnologia da informação, com uma grande e ativa comunidade, possuindo uma crescente base de dados sobre diversos assuntos ligados a programação (BOSU et al., 2013). O *StackOverflow* é modular, existindo o *StackOverflow* dos Estados Unidos, da Espanha, do Brasil e entre outros países. Desta forma, há uma melhor qualidade na troca de conhecimento entre as comunidades de usuários de cada *StackOverflow*.

Neste trabalho, o *Chatterbot* Laura extrai seus conhecimentos a partir das postagens sobre programação Java do fórum online *StackOverflow* destinado a comunidade brasileira. Isso

¹ <https://pt.stackoverflow.com/>

se deve ao fato de que, a linguagem Java é uma das linguagens mais utilizadas nos cursos de Tecnologia da Informação.

Laura possui uma arquitetura Multiagente, em que, cada módulo principal será representado como um sistema multiagente, em que os agentes presentes no sistema terão responsabilidades específicas. E, por meio da troca de mensagens, irão interagir para atingir o objetivo principal de Laura: responder dúvidas sobre programação Java. A arquitetura multiagente, deve prover um sistema distribuído, modular e totalmente autônomo. A autonomia é importante para o presente trabalho, visto que, os processos e tarefas realizadas por Laura devem ser executados periodicamente sem a necessidade de supervisão.

Laura faz uso da técnica de clusterização, que consiste em agrupar dados baseados em uma função de similaridade. A cada nova entrada de uma pergunta feita pelo usuário, é feito o cálculo de similaridade com as perguntas representantes dos clusters já existentes na base de conhecimento de Laura, atribuindo-se um cluster exclusivo à pergunta recebida, com suas respectivas perguntas similares (LEISCH, 2006). Perguntas são similares quando, após todo o processo de clusterização, elas possuem um valor de similaridade elevado.

Um ranking de todas as respostas presentes na base de conhecimento de Laura foi criado a partir do desenvolvimento de métricas baseadas nas regras adaptadas do framework proposto por Huang, Zhou e Yang (2007) para eleger a melhor resposta. Com isso, Laura responde a pergunta recebida com as respostas mais bem posicionadas que respondam esta pergunta, garantindo assim, a qualidade da informação.

O restante deste trabalho está organizado da seguinte maneira. Na Seção 2 é apresentada a fundamentação teórica do trabalho. A Seção 3 apresenta os trabalhos relacionados. A Seção 4 apresenta os procedimentos metodológicos realizados para o desenvolvimento de Laura. A Seção 5 apresenta o desenvolvimento e os resultados do trabalho. Por fim, a Seção 6 apresenta as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos utilizados neste trabalho. A Seção 2.1 define e introduz conceitos sobre *Chatterbots*. Na Seção 2.2 é apresentado o conceito de clusterização. Na Seção 2.3 é apresentada a definição e os conceitos sobre medidas de distância. Na Seção 2.4 é apresentada os conceitos de representação de documentos. Por fim, a Seção 2.5 introduz os conceitos de arquitetura multiagente.

2.1 Chatterbots

Um *Chatterbot* é um agente de conversação que possui o objetivo de interagir com usuários em um determinado domínio, utilizando de linguagem natural para se comunicar. Na maioria das vezes, um *Chatterbot* serve para responder perguntas de um determinado usuário, simulando uma pessoa. Segundo Teixeira et al. (2005), o objetivo é responder as perguntas de tal forma que as pessoas tenham a impressão de estar conversando com outra pessoa e não com um programa de computador.

De acordo com Huang, Zhou e Yang (2007), muitos *Chatterbots* foram implantados na internet com a finalidade de buscar informações, fornecer orientação em sites e responder às perguntas frequentes, em um domínio estritamente limitado. Ultimamente, além destas típicas tarefas, os *Chatterbots* vêm sendo utilizados também na área da educação à distância, recebendo o papel de tutores virtuais.

O primeiro sistema *Chatterbot* criado foi ELIZA (WEIZENBAUM, 1966). ELIZA foi desenvolvido por Joseph Weizenbaum em 1966, é um dos primeiros programas com inteligência artificial. Todo seu código possuía 204 linhas escritas na linguagem de programação SLIP (*Symmetric List Processor*), e foi feita para simular uma psicoterapeuta que fornecia consultas aos usuários (WEIZENBAUM, 1966). Com uma ideia simples, ELIZA se relacionava bem com o usuário, porém não tinha a capacidade de recordar trechos passados, ou seja, ELIZA não guardava informações obtidas nas interações com seus usuários. Segue um exemplo de interação com o *Chatterbot* ELIZA:

- ELIZA: *Hello, let's talk.* (Olá, vamos conversar.)
- Usuário: *Hi.* (Oi.)
- ELIZA: *How are you today.. What would you like to discuss?* (Como você está hoje ... O que você gostaria de discutir?)

- Usuário: *I am sad today.* (Estou triste hoje.)
- ELIZA: *How do you feel about being sad?.* (Como você se sente sobre estar triste?)
- Usuário: *I feel awful.* (Me sinto horrível.)
- ELIZA: *Tell me more about such feelings.* (Conte-me mais sobre tais sentimentos.)
- Usuário: *I can't explain. I feel so lonely.* (Não posso explicar. Eu me sinto tão sozinho.)
- ELIZA: *Do you believe it is normal to be so lonely?* (Você acredita que é normal estar tão solitário?)
- Usuário: *I do not know.* (Eu não sei.)
- ELIZA: *Talk to me.. I can help you!* (Fale comigo.. Eu posso te ajudar!)

Com a leitura do exemplo de interação com ELIZA acima, pode-se notar que ELIZA possuía uma maneira gentil e atenciosa de se comunicar com os usuários.

A maioria dos *Chatterbots* existentes consistem em módulos de gerenciamento de diálogo para controlar o processo de conversação e possuem bases de conhecimento para responder à entrada recebida pelo usuário (HUANG; ZHOU; YANG, 2007). As bases de conhecimento dos *Chatterbots* são tipicamente implementadas possuindo um conjunto de modelos que correspondem às entradas do usuário e, a partir dessas entradas, geram respostas.

Por diversas vezes o que define o quão bem um sistema *Chatterbot* se comunica com seus usuários, além de uma ampla base de conhecimento, é como a informação é obtida a partir dessa base de conhecimento. Sendo a construção da base de conhecimento de um *Chatterbot*, um dos principais desafios para os desenvolvedores.

É importante entender a intenção do usuário ao fazer uma pergunta: seja ela sobre o nome de um local, sobre uma pessoa ou uma organização (LEE et al., 2001). Para isso, é necessário que técnicas como a síntese de recuperação de informações, processamento de linguagem natural e aprendizado de máquina sejam realizadas (BRITO, 2017). No presente trabalho, será criado um *Chatterbot* para responder dúvidas sobre programação Java.

2.2 Clusterização

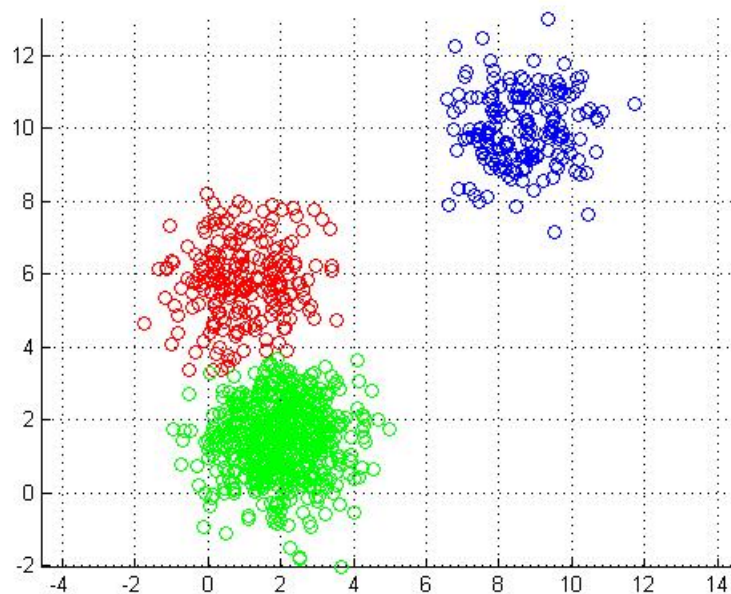
Clusterização é uma das diversas técnicas existentes de Mineração de dados. Segundo Arockiam, Baskar e Jeyasimman (2012), clusterização é o processo de agrupar um conjunto de objetos físicos ou abstratos em classes de objetos que são similares entre si. De uma forma geral, obter a solução para um problema de clusterização corresponde ao processo de agrupar os elementos (objetos) de uma base de dados (conjunto) de tal forma que os grupos formados, ou

clusters, representem uma configuração em que cada elemento possua uma maior similaridade com qualquer elemento do mesmo *cluster* do que com elementos de outros *clusters* (OCHI; DIAS; SOARES, 2004).

A clusterização é um processo de aprendizagem não supervisionada. Neste tipo de aprendizagem não existe uma categorização prévia dos dados, sendo o algoritmo de clusterização responsável por descobrir os padrões, observações, itens ou características dos dados. Segundo Jain, Murty e Flynn (1999), a clusterização é um dos primeiros passos na análise de mineração de dados, pois identifica *clusters* de registros relacionados que podem ser usados como ponto de partida para explorar possíveis relacionamentos adicionais.

A Figura 1 exemplifica a aplicação da técnica de clusterização em um determinado conjunto de dados.

Figura 1 – Exemplo de clusterização



Fonte – Jain, Murty e Flynn (1999)

A Figura 1 mostra os *clusters* que foram criados após ser realizado o processo de clusterização, sendo 3 *clusters* ao todo. Como mostrado acima, cada *cluster* possui objetos que têm cores semelhantes entre si. Pode-se perceber que a clusterização pode ser útil em várias situações exploratórias de análise de padrões, classificação de dados entre outras atividades. No presente trabalho, para a etapa de clusterização, será utilizado um algoritmo de clusterização para agrupar as perguntas, de domínio da linguagem Java, extraídas do fórum online *StackOverflow*.

2.2.1 *K-medoids*

O algoritmo *K-medoids* é um dos vários algoritmos de clusterização, sendo intrinsecamente relacionado ao algoritmo *K-means* Jain (2010), e ao algoritmo *MedoidShift* Sheikh, Khan e Kanade (2007). O algoritmo *K-medoids* é particional, ou seja, divide o conjunto de dados em grupos distintos, tentando minimizar a distância entre os objetos, formando *clusters* e definindo objetos como representantes destes *clusters*. O algoritmo *K-medoids* utiliza uma medida de similaridade (definida na Seção 2.3) para definir a distância entre os objetos.

A estratégia básica do algoritmo *K-medoids* é encontrar *K clusters* em *n* objetos. Primeiro, o valor de *K* é definido previamente e será a quantidade de *clusters* a serem criados. Posteriormente, um objeto representativo é selecionado aleatoriamente, chamado de *medoid*, para cada *cluster*. Um *medoid* pode ser definido como o objeto de um determinado *cluster* cuja dissimilaridade média para todos os objetos no *cluster* é mínima. Isto é, é o ponto mais localizado centralmente no *cluster*. Cada objeto restante no conjunto de dados é clusterizado com o *medoid* ao qual é o mais semelhante, ou seja, o *medoid* que possui o maior valor de similaridade com este objeto. Assim, o método de particionamento pode ainda ser realizado com base no princípio de minimizar a soma das dissimilaridades entre cada objeto e seu ponto de referência (*medoid*) correspondente (VELMURUGAN; SANTHANAM, 2010).

