UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE SISTEMAS DE INFORMAÇÃO

Bruno Budel Rossi

SISTEMA INTELIGENTE DE BUSCA DE ISSUES EM REPOSITÓRIOS DE CÓDIGO

Santa Maria, RS
2023

Bruno Budel Rossi

SISTEMA INTELIGENTE DE BUSCA DE ISSUES EM REPOSITÓRIOS DE CÓDIGO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Bacharel em Sistemas de Informação.**

Orientadora: Profª. Drª. Lisandra Manzoni Fontoura

Santa Maria, RS
2023

Bruno Budel Rossi

SISTEMA INTELIGENTE DE BUSCA DE ISSUES EM REPOSITÓRIOS DE CÓDIGO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Bacharel em Sistemas de Informação.**

Lisandra Manzoni Fontoura, Drª. (UFSM)
(Presidente/Orientadora)

Celio Trois, Dr. (UFSM)

Giliane Bernardi, Drª. (UFSM)

Santa Maria, RS
2023

AGRADECIMENTOS

Agradeço primeiramente a minha mãe, Rozemari Budel Rossi e pai, Nelso Rossi, por todo apoio e suporte prestado durante a graduação. Obrigado por acreditarem em mim e me apoiarem nos momentos em que mais precisei, sem vocês ao meu lado, nada disso seria possível.

Agradeço a minha namorada, Eliane Fischborn, por todo suporte, compreensão e por acreditar na minha capacidade em todos os momentos. Com seu apoio tudo foi mais leve e agradável.

Agradeço a minha orientadora, professora Lisandra, que além de ter me orientado neste trabalho, também me deu suporte na realização de outras pesquisas durante a graduação. Obrigado por todo suporte, paciência, dedicação e leveza com que conduz seu trabalho.

Ao meu amigo, Evandro Luis Filho, por suas orientações e por não medir esforços para me auxiliar, sua ajuda foi fundamental para conclusão deste trabalho.

A todos os meus amigos e colegas que fizeram parte desta jornada ao meu lado, obrigado por todas as colaborações e momentos de alegria. Espero poder levar essas amizades por toda vida.

Aos demais professores do Curso de Sistemas de Informação que sempre mostram dedicação e comprometimento em transmitir seus conhecimentos. Cada ensinamento será levado para toda a vida.

Ao Programa de Educação Tutorial do curso de Sistemas de Informação e todos os colegas que fizeram parte do grupo junto comigo. Todos os momentos divididos ao lado de vocês foram de alegria e aprendizado.

Aos participantes desta pesquisa por compartilharem seu tempo durante a validação deste trabalho.

RESUMO

SISTEMA INTELIGENTE DE BUSCA DE ISSUES EM REPOSITÓRIOS DE CÓDIGO

AUTOR: Bruno Budel Rossi
ORIENTADORA: Profª. Drª. Lisandra Manzoni Fontoura

A preocupação em como desenvolvedores de software usam seu tempo e as tentativas de automatizar processos manuais têm tomado uma proporção maior a cada ano. Em Engenharia de Software, sistemas de recomendação auxiliam desenvolvedores a encontrar informações que se deve conhecer, avaliar alternativas e auxiliar na navegação de dados. Atualmente, repositórios de código compartilhado com base em Git fazem parte do dia a dia de grande parte dos desenvolvedores de software. Estes repositórios disponibilizam várias funcionalidades aos seus usuários, dentre essas, as *issues* são uma maneira de documentar problemas, solicitar alguma documentação, prover *feedbacks* e rastrear o fluxo de trabalho sobre um problema, entretanto, repositórios de código ativos tendem a gerar muitas issues, o que por sua vez acaba tornando a tarefa de recuperar *issues* relevantes algo mais custoso aos seus usuários. A proposta deste trabalho é a elaboração de um sistema de recomendação capaz de analisar *issues*, utilizando aprendizado de máquina não supervisionado, através do algoritmo de processamento de linguagem natural Word2Vec, com objetivo de facilitar a busca de *issues* e auxiliar na tomada de decisão a partir de *issues* armazenadas em um repositório de código. Para atingir este objetivo, foi necessário extrair as *issues* do repositório de código, realizar o pré-processamento dos campos previamente definidos, criar dicionários de *unigram*, *bigram* e *trigram* e, por fim, realizar o treinamento da rede neural utilizada pelo algoritmo Word2Vec, utilizando como entrada os dicionários criados anteriormente. Após o modelo já treinado, o usuário será capaz de buscar *issues* através de uma entrada de dados, esta entrada pode conter uma, duas ou três palavras e deve obter como retorno um conjunto de *issues* mais similar semanticamente com a entrada. As avaliações mostraram que o sistema de recomendação realiza sugestões relevantes, auxilia na navegação e facilita a tomada de decisão no âmbito de issues em repositórios de código.

Palavras-chave: *Issues*. Sistemas de recomendação. Processamento de Linguagem Natural. Aprendizado de Máquina Não Supervisionado. Redes Neurais.

ABSTRACT

INTELLIGENT SYSTEM FOR SEARCHING ISSUES IN CODE REPOSITORY

AUTHOR: Bruno Budel Rossi
ADVISOR: Profª. Drª. Lisandra Manzoni Fontoura

The concern about how software developers use their time and attempts to automate manual processes have taken on a greater proportion every year. In Software Engineering, recommender systems help developers find information that should be known, evaluate alternatives and aid data navigation. Currently, Git-based shared code repositories are part of everyday life for most software developers. These repositories make several functionalities available to their users, among them, issues are a way to document problems, request some documentation, provide feedback and track the workflow on a problem, however, active code repositories tend to generate many issues, which which in turn ends up making the task of recovering relevant issues something more costly for its users. The purpose of this work is the elaboration of a recommendation system capable of analyzing issues, using unsupervised machine learning, through the Word2Vec natural language processing algorithm, with the objective of facilitating the search for issues and assisting in decision making based on stored issues. in a code repository. To achieve this goal, it was necessary to extract the issues from the code repository, pre-process the previously defined fields, create unigram, bigram and trigram dictionaries and finally train the neural network used by the Word2Vec algorithm, using as input previously created dictionaries. After the model has already been trained, the user will be able to search for issues through a data entry, this entry may contain one, two or three words and should return a set of issues more semantically similar to the entry. The evaluations showed that the recommendation system makes relevant suggestions, helps with navigation and facilitates decision-making regarding issues in code repositories.

Keywords: Issues. Recommender System. Natural Language Processing. Unsupervised Machine Learning. Neural Network.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CI	*Integrated Circuit*
CI/CD	*Continuous Integration/Continuous Delivery*
REST	*Representational State Transfer*
NLP	*Natural Language Processing*

SUMÁRIO

1. INTRODUÇÃO

Atualmente, grande parte das organizações utiliza repositórios de código compartilhado para armazenar seus códigos, estes repositórios tipicamente utilizam *issues* para lidar com problemas. Esta por sua vez, permite com que o colaborador acompanhe o desenvolvimento, deixe algum *feedback* ou documentação do projeto, rastreie todo o fluxo de trabalho em que uma *issue* está envolvida, além disso, *issues* são capazes de melhorar a comunicação entre seus colaboradores, com funcionalidades como menção a um colaborador ou até mesmo outra *issue* (GITHUB, 2022). Ferramentas de código compartilhado tipicamente possuem algumas funcionalidades que auxiliam o uso de *issues*, como maneiras de classificá-las, por exemplo, as *labels,* através delas, é possível filtrar as *issues*, além disso, é comum a adição de imagens, links e trechos de código, possibilitando um maior entendimento do problema.

Repositórios de software como o Git tornaram-se fonte relevante de informação para desenvolvedores de software. Por meio da análise de *issues,* do projeto atual ou projetos passados, pode-se obter informações pertinentes para a avaliação de um novo requisito, mudanças em requisitos ou até mesmo reuso de soluções adotadas para resolução de problemas similares que podem auxiliar em novas implementações. *Issues* possuem grande impacto no sucesso da comunicação em um repositório Git, além de impactar positivamente na quantidade de *forks* e estrelas de favorito de um repositório, isto se deve a capacidade que as *issues* possuem de documentar conhecimento dos membros da equipe e de sua comunidade (BRISSON; NOEI; LYONS, 2020).

Os desenvolvedores são continuamente apresentados a novas tecnologias, componentes e ideias. Os sistemas em que trabalham têm mais código e dependem de bibliotecas maiores, ou seja, dominar uma linguagem de programação não é mais suficiente para garantir a proficiência no desenvolvimento de software, os desenvolvedores também devem aprender a navegar por grandes bases de código e bibliotecas de classes (ROBILLARD; WALKER; ZIMMERMANN, 2010)**.** O processo de desenvolvimento de software pode ser visto como um processo emergente porque muitas ações, como as *issues,* por exemplo, surgem durante o processo como resultado da disponibilidade de informações adicionais, uma situação mitigada em processos ágeis de desenvolvimento de software (ABRANTES; TRAVASSOS, 2011).

Autores sugerem que a Engenharia de Software necessita de técnicas e ferramentas automatizadas, adaptabilidade e escalabilidade para acompanhar o aumento da produtividade

do desenvolvedor (BENNACEUR; MEINKE, 2018). Em Engenharia de Software, os sistemas de recomendação auxiliam desenvolvedores a encontrar informações que se deve conhecer, avaliar alternativas e auxiliar na navegação em grandes quantidades de dados (ROBILLARD; WALKER; ZIMMERMANN, 2010). Sistemas de recomendação tomaram cada vez mais lugar em nossas vidas, estas ferramentas de software são capazes de fornecer sugestões com base nas necessidades de um usuário, estas técnicas são usadas em *e-commerce* como o da Amazon, vídeos como o YouTube, *streaming* como Netflix ou mídias sociais como Facebook (DAS; SAHOO; DATTA, 2017). Neste cenário, um sistema de recomendação de *issues* deve ser capaz de melhorar o desempenho de desenvolvedores e auxiliar os mesmos na tomada de decisões, busca sobre histórico do projeto e documentação de problemas do projeto. Um sistema de recomendação, no que lhe concede, é uma ferramenta que utiliza uma série de algoritmos, análise de dados e, principalmente, *machine learning* para coletar informações e realizar suas recomendações.

O *machine learning* usa computadores para simular a aprendizagem humana e permite que eles identifiquem e adquiram conhecimento do mundo real, além de melhorar o desempenho de algumas tarefas com base nesse novo conhecimento (PORTUGAL; ALENCAR; COWAN, 2018). Mikolov *et al*. (2013), perceberam o crescimento destas técnicas e sua capacidade em analisar grandes quantidades de dados, com isso muitas possibilidades podem ser exploradas, como a construção de um sistema de recomendação capaz de analisar *issues* em grandes repositórios de código.

O sistema de recomendação desenvolvido utiliza o algoritmo de processamento de linguagem natural Word2Vec, uma representação distribuída de palavras aprendidas por redes neurais. Esta técnica busca fugir da camada oculta não linear de um modelo tradicional de redes neurais e foca em treinar uma rede neural mais simples, capaz de processar grandes quantidades de dados (MIKOLOV *et al*., 2013). A ideia central desta técnica consiste em fugir da complexidade causada pela camada oculta não linear de um modelo tradicional em redes neurais e focar em treinar uma rede neural rasa, capaz de representar os dados com a mesma precisão das redes neurais mais complexas, mas com a possibilidade de ser treinada com uma quantidade de dados consideravelmente maior.

A proposta deste trabalho é a elaboração de um sistema de recomendação capaz de analisar *issues*, utilizando aprendizado de máquina não supervisionado, através do algoritmo de NLP Word2Vec, com objetivo de facilitar a busca de *issues* e auxiliar na tomada de decisão

que envolva *issues* em um repositório de código. Para atingir este objetivo, é necessário extrair as *issues* do repositório de código, realizar o pré-processamento dos campos previamente definidos, criar dicionários de *unigram*, *bigram* e *trigram*, e por fim, realizar o treinamento da rede neural utilizada pelo algoritmo Word2Vec, utilizando como entrada os dicionários criados anteriormente. Após o modelo já treinado, o usuário será capaz de buscar *issues* através de uma entrada de dados, esta entrada pode conter uma, duas ou três palavras e deve obter como retorno um conjunto de *issues* mais similar semanticamente com a entrada.

Este trabalho está estruturado da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica, abordando conceitos como repositórios de código, algoritmos de aprendizado de máquina, redes neurais, processamento textual e sistemas de recomendação. O Capítulo 3 apresenta uma análise crítica dos trabalhos relacionados. O Capítulo 4 apresenta como o sistema de recomendação de *issues* foi desenvolvido. O Capítulo 5 explica como foi realizada a validação do trabalho através de um experimento controlado, bem como os resultados obtidos neste estudo. Por fim, no Capítulo 6 é realizada a conclusão do trabalho e descrito os trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos usados para o desenvolvimento deste trabalho. Na Seção 2.1 são definidos os conceitos de Git, repositório de código compartilhado e *issues.* Na Seção 2.2 são definidas as diferenças entre aprendizagem de máquina supervisionadas e não supervisionadas. Em seguida, na Seção 2.3 é definido o conceito de redes neurais e como sua arquitetura é formulada. A Seção 2.4 apresenta o conceito de processamento de linguagem natural bem como algumas técnicas utilizadas neste trabalho para preparar os dados textuais para posterior uso pelo sistema de recomendação. A Seção 2.5 apresenta os principais conceitos para o entendimento do algoritmo Word2Vec, bem como suas principais arquiteturas. Por fim, na Seção 2.6 são abordados os conceitos referentes a sistemas de recomendação, além das técnicas escolhidas para a implementação desta abordagem.

2.1. REPOSITÓRIOS DE CÓDIGO

É importante reconhecer que os repositórios de código compartilhado e os sistemas de controle de versão são duas entidades separadas. Git é um sistema de controle de revisão distribuído disponível em todas as principais plataformas de desenvolvimento por meio de uma licença de software livre. Os repositórios de código compartilhado são provedores de hospedagem de repositório Git que simplificam muitas tarefas de gerenciamento de repositório por meio de uma interface de usuário baseada na Web, ao mesmo tempo, em que promovem a cooperação em projetos de código aberto (SPINELLIS, 2012). Não é possível utilizar serviços de hospedagem de repositórios de código sem um sistema subjacente de controle de versão.

Git é uma ferramenta de controle de versionamento grátis e de código aberto, projetado para lidar com todo tipo de projeto, desde projetos pequenos a muito grandes em termos de velocidade e eficiência (GIT, 2022). O Git é um projeto de código aberto maduro e com manutenção ativa desenvolvido em 2005 por Linus Torvalds, criador do kernel do sistema operacional Linux. Um número impressionante de projetos de software depende do Git para controle de versão, incluindo projetos comerciais e de código-fonte aberto. A capacidade de associar um repositório local a vários repositórios remotos permite que os desenvolvedores e seus gerentes criem fluxos de trabalho distribuídos (SPINELLIS, 2012), deste modo,

possibilitando um desenvolvimento em paralelo, sendo essencial para muitos times de desenvolvimento de software, logo, os desenvolvedores que dominam esta ferramenta, possuem destaque no desenvolvimento de software.

Tendo uma arquitetura distribuída, o Git é um exemplo de SCVD (Sistema de Controle de Versão Distribuído). Em vez de ter apenas um único local para o histórico completo da versão do software, conforme é comum em sistemas de controle de versão outrora populares como CVS ou *Subversion* (também conhecido como SVN), no Git, a cópia de trabalho de todo desenvolvedor do código também é um repositório que pode conter o histórico completo de todas as alterações. Além de ser distribuído, o Git foi projetado com desempenho, segurança e flexibilidade em mente (GIT, 2022).

O maior e mais famoso serviço de hospedagem atualmente é o GitHub, ele é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o Git. Ele permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribuam em projetos privados e/ou *Open Source* de qualquer lugar do mundo, como mais de 80 milhões de desenvolvedores, o GitHub abriga mais de 200 milhões de projetos (GITHUB, 2022).

Além do Github, existem inúmeros repositórios de código compartilhado ganhando popularidade e outros já consolidados, dentre as funcionalidades destes serviços, é possível criar *branches* de desenvolvimento, gerenciar *pull requests,* criar ferramentas de integração, entre outras. Existem repositórios de código compartilhado já consolidados, como o GitLab, que se destaca por oferecer uma gama de serviços com foco em DevOps, integração contínua e entrega contínua (CI/CD), entre outros (GITLAB, 2022).

Outra importante funcionalidade são as *issues*, essa por sua vez, permite aos desenvolvedores escrever mensagens em repositórios de código, com objetivo de prover *feedbacks* ou relatar *bugs*. *Issues* podem conter textos, imagens, links, métodos de menção e marcação, entre outros, facilitando a comunicação entre desenvolvedores. Além disso, as *issues* são uma maneira que programadores avaliam a expressão e qualidade que um repositório possui, já que repositórios que possuem uma maior atenção com suas *issues* tendem a ser mais confiáveis e atualizados para se basear no desenvolvimento. Outra importante função das *issues* é servir como documentação, seja como uma menção em um *pull request* ou para relatar lições aprendidas, deste modo, auxiliando outros desenvolvedores a criar ou buscar *issues* em repositórios (WANG *et al.,* 2022).

2.2. APRENDIZADO DE MÁQUINA

Sistemas inteligentes são sistemas computacionais que são capazes de processar dados de entrada e entender e ajustar padrões, com objetivo de otimizar seus resultados de saída, conforme os objetivos esperados para aquele algoritmo. Em 1959, Arthur Samuel definiu aprendizado de máquina ou *machine learning* em inglês, como o campo de estudo que possibilita aos computadores aprenderem sem serem explicitamente programados (SIMON, 2013). Neste cenário, o aprendizado de máquina é um método de análise de dados que automatiza a construção de modelos analíticos, focando no treinamento desses algoritmos para melhorar seu desempenho e, consequentemente, seus resultados. Esse processo está ligado com a redução de dimensionalidade, classificação e associação dos dados e previsão de comportamentos.

Algoritmos de aprendizado de máquina são classificados em dois principais segmentos, os chamados supervisionados, que necessitam de uma supervisão para melhorar seus resultados e os não supervisionados, que fazem esse processo de maneira independente (RUSSELL; NORVIG, 2014). Nesta Seção são apresentados esses dois tipos de algoritmos, especificando suas características e diferenças.

2.2.1. APRENDIZAGEM DE MÁQUINA SUPERVISIONADA

A aprendizagem supervisionada é um método de análise de dados que realiza o treinamento dos algoritmos de forma interativa com dados para os quais suas respostas já sejam conhecidas. Ou seja, sempre depende de um padrão de valores de entrada e da comparação das respostas do sistema com aquelas buscadas pelo algoritmo em questão. Conforme o algoritmo é treinado, seus padrões vão sendo ajustados a fim de diminuir o erro e otimizar as respostas. Os problemas solucionados através da aprendizagem supervisionada são divididos em problemas de regressão e classificação de dados, como exemplificado na Figura 2.1.

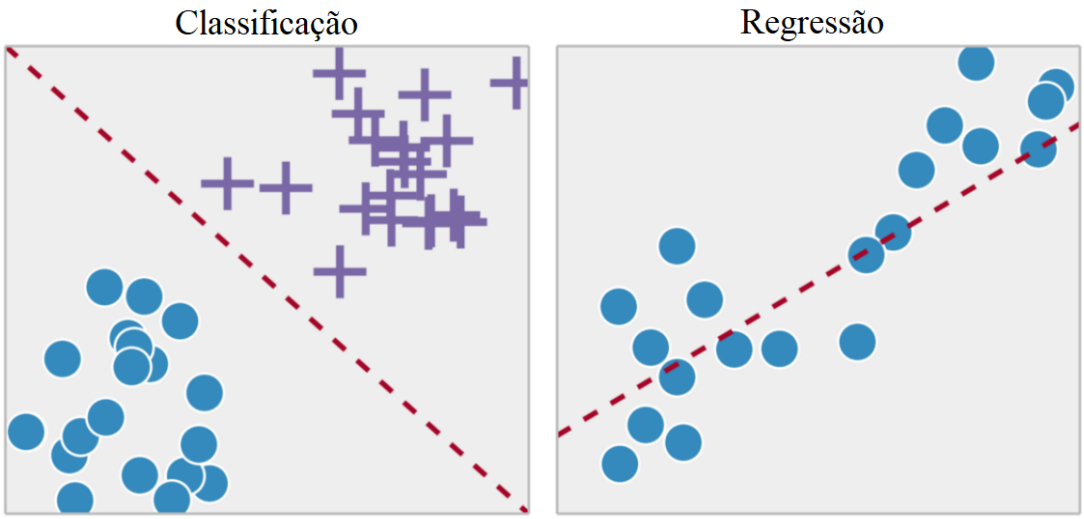
Quando o resultado esperado for de um conjunto finito de valores, como ensolarado, nublado ou chuvoso, o problema da aprendizagem será chamado de classificação, e será chamado de classificação booleana ou binária se houver apenas dois valores. Quando o

resultado esperado for um número, como temperatura de amanhã, o problema de aprendizagem é chamado de regressão (RUSSELL; NORVIG, 2014).

Nesta abordagem, o algoritmo usado é escolhido com base na classificação dos dados do problema, este, recebe entradas já categorizadas para realizar o treinamento, a cada interação, seus parâmetros são ajustados para otimizar os resultados da saída, que podem ter como objetivo, por exemplo, minimizar o erro, maximizar a precisão ou a acurácia. Após esta etapa, é tipicamente realizada uma etapa de validação, na qual o algoritmo é submetido a entradas sem classificação, buscando validar seus resultados e, se necessário, uma nova etapa de treinamento pode ser realizada, buscando melhorar a precisão do algoritmo.

A Figura 2.1 ilustra a diferença entre problemas de classificação e regressão.

Figura 2.1 - Problema de Classificação e Regressão.



Fonte: Produção do próprio autor.