O algoritmo *K-medoids* é indicado para solucionar problemas de clusterização em que o valor médio dos objetos em um *cluster* não seja o melhor ponto de referência para aquele *cluster*, uma vez que um objeto com um grande valor pode distorcer substancialmente a distribuição de dados (VELMURUGAN; SANTHANAM, 2010).

2.2.2 *Clusterização Hierárquica Aglomerativa*

Algoritmos particionais, como o algoritmo *K-medoids* (Seção 2.2.1), baseiam-se na classificação de um número inicial de grupos e na realocação iterativa de objetos entre estes grupos para a convergência. Em contraste, algoritmos de clusterização hierárquicos combinam ou dividem grupos existentes, criando uma estrutura hierárquica que reflete a ordem em que os grupos são mesclados ou divididos (ABBAS, 2008).

Existem duas estratégias para realizar uma clusterização hierárquica:

- Aglomerativa
 - Cada objeto inicia em seu próprio *cluster*, e os pares de *clusters* são mesclados à

medida em que se move para cima na hierárquia.

- Divisiva
 - Todos os objetos iniciam em um mesmo *cluster*, e as divisões são executadas de forma recursiva à medida em que se move para baixo na hierárquia.

Segundo Abbas (2008), as vantagens em se utilizar um algoritmo de clusterização hierárquica são:

- Flexibilidade integrada em relação a um nível de granularidade.
- Facilidade de tratamento de quaisquer medidas de semelhança ou distância.
- Aplicabilidade a qualquer tipo de atributo
- Versatilidade

Clusterização hierárquica aglomerativa é bastante utilizada nos cenários de clusterização de documentos. Em que, a abordagem de criar *clusters* de baixo para cima, a partir de cada documento, resulta em uma informação bem estruturada em hierarquia destes documentos. Possibilitando uma análise geral de todos estes documentos (ROKACH; MAIMON, 2005).

No presente trabalho, será realizada uma análise de desempenho entre os algoritmos de clusterização hierárquica aglomerativa e *K-medoids* (descrito na Seção 2.2.1) a fim de escolher o melhor algoritmo de clusterização para o presente trabalho.

2.3 Medidas de Distância

Antes de realizar a clusterização, deve-se determinar uma medida de semelhança / distância. A medida reflete o grau de proximidade ou separação dos objetos-alvo e deve corresponder as características que se acredita para distinguir os clusters. Em muitos casos, essas características dependem dos dados ou do contexto do problema em mão, e não há nenhuma medida que seja universalmente melhor para todos tipos de problemas de agrupamento (HUANG, 2008). Uma medida de distância válida deve ser simétrica (ROKACH; MAIMON, 2005).

2.3.1 *Levenshtein Distance*

A distância de Levenshtein permite a quantificação de várias operações de edição de texto, incluindo a supressão, substituição e exclusão de símbolos individuais. A distância de *Levenshtein* (*Levenshtein Distance*), popularmente conhecida por *EditDistance*, pode ser

definida como o custo mínimo de transformar uma sequência de caracteres em outra através de uma sequência de operações de edição ponderadas. A distância de *Levenshtein* é bastante utilizada para o reconhecimento de padrões e também para aplicações que necessitam determinar o quão duas cadeias de caracteres são semelhantes, como os corretores ortográficos.

2.3.2 *Jaro–Winkler Distance*

Jaro–Winkler distance é uma medida utilizada para medir a distância de edição entre duas sequências de caracteres (STOILLOS; STAMOU; KOLLIAS, 2005). É uma variante proposta em 1990 por William E. Winkler da medida *Jaro Distance* criada por Matthew A. Jaro em 1989. A medida *Jaro Distance* calcula o número mínimo de transposições de um único caractere necessárias para transformar uma palavra em outra. Quanto menor o valor da distância de Jaro–Winkler para duas sequências de caracteres, mais semelhantes são as sequências.

A medida *Jaro–Winkler Distance* é definida pela fórmula a seguir:

$$d = \frac{m}{3a} + \frac{m}{3b} + \frac{m-t}{3m}$$

Em que:

- m é o número de correlações entre caracteres.
- a e b são os tamanhos do primeiro e do segundo texto, respectivamente.
- t é a quantidade de transposições.
- d é o resultado da similaridade.

2.3.3 *Cosine Similarity*

Quando documentos são representados como vetores de termos, a semelhança entre dois documentos corresponde à correlação entre os seus vetores. Isso é quantificado com o cosseno do ângulo entre os vetores. Esta técnica é chamada de *Cosine Similarity*. A semelhança dos cossenos é uma das medidas mais populares aplicada a documentos de texto (HUANG, 2008). Abaixo é demonstrada a fórmula do cálculo da *Cosine Similarity*:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$$

Em que:

- \mathbf{x} e \mathbf{y} são vetores m dimensionais sobre o conjunto $T = \{t_1, t_2, \dots, t_n\}$ de termos.
- Cada dimensão representa um termo com seu respectivo peso no documento, o que não é negativo. Como resultado, o valor da *Cosine Similarity* não é negativo e está limitado entre $[0,1]$.

2.3.4 *Elbow method*

Elbow method em português significa "Método do cotovelo" é o método mais antigo para determinar o número real de *clusters* em um conjunto de dados. A ideia é que se comece com $K = 2$, e continue aumentando a cada etapa em 1, calculando seus *clusters* e o custo que vem com o treinamento. Em algum momento, com algum valor K , o custo cai drasticamente, e depois disso atinge um platô quando o valor de K é aumentado ainda mais. Este é o valor de K que você deseja (KODINARIYA; MAKWANA, 2013).

Neste trabalho, o *Elbow method* será aplicado ao algoritmo de clusterização escolhido, juntamente com as medidas de distância: *Levenshtein Distance*(Seção 2.3.1), *Jaro-Winkler Distance*(Seção 2.3.1) e *Cosine Similarity*(Seção 2.3.3) como fator principal da análise de desempenho destas medidas, a fim de se escolher a melhor medida de distância para o presente trabalho.

2.4 Representação de Documentos

Existem várias maneiras de modelar um documento de texto. Por exemplo, o documento pode ser representado como um "saco" de palavras (do inglês, *bag of words*), onde as palavras aparecem de forma independente e a ordem é imaterial. O modelo "saco" de palavras é amplamente utilizado na recuperação de informação e extração de texto. As palavras são contadas no "saco", que difere da definição matemática de conjunto. Cada palavra corresponde a uma dimensão nos dados resultantes e cada documento torna-se então um vetor consistindo de valores não negativos (HUANG, 2008).

2.4.1 *Term frequency–inverse document frequency*

Term frequency–inverse document frequency (tf-idf) é uma medida que tem como objetivo principal indicar a importância de uma determinada palavra em um documento de texto, levando em consideração uma coleção de outros documentos. É frequentemente utilizada como fator de ponderação na recuperação de informações e na mineração de textos. O valor *tf-idf* de uma palavra aumenta proporcionalmente de acordo com o aumento das ocorrências desta palavra no documento de texto. Contudo, esse valor é equilibrado pela frequência da palavra no documento de texto. Isso auxilia a distinguir o fato da ocorrência de algumas palavras serem geralmente mais comuns que outras.

A frequência (*tf*) de um termo *t* em um dado documento *d* é calculada da seguinte maneira:

$$TF(t, d) = \frac{N(t)}{p}$$

Em que:

- $N(t)$ é o número de vezes em que o termo *t* aparece no documento *d*
- p é a quantidade total de termos no documento *d*

A frequência de documentos inversa (*idf*) de um termo *t* para um conjunto de documentos *D* é calculada da seguinte maneira:

$$IDF(t, D) = \log\left(\frac{G}{K(t)}\right)$$

Em que:

- G é a quantidade total de documentos
- $K(t)$ é a quantidade de documentos que possuem o termo *t*

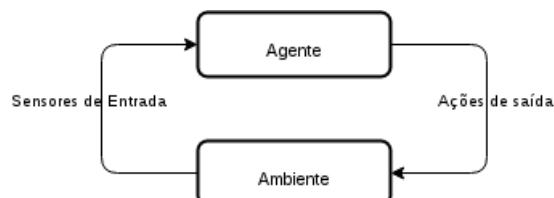
Logo, o valor *tf-idf* de um termo *t* em um documento *d* de um conjunto de documentos *D* pode ser calculado como:

$$TFIDF(t, d, D) = TF(t, d) * IDF(t, D)$$

2.5 Arquitetura Multiagente

Para que os conceitos de arquitetura multiagente fiquem claros, serão apresentados os conceitos e definições de agentes inteligentes e do paradigma multiagente. Um agente é um sistema computacional que está situado em um ambiente capaz de realizar ações de forma autônoma / independente com o intuito de atingir seus objetivos (WOOLDRIDGE, 2009). A Figura 2 representa uma visão abstrata de um agente.

Figura 2 – Um agente em seu ambiente



Fonte – Wooldridge (2009)

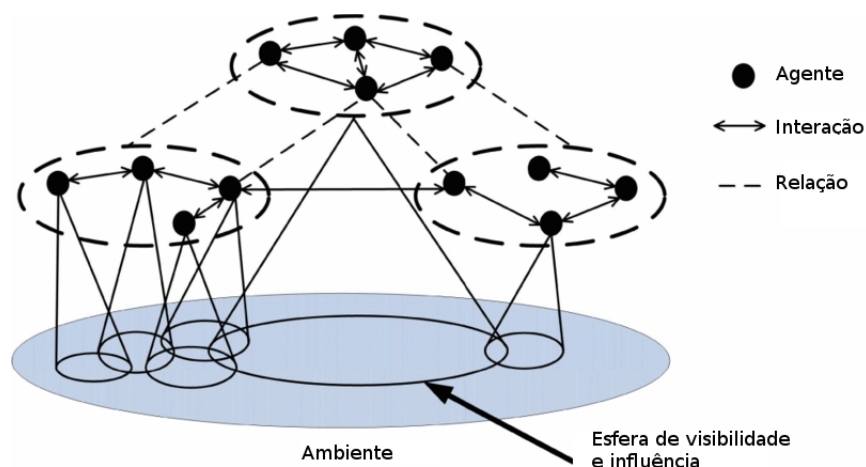
A partir da Figura 2, pode-se perceber que um agente possui sensores de entrada e ações de saída. As ações de saída geradas pelo agente têm como objetivo afetar o ambiente em que o mesmo se encontra. Normalmente, um agente possui um repertório de ações disponíveis e

limitadas. O conjunto das ações possíveis representa a capacidade que o agente possui para mudar o ambiente em que se está inserido (WOOLDRIDGE, 2009). Agentes inteligentes possuem uma visão parcial do ambiente em que estão inseridos. Como mostrado na Figura 3, os agentes possuem sensores de entrada que são capazes de perceber as mudanças do ambiente, fazendo com que o agente esteja ciente de tudo que acontece em sua volta. Com essa capacidade de percepção das mudanças do ambiente, o agente se torna capaz de se adaptar às mudanças deste ambiente.

Agentes possuem a capacidade de se comunicar uns com os outros por meio da troca de mensagens. Esta capacidade de comunicação é o que faz com que agentes cooperem entre si para o bem de uma organização maior. Um agente é orientado por um conjunto de objetivos. Para atingir seus objetivos, o agente deve ser capaz de raciocinar e aprender a partir das informações extraídas do seu ambiente ou pela troca de mensagens com outros agentes.

Os agentes não atuam sozinhos em um determinado sistema, mas sim com outros agentes. Um sistema multiagente consiste de vários agentes que interagem entre si (WOOLDRIDGE, 2009). Os agentes podem colaborar, cooperar, negociar ou competir para atingir seus objetivos ou os objetivos gerais do seu sistema. Os agentes não são as únicas entidades de um sistema multiagente, havendo ambientes, organizações, papéis e objetos. Nos sistemas multiagente, os agentes podem assumir papéis em organizações, devendo cumprir regras e realizar tarefas específicas para o bem da organização em que está inserido. A Figura 3, representa a arquitetura de um típico sistema multiagente.

Figura 3 – Representação de um típico sistema multiagente



Fonte – Xu, Zhong e Zhang (2014)

A partir da Figura 3 pode-se perceber que os agentes interagem entre si. Esta

interação ocorre por meio da troca de mensagens, e é muito importante para que os agentes atualizem suas bases de conhecimento sobre os outros agentes, organizações e o próprio ambiente, adquirindo e trocando assim, muita informação pelo sistema. Também pode-se perceber pela Figura 3, que os agentes formam grupos entre si, chamados de organizações e possuem relações. Essas relações entre os agentes podem ser criadas a partir da participação de uma mesma organização ou pela interação entre organizações diferentes.

A Figura 3 demonstra que agentes possuem uma visão limitada do ambiente em que estão inseridos, e que a partir das relações e das interações, a visibilidade e influência dos agentes aumenta, ou seja, a questão social em um sistema multiagente é de suma importância para o sucesso do sistema. No presente trabalho, será desenvolvida e aplicada uma arquitetura multiagente para o sistema *Chatterbot*.

3 TRABALHOS RELACIONADOS

Nesta seção, é apresentado o trabalho de Huang, Zhou e Yang (2007), que propõe um *framework* para a extração de dados de fóruns online para servir como base de conhecimento a sistemas *Chatterbots*. É apresentado também, o trabalho de Batista et al. (2010) que fornece um *Chatterbot* com arquitetura multiagente para ambientes virtuais de ensino a distância. Por fim, é apresentado o trabalho de Brito (2017) que propõem um *Chatterbot* para uma página virtual de um curso superior.

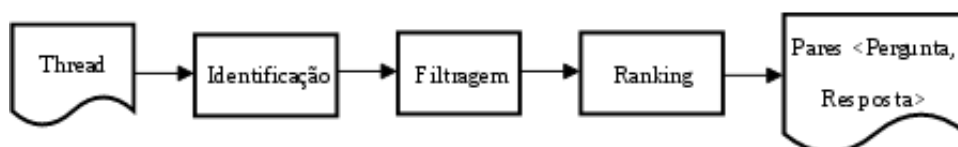
3.1 Extracting Chatbot Knowledge from Online Discussion Forums

Huang, Zhou e Yang (2007) propõem uma nova abordagem para a extração de pares $\langle \text{Pergunta}, \text{Resposta} \rangle$ de alta qualidade de fóruns de discussão online como conhecimento para *Chatterbots*. Dado um determinado fórum online, os pares são extraídos utilizando-se um *framework* em cascata. Em uma visão geral, todo o processo proposto por Huang, Zhou e Yang (2007) pode ser descrito pelas etapas a seguir:

1. As respostas logicamente relevantes ao título da pergunta são extraídas com um classificador SVM de todas as respostas disponíveis para esta pergunta, com base em correlações de estrutura e conteúdo.
2. Os pares $\langle \text{Pergunta}, \text{Resposta} \rangle$ extraídos são classificados por um algoritmo SVM que realiza um ranking com base nas qualidades do conteúdo de cada par. Os *labels* foram determinados por 3 especialistas contratados.
3. Finalmente, os pares $\langle \text{Pergunta}, \text{Resposta} \rangle$ são selecionados como conhecimento para o *Chatterbot*.

A Figura 4 ilustra as etapas do *framework* proposto por Huang, Zhou e Yang (2007).

Figura 4 – Etapas do processo em cascata do *framework* proposto por Huang, Zhou e Yang (2007)



Fonte – Huang, Zhou e Yang (2007)

O *framework* recebe como entrada uma *thread* ou seja, uma determinada postagem

de um fórum online. A primeira etapa (Identificação) aplica um classificador SVM às respostas candidatas para identificar se as mesmas possuem algum segmento relacionado à pergunta. Em seguida, a segunda etapa (Filtragem) filtra as respostas que contém sentimento intemperante, palavras obscenas e/ou informações pessoais com uma lista de palavras-chave predefinidas. A resposta que é mais longa do que um comprimento predefinido é também filtrada, sendo excluída. Finalmente, o módulo de classificação de respostas (*Ranking*) é usado para extrair as respostas descritivas, informativas e confiáveis para a pergunta. Após todos esses processos, o resultado final é formado por pares de <Pergunta, Resposta> de alta qualidade.

O presente trabalho toma como base o *framework* proposto por Huang, Zhou e Yang (2007) para a criação de parâmetros de classificação das respostas extraídas do fórum online *StackOverflow*.

3.2 Multi-Agent Systems in a Computational Environment of Education: A Chatterbot Case Study

Batista et al. (2010) propõem um sistema *Chatterbot* baseado em diálogo para ambientes virtuais de ensino a distância utilizando uma arquitetura multiagente. Para Batista et al. (2010), a capacidade de aplicar sistemas interativos no processo de ensino, através de ambientes virtuais educativos, proporciona situações interativas em que o aluno pode evoluir no processo de construção de conhecimento. De acordo com os autores, um sistema que possua uma interface cognitiva, que permita o estabelecimento de diálogos em linguagem natural, torna a interação estudante-sistema mais atraente, e com um nível mais elevado de empatia.

O *Chatterbot* proposto por Batista et al. (2010) é uma ferramenta de apoio para esclarecer dúvidas de alunos sobre a linguagem de programação Java. O *Chatterbot* desenvolvido pelos autores possui uma base estática de conhecimento que foi criada manualmente pelos mesmos. Ao se receber uma nova pergunta, a mesma é associada com alguma pergunta já existente na base de conhecimento por meio de casamento de padrão. Com isso, o *Chatterbot* retorna a resposta associada a esta pergunta.

O presente trabalho também utiliza uma arquitetura multiagente. Esta arquitetura foi baseada na arquitetura proposta pelo trabalho de Batista et al. (2010), que é uma arquitetura distribuída com troca de mensagens entre os agentes, que representam serviços do sistema. O presente trabalho se difere do trabalho de Batista et al. (2010) na construção da base de conhecimento do *Chatterbot* e na recuperação das respostas. A extração de conhecimento do

presente trabalho é dinâmica, fazendo com que o conhecimento do *Chatterbot* cresça de acordo com o crescimento da base de dados do *StackOverflow*. O que não é possível pela abordagem de construção de conhecimento proposta por Batista et al. (2010).

3.3 Desenvolvimento de um Chatterbot para a Página Web de um Curso de Nível Superior

Brito (2017) propõe o desenvolvimento de um *Chatterbot* capaz de responder perguntas sobre o curso superior de Sistemas de Informação (SI), da Universidade Federal do Ceará - Campus Quixadá. O objetivo do *Chatterbot* desenvolvido por Brito (2017) é responder as principais perguntas sobre o curso de SI, de maneira rápida e com uma alta taxa de disponibilidade, além de conseguir manter um diálogo com várias pessoas simultaneamente.

O *Chatterbot* proposto por Brito (2017) possui duas bases de conhecimento: base de conhecimentos técnicos e base de conhecimentos extras. Conhecimentos técnicos são as informações diretamente ligadas ao curso de SI. Essa base de conhecimento foi extraída da página web do curso. Conhecimentos extras são as informações alheias ao curso de SI. A base de conhecimentos extras consiste em uma série de sub-áreas externas ao curso de SI, que tornam o *Chatterbot* mais social.

Para desenvolver o *Chatterbot*, Brito (2017) aplicou uma arquitetura com três módulos: uma interface humano-computador onde o usuário se comunica com o *Chatterbot*; um sistema de gerenciamento de conteúdo (CMS) para gerenciar informações variáveis; e o próprio *Chatterbot*.

A implementação do servidor do *Chatterbot* foi feita com a linguagem de programação Python. O *Chatterbot* fica em constante aguardo por mensagens. Ao receber uma mensagem ele encaminha a mensagem para o *IBM Watson Conversation*, que é uma API online para processamentos de linguagem natural e definição de regras para aplicação de técnicas de inteligência artificial. Brito (2017) definiu regras no *Watson*, em que ele recebe uma pergunta e, baseado nas regras definidas, elabora uma resposta e retorna a resposta ao servidor do *Chatterbot*. Uma vez que o *Chatterbot* recebe a resposta, é feita uma análise se uma consulta ao CMS é necessária. Se sim, uma consulta é feita ao CMS para obter as informações necessárias e em seguida, a resposta é retornada.

O presente trabalho se baseia no trabalho de Brito (2017) no que se refere a divisão da base de conhecimento em bases do domínio da aplicação e de conhecimentos sociais. O

presente trabalho irá possuir uma base de conhecimento específica para o domínio do problema e uma base com informações sociais, a fim de garantir que Laura possua um comportamento que simule os cumprimentos e comportamento social humano.

3.4 Comparativo entre os Trabalhos

Esta Seção apresenta o Quadro de comparação entre os trabalhos citados anteriormente com o presente trabalho.

Quadro 1 – Comparativo entre os Trabalhos

Trabalho	Conhecimento do Chatterbot	Arquitetura	Qualidade das Respostas
(HUANG; ZHOU; YANG, 2007)	O conhecimento do <i>Chatterbot</i> é extraído a partir de fóruns online de discussão	O <i>framework</i> possui uma arquitetura bem definida, segmentada em 3 módulos: Identificação, Filtragem, e <i>Ranking</i>	O <i>framework</i> aplica etapas para analisar e filtrar as melhores respostas
(BATISTA et al., 2010)	O conhecimento do <i>Chatterbot</i> foi criado pelo autor do trabalho manualmente	Possui uma arquitetura multiagente bem definida, na qual, agentes representam as funcionalidades do <i>Chatterbot</i>	A qualidade das respostas não é trabalhada, toda a base de respostas foi escrita pelo autor
(BRITO, 2017)	O conhecimento do <i>Chatterbot</i> foi extraído a partir da página <i>web</i> do curso de SI	Define uma arquitetura que contempla tecnologias e APIs de terceiros	A qualidade das respostas foi avaliada por meio de testes de validação com usuários reais
Este trabalho	O conhecimento do <i>Chatterbot</i> é extraído a partir do fórum online <i>StackOverflow</i>	Arquitetura multiagente, na qual cada funcionalidade do sistema é representada como um agente inteligente	A qualidade das respostas é calculada a partir da aplicação de uma métrica definida

Fonte – Elaborado pelo autor

4 PROCEDIMENTOS METODOLÓGICOS

O desenvolvimento do *Chatterbot* Laura foi planejado para ocorrer em 7 etapas. A primeira etapa define a arquitetura do sistema, baseada em um sistema multiagente. Na segunda etapa é realizada a coleta dos dados necessários para a base de conhecimento de Laura. Na terceira etapa são realizadas as técnicas de pré-processamento nos dados obtidos. Na quarta etapa ocorre a análise de desempenho dos algoritmos *K-Medoids* e Clusterização Hierárquica Aglomerativa. Na quinta etapa ocorre a análise de desempenho das medidas de distância. Na sexta etapa ocorre a clusterização das perguntas obtidas a partir do fórum *StackOverflow*. Por fim, na sétima etapa é definida e aplicada uma métrica para criar um *ranking* das respostas. Os resultados de cada etapa estão presentes no capítulo 5 do presente trabalho.