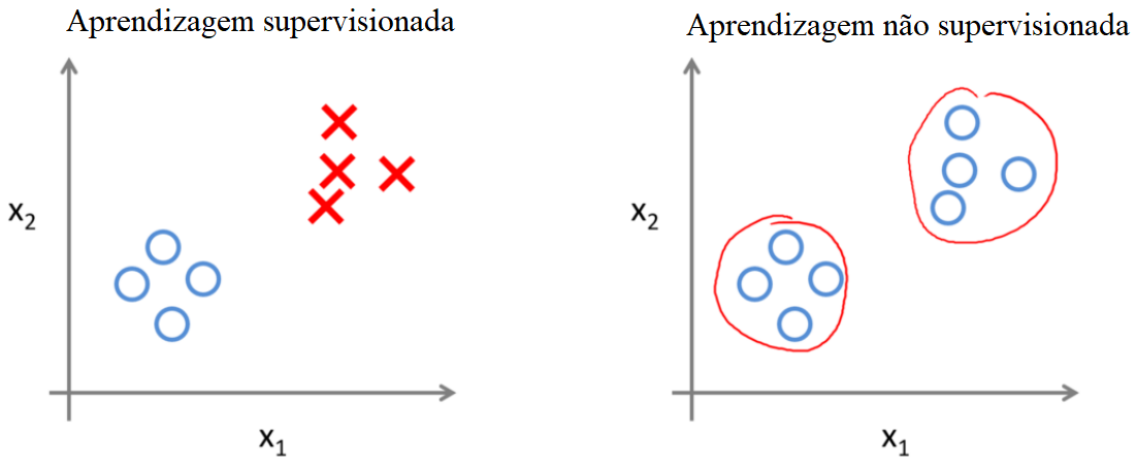
2.2.2. APRENDIZAGEM DE MÁQUINA NÃO SUPERVISIONADA

Na aprendizagem não supervisionada, o algoritmo aprende com os padrões de entrada, embora não seja fornecida nenhuma classificação explícita. A tarefa mais comum de aprendizagem não supervisionada é o agrupamento, ou seja, a detecção de grupos de exemplos de entradas potencialmente úteis. Por exemplo, um taxista pode desenvolver gradualmente um conceito de "dia de tráfego bom" e "dia de tráfego ruim", através de suas experiências, sem nunca ter aprendido formalmente estes conceitos por meio de um professor (RUSSELL; NORVIG, 2014).

Nesta abordagem, conforme os dados vão sendo recebidos, o próprio algoritmo é responsável por detectar relações e padrões presentes nos dados. A aprendizagem não supervisionada não prevê soluções específicas para realizar o treinamento e validação dos resultados, ou seja, não há um *feedback* explícito sobre os resultados previstos.

A Figura 2.2 abaixo ilustra a diferença entre a abordagem supervisionada e não supervisionada.

Figura 2.2 - Diferença entre as abordagens de aprendizado de máquina.



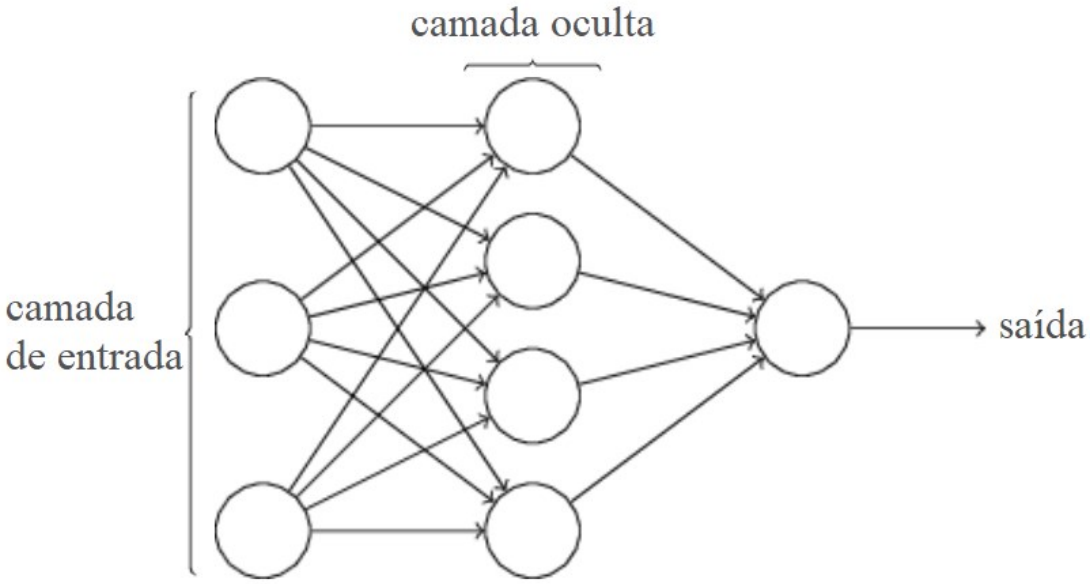
Fonte: Produção do próprio autor.

2.3. REDES NEURAIS

A Rede Neural é uma técnica de aprendizado de máquina que é inspirada e se assemelha ao sistema nervoso humano e a estrutura do cérebro, estas técnicas são capazes de realizar o aprendizado de máquina bem como o reconhecimento de padrões (ZHANG *et al*., 2019). Redes neurais consistem em unidades de processamento organizadas nas camadas de entrada, oculta e de saída. Os nós ou unidades em cada camada são conectados a nós em camadas adjacentes, cada conexão tem um valor de peso, o treinamento é realizado por meio de um processo iterativo de ajustes aplicado a seus pesos e o aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas (SHRESTHA; MAHMOOD, 2019). Redes neurais artificiais geralmente são apresentadas como sistemas de neurônios interconectados, que podem computar valores de entradas, simulando o comportamento de redes neurais biológicas.

A Figura 2.3 representa um exemplo de arquitetura de redes neurais, a camada mais à esquerda nesta rede é chamada de camada de entrada e os neurônios dentro da camada são chamados de neurônios de entrada. A camada mais à direita ou a saída contém os neurônios de saída ou, como neste caso, um único neurônio de saída. A camada do meio é chamada de camada oculta, já que os neurônios nessa camada não são de entradas ou saídas, a figura em questão possui apenas uma única camada oculta, mas algumas redes possuem múltiplas camadas ocultas.

Figura 2.3 - Arquitetura de uma Rede Neural.



Fonte: Produção do próprio autor.

As redes neurais podem ser usadas em uma variedade de problemas, incluindo reconhecimento de padrões, classificação, agrupamento, redução de dimensionalidade, visão computacional, processamento de linguagem natural, problemas de regressão, problemas de análise preditiva, entre outros. Já a implementação de redes neurais divide-se nos seguintes passos: adquirir e testar o conjunto de dados do treinamento; realizar o treinamento da rede neural e realizar previsões com dados de teste (SHRESTHA; MAHMOOD, 2019).

2.4. PROCESSAMENTO DE LINGUAGEM NATURAL

Processamento de linguagem natural (ou *natural language processing* em inglês) é uma das vertentes de inteligência artificial e um subcampo da linguística. Ele contém um conjunto de funcionalidades que ajudam o computador a entender, interpretar e manipular a linguagem humana e, em seguida, torná-las compreensíveis para a máquina (EISENSTEIN, 2019).

Embora abordar uma visão de texto completo (ou em inglês *full-text*) dos documentos seja possível, com coleções muito grandes de palavras pode ser interessante reduzir o conjunto de palavras-chave representativas. Isto pode ser feito eliminando-se as *stopwords*, como artigos e preposições, aplicando-se *stemming*, ou seja, reduzindo as palavras distintas a sua raiz

gramatical, removendo sinais de pontuação, espaços desnecessários e possíveis caracteres inesperados (BAEZA-YATES; RIBEIRO-NETO, 2013).

Um termo muito utilizado neste contexto é o *bag of words*, que nada mais é que uma forma de representar as palavras de um documento ou texto de acordo com sua ocorrência, ou seja, não considera a ordem ou a estrutura das palavras no documento, apenas se ela aparece ou a frequência com que aparece no documento (WITTEN *et al*., 2017).

Estas operações ou transformações aplicadas sobre o texto reduzem a complexidade e modificam a representação do texto inteiro para um conjunto de termos de indexação, consequentemente reduzindo o custo computacional (BAEZA-YATES; RIBEIRO-NETO, 2013). A etapa de processamento textual é de grande importância nos resultados obtidos por qualquer modelo de aprendizagem de máquina que use textos como base. Peter Norvig, escritor de renomados livros na área de inteligência artificial, disse que, “mais dados batem algoritmos inteligentes, mas dados melhores batem mais dados”. Ou seja, mesmo utilizando dos mais poderosos algoritmos de aprendizagem de máquina, os resultados não serão tão bons se os dados não forem significativos.

2.5 WORD2VEC

John Rupert Firth foi um linguista conhecido, que na década de 1950, dentre muitas obras, publicou contribuições ao estudo de contextualização do significado, seus estudos observaram que cada palavra é caracterizada, e até certo ponto definida, pelas outras palavras em seu entorno, ou seja, estudos sobre a importância de contexto em que uma palavra está inserida sempre foram realizados.

Word2Vec é uma técnica de NLP publicada por Mikolov *et al*. (2013), no qual eles observaram que com o progresso das técnicas de *machine learning*, foi possível treinar modelos mais complexos em um conjunto de dados muito maior e que essa abordagem normalmente supera modelos mais simples. A ideia central desta técnica consiste em fugir da complexidade causada pela camada oculta não linear de um modelo tradicional em redes neurais e focar em treinar uma rede neural rasa, capaz de representar os dados com a mesma precisão das redes neurais mais complexas, mas com a possibilidade de ser treinada com uma quantidade de dados consideravelmente maior. Esta técnica tem como proposta, analisar palavras não mais como

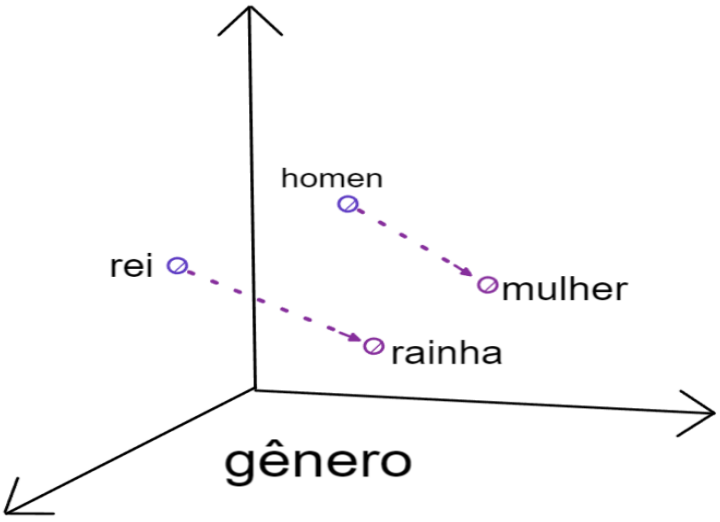
unidades atômicas, mas dentro de um contexto, possibilitando uma análise semântica dessas palavras.

Para compreender esta técnica é necessário definir o termo *embedding,* esta estrutura consiste em um mapa de vetores multidimensionais, no qual cada palavra é representada como um vetor numérico, este por sua vez possui um valor em escala de -1 a 1 para palavra ou dimensão do vocabulário, quanto mais próximo de -1 é este valor, mais próxima é esta palavra de outra em questão. Com essa abordagem tem-se liberdade com o tamanho dos *embeddings,* pois é possível escolher o tamanho dos vetores e observar uma relação semântica entre as palavras dentro de um contexto.