4.1 Desenvolvimento de uma Arquitetura Multiagente

Nesta etapa, foi desenvolvida uma arquitetura multiagente com o objetivo de facilitar os processos de extração de conhecimento necessários para Laura, bem como os processos internos de interação com os usuários. Esta arquitetura também fornece um sistema distribuído, modular e autônomo. A arquitetura desenvolvida divide o sistema em agentes. Em que cada agente é responsável por uma função do sistema. Como, por exemplo, há o agente responsável em automatizar o processo de clusterização das perguntas, o agente que realiza o ranking das respostas, entre outros.

4.2 Coleta dos Dados

O *StackOverflow* destinado à comunidade brasileira foi o escolhido para servir como base de conhecimento do *Chatterbot* Laura. Esta escolha foi definida pelos seguintes motivos:

- É um dos maiores fóruns sobre programação existentes no Brasil, possuindo dados diversificados e conhecimentos disponibilizados sobre programação por diversos usuários brasileiros.
- Possui uma base de dados totalmente em português.
- Possui dois métodos de acesso ao conteúdo de sua base de dados, via requisição HTTPS ou por meio da cópia do banco de dados do fórum.

Para o presente trabalho, o uso do *dump*¹ da base de dados do fórum *StackOverflow*,

¹ <https://archive.org/details/stackexchange>

foi a opção escolhida para a coleta de dados. Esta opção de coleta de dados foi escolhida, por não se fazer necessária a criação constante de requisições *HTTPS* para a *API* do *StackOverflow*, que tornaria mais lento o acesso aos dados. E também, pelo fato de que, possuir a base de dados completa facilita as próximas etapas do trabalho, como o pré-processamento e a clusterização destes dados.

O *dump* utilizado no presente trabalho contém a base de dados do fórum *StackOverflow* da comunidade brasileira, desde o início do fórum até março de 2017.

4.3 Pré-processamento dos Dados

Após a Coleta dos dados descrita pela Seção 4.2, se faz necessário realizar o pré-processamento dos dados obtidos. A etapa de pré-processamento dos dados tem o objetivo de limpar os dados, retirando assim, possíveis ruídos e inconsistências que venham a influenciar a clusterização destes dados.

Foi feita uma análise nos dados obtidos do fórum *StackOverflow*. Primeiramente foi feito o descarte de tabelas que possuíam dados irrelevantes ao *Chatterbot* Laura. Feito isso, as tabelas remanescentes tiveram seus atributos analisados um por um. Para que, atributos não relevantes a Laura também fossem descartados.

Após todo o descarte de informações não relevantes a Laura, foi aplicado um processador textual nas perguntas extraídas da base de dados. Este processador textual, retirou todos os caracteres repetidos, acentos, *links* e símbolos que possam estar inseridos nas perguntas. Além desta limpeza, as *stop words* também foram removidas. Os sufixos das palavras também foram desconsiderados, restando apenas a palavra em sua forma primitiva.

4.4 Análise de Desempenho dos Algoritmos K-Medoids x Clusterização Hierárquica Aglomerativa

Com o objetivo de garantir que o algoritmo de clusterização utilizado para clusterizar as perguntas da base de dados de Laura seja o mais eficaz, foi realizada uma análise de desempenho. A análise de desempenho comparou os algoritmos *K-Medoids* e Clusterização Hierárquica Aglomerativa quanto ao tempo de execução. Ambos algoritmos foram implementados a partir de seus respectivos pseudocódigos encontrados na literatura.

4.5 Análise de Desempenho das Medidas de Distância

Como método de validação da medida de distância utilizada pelo algoritmo de clusterização *K-Medoids*, foi realizada uma análise de desempenho com as medidas de distância descritas nas Seções 2.3.1, 2.3.2 e 2.3.3. Esta análise de desempenho utilizou o *Elbow method*, descrito na Seção 2.3.4, como método de validação das distâncias.

4.6 Clusterização das Perguntas extraídas do *StackOverflow*

Como o objetivo de Laura é responder perguntas sobre programação Java, se faz necessário que ela consiga, a partir de seu conhecimento, identificar perguntas que sejam semelhantes à pergunta recebida. A partir de uma pergunta recebida, Laura deve ser capaz de assimilar esta pergunta com as demais perguntas existentes em sua base de conhecimento. As perguntas que forem semelhantes à pergunta recebida, provavelmente, possuem respostas candidatas à pergunta recebida. Para isso, se fez necessário realizar a clusterização das perguntas.

Para a realização do processo de clusterização, foi aplicado o algoritmo de clusterização que obteve os melhores resultados na análise de desempenho realizada (descrita pela Seção 4.4). Apenas as perguntas de domínio Java foram clusterizadas.

4.7 Desenvolvimento de uma Métrica para a criação de um Ranking das Respostas

Para que Laura responda aos usuários somente com respostas de alta qualidade, uma métrica de classificação para as respostas presentes na base de dados foi desenvolvida. Esta métrica é utilizada para criar um *ranking* das respostas relacionadas à uma dada pergunta da base de dados. Esta métrica de classificação foi baseada em regras de classificação de respostas presentes no *framework* proposto por Huang, Zhou e Yang (2007).

5 RESULTADOS E DISCUSSÕES

Neste capítulo são apresentados e discutidos os resultados obtidos através da execução deste trabalho. Na Seção 5.1 são descritos e discutidos os dados obtidos a partir do fórum *StackOverflow*. Na Seção 5.2 são descritas as tabelas remanescentes após a aplicação da técnica de pré-processamento nos dados obtidos. Na Seção 5.3 são apresentados os resultados obtidos durante o desenvolvimento da Arquitetura do sistema. Na Seção 5.4 é descrito e discutido os resultados da análise de desempenho dos algoritmos *K-Medoids* e Clusterização Hierárquica Aglomerativa. Na Seção 5.5 é descrito e discutido os resultados da análise de desempenho das medidas de distância. Na Seção 5.6 são apresentados os resultados da Clusterização das perguntas. Na Seção 5.7 é descrita a métrica para a criação do *ranking* das respostas e são apresentados os resultados da aplicação da mesma. Por fim, na Seção 5.8, são apresentados os resultados dos experimentos feitos com Laura.

5.1 Resultados da Coleta dos Dados

Como explicado na Seção 4.2, foi realizado o download do *dump* da base de dados do *StackOverflow*. A base é constituída por oito tabelas. As tabelas da base de dados são descritas pela Tabela 1, juntamente com a quantidade dos seus registros.

Tabela 1 – Tabelas da base de dados do *StackOverflow* com suas respectivas quantidades de registros.

Nome da Tabela	Quantidade de Registros
<i>Badges</i>	112.900
<i>Comments</i>	266.592
<i>PostHistory</i>	475.356
<i>PostLinks</i>	16.460
<i>Posts</i>	149.384
<i>Tags</i>	2.553
<i>Users</i>	44.592
<i>Votes</i>	486.174

Fonte – Elaborado pelo autor

A tabela *Badges* possui dados referentes à classificação dos usuários quanto a suas interações no fórum *StackOverflow*. A tabela *Comments* possui todos os comentários realizados

pelos usuários nas postagens do fórum *StackOverflow*. A tabela *PostHistory* possui todas as alterações realizadas nas postagens existentes do fórum *StackOverflow*. A tabela *PostLinks* possui os registros referentes as postagens que referenciam outras postagens existentes do fórum *StackOverflow*. A tabela *Posts* possui os registros referentes as postagens criadas no fórum *StackOverflow*, tanto postagens do tipo pergunta quanto do tipo resposta. Considerada para este trabalho como a tabela mais importante de toda a base de dados. A tabela *Tags* possui os diversos rótulos existentes no fórum *StackOverflow*. A tabela *Users* possui os registros de todos os usuários cadastrados no *StackOverflow*. A tabela *Votes* possui os registros de todos os votos efetuados nas postagens existentes do fórum *StackOverflow*.

5.2 Resultados do Pré-processamento

Como descrito na Seção 4.3, foi realizado o pré-processamento dos dados obtidos do *StackOverflow*. Primeiramente, as tabelas existentes na base foram analisadas e foi observado que certas tabelas não forneciam informações relevantes à Laura. As tabelas que não forneciam informações relevantes para Laura foram:

- *PostHistory*
- *PostLinks*
- *Tags*
- *Votes*
- *Comments*
- *Badges*

Após o processo de descarte das tabelas, foi realizada uma análise nos atributos das tabelas remanescentes. Esta análise nos atributos é essencial para a exclusão de atributos desnecessários, que, invés de atribuir maior valor de informação à Laura, podem acabar por poluir sua base de conhecimento. Após a exclusão de atributos desnecessários, as tabelas remanescentes ficaram da seguinte forma:

- *Posts*
 - *Id*
 - * Identificação da postagem
 - *PostType*
 - * Tipo da postagem, pode ser uma pergunta ou uma resposta
 - *AcceptedAnswerId*

- * Caso seja uma pergunta, possui a identificação da resposta escolhida pelo autor da pergunta
- *CreationDate*
 - * Data de criação da postagem
- *ParentId*
 - * Caso seja uma resposta, possui a identificação da pergunta a qual esta postagem responde
- *Score*
 - * Valor de pontuação de uma postagem
- *Body*
 - * No caso de uma resposta, é o conteúdo da mesma
- *OwnerUserId*
 - * Identificação do usuário que escreveu a postagem
- *Title*
 - * No caso de uma pergunta, é o título da mesma
- *FavoriteCount*
 - * Quantidade de favoritos dado a uma postagem
- *Tags*
 - * São termos que classificam a postagem em uma ou mais categorias
- *Users*
 - *Id*
 - * Identificação do usuário
 - *Reputation*
 - * Valor da reputação do usuário no fórum
 - *CreationDate*
 - * Data de criação da conta do usuário
 - *LastAccessDate*
 - * Data do último acesso do usuário
 - *Views*
 - * Quantidade de visualizações do perfil do usuário
 - *UpVotes*
 - * Quantidade de votos positivos recebidos pelo usuário

– *DownVotes*

- * Quantidade de votos negativos recebidos pelo usuário

Após os descartes de atributos desnecessários das tabelas *Posts* e *Users*, os dados referentes à coluna *Title*, presentes na tabela *Posts*, passaram por um processamento de texto com o objetivo de retirar possíveis inconsistências e ruídos (dados sujos). Visto que, para a clusterização das postagens do tipo *pergunta*, apenas o título das postagens será levado em consideração para a formação dos *clusters*. Para o processamento dos títulos das postagens, foi utilizado o *Framework MineText*¹, o qual é um *Framework* desenvolvido em Python para Mineração e processamento de texto.

Com o uso do *Framework MineText* foram retirados os acentos, *links*, caracteres repetidos, símbolos, *stop words* e o sufixos dos títulos. Todo este processamento de texto foi realizado levando em consideração o idioma português.

5.3 Arquitetura Multiagente Desenvolvida

Como descrito pela Seção 4.1 do presente trabalho, uma arquitetura multiagente foi desenvolvida com o objetivo de tornar Laura autônoma e distribuída, facilitando processos e tarefas realizados. A Figura 5 descreve em uma visão de alto nível a arquitetura desenvolvida.