Word2Vec é um algoritmo que obtém vantagem sobre em grandes quantidades de dados, porém, quanto mais dados se obtém, maior será a dimensionalidade dos vetores de palavras, consequentemente levará mais tempo para treinar o algoritmo e melhor será os resultados de saída.

A Figura 2.4 demonstra um exemplo de *embedding*, em um contexto "gênero", podemos observar que palavras com certa similaridade semântica estão próximas neste contexto.

Figura 2.4 - Técnica de embedding em um contexto de "Gênero".



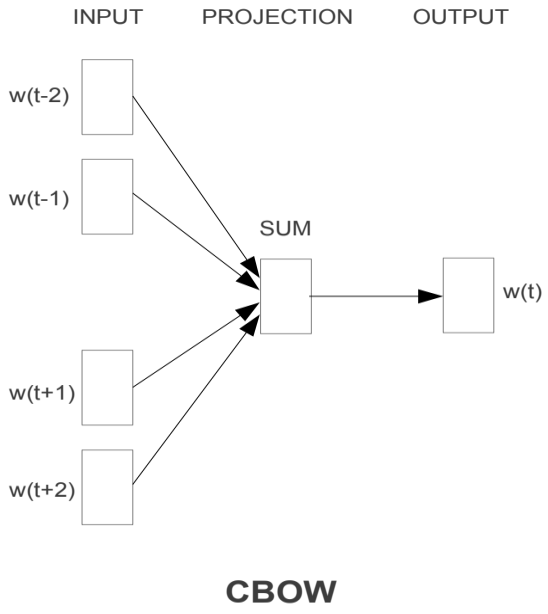
Fonte: Produção do próprio autor.

Mikolov *et al*. (2013) definiram dois principais modelos de redes neurais em sua arquitetura, sendo eles, CBOW e Skip-gram, ambos possuem uma camada de entrada, uma camada oculta e uma camada de saída.

Continuous bag of words (CBOW) é um modelo que busca descobrir a palavra central de uma sentença, baseando-se nas palavras que o cercam, neste modelo o treinamento da rede neural é feito percorrendo todo o espaço de palavras do vocabulário, a cada palavra observada a rede neural é treinada observando as demais palavras em sua volta, o raio de busca destas palavras pode ser variável e, consequentemente, influencia diretamente nos resultados e tempos de treinamento (MIKOLOV *et al.*, 2013).

A Figura 2.5 mostra o modelo CBOW representado, na qual na camada de entrada são recebidos os dados pré-processados e em sua saída terá uma palavra que representará a maior probabilidade para a palavra alvo.

Figura 2.5 - Funcionamento do modelo CBOW.

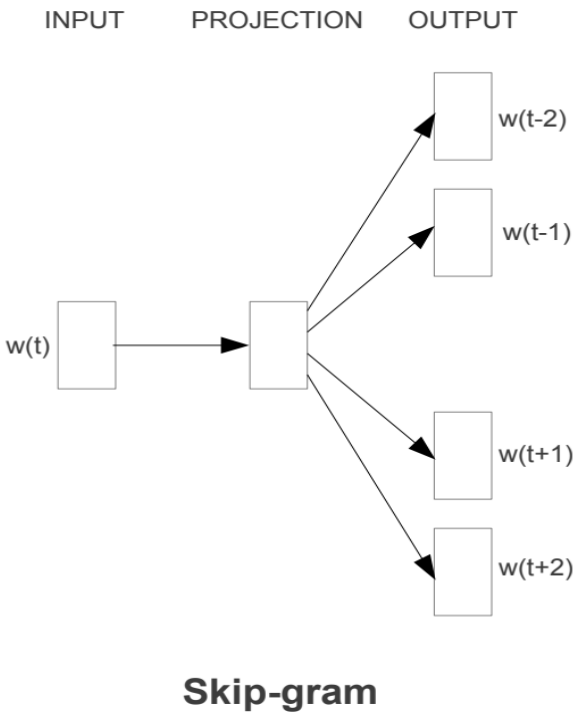


Fonte: MIKOLOV et al., 2013.

Skip-Gram é o segundo modelo proposto por Mikolov *et al*. (2013), este modelo pode ser facilmente comparado com o primeiro. Neste caso ao invés do modelo tentar descobrir a palavra central, o processo é o inverso, da palavra central, tentaremos descobrir as palavras de contexto, neste modelo o processo de treinamento é similar ao CBOW, porém o que era usado na camada de entrada deverá ser usado na camada de saída e vice-versa.

A Figura 2.6 mostra o modelo Skip-Gram representado, no qual a camada de entrada, temos apenas a palavra alvo, e em sua saída temos diferentes probabilidades, cada uma contendo palavras de contexto possíveis.

Figura 2.6 - Funcionamento do modelo Skip-gram.



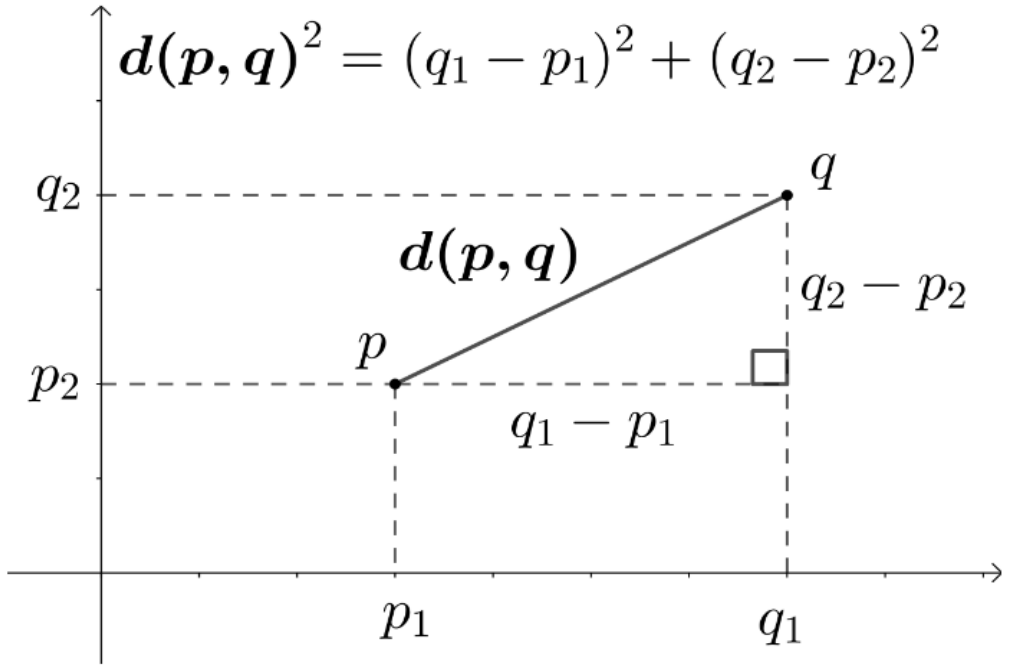
Fonte: MIKOLOV et al., 2013.

Outro fator importante sobre o Word2Vec é como este algoritmo calculado à similaridade entre vetores, uma das maneiras mais conhecidas de calcular a semelhança entre vetores é pela distância euclidiana, o qual é a forma padrão de calcular a distância entre eles (LAI, S. *et al*, 2016). Na figura 2.7 pode-se observar que a distância $d(p,q)$ corresponde a menor distância possível entre p e q, ou segmento de reta entre os dois pontos, logo sua fórmula é a aplicação do teorema de Pitágoras. A Equação 2.1 demonstra como calcular a menor distância possível entre dois vetores.

$$d(p,q) = \sqrt{(p1 - q1)^2 + (p2 - q2)^2}$$

(2.1)

Figura 2.7 - Distância euclidiana entre duas dimensões.



Fonte PATEL; UPADHYAY, 2020.

Para calcular a distância entre vetores de tamanho arbitrário n, generalizamos para a Equação 2.2, vale lembrar que, no contexto de *embeddings*, estaremos sempre comparando vetores de mesmo tamanho, ou seja, o valor escolhido para a dimensão do *embedding*.

$$d(p,q) = \sqrt{(p1 - q1)^2 + (p2 - q2)^2 + \cdots + (pi - qi)^2 + \cdots + (pn - qn)^2}$$

(2.2)

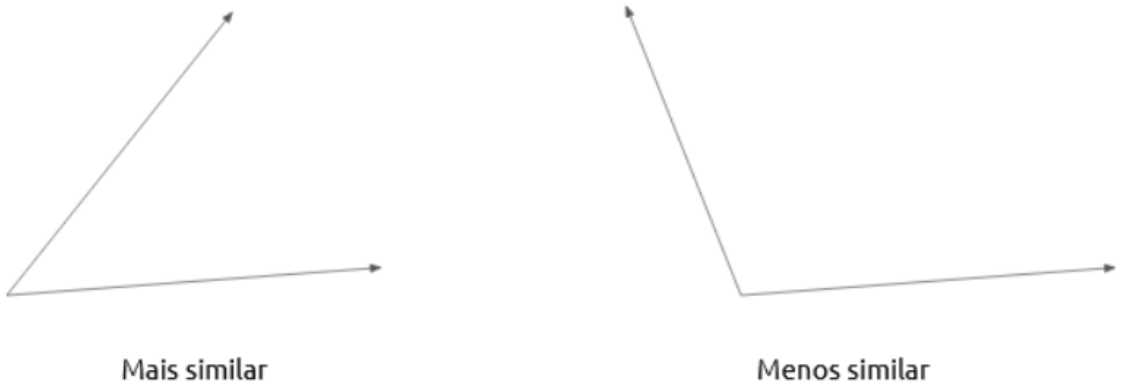
Observe que $p1, p2, \ldots, pn \; e \; q1, q2, \ldots, qn$ são as componentes dos vetores $p \; e \; q$, respectivamente. Por exemplo, p2 é o valor do vetor na linha 2. Diz-se, no entanto, que a distância euclidiana entre *embeddings* é uma medida de dissimilaridade, ou seja, quanto maior, mais dissimilar, pois esperamos que quanto mais distantes os vetores, menos semelhantes eles sejam.

No entanto, se tratando de vetores de palavras, a direção deles é um componente importante para a aquisição do significado, portanto, é mais interessante saber o ângulo entre dois vetores do que a distância entre eles, deste modo, a medida de similaridade mais usada é a similaridade de cossenos (LAI, S. *et al*, 2016). Sabe-se que o valor do cosseno entre dois ângulos é um número que varia entre -1 e 1, quando o ângulo entre eles se aproxima de 0°, o cosseno se aproxima de 1, já quando os vetores são ortogonais, ou seja, ângulo de 90°, o cosseno é igual a 0, e, por fim, quando os vetores são opostos, o cosseno se aproxima de -1. Assim, pode-se considerar que cosseno próximo de 1 indica uma grande similaridade entre os vetores, enquanto cosseno próximo de -1 indica o oposto (JATNIKA; BIJAKSANA; SURYANI, 2019). A Equação 2.3 representa a maneira de realizar esse cálculo.

$$similaridade = cos\ cos\ \emptyset = \frac{< a, b >}{||a||.||b||} \tag{2.3}$$

Quanto maior o ângulo entre dois vetores, valores entre 0º e 180º, menor o cosseno, ou seja, maior a similaridade. A Figura 2.8 exemplifica diferentes ângulos entre vetores.