Como representado na Figura 5, Laura é um sistema multiagente, possuindo agentes inteligentes com responsabilidades bem definidas. A partir da Figura 5, pode-se perceber que há 6 tipos de agentes, que serão explicados a seguir, são eles: Interface, Analista, Social, Domínio, Tensor e Processador.

5.3.1 Agente Interface

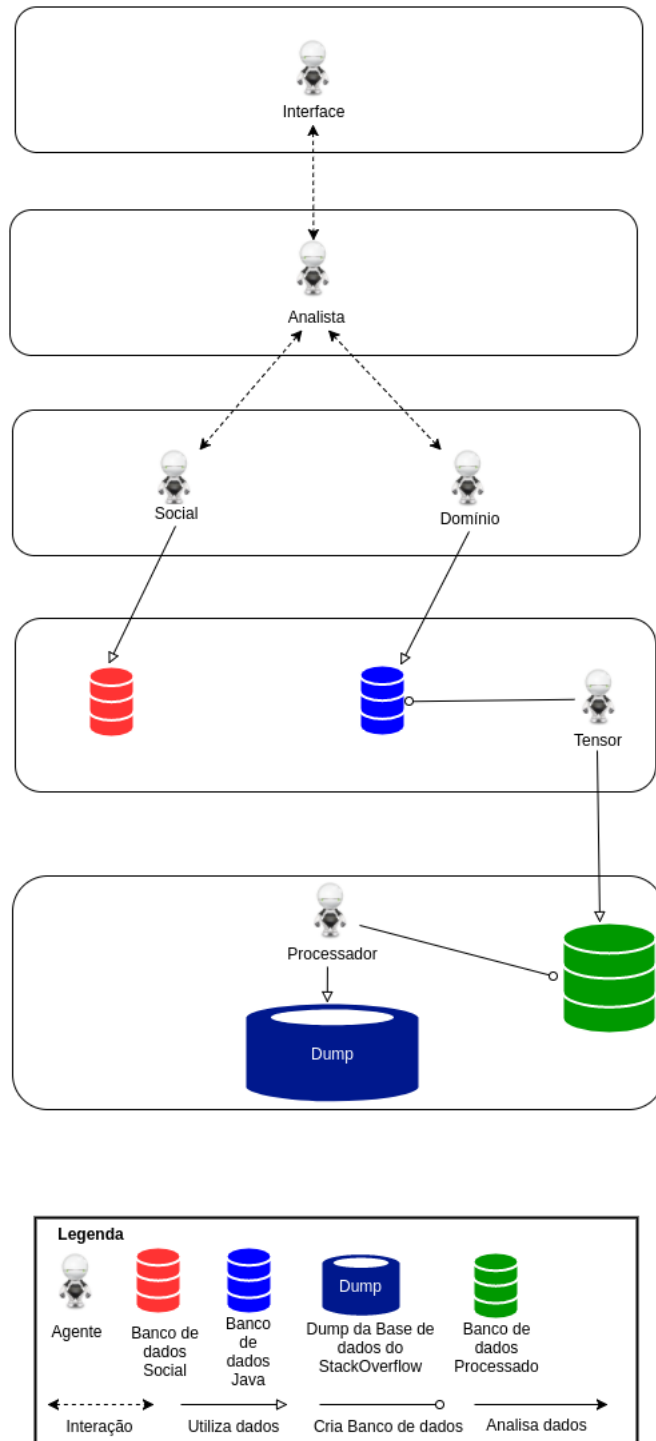
Agente responsável por interagir com os usuários, recebendo mensagens e enviando as respostas aos usuários. Estas mensagens são enviadas via requisição HTTPS tipo Post.

5.3.2 Agente Analista

Agente responsável por filtrar uma mensagem recebida, e categorizá-la como sendo uma mensagem *social* ou uma mensagem do *domínio*, enviando a mensagem para o agente responsável. Esta classificação é feita com base em duas listas contendo 35 palavras cada. Uma

¹ <https://github.com/gustavoaires/minetext>

Figura 5 – Visão em alto nível da Arquitetura Multiagente



Fonte – Autor

lista com palavras referentes a mensagens sociais, tais como: dia, bem, oi, oie, olá, conversar, entre outras. E a outra com palavras referentes a mensagens de domínio, tais como: java, WEB, POO, programação, inteiro, vetores, matriz, JPA, entre outras.

O agente Analista verifica a mensagem recebida comparando-a com listas de termos sociais e de domínio. Estas listas de termos foi definida manualmente. Por exemplo, a lista de termos sociais possui alguns termos, como: oi, olá, tudo bom?, entre outros. Já a lista de termos de domínio possui termos que remetem ao domínio de programação Java, tais como: java, array, list, integer, entre outros.

Para cada termo presente na mensagem recebida, é atribuído 1 para a classe social ou para a classe domínio, a classe que possuir maior valor é a classe na qual a mensagem será classificada.

5.3.3 *Agente Social*

Agente responsável por receber mensagens sociais, e recuperar uma resposta adequada para a mensagem recebida. O agente Social recebe a mensagem, processa-a verificando suas palavras e depois busca em um mapa uma *key* mais semelhante a esta mensagem e então retorna o valor para esta *key*. O mapa é construído a partir de uma base estática de mensagens sociais. Estas mensagens são necessárias para que Laura responda perguntas sociais recebidas pelos usuários. Como o foco deste trabalho não é sobre a interação social de Laura com os usuários, este assunto não será aprofundado.

5.3.4 *Agente Domínio*

Agente responsável por receber mensagens do domínio, ou seja, dúvidas sobre programação Java. Primeiramente, o agente realiza o processamento textual da mensagem recebida, mesmo processo realizado nas perguntas existentes na base de dados, descrito na Seção 4.3. Após o processamento textual, o agente Domínio compara a mensagem com os *medoids* existentes na base de conhecimento, atribuindo a mensagem ao *cluster* com o *medoid* mais semelhante. A semelhança é calculada a partir da medida *Cosine Similarity* entre a mensagem e os *medoids*. Por fim, a mensagem é novamente comparada com todas as perguntas presentes no *cluster*, em que, a resposta da pergunta mais semelhante à mensagem recebida é retornada pelo agente.

5.3.5 *Agente Tensor*

Agente responsável por, a partir da base de dados processada, filtrar apenas perguntas e respostas do domínio Java. Esse filtro é feito a partir das *tags* presentes nos dados obtidos pela base do fórum *StackOverflow* detalhadamente descrita na Seção 5.1.

5.3.6 *Agente Processador*

Agente responsável por, a partir do *dump* da base de dados do fórum *StackOverflow*, criar a base de conhecimento do Chatterbot. O agente Processador verifica de 15 em 15 dias se há uma nova versão do *dump* da base de dados disponível para download. Este prazo de tempo é parametrizável. A escolha deste período de tempo foi feita com base na métrica de tempo para a disponibilização de uma nova versão da base de dados. Em que a atualização do *dump* fornecido, costuma ser de 3 em 3 meses. Se houver uma nova versão disponível, o Processador realiza o download da base e aplica as atividades de pré-processamento, descritas pela Seção 4.3, nesta base.

5.4 Resultados da Análise de Desempenho dos Algoritmos K-Medoids x Clusterização Hierárquica Aglomerativa

Como descrito na Seção 4.4, uma análise de desempenho é necessária para comparar os algoritmos de clusterização *K-Medoids* e Clusterização Hierárquica Aglomerativa a fim de se obter o melhor algoritmo para clusterizar as perguntas da base de conhecimento de Laura. Todas as perguntas do domínio Java foram selecionadas como base para o experimento, somando ao todo 7.282 perguntas. A análise de desempenho visa conferir o tempo de execução de ambos os algoritmos. A Tabela 2 exibe o tempo de execução obtido para a clusterização das perguntas de ambos os algoritmos.

Tabela 2 – Tempo de execução dos algoritmos de clusterização.

Algoritmo	Tempo de Execução
<i>K-Medoids</i>	0h22min42s
Clusterização Hierárquica Aglomerativa	1h03min12s

Fonte – Elaborado pelo autor

Para realizar a clusterização, os dois algoritmos utilizaram a medida de similaridade cosseno. Para o algoritmo *K-Medoids* foi escolhido um valor de K igual a 20. Esta escolha de K foi com base na análise das características dos dados. Como mostrado na Tabela 2, o algoritmo *K-Medoids* levou menos tempo para clusterizar as perguntas. Já o algoritmo de Clusterização Hierárquica Aglomerativa levou bem mais tempo. Isso pode se dar ao fato de que ele realiza um processo computacionalmente mais custoso, criando no começo um *cluster* para cada pergunta até restarem somente dois *clusters*.

Após a análise do tempo de execução dos algoritmos, os *clusters* criados foram analisados. Os *clusters* criados pelo algoritmo *K-Medoids* ficaram bem divididos, em sua maioria, as perguntas ficaram em *clusters* com outras perguntas bastante similares. Já para o algoritmo de Clusterização Hierárquica Aglomerativa, foi formada uma árvore hierárquica de *clusters*, com isso a visualização ficou um pouco difícil, visto que a árvore criada ficou muito extensa. No entanto, os *clusters* resultantes também possuíam um agrupamento de perguntas similares.

Com esta análise de desempenho, ficou claro que o algoritmo *K-Medoids* teve a execução mais rápida. Já o algoritmo de Clusterização Hierárquica Aglomerativa possuiu um tempo de execução bastante superior. Logo, após estes resultados, o algoritmo *K-Medoids* acabou sendo melhor do que o algoritmo de Clusterização Hierárquica Aglomerativa para o domínio deste trabalho.

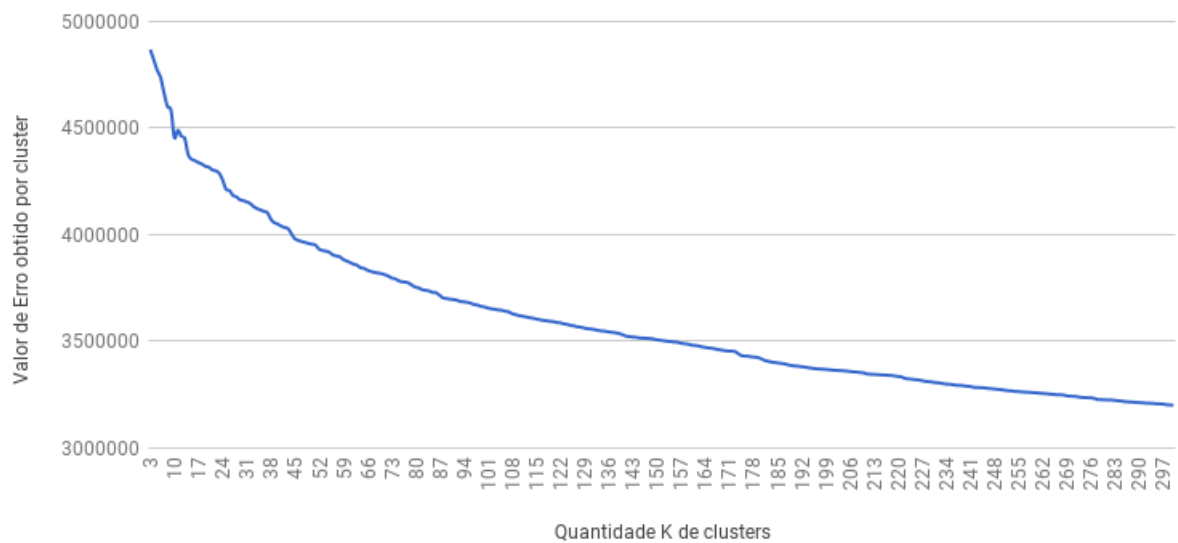
5.5 Resultados da Análise de Desempenho das Medidas de Distância

Como descrito pela Seção 4.5 foi realizada uma análise de desempenho com o objetivo de se escolher a melhor medida de distância a ser utilizada pelo algoritmo *K-Medoids*. Para a realização da análise foram selecionadas todas as 7.282 perguntas de domínio Java da base de conhecimento. Essas perguntas serão clusterizadas pelo algoritmo *K-Medoids* utilizando cada medida de distância. Foi definido um *range* para K começando com 2 indo até 300. Para analisar o desempenho das medidas de distância, foi utilizado o *Elbow Method* descrito na Seção 2.3.4.