Figura 2.8 - Ângulos em vetores.



Fonte: Produção do próprio autor.

2.6 SISTEMA DE RECOMENDAÇÃO

Um sistema de recomendação também pode ser tratado como um sistema de *machine learning*, ele tem como objetivo sugerir um item que possivelmente é relevante para determinado usuário, estes sistemas são ainda mais eficientes quando um usuário precisa escolher ou observar uma grande quantidade de dados. Esta abordagem desempenha uma função indispensável em diversos sistemas de informação, facilitando a tomada de decisão e potencializando negócios (ZHANG *et al*., 2019).

Em geral, os modelos de recomendação são gerados com base nas preferências do usuário, recursos dos itens, interações anteriores entre usuários e itens e algumas outras informações adicionais, como dados temporais, além disso, estes modelos são geralmente classificados em três categorias, sendo elas: filtragem colaborativa, filtragem baseada em conteúdo e filtragem híbrida (DAS; SAHOO; DATTA, 2017).

Na filtragem colaborativa realiza análises sobre as interações entre usuários e itens, seja por um *feedback* explícito, como classificações anteriores ou implícitos, como históricos de acesso. Já a filtragem baseada em conteúdo, por sua vez, busca trazer recomendações baseadas em características de usuários similares ou de itens já consumidos previamente por este usuário, estes itens podem ser vídeos, imagens, textos, etc. Os modelos híbridos são sistemas de recomendação que integram dois ou mais tipos de estratégias de recomendação (ZHANG *et al*., 2019).

Na área Engenharia de Software, não há uma arquitetura de referência para sistemas de recomendação, tipicamente os sistemas são construídos fortemente acoplados a seus dados de entrada, pois normalmente se trata de um cenário específico, deste modo, a tarefa de pré-processamento e entendimento do contexto é de suma importância na construção do sistema de recomendação (ROBILLARD *et al*., 2014).

3. TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos relacionados a esta proposta, sendo estes: Wang *et al*. (2022); Cosh (2016); Cabot *et al*. (2015); Kallis *et al*. (2021) e Baek e Chung (2020).

O trabalho de Wang *et al.* (2022) propõe um método para identificar *labels* sinônimas automaticamente e recomendá-las para diferentes projetos de código aberto, sua abordagem consiste em dois processos principais: treinamento e previsão. A etapa de treinamento inclui extração de recursos de texto e construção de *labels* abstratos, estes são o resultado de um agrupamento de *labels* similares obtidos pelo algoritmo k-means (MACQUEEN, 1967). O processo de previsão inclui previsão de *labels* abstrato e pesquisa de *labels* personalizada, para isto o trabalho utilizou-se de alguns métodos de pré-processamento, como transformação do texto para caixa baixa, remoção de *links*, códigos, imagens e descarte de alguns *labels* referentes a status, além disso, o conceito de *embeddings* também é aplicado para converter texto em vetores semânticos, após isso é aplicando SENTENCE-BERT (REIMERS; GUREVYCH, 2019), uma modificação da rede BERT pré-treinada que usa uma estrutura tripla de redes siamesas para derivar *embeddings* de frases ou documentos semanticamente significantes, estes resultados podem ser comparados usando similaridade de cosseno.

Cosh (2016) em seu trabalho propôs um método capaz de aplicar técnicas de NLP para extrair automaticamente os temas de artigos analisados, com objetivo de descobrir qual área de Engenharia de Software é mais pesquisada. Seu método consiste em criar um corpus de palavras com as quais cada artigo pudesse ser comparado, sendo assim, todos os documentos foram extraídos e as frequências de cada palavra contadas através da representação de *Bag of Words.* Deste modo foi possível comparar artigos usando o cálculo de Log-Likelihood (WILKS, 1938), que identifica as palavras que são usadas mais do que o esperado, ao invés de apenas as mais frequentes, com isso palavras que aparecem muitas vezes ficaram mal classificadas pelo algoritmo, mas ainda assim não são descartadas. Após isso é realizado um agrupamento de artigos com base em seu tema principal, isto foi realizado usando o cálculo do coeficiente RV (ROBERT; ESCOUFIER, 1976) para medir a proximidade de duas matrizes, por meio da sua

pontuação Log-Likelihood (WILKS, 1938), sendo 1 para idêntico e 0 para nenhuma relação, o único processo manual foi a nomeação dos novos grupos a cada interação de agrupamento, porém nenhum mecanismo possibilita que um artigo seja incluído em mais de uma classe.

Cabot *et al*. (2015) observaram uma população de mais de três milhões de projetos do GitHub e forneceram alguns *insights* sobre como os *labels* são usados nestes. Os resultados revelam que, mesmo que o mecanismo de *labels* seja pouco utilizado, o uso de *labels* favorece a resolução de problemas. Sua análise também sugere que nem todos os projetos usam *labels* da mesma maneira, ou seja, alguns *labels* são apenas para hierarquia de prioridade no desenvolvimento, outros são usados como forma de documentação ou solicitação de documentação, enquanto outros são usados para sinalizar uma evolução temporal à medida que o fluxo de trabalho avança.

Kallis *et al*. (2021) apresentam em seu trabalho o Ticker Tagger, um aplicativo desenvolvido para funcionar em qualquer repositório de código hospedado no GitHub com objetivo de atribuir automaticamente uma *label* a novas *issues* abertas no repositório alvo, estas *labels,* por sua vez podem ser de três tipos: *Bug*, *Enhancement* ou *Question*. Para a implementação foi usado o algoritmo de aprendizado de máquina FastText, pois este algoritmo possui um baixo custo computacional, principalmente em relação ao espaço em disco, como entrada do algoritmo foi concatenado o título e a descrição das *issues*, o texto resultante é então tokenizado e este token representa a fonte para obter a representação do *Bag of Words* da *issue*, cada palavra desta representação é um vetor n-grama de caracteres. Para os testes deste trabalho, o aplicativo foi comparado com o algoritmo J48 em mesmos cenários de pré-processamento e entrada de dados, como resultado destes testes, foi descoberto que o algoritmo FastText supera o J48 em praticamente todos os cenários, também foi possível observar uma dificuldade em sugestões interpretadas com o tipo *Question,* segundo os autores isto provavelmente se deve a questão de que *issues* do tipo *Question* usam muito as palavras como "*how*" e "*what*" e o algoritmo acaba por atribuir erroneamente estar *issues* ao tipo *Question* quando podem se tratar de outras classes, além disso eles ressaltam que essa menor precisão em *issues* do tipo pergunta pode ser consequência de que são feitas perguntas de uma gama muito grande de assuntos. Outros pontos observados é que *issues* com *snippets* de código não impactam significativamente nos resultados finais da aplicação, enquanto inconsistências de idiomas, como mistura de línguas tendem a ter um desempenho pior em relação a entrada de dados sem estas características.

Baek e Chung (2020) propuseram um estudo sobre um sistema de recomendação de mídias em redes sociais usando mineração de relacionamentos sociais baseada em Word2Vec. O método proposto coleta metadados de conteúdo multimídia por gênero, título e palavra-chave, e então os agrupa por gênero usando o algoritmo k-means (MACQUEEN, J. B.,1967). Após isso, realiza um pré-processamento removendo *stop words* e expressões especiais e, em seguida, as palavras de sentimento são extraídas dos comentários relacionados ao conteúdo através de uma abordagem usando dicionário, para então serem classificadas em tipos três categorias, positivos, negativos e neutros por meio do algoritmo Máquina de Vetores de Suporte. Este consegue analisar dados, reconhecer padrões e no contexto deste trabalho é usado para análise de regressão, com objetivo de através do seu treinamento manter a precisão da classificação mesmo com mudanças de comportamento do usuário, além de diminuir o nível de dimensões. Com o uso do Word2Vec, as palavras de sentimento nos comentários de cada usuário são convertidas em vetores e quantificadas. Como as relações semânticas das palavras são consideradas, é usado similaridade dos cossenos para calcular uma relação social. Para avaliação de desempenho, o método proposto é comparado com o método de recomendação usando filtragem colaborativa e o método de recomendação usando filtragem baseada em conteúdo, como resultado o trabalho observou que a abordagem proposta se saiu melhor que as abordagens tradicionais, uma vez que considera fatores complementares tanto do método de recomendação implícito quanto do método de recomendação explícito.

Ressalta-se que nenhum trabalho foi encontrado com objetivo de criar um sistema de recomendação sobre uma análise de *issues* em repositórios de código usando a técnica de NLP Word2Vec. Relacionados a esta proposta, o trabalho de Wang *et al*. (2022) identifica *labels* sinônimas e as recomenda, assim como Baek e Chung (2020) implementam um sistema de recomendação de mídias sociais, em ambos os trabalhos pode-se observar a importância da etapa de pré-processamento e a necessidade, em certos contextos, de remover algumas informações que podem estar descontextualizadas ou prejudicar a base de dados que será usada como entrada no algoritmo de NLP.

Os trabalhos acima utilizam técnicas de *machine learning* antes de usar alguma técnica de NLP. Baek e Chung (2020) utilizaram o algoritmo SVM para tratar dos sentimentos e para isto, foi necessário usar k-means (MACQUEEN, 1967). Neste trabalho, tem-se como objetivo apenas realizar uma análise semântica das palavras em questão, deste modo, apenas aplicando outras técnicas de pré-processamento. Além disso, observando os resultados obtidos por Baek e Chung (2020), pode-se notar o ótimo desempenho obtido nos resultados do sistema de

recomendação usando o algoritmo NLP Word2Vec e similaridade dos cossenos, este por sua vez se saiu melhor do que abordagens de filtragem colaborativa, filtragem baseada em conteúdo e filtragem híbrida.

Cabot *et al.* (2015) realizam uma extensa análise sobre *labels*, estas, por sua vez são utilizadas para classificar as *issues*, logo é possível afirmar que os bons resultados obtidos em projetos que fazem bom uso de *labels* se estendem a um bom uso de *issues*. Além disso, eles observaram que nem todos os projetos usam estes recursos da mesma maneira, Kallis *et al.* (2021) notaram resultados piores em alguns tipos de *issues* que são do tipo pergunta e em *issues* cujo conteúdo mistura mais de um idioma. Assim como afirma Robillard *et al*. (2014), sistemas de recomendação tendem a ser construídos fortemente acoplados a seus dados de entrada, deste modo, este trabalho realiza a remoção de algumas *issues* que não fazem sentido no contexto do repositório de código e tendem a piorar os resultados obtidos pelo sistema de recomendação.

Cosh (2016) em seu trabalho não aplicou nenhuma técnica para identificar mais de um tema por artigo, o que pode implicar em resultados com alguma imprecisão, já que muitos artigos devem pertencer a mais de um tema, pensando neste ponto, este trabalho disponibiliza a possibilidade de usar como entrada, mais de uma palavra, o que disponibiliza ao usuário uma maior precisão no que o mesmo busca no sistema de recomendação.

Quando analisados as técnicas de NLP utilizadas, como, por exemplo, SENTENCE-BERT em Wang *et al*. (2022) ou *Bag of Words* com um agrupamento manual e uso de cálculos como coeficiente RV e sua pontuação Log-Likelihood em Cosh (2016). Pode-se observar que estas técnicas obtiveram bons resultados em suas propostas, porém este trabalho escolheu utilizar o Word2Vec, pois tem-se o objetivo de recomendar *issues* similares com base em algumas palavras de entrada e esta técnica prioriza a análise semântica de palavras.

Na Tabela 3.1 consta uma comparação rápida entre temas abordados pelos trabalhos relacionados citados acima e este trabalho.

Tabela 3.1 – Comparação entre o trabalho proposto e os trabalhos relacionados.

Trabalhos	**Realiza pré-processamento**	**Utiliza NLP**	**Analisa *issues* ou *labels***	**Cria um sistema de recomendação**
Wang *et al*. (2022)	X	X	X	X
Cosh (2016)	X	X		
Cabot *et al*. (2015)			X	
Kallis *et al*. (2021)	X	X	X	
Baek e Chung (2020)	X	X		X
Este trabalho	**X**	**X**	**X**	**X**

Fonte: Produção do próprio autor.

4. SISTEMA INTELIGENTE DE BUSCA DE ISSUES EM REPOSITÓRIOS DE CÓDIGO

A proposta deste trabalho é a elaboração de um sistema de recomendação capaz de analisar *issues*, utilizando aprendizado de máquina não supervisionado, através do algoritmo de NLP Word2Vec, com objetivo de facilitar a busca de *issues* e auxiliar na tomada de decisão que envolva *issues* em um repositório de código. Para atingir este objetivo, foi necessário extrair as *issues* do repositório de código, realizar o pré-processamento dos campos previamente definidos, criar dicionários de *unigram*, *bigram* e *trigram* e por fim realizar o treinamento da rede neural utilizada pelo algoritmo Word2Vec, utilizando como entrada os dicionários criados anteriormente. Após o modelo já treinado, o usuário será capaz de buscar *issues* através de uma entrada de dados, esta entrada pode conter uma, duas ou três palavras e deve obter como retorno um conjunto de *issues* mais similar semanticamente com a entrada.

Para facilitar a implementação de conceitos do Word2Vec, como criação de *embeddings*, criação do modelo Word2Vec e busca de palavras mais similares, foi utilizado à biblioteca *open source* Gensim (GENSIM, 2022).

Este capítulo apresenta os métodos usados para criação do sistema de recomendação de *issues*. A Seção 4.1 apresenta a metodologia de pesquisa utilizada para o desenvolvimento deste trabalho. A Seção 4.2 apresenta a definição e processamento dos atributos de interesse. Em seguida, a Seção 4.3 explica como foi realizado o processo de criação dos dicionários de palavras. A Seção 4.4 explica como foi realizado o treinamento do modelo Word2Vec. Por fim, a Seção 4.5 apresenta fluxo de dados do sistema de recomendação.

4.1 METODOLOGIA DE PESQUISA

A metodologia de pesquisa adotada neste trabalho seguiu a seguinte sequência de passos:

Escolha do repositório: o repositório de código do projeto SIS-ASTROS GMF foi escolhido devido a ter uma quantidade e *issues* suficientes para a realização do experimento controlado, além disso, a equipe do projeto poderia ser entrevista para avaliar as recomendações feitas pela ferramenta. Na Seção 5.2. são descritas mais informações sobre essa escolha.

Análise das *issues* do projeto: a partir da escolha do repositório, as *issues* foram analisadas visando selecionar os atributos de interesse para serem extraídos e pré-processados, a Seção 4.2 explica como esta etapa foi realizada.

Criação de dicionários: o passo seguinte foi a criação dos dicionários para as abordagens *unigram*, *bigram* e *trigram*, estes por sua vez, foram utilizados no treinamento do modelo Word2Vec, a Seção 4.3 explica como estes dicionários foram criados e sua importância no contexto da aplicação.

Treinamento do modelo Word2Vec: nesta etapa foi realizado o treinamento do modelo Word2Vec, utilizando os dicionários criados previamente, os detalhes deste processo são descritos na Seção 4.4.

Validação da proposta: um experimento controlado foi realizado buscando validar as seguintes hipóteses: o sistema de recomendação pode facilitar a navegação entre *issues* do repositório de código; as *issues* retornadas pelo sistema de recomendação são relevantes; o sistema de recomendação é capaz de auxiliar na tomada de decisão sobre eventos relacionados à *issues*. Para tal validação, um *script* foi criado buscando facilitar a realização dos testes, auxiliar os desenvolvedores a utilizar o sistema de recomendação e facilitar o processo de entrada e saída do sistema. O experimento controlado está descrito no Capítulo 5.

4.2 DEFINIÇÃO E PROCESSAMENTO DOS ATRIBUTOS DE INTERESSE

Como parte do objetivo deste trabalho é obter as *issues* mais similares semanticamente a uma entrada, é de fundamental importância a extração de dados das *issues* que possuem algum significado semântico por definição. Esta etapa corresponde à definição dos atributos, que são extraídos de cada *issue* coletada.

Dois atributos de texto foram escolhidos para serem processados e usados pelo modelo Word2Vec. O primeiro atributo extraído foi a descrição da *issue*, este campo contém a descrição detalhada da *issue* em questão e, tipicamente, possui uma quantidade considerável de palavras, este conjunto de palavras sempre possui significado semântico. O segundo atributo extraído foi o título da *issue*, embora em alguns casos o título possa conter poucas palavras, na maioria dos casos, é necessária uma quantidade considerável de palavras para descrever resumidamente uma *issue* em seu título, além disso, o título também possui significado semântico. Por fim, o

identificador único de cada *issue* também foi extraído, a fim de ser salvo para ser utilizado como referência em qualquer processo de busca de *issues*.

A fim de aprimorar a precisão do modelo Word2Vec, alguns pré-processamentos foram realizados nos atributos de título e descrição, os itens abaixo definem cada um destes processamentos:

- **Codificação UTF-8 e remoção de caracteres especiais:** caracteres especiais como pontuações e símbolos tendem a ser considerados como palavras quando o processo de *embedding* é realizado, além disso, estes caracteres não influenciam no entendimento de um contexto e acabam deixando as principais palavras do contexto mais distantes umas das outras, o que consequentemente piora o desempenho do modelo. A transformação para codificação UTF-8 segue o mesmo propósito, porém é implementada para remover possíveis caracteres especiais causados por erros de codificação.
- **Substituição de palavras acentuadas:** palavras acentuadas foram substituídas por suas respectivas palavras sem acento, com objetivo de evitar casos em que uma palavra é considerada diferente de outra apenas por ser ou não ser acentuada, além disso, é comum, programadores não utilizarem acentos em alguns casos, já que acentos não são reconhecidos por linguagens de programação, este fato torna ainda mais necessário um tratamento para palavras acentuadas.
- **Remoção de *stopwords*:** assim como a remoção de caracteres especiais, as remoções *stopwords* reduzem a distância de palavras importantes e não possuem grande impacto no entendimento do contexto das frases, sendo assim, a remoção de *stopwords* otimizam ainda mais o desempenho do modelo Word2Vec.

O Algoritmo 1, escrito utilizando a linguagem Python (Python Software Foundation, 2023), demonstra como foi realizada a implementação das técnicas de pré-processamento citadas acima, estas funções também são aplicadas nas palavras de entrada fornecidas pelo usuário e, por este motivo, foram implementadas recebendo como parâmetro apenas uma variável do tipo *string* e não o conjunto de todas as descrições ou títulos. Além disso, o Algoritmo 1 demonstra o uso das funções desenvolvidas sobre os títulos das *issues*, porém vale ressaltar que o mesmo processo foi realizado para as descrições.

Algoritmo 1 Algoritmo para processamento dos atributos

```
1.  import nltk
2.  import re
3.  import unicodedata
4.
5.  def remove_stopwords(title):
6.      nltk.download('stopwords')
7.      stopwords = set(nltk.corpus.stopwords.words('portuguese'))
8.
9.      txt_tokenized = nltk.word_tokenize(title)
10.     txt_clean = [word for word in txt_tokenized if word not in stopwords]
11.     return " ".join([str(i) for i in txt_clean])
12.
13. def remove_accents(input_str):
14.     input_str = input_str.lower().strip()
15.     nfkd = unicodedata.normalize('NFKD', input_str)
16.     handled_str = u"".join([c for c in nfkd if not unicodedata.combining(c)])
17.     return re.sub('[^a-zA-Z0-9 \\\]', '', handled_str)
18.
19. def clean_sentence(sentence):
20.     sentence = re.sub(r'[^a-z0-9\s]', '', sentence)
21.     sentence = sentence.replace('_', ' ')
22.     return re.sub(r'\s{2,}', ' ', sentence)
23.
24. df = pd.DataFrame()
25. df['title'] = base_issues_df['title'].apply(lambda x: remove_stopwords(x))
26.
27. df['title'] = df['title'].apply(lambda x: remove_accents(x))
28. df['title'] = df['title'].apply(lambda x: clean_sentence(x))
```

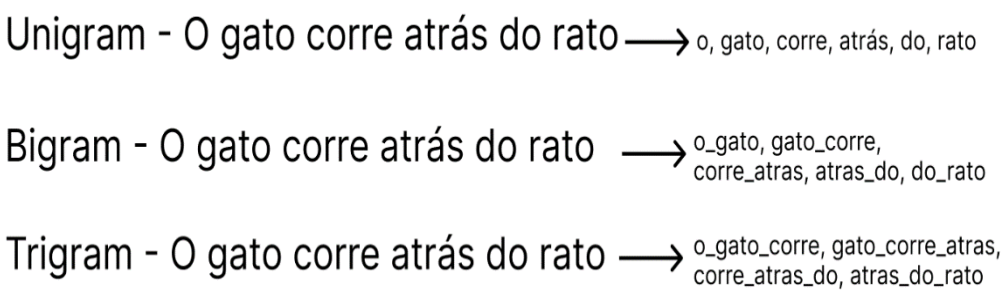
Fonte: Produção do próprio autor.

4.3 CRIAÇÃO DO DICIONÁRIO DE PALAVRAS

Um dos principais conceitos por trás do funcionamento do Word2Vec é a implementação de *embeddings.* Neste caso, foi necessário a tokenização das palavras em cada campo de texto envolvido, sendo assim, dicionários para as abordagens *unigram*, *bigram* e *trigram* foram criados, no caso da abordagem *unigram*, apenas a tokenização de cada campo das *issues* foi realizada, já para *bigram* e *trigram*, com o auxílio da biblioteca Gensim e de sua classe *Phrases*, cada palavra em um campo das *issues* é concatenada com a próxima palavra da frase e/ou a anterior e essa concatenação é separada pelo caractere *underline*. O resultado da concatenação de palavras é chamado de expressão por este trabalho.

Na Figura 4.1 pode ser visualizado como foi realizado o processo de criação dos dicionários usados, desconsiderando qualquer tipo de pré-processamento necessário.