A Figura 6 ilustra os resultados obtidos pela análise de desempenho da medida *Levenshtein Distance*.

Pela Figura 6 é perceptível que os valores de erros obtidos pela *Levenshtein Distance* foram muito altos e que até o fim não houve um equilíbrio nos valores. Dessa forma, os *clusters*

Figura 6 – Análise de Desempenho - *Levenshtein Distance*

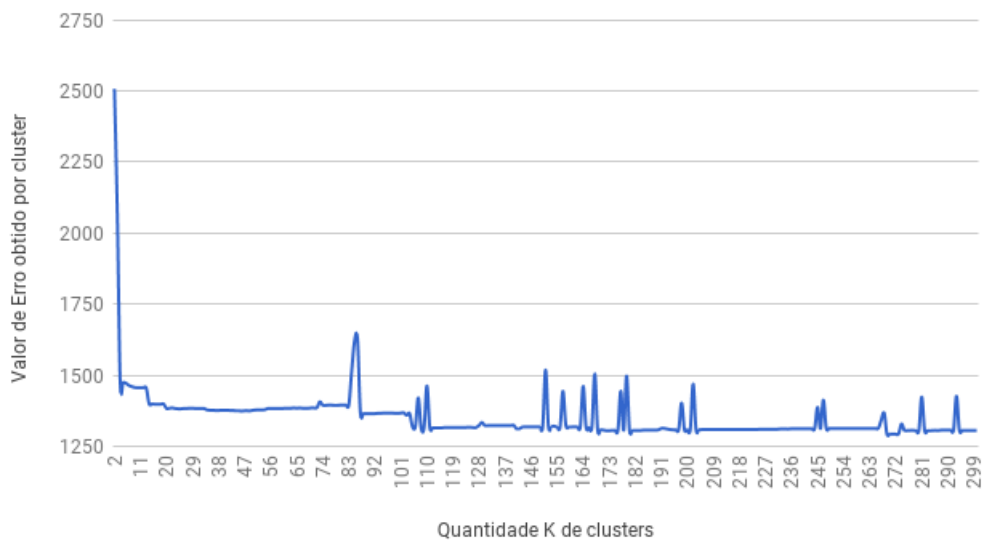


Fonte – Elaborado pelo Autor

criados com esta medida de distância não obtiveram uma boa qualidade. Com isso, a *Levenshtein Distance* foi descartada para o domínio deste trabalho.

A Figura 7 ilustra os resultados obtidos pela análise de desempenho da medida *Jaro-Winkler Distance*.

Figura 7 – Análise de Desempenho - *Jaro-Winkler Distance*



Fonte – Elaborado pelo Autor

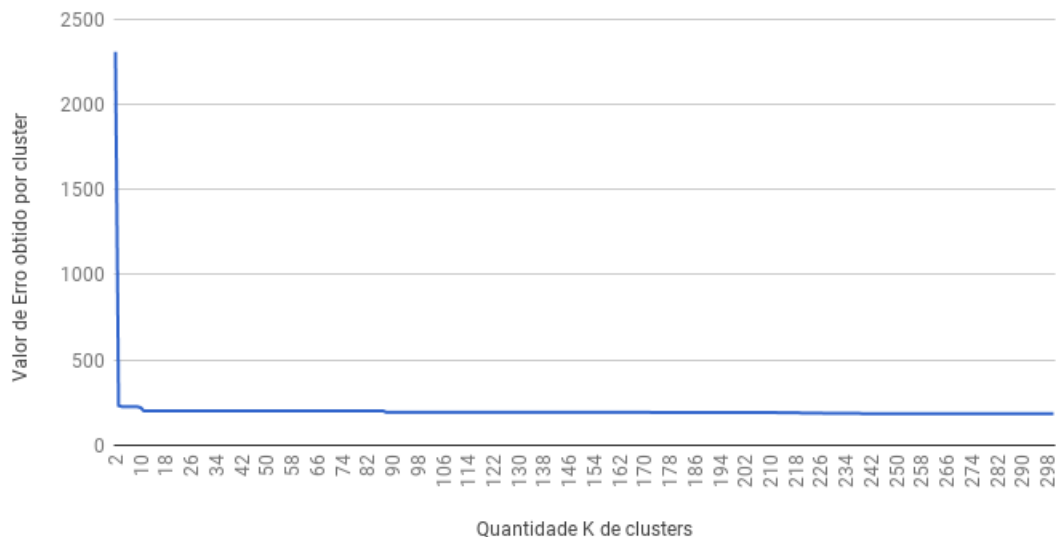
Pela Figura 7 é perceptível que os valores de erros obtidos pela medida *Jaro-Winkler*

Distance não foram tão altos quanto os da *Levenshtein Distance*. No entanto, também não foi possível obter uma efeito platô dos valores dos erros. Muito pelo contrário, houve uma grande variação nos valores dos erros de acordo com o valor de K . Dessa forma, não é possível assegurar a qualidade dos *clusters* criados. Com isso, a *Jaro-Winkler Distance* também foi descartada para o domínio deste trabalho.

A Figura 8 ilustra os resultados obtidos pela análise de desempenho da medida *Cosine Similarity*.

Pela Figura 8 é perceptível que os valores de erros obtidos pela medida *Cosine Similarity* foram os menores até o momento. E que, após K igual a 90 o valor dos erros obtidos chegou a um estado platô, ou seja, não houve variação nos valores. O que pode-se extrair a partir disso, é que 90 é um ótimo valor para K . Esta análise deve ser realizada novamente sempre que a base de conhecimento de Laura for alterada, para obter o melhor valor de K da base atual. Dessa forma, a medida *Cosine Similarity* foi a que proveu maior qualidade aos *clusters*. Com isso, a *Cosine Similarity* foi escolhida como medida de distância para o presente trabalho.

Figura 8 – Análise de Desempenho - *Cosine Similarity*



Fonte – Elaborado pelo Autor

5.6 Resultados da Clusterização das Perguntas

Após a realização das análises de desempenho, foi possível assegurar que utilizando o algoritmo *K-Medoids*, para $K = 90$ com a medida de similaridade cosseno, os *clusters* criados

possuíam uma ótima qualidade. Com isso, todas as perguntas do domínio da linguagem de programação Java, 7.282 perguntas, foram clusterizadas. Para realizar a clusterização, toda as 7.282 perguntas passaram pelo processo de vetorização, utilizando-se o cálculo *tf-idf*. Isso se deve ao fato da medida de similaridade cosseno trabalhar apenas com vetores numéricos não nulos. A Tabelas 3, 4 e 5 descrevem os 90 *clusters* criados, mostrando seus respectivos *medoids* e suas respectivas quantidades de perguntas.

Tabela 3 – *Clusters* criados - Parte 1

<i>Cluster</i>	<i>Medoid</i>	Quantidade de Perguntas
1	Enviar dados para banco de dados	558
2	Gerando numeros sem repetir usando array ou matriz	52
3	Como utilizar JProgressBar?	39
4	Spring Security com Java EE	36
5	Genéricos, estender a uma X ou Y	22
6	Como converter um projeto Java em Eclipse para um projeto Maven?	84
7	erro na minha aplicação java	201
8	Dúvida sobre NetBeans	99
9	Como fazer meus próprios comandos e atalhos no Eclipse?	709
10	Erro: Parameter index out of range	24
11	Quais são as diferenças entre HashMap e Hashtable?	83
12	Problema com Hibernate	105
13	Adicionar elemento em Lista (Arraylist)	97
14	O que resta para configurar com Spring boot?	55
15	Juntar duas Arraylist	44
16	Como adicionar elemento dentro de um arrayList?	68
17	Como escrever em um arquivo.json	142
18	Atualizar JTable após cada alteração de dados em uma célula	31
19	Aplicação não utiliza todos os núcleos do processador?	37
20	Como converter uma string em data ou date?	116
21	Atualizar projeto ao executar Maven	81
22	Problema https Redirecionamento em excesso	301
23	Integração entre Spring e JPA	85
24	Impedindo JFrame de ser movimentado na tela	31
25	Banco de dados SQLite e Android	45
26	Modificar símbolo de tag no PDF gerado através do iText	20
27	AsyncTask Android	150
28	Como converter um byte[] para uma imagem?	93
29	Ocultar coluna na hora da edição	30
30	Como inserir dados em INPUTS pelo JAVA	166

Fonte – Elaborado pelo autor

A partir do que foi exposto pelas Tabelas 3, 4 e 5 pode-se perceber que os *clusters* gerados são bem variados, dividindo as perguntas em subdomínios relacionados à programação

Tabela 4 – *Clusters* criados - Parte 2

<i>Cluster</i>	<i>Medoid</i>	Quantidade de Perguntas
31	relatorio iReport no servidor	84
32	Problemas com Update Java e mysql	31
33	Bat para finalizar processo	26
34	Implementando CDI no jpa	32
35	Configurar o IP para acesso ao servidor utilizando Hibernate	33
36	Definindo o construtor de um Objeto	112
37	Organização dos códigos gerados pelo WindowBuilder	37
38	Como inserir String do res/string no array String	216
39	É possível alterar o tipo da variável em Java?	57
40	Qual o problema com essa query da minha JSP?	4
41	Preencher textview android studio/sql server	44
42	Como usar mais de uma extends?	181
43	É possível cancelar uma conexão JDBC?	35
44	erro no \$http com ionic e spring	458
45	Não consigo inserir no MongoDB usando Java	70
46	Tela de login - não consigo logar um usuario,admin ou user	31
47	panelGrid do JSF não funciona	143
48	Como modificar/evoluir um cache distribuído	7
49	Duvida com regra de negócio entre classe aluno e turma	9
50	Refatorar com eclipse	26
51	Problema ao retornar valor de um método	65
52	Criar máscara para campos	73
53	Dúvida ao gerar REST de retorno de procedure no jersey	8
54	Evitar as especificações no Android?	21
55	Serialização LocalDateTime	10
56	ActionBar tamanho texto dinâmico	50
57	Não consigo enviar valores para Webservice	34
58	Como tratar exceptions com JTA?	14
59	Java - Leitor de Disco (Linux)	11
60	Minimizar + SytemTray	10

Fonte – Elaborado pelo autor

Java. Também percebe-se que a quantidade de perguntas por *cluster* foi bastante variada. Um possível motivo para isto ter acontecido é a natureza dos dados. Mesmo todas as perguntas sendo sobre Java, há uma grande variação nas características específicas destas perguntas, como tecnologia, ferramentas e *frameworks*.