Figura 4.1 - Como é criado um dicionário para geração de embeddings.



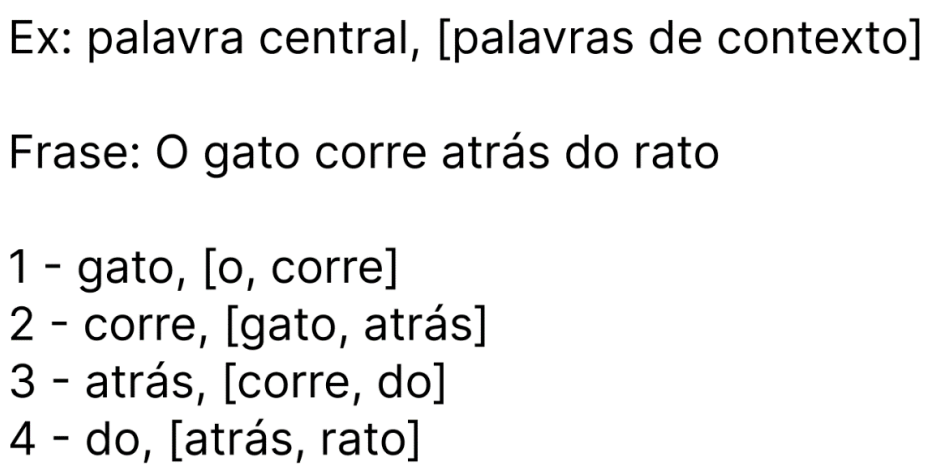
Fonte: Produção do próprio autor.

4.4 TREINAMENTO DO MODELO WORD2VEC

O modelo Word2Vec foi treinado usando a arquitetura skip-gram, nesta abordagem, dada uma palavra central, o modelo tenta descobrir as palavras de contexto, também é possível observar que, para abordagens *ngram* o conceito é o mesmo, já que várias palavras são

concatenadas usando o caractere *underline* e, consequentemente, são considera com uma única palavra pelo algoritmo. A Figura 4.2 exemplifica como esta abordagem é utilizada para descobrir a palavra de contexto.

Figura 4.2 - Funcionamento da arquitetura skip-gram.



Fonte: Produção do próprio autor.

O Algoritmo 2 é um exemplo simplificado do processo de criação do modelo Word2Vec e de busca de uma expressão similar utilizando a biblioteca Gensim, este algoritmo foi escrito utilizando a linguagem Python (Python Software Foundation, 2023).

Algoritmo 2 Algoritmo para criação do modelo Word2Vec

```
1. from gensim.models import Word2Vec
2.
3. model = Word2Vec(sentences=corpus, window=5, min_count=1, workers=12, sg=1)
4. res = model.wv.most_similar(expression, topn=5)
```

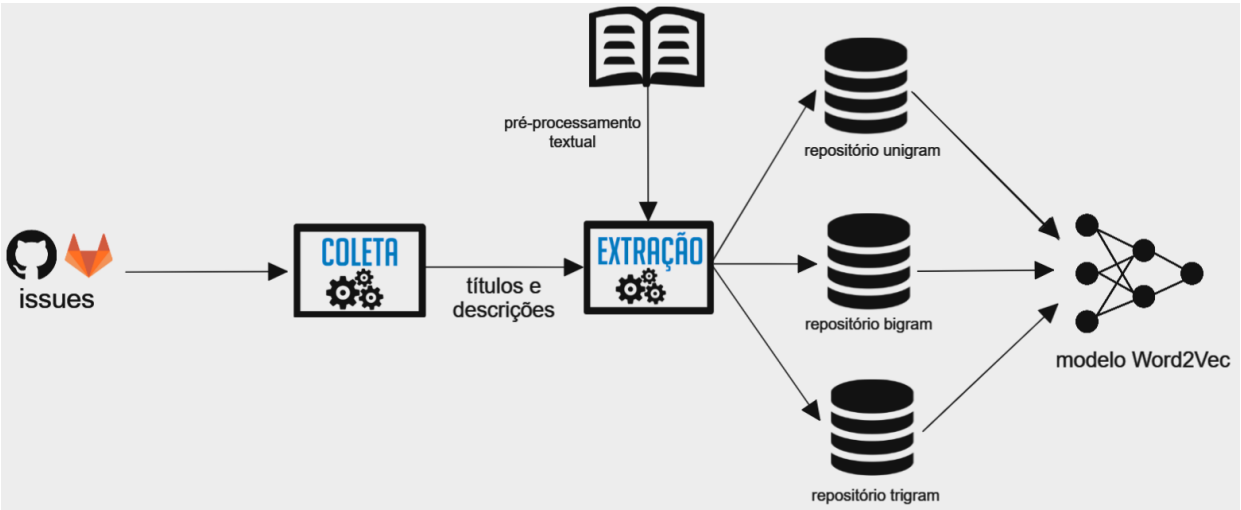
Fonte: Produção do próprio autor.

Alguns parâmetros foram definidos para criação do modelo Word2Vec, utilizando a biblioteca Gensim (GENSIM, 2022), sendo eles:

- **sg:** define que a arquitetura da rede neural utilizada pelo modelo, neste caso o valor 1 foi atribuído e representa a arquitetura skip-gram, o valor padrão é zero e representa a arquitetura de rede neural CBOW.
- **corpus:** este parâmetro representa o conjunto total de palavras que será utilizado pelo modelo, deste modo, os dicionários criados anteriormente foram utilizados para treinamento do modelo Word2Vec.
- **workers:** número de threads que podem ser utilizadas pelo algoritmo, neste caso, o valor atribuído foi de 12 threads, o máximo suportado pela máquina usada nos testes.
- **window:** este parâmetro representa o tamanho da janela usada pelo algoritmo de rede neural, ou seja, quantas palavras antes e depois da palavra alvo serão consideradas no treinamento, neste trabalho utilizamos o valor 5, padrão da biblioteca.
- **min_count:** define um valor mínimo que uma determinada palavra ou conjunto de palavras concatenadas deve aparecer no corpus para ser considerada no treinamento, para este parâmetro este trabalho utilizou o valor 1, buscando evitar que palavras importantes fossem desconsideradas apenas levando em conta sua frequência.

Na Figura 4.3 é apresentada uma visão da arquitetura do modelo Word2Vec criado, bem como o fluxo de dados da coleta das *issues* até a criação do modelo.

Figura 4.3 - Arquitetura do modelo Word2Vec criado.



Fonte: Produção do próprio autor.

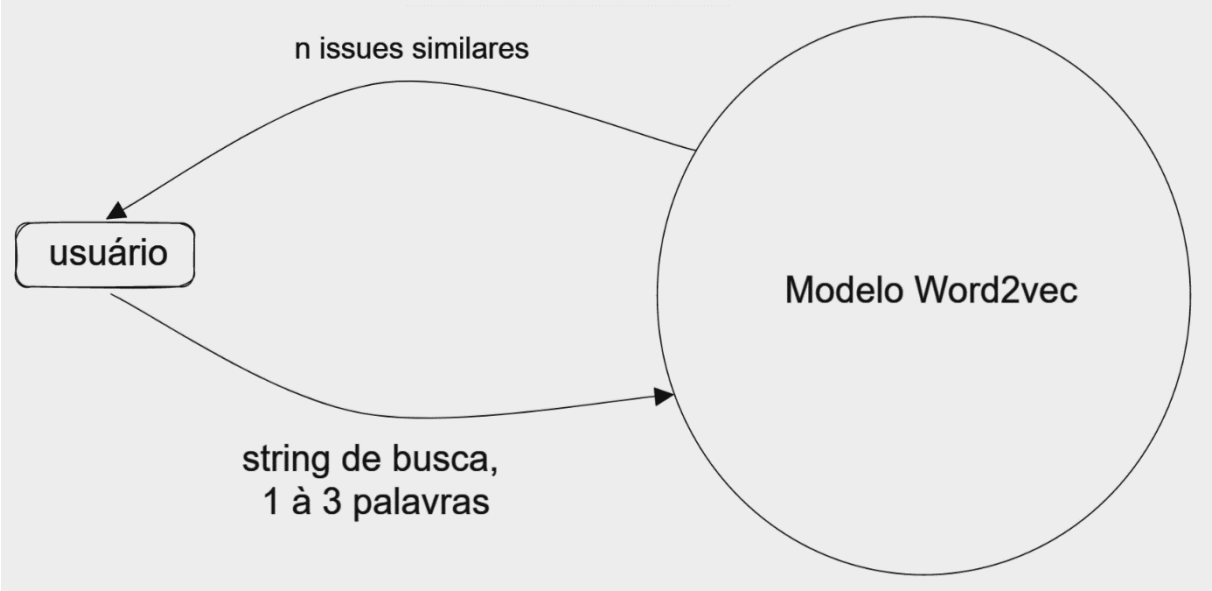
4.5 FLUXO DE DADOS DO SISTEMA DE RECOMENDAÇÃO

Após o treinamento do modelo Word2Vec, o usuário é capaz de fornecer como entrada ao sistema de recomendação, uma, duas ou três palavras, após isso, o modelo Word2Vec retorna as expressões mais bem classificadas, por ordem crescente e, através de uma configuração prévia, as cinco primeiras expressões são retornadas.

Para cada uma das cinco expressões retornadas pelo modelo Word2Vec, o sistema de recomendação realiza uma busca de três *issues* que possuem mais repetições destas expressões. Esta busca é realizada nas *issues* pré-processadas, devido ao fato das expressões retornadas pelo modelo Word2Vec serem do mesmo formato de suas entradas. Através do identificador único extraído previamente, conforme explicado na Seção 4.1, é possível encontrar as referências originais das *issues* obtidas neste processo, sem realizar o processo inverso de pré-processamento de texto realizado na criação dos dicionários *ngrams*. Por fim, em caso de alguma expressão retornar alguma *issue* já encontrada por outra expressão, a *issue* duplicada em questão é removida do conjunto de *issues* resultante, caso não seja encontrado nenhuma duplicata, todas as *issues* são mantidas. Ao final do processo, este conjunto é exibido ao usuário do sistema de recomendação. É importante ressaltar que o modelo não é treinado novamente, caso o usuário forneça uma entrada que não exista no corpus do modelo Word2Vec.

A Figura 4.4 mostra o fluxo de dados do sistema de recomendação e como ocorre a interação com o modelo Word2Vec desenvolvido.

Figura 4.4 - Diagrama de fluxo de dados do sistema de recomendação.



5. EXPERIMENTO CONTROLADO

Este capítulo apresenta o experimento controlado realizado para validação deste trabalho. Na Seção 5.1 são apresentados os objetivos e hipóteses que a validação deseja verificar. Em seguida, a Seção 5.2 explica como foi realizada a preparação para validação e coleta de dados. Na Seção 5.3 são explicados os processos de coleta de dados. Por fim, na Seção 5.4 é realizada uma análise e explicação dos resultados obtidos.

5.1 OBJETIVOS

A proposta deste trabalho é a elaboração de um sistema de recomendação capaz de analisar *issues*, buscando facilitar sua busca e auxiliar na tomada de decisão que envolva *issues* em um repositório de código. Portanto, o objetivo é verificar se a proposta ajuda desenvolvedores a enfrentar os seguintes desafios: auxiliar na busca de *issues* no repositório de código, recomendar *issues* relevantes no contexto das palavras de entrada e auxiliar na tomada de decisão em eventos relacionados a isso.