Tabela 5 – *Clusters* criados - Parte 3

<i>Cluster</i>	<i>Medoid</i>	Quantidade de Perguntas
61	Profile network java (jdbc)	6
62	PersistenceUnit: EntityManagerFactory	11
63	Como passar valores de uma activity para outra	138
64	Ler todos os arquivos txt em uma pasta	65
65	Problema com SUM na NamedQuery	15
66	Alert Dialog personalizado com imagem Gif	10
67	Como migrar Swing?	27
68	NoClassDefFoundError - Apenas em execução via Jenkins	16
69	Problemas com caracteres no MySQL com Java	17
70	Onde colocar comentários e javadoc ? Model ou Controller?	12
71	Retorno null JSP	107
72	Cardview dentro de reciclerview	8
73	JFreeChart - Gráficos em Java Swing não aparecem na tela	29
74	O que é "Princípio da Inversão de Dependência"(DIP)?	6
75	DataTable com jsf	77
76	Banco de dados + java	60
77	Chamar um método da classe	207
78	Recuperar itens de ListView	67
79	Java não compila .java	595
80	DataTable sem a mensagem "No records found."?	44
81	Stream() e parallelStreams()	8
82	Jogo das 3 portas Java	19
83	Qt Java - Pegar imagem da galeria	37
84	Mover várias mensagens JavaMail	9
85	Dados não persistem no arquivo txt	27
86	Erro ao criar Google Maps Activity	25
87	AlertDialog a partir de outra classe	50
88	Android não reconhece parse.parse.initialize	14
89	Qual ideal começar, Swing ou JavaFX?	12
90	Problema ao imprimir o conteúdo de um vetor	23

Fonte – Elaborado pelo autor

5.7 Resultados da Criação e Aplicação da Métrica para o Ranking das Respostas

Como descrito pela Seção 4.7, foi criada uma métrica para a criação de um ranking das respostas presentes na base de conhecimento de Laura. A métrica desenvolvida possui as seguintes notas:

- *N1*
 - A resposta foi a escolhida pelo usuário que fez a pergunta?
- *N2*
 - O *score* da resposta é alto?
- *N3*
 - A resposta foi criada pelo usuário que fez a pergunta?
- *N4*

- A resposta é impessoal?
- *N5*
 - A resposta não contém palavras obscenas?
- *N6*
 - A reputação do autor da resposta é boa?
- *N7*
 - A popularidade do autor da resposta é alta?
- *N8*
 - Quantidade de votos positivos do autor da resposta
- *N9*
 - Quantidade de votos negativos do autor da resposta
- *N10*
 - A longevidade da participação do autor no fórum
- *N11*
 - Quantidade total de respostas do autor no fórum

As notas: *N1*, *N3*, *N4*, *N5*, *N7*, *N10*, *N11* foram retiradas do trabalho de Huang, Zhou e Yang (2007). As notas restantes foram criadas com base no domínio do presente trabalho. Cada nota pode possuir valores de 0 até 1.

A nota *N1* pode receber apenas 0 ou 1. Quando a resposta foi a escolhida pelo o autor da pergunta então a nota recebe 1, caso contrário, recebe 0. A nota *N2* pode assumir os valores 0, 0.25, 0.5, 0.75 ou 1. Ela recebe 0 caso o *score* da resposta seja menor que 0; 0.25 caso seja maior/igual que o menor valor diferente de 0 de *score* e menor que o valor médio de *scores* da base; 0.5 caso seja maior/igual que o valor médio de *scores* da base; 0.75 caso seja maior/igual que 75% do maior valor de *score* da base e 1 caso seja igual ao maior valor de *score* da base.

A nota *N3* pode receber 0 ou 1. 1 caso a resposta tenha sido dada pelo próprio usuário que fez a pergunta e 0 caso contrário. A nota *N4* pode receber os valores 0 ou 1. 1 caso a resposta seja impessoal e 0 caso contrário. Para avaliar se a resposta é impessoal, foram selecionados 35 termos pessoais. Caso a resposta contenha algum desses termos, ela não é impessoal. A nota *N5* pode receber os valores 0 ou 1. 1 caso a resposta não possua palavras obscenas e 0 caso contrário. Para avaliar se a resposta contém palavras obscenas, foram selecionados 50 termos obscenos. Caso a resposta contenha algum desses termos, ela é obscena.

A nota *N6* pode assumir os valores 0, 0.25, 0.5, 0.75 ou 1. Ela recebe 0 caso a reputação do autor seja 0; 0.25 caso seja maior/igual que o menor valor diferente de 0 da base e menor que o valor médio da base; 0.5 caso seja maior/igual que o valor médio da base; 0.75 caso seja maior/igual que 75% do maior valor da base e 1 caso seja igual ao maior valor de reputação da base.

A nota *N7* pode assumir os valores 0, 0.25, 0.5, 0.75 ou 1. Ela recebe 0 caso o número de visualizações do perfil do autor da resposta seja igual a 0; 0.25 caso seja maior/igual que o menor valor de visualizações diferente de 0 da base e menor que o valor médio da base; 0.5 caso seja maior/igual que o valor médio da base; 0.75 caso seja maior/igual que 75% do maior valor da base e 1 caso seja igual ao maior valor de visualizações da base.

A nota *N8* pode assumir os valores 0, 0.25, 0.5, 0.75 ou 1. Ela recebe 0 caso o número de votos positivos do autor da resposta seja igual a 0; 0.25 caso seja maior/igual que o menor valor diferente de 0 da base e menor que o valor médio da base; 0.5 caso seja maior/igual que o valor médio da base; 0.75 caso seja maior/igual que 75% do maior valor da base e 1 caso seja igual ao maior valor de votos positivos da base.

A nota *N9* pode assumir os valores 0, 0.25, 0.5, 0.75 ou 1. Ela recebe 1 caso o número de votos negativos do autor da resposta seja igual a 0; 0.75 caso seja maior/igual que o menor valor diferente de 0 da base e menor que o valor médio da base; 0.5 caso seja maior/igual que o valor médio da base; 0.25 caso seja maior/igual que 75% do maior valor da base e 0 caso seja igual ao maior valor de votos negativos da base.

A nota *N10* pode assumir os valores 0.2, 0.4, 0.6, 0.8 e 1. Ela recebe 0.2 caso o autor da resposta tenha criado sua conta em 2017; 0.4 caso tenha criado em 2016; 0.6 caso tenha criado em 2015; 0.8 caso tenha criado em 2014 e 1 caso tenha criado em 2013.

A nota *N11* pode assumir os valores 0, 0.25, 0.5, 0.75 e 1. Ela recebe 0 caso o autor da resposta só tenha respondido 1 única vez até o presente momento; 0.25 caso já tenha respondido mais de 1 resposta e menos que 5 respostas; 0.5 caso já tenha respondido entre 5 a 10 respostas; 0.75 caso já tenha respondido entre 10 a 15 respostas e 1 caso já tenha respondido mais de 15 respostas.

A métrica pode ser definida pela fórmula a seguir:

$$m = \frac{N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11}{11}$$

Em que, *m* é o valor atribuído à uma determinada resposta. As respostas com os valores mais próximos de 1 possuem uma melhor qualidade e são as utilizadas por Laura para

responder aos usuários. Essa métrica foi aplicada à todas as respostas ligadas à perguntas de domínio Java presentes na base de conhecimento, somando ao todo 8.378 respostas. O menor valor do *rank* de uma resposta foi de 0.22. O valor médio do *rank* foi de 0.49. O maior valor do *rank* das respostas é de 0.78.

5.8 Experimentos com Laura

Nesta Seção são apresentados experimentos de interação com Laura. Foram realizadas algumas perguntas e as respostas dadas a estas perguntas foram analisadas. Todas as perguntas feitas, foram retiradas aleatoriamente da base de conhecimento de Laura. Todas as respostas dadas por Laura, para as perguntas dos experimentos, não possuíam palavras obscenas e também não são pessoais.

5.8.1 Experimento 1

- Pergunta:
 - Quais as diferenças de `public`, `default`, `protected` e `private` no Java?
- Resposta:
 - *Não existe nenhuma regra, somente boas práticas. Vamos do começo.*
Pra começar existem somente 3 modificadores (`private`, `protected` e `public`), e com isso temos 4 níveis de visibilidade. Os níveis são os que você disse: `private`, `default`, `protected` e `public`
Private: A única classe que tem acesso ao atributo é a própria classe que o define, ou seja, se uma classe Pessoa declara um atributo privado chamado nome, somente a classe Pessoa terá acesso a ele.
Default: Tem acesso a um atributo default (identificado pela ausência de modificadores) todas as classes que estiverem no mesmo pacote que a classe que possui o atributo.
Protected: Esse é o que pega mais gente, ele é praticamente igual ao default, com a diferença de que se uma classe (mesmo que esteja fora do pacote) estende da classe com o atributo `protected`, ela terá acesso a ele. Então o acesso é por pacote e por herança.
Public: Esse é fácil, todos tem acesso :)

nível de visibilidade envolve encapsulamento. É sempre dito como boa prática que atributos internos devem ser privados, pois classes externas nem devem saber que ele existe. O que a classe expõe são suas funcionalidades, sua API, se preferir. Expor atributos internos pode causar sérios problemas de segurança. Se tem algo que é inerente à implementação, que pode vir a mudar no futuro, provavelmente deve ser privado.

A resposta dada por Laura foi uma resposta bastante completa e clara. Esta resposta conseguiu responder a intenção da pergunta feita, descrevendo as diferenças entre os modificadores Java. O resultado também mostra que a resposta dada foi um tanto quanto longa. Isso pode ser um problema, pois pode fazer com que o usuário fique desestimulado em ler toda a resposta. Comparada as outras respostas candidatas para esta pergunta na base, esta foi a que melhor respondeu.

5.8.2 Experimento 2

- Pergunta:
 - Por que existem as anotações de controllers e services?
- Resposta:
 - *Ao contrário da anotação genérica @Component, as anotações específicas marcam as classes com estereótipos, assim como na UML. Dessa forma, se uma classe é anotada com @Service você pode pressupor que ela contém regras de negócio, se a anotação é @Repository é óbvio que a classe implementa o pattern Repository (não é igual ao DAO, mas chega a ser parecido) e se a anotação é @Controller você também pode associar diretamente a um controlador do modelo MVC.*

Neste experimento, a resposta dada por Laura foi bastante objetiva e clara. Conseguindo responder a pergunta realizada. Uma característica importante dessa resposta, é que ela não foi longa mas conseguiu exemplificar de maneira muito clara o que foi questionado pela pergunta recebida.

5.8.3 Experimento 3

- Pergunta:
 - Como faço para remover acentos de uma String no Java?
- Resposta:

– *Eu costumo usar regex junto com a classe Normalizer. Assim:*

```
public static String removerAcentos(String str) { return Normalizer.normalize(str,
Normalizer.Form.NFD).replaceAll(
p{ASCII}]; ” ”); }
```

Neste experimento, a resposta dada a pergunta não envolveu apenas texto simples, mas também, códigos. Isso é importante para aumentar a qualidade da resposta, pois um exemplo dado pode ajudar bastante ao aluno que está com dúvida. No entanto, a resposta conter código não interfere no cálculo do seu *rank*. A resposta foi bastante clara, não sendo longa e ainda possuiu um exemplo claro de aplicação, resultando em uma ótima resposta para a pergunta dada.

5.8.4 Experimento 4

- Pergunta:

- Quais as bibliotecas para API REST em Java?

- Resposta:

- *Existem várias opções para trabalhar com REST em Java. Existe uma implementação do Spring Framework (Spring MVC) e o padrão JEE JAX-RS, que conta com diversas implementações, dentre as quais as que vejo sendo mais utilizadas são Jersey e RESTEasy. Para ver um exemplo de como utilizar o Spring MVC para implementar REST veja este guia. Veja também o User Guide do Jersey, nele há toda a documentação necessária. Algo que você deve considerar é seu ambiente de produção. Como o JBoss Application Server vem com o RESTEasy, faz sentido você adotar esta implementação caso use JBoss, pois vai facilitar a sua vida. Eu já implantei uma aplicação com Jersey num JBoss AS 7.1 e posso dizer que não foi sem dificuldades, a ponto de precisar alterar arquivos de configuração dos módulos do container. Por outro lado, se seu servidor de aplicação é o Glassfish, será mais fácil ir com o Jersey que, assim como o Glassfish, é a implementação de referência.*

Pode-se perceber a partir do resultado deste experimento, que a resposta dada foi bastante útil para responder a pergunta feita. A resposta trouxe vários exemplos e ainda explicou um pouco sobre os exemplos, tornando uma resposta bem completa. No entanto, a resposta foi um pouco longa. E isto pode ser um problema para o usuário que pode ficar desestimulado em lê-la por completo.

5.8.5 Experimento 5

- Pergunta:
 - Primefaces DataTable Lazy, tabela sem apresentar registros?
- Resposta:
 - *Resolvido! Faltava setar a quantidade total de registros na propriedade RowCount.*

ChamadoLazyList

```
public ChamadoLazyList() { this.setRowCount(new ChamadoDao().getQtChamados()); }
```

Neste experimento, a resposta escolhida por Laura foi uma resposta que foi respondida pelo próprio autor da pergunta da base mais semelhante à pergunta recebida. Por ser uma resposta que foi feita pelo próprio autor da pergunta, ela tanto possui a característica de ser aceita como correta pelo autor, quanto a de que foi a respondida por ele mesmo. Isso implicou em uma boa nota para esta resposta.

5.8.6 Experimento 6

- Pergunta:
 - Quais as vantagens das Expressões Lambda presentes no Java 8?
- Resposta:
 - *Com Java 8 você pode resolver utilizando expressão lambda para filtrar apenas os registros em que a cidade está contida dentro do array de filtros de cidade.*

Neste experimento, a resposta escolhida por Laura foi uma resposta que não respondeu a pergunta feita. A resposta menciona sobre Java 8 e sobre lambda, porém, não fala sobre as vantagens em se utilizar. Pelo contrário, é uma resposta específica demais, pois fala sobre filtrar registros.

5.8.7 Experimento 7

- Pergunta:
 - Configurar Jenkins para gerar build a cada commit?
- Resposta:
 - *Se os arquivos estavam com status "modificado", eles já existiam no index do Git, o que é uma boa notícia. Você pode recuperar todos as revisões de arquivos que*

foram salvas mas abandonadas com o comando: git fsck --lost-found. Esse comando cria duas pastas dentro do seu diretório .git: .git/lost-found/commits e .git/lost-found/other e salva nessas pastas todos os arquivos que entraram no histórico do GIT quando você salvou, mas que não foram utilizados, não tem relação no índice atual do projeto. Dependendo de há quanto tempo essa cópia local vem sendo usada isso pode ser algumas centenas de arquivos, mas é grande a chance de você recuperar pelo menos uma parte do seu trabalho ali.

Neste experimento, a resposta escolhida por Laura foi uma resposta que não respondia a pergunta feita. A resposta dada remete à alguma pergunta sobre Git. Logo, Laura falhou ao responder a pergunta com uma resposta de domínio diferente. Isso pode ter ocorrido pelo fato da pergunta possuir a palavra *commit*, indicando a possibilidade da pergunta ter sido clusterizada em um *cluster* sobre Git e não sobre Jenkins.

6 CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentado o desenvolvimento do *Chatterbot* Laura, com o objetivo de responder perguntas sobre programação Java. Laura pode ser uma possível ferramenta para solucionar o problema da falta da disponibilidade na aprendizagem interativa. Uma das vantagens em se utilizar Laura para responder perguntas sobre programação Java, é a sua disponibilidade. Mesmo existindo os professores e monitores para auxiliar os alunos em suas dúvidas, eles geramente são limitados aos horários de aula e de monitoria, diferentemente de Laura.

Outra vantagem, é sua a base de conhecimento. Laura possui uma base de conhecimento muito rica, criada a partir de técnicas de mineração de dados na base do fórum *StackOverflow*. Laura também possui uma arquitetura multiagente autônoma e distribuída. Isso é muito importante para a realização dos processos necessários para a extração contínua e automática de conhecimento. Bem como, a realização dos processos necessários para a interação com os usuários.

Com os experimentos realizados, pode-se concluir que Laura conseguiu responder bem as perguntas recebidas. Com respostas bem classificadas, claras, sem palavras obscenas e bem votadas. Porém, ela errou em duas perguntas, respondendo-as com respostas que não respondiam o que foi perguntado.

Se faz necessário um processamento textual nas respostas, a fim de que as mesmas não fiquem muito longas e não desestimulem os usuários. Também se faz necessária uma validação com os usuários, para que a interação com Laura seja analisada a fim de verificar sua aceitação entre os alunos.

Laura aprende somente novas perguntas, clusterizando-as no momento em que as recebe dos usuários. No entanto, seria interessante que Laura pudesse aprender novas respostas com os usuários. Portanto, como trabalho futuro é necessário realizar o desenvolvimento de um módulo de aprendizagem de máquina para que Laura possa aprender novas respostas a partir das interações com os usuários.

Como outro trabalho futuro, sugere-se a utilização das *tags* para particionar os tópicos java. Também como trabalho futuro, se faz necessário validar a métrica de classificação de respostas criada a fim de se provar sua efetividade no ranqueamento de respostas. Seria interessante também, realizar o ranqueamento das respostas levando-se em consideração a pergunta que está sendo feita, a fim de se comparar os valores obtidos e a qualidade das

respostas.

REFERÊNCIAS

- ABBAS, O. A. Comparisons between data clustering algorithms. **International Arab Journal of Information Technology (IAJIT)**, v. 5, n. 3, 2008.
- AROCKIAM, L.; BASKAR, S.; JEYASIMMAN, L. Clustering techniques in data mining. **Asian Journal of information Technology**, v. 11, n. 1, p. 40–4, 2012.
- BATISTA, A. F.; MARIETTO, M. G.; BARBOSA, G. C.; FRANÇA, R. S.; NORONHA, E. A. Multi-agent systems in a computational environment of education: A chatterbot case study. **International Journal for Infonomics (IJI)**, v. 3, p. 3, 2010.
- BOSU, A.; CORLEY, C. S.; HEATON, D.; CHATTERJI, D.; CARVER, J. C.; KRAFT, N. A. Building reputation in stackoverflow: an empirical investigation. In: IEEE PRESS. **Proceedings of the 10th Working Conference on Mining Software Repositories**. [S.l.], 2013. p. 89–92.
- BRITO, F. N. **Desenvolvimento de um Chatterbot para a página web de um curso de nível superior**. Monografia (Graduação em Engenharia de Software) — Universidade Federal do Ceará, Campus de Quixadá, Quixadá, 2017.
- CHARLTON, P.; LUCKIN, R. Time to reload? computational thinking and computer science in schools. **What researches says**, 2012.
- HUANG, A. Similarity measures for text document clustering. In: **Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand**. [S.l.: s.n.], 2008. p. 49–56.
- HUANG, J.; ZHOU, M.; YANG, D. Extracting chatbot knowledge from online discussion forums. In: **IJCAI**. [S.l.: s.n.], 2007. v. 7, p. 423–428.
- JAIN, A. K. Data clustering: 50 years beyond k-means. **Pattern recognition letters**, Elsevier, v. 31, n. 8, p. 651–666, 2010.
- JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. **ACM computing surveys (CSUR)**, Acm, v. 31, n. 3, p. 264–323, 1999.
- KODINARIYA, T. M.; MAKWANA, P. R. Review on determining number of cluster in k-means clustering. **International Journal**, v. 1, n. 6, p. 90–95, 2013.
- LEE, G. G.; SEO, J.; LEE, S.; JUNG, H.; CHO, B.-H.; LEE, C.; KWAK, B.-K.; CHA, J.; KIM, D.; AN, J. et al. Siteq: Engineering high performance qa system using lexico-semantic pattern matching and shallow nlp. In: **TREC**. [S.l.: s.n.], 2001.
- LEISCH, F. A toolbox for k-centroids cluster analysis. **Computational statistics & data analysis**, Elsevier, v. 51, n. 2, p. 526–544, 2006.
- NIRANJAN, M.; SAIPREETHY, M.; KUMAR, T. An intelligent question answering conversational agent using naïve bayesian classifier. In: IEEE. **Technology Enhanced Education (ICTEE), 2012 IEEE International Conference on**. [S.l.], 2012. p. 1–5.
- OCHI, L. S.; DIAS, C. R.; SOARES, S. S. F. Clusterização em mineração de dados. **Instituto de Computação-Universidade Federal Fluminense-Niterói**, 2004.

- PONZANELLI, L.; BAVOTA, G.; PENTA, M. D.; OLIVETO, R.; LANZA, M. Mining stackoverflow to turn the ide into a self-confident programming prompter. In: ACM. **Proceedings of the 11th Working Conference on Mining Software Repositories**. [S.l.], 2014. p. 102–111.
- ROKACH, L.; MAIMON, O. Clustering methods. In: **Data mining and knowledge discovery handbook**. [S.l.]: Springer, 2005. p. 321–352.
- SHAWAR, B. A.; ATWELL, E. Chatbots: are they really useful? In: **LDV Forum**. [S.l.: s.n.], 2007. v. 22, n. 1, p. 29–49.
- SHEIKH, Y. A.; KHAN, E. A.; KANADE, T. Mode-seeking by medoidshifts. In: IEEE. **Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on**. [S.l.], 2007. p. 1–8.
- STOILLOS, G.; STAMOU, G.; KOLLIAS, S. A string metric for ontology alignment. **The Semantic Web–ISWC 2005**, Springer, p. 624–637, 2005.
- TAN, J. T. C.; DUAN, F.; INAMURA, T. Multimodal human-robot interaction with chatterbot system: Extending aiml towards supporting embodied interactions. In: IEEE. **Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on**. [S.l.], 2012. p. 1727–1732.
- TEIXEIRA, S.; RAMIRO, T. B.; OLIVEIRA, E. de; MENEZES, C. S. de. Chatterbots em ambientes de aprendizagem—uma proposta para a construção de bases de conhecimento. In: **Anais do Workshop de Informática na Escola**. [S.l.: s.n.], 2005. v. 1, n. 1.
- VELMURUGAN, T.; SANTHANAM, T. Computational complexity between k-means and k-medoids clustering algorithms for normal and uniform distributions of data points. **Journal of computer science**, v. 6, n. 3, p. 363, 2010.
- WEIZENBAUM, J. Eliza—a computer program for the study of natural language communication between man and machine. **Communications of the ACM**, ACM, v. 9, n. 1, p. 36–45, 1966.
- WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.
- XU, X.; ZHONG, Y.; ZHANG, L. Adaptive subpixel mapping based on a multiagent system for remote-sensing imagery. **IEEE Transactions on Geoscience and Remote sensing**, IEEE, v. 52, n. 2, p. 787–804, 2014.