Objetivos da avaliação

Avaliar se a abordagem proposta, suportada pela ferramenta, é capaz de solucionar os principais desafios relacionados à recomendação de *issues*.

Hipóteses:

H1: O sistema de recomendação pode facilitar a navegação entre *issues* do repositório de código;

H2: As *issues* retornadas pelo sistema de recomendação são relevantes;

H3: O sistema de recomendação é capaz de auxiliar na tomada de decisão sobre eventos relacionados.

5.2 PREPARAÇÃO PARA COLETA DE DADOS

Para apoiar a abordagem, foi implementada uma ferramenta em formato *script* para realização dos testes, buscando auxiliar os desenvolvedores a utilizar o sistema de recomendação e facilitando o processo de entrada e saída do sistema. Esta ferramenta foi desenvolvida utilizando a linguagem Python (Python Software Foundation, 2023) e o ambiente de desenvolvimento Anaconda (Anaconda The World's most popular Data Science Platform, 2023) juntamente com o Jupyter (Project Jupyter, 2023), uma ferramenta *Literate Computing*, flexível e que permite ao seu usuário configurar um ambiente de desenvolvimento, que possuem várias dependências de bibliotecas externas, como a biblioteca Gensim e bibliotecas que facilitam no pré-processamento dos dados.

O repositório de *issues* utilizado foi o repositório do projeto SIS-ASTROS GMF (Sistema Integrado de Simulação Integrado para o Sistema ASTROS - Grupo de Mísseis e Foguetes). SIS-ASTROS GMF é um projeto de pesquisa entre a Universidade Federal de Santa Maria (UFSM) e o Exército Brasileiro que visa o desenvolvimento de um simulador virtual tático para treinamento de militares em operações relativas ao emprego de um Grupo de Mísseis e Foguetes. Este repositório contém aproximadamente 1100 *issues* na data de realização dos testes.

5.3 COLETA DE DADOS

A validação da ferramenta foi realizada por dois desenvolvedores do projeto SIS-ASTROS GMF, que utilizam o repositório de código em seu dia a dia e possuem conhecimento sobre as *issues* do projeto. Para realizar este processo, uma entrevista com os desenvolvedores foi realizada com objetivo de validar as hipóteses deste estudo.

O processo de validação começou com os desenvolvedores digitando uma, duas ou três palavras para realização da busca, após isso o sistema de recomendação retorna as *issues* resultantes, a quantidade deste resultado varia conforme o sistema de recomendação elimina *issues* duplicadas, em média cada retorno conteve de onze a quinze *issues*. Cada resultado foi avaliado individualmente pelos desenvolvedores, eles discutiram e entraram em um consenso se a *issue* em questão é ou não relevante no contexto das palavras digitada, após esta análise,

uma porcentagem de acerto foi calculada para as palavras digitadas, na qual 100% significam que as *issues* resultantes foram totalmente relevantes no contexto das palavras digitadas e 0% significa que elas não foram relevantes.

Em resumo, os desenvolvedores validaram doze expressões, sendo, cinco *unigrams*, seis *bigrams* e uma *trigram*, para cada expressão retornada foi atribuída uma porcentagem de acerto das *issues* resultantes.

5.4 ANÁLISE DOS RESULTADOS

A validação demonstrou que a abordagem facilita a navegação de *issues*, auxiliando os programadores a encontrar mais facilmente *issues* relacionadas a entrada de dados, também foi possível observar que, na maioria dos casos, as *issues* retornadas pelo sistema de recomendação possuem relevância no contexto das palavras fornecidas como entrada. Além disso, mesmo em casos que a porcentagem de acerto não foi tão expressiva, as *issues* que foram avaliadas como significantes pelos programadores possuem grande similaridade com as palavras de entrada e pode auxiliar os programadores em seus objetivos.

A Tabela 4.2 abaixo possui uma relação das palavras testadas e suas respectivas porcentagens de acerto, bem como a categoria que pertencem.

Tabela 4.2 – Análise dos resultados.

(continua)

Categoria	**Palavra(s)**	**Porcentagem de acerto**
unigram	botão	100%
unigram	modelo	100%
unigram	navegação	65%
unigram	avaliar	80%
unigram	física	95%
unigram	menu	85%
bigram	navegação local	55%

(conclusão)

bigram	modelo performance	70%
bigram	*cordenate picker*	70%
bigram	menu radial	85%
bigram	menu medidas	65%
trigram	especificar implementação funcionalidade	33%

Fonte: Produção do próprio autor.

As validações para expressões *trigrams* encontraram algumas dificuldades, devido ao fato de o modelo não ser treinado toda vez que uma palavra de entrada não é encontrada no vocabulário, o que ocasionou em certa dificuldade pelos programadores de encontrarem boas palavras para realização dos testes.

Outro exemplo foi o caso de expressões envolvendo as palavras “menu medidas”, que obteve uma porcentagem de acerto baixa devido ao fato de ter poucas ocorrências nas *issues* e de várias descrições de *issues* do repositório de código em questão possuírem um *template*, que se repete em muitos casos e possui a palavra “menu” como um seus principais itens, além disso, esta hipótese foi reforçada durante os testes, devido ao fato das palavras “menu radial” terem obtido ótimos resultados, já que é uma sequência de palavras comum dentro do contexto do repositório de código. A palavra “menu” obteve um bom resultado, pois mesmo se referindo a vários menus diferentes, possui o mesmo significado.

Referências a diretórios ou a arquivo do projeto também foram encontradas em alguns casos em que a porcentagem de acerto foi relativamente baixa, embora a etapa de pré-processamento tenha removido caracteres especiais, referências a arquivos ou diretórios acabaram não sendo removidas por completo, o que afetou o resultado de algumas *issues* retornadas pelo sistema de recomendação.

Outra questão observada durante a validação foi que, em alguns casos, nomes de programadores ou de gerentes do projeto são citados na descrição das *issues,* o que acaba impactando negativamente nos resultados do modelo Word2Vec como um todo, pois estes

casos não foram tratados durante a etapa de pré-processamento, já que se trata de uma convenção da equipe do projeto.

Deste modo, mesmo com algumas limitações, foi possível observar que o sistema de recomendação de *issues* é capaz de auxiliar os programadores na tomada de decisão em eventos envolvendo *issues*, além de facilitar a busca e recomendar *issues* relevantes para os programadores que o utilizam.

6. CONCLUSÃO

Neste capítulo são descritas as considerações finais sobre o trabalho, enfatizando os resultados obtidos e os trabalhos futuros. Na Seção 6.1 são resumidos os objetivos e resultados deste trabalho. Em seguida, na Seção 6.2 o trabalho é finalizado e são descritas as perspectivas de trabalhos futuros.

6.1 REVISÃO GERAL E RESULTADOS

Neste trabalho foi desenvolvido um sistema de recomendação de *issues,* utilizando técnicas de aprendizado de máquina não supervisionado e o algoritmo de NLP Word2Vec, na implementação desta ferramenta, uma etapa de pré-processamento de *issues* foi realizada, esta etapa inclui remoção de *stopwords*, codificação em UTF-8, remoção de caracteres especiais e substituição de palavras acentuadas por sua respectiva palavra sem acento. Após isso, um dicionário de *unigrams*, *bigrams* e *trigrams* foi criado para realização do treinamento da rede neural utilizada pelo algoritmo Word2Vec. Por fim, os programadores que utilizam este sistema de recomendação são capazes de fornecer palavras de entrada e obter como resultado as *issues* recomendadas pelo sistema de recomendação.

Após isso, um experimento controlado foi realizado buscando validar as seguintes hipóteses: o sistema de recomendação pode facilitar a navegação entre *issues* do repositório de código; as *issues* retornadas pelo sistema de recomendação são relevantes; o sistema de recomendação é capaz de auxiliar na tomada de decisão sobre eventos relacionados à *issues*. Para tal validação, um script utilizando a linguagem Python foi criado com objetivo de facilitar a interação com os programadores, este script foi responsável por realizar as interações entre os usuários e o sistema de recomendação.

Por fim, a abordagem foi validada através de uma entrevista, onde doze expressões contendo *unigrams*, *bigrams* e *trigrams* foram testadas por programadores do projeto SIS-ASTROS GMF. O repositório de código utilizado possui aproximadamente 1100 *issues*. Durante a validação, uma porcentagem de acerto dos resultados do sistema de recomendação foi estimada pelos desenvolvedores. A partir disso, foi possível perceber que o sistema de recomendação teve ótimo desempenho ao recomendar *unigrams*, teve um bom desempenho ao recomendar *bigrams*, mas não obteve grandes resultados ao recomendar *trigrams,* devido ao

fato de não realizar um novo treinamento a cada palavra de entrada desconhecida pelo algoritmo Word2Vec. Por fim, conclui-se que, embora o sistema de recomendação não tenha uma significativa porcentagem de acerto em alguns casos, ele foi capaz de retornar *issues* relevantes, auxiliar na navegação de *issues* e, consequentemente, facilitar a tomada de decisão em eventos envolvendo *issues.*

6.2 TRABALHOS FUTUROS

Como trabalhos futuros, pretende-se melhorar a implementação do sistema de recomendação por meio de uma conexão com a API dos repositórios de código online, assim, possibilitando a qualquer usuário, extrair *issues* de qualquer repositório de código a sua escolha. Além disso, podem ser elencados como trabalhos futuros a realização de um experimento utilizando um modelo *ngram* com Word2Vec, este modelo deve ser capaz de lidar com expressões sem tamanho máximo, embora aumentar a quantidade de palavras possa não resultar em um crescimento exponencial positivo nos resultados, utilizar de mais palavras tende a retornar resultados melhores até certo limite. Outra possível abordagem para trabalhos futuros é a utilização de outras técnicas de processamento de linguagem natural para processamento de texto, como *bag of words* ou utilização de sinônimos caso uma palavra não seja encontrada.

Outro ponto de possível melhoria de implementação é a realização de um novo treinamento a cada interação do usuário que resulta em uma palavra que não é encontrada no vocabulário do modelo Word2Vec, facilitando a interação com o sistema de recomendação e, possivelmente, melhorando os resultados obtidos. Além disso, durante a etapa de validação, observou-se que nomes de usuários e diretórios do projeto alvo estavam presentes nas *issues* e poderiam ser tratados ou mais explorados, por exemplo, fornecendo a possibilidade ao usuário do sistema de recomendação de informar alguns nomes próprios ou apelidos referentes a pessoas envolvidas no projeto, para que eles sejam removidos dos campos utilizados pelo modelo Word2Vec.

REFERÊNCIAS

ABRANTES, J. F., TRAVASSOS, G. H. **Common Agile Practices in Software Processes**. In: International Symposium on Empirical Software Engineering and Measurement, pp. 355-358, Banff, AB, 2011.

Anaconda - The World's most popular Data Science Platform. **Anaconda documentation**. Anaconda - The World's most popular Data Science Platform, 2023. Disponível em: https://docs.anaconda.com/. Acesso em: 12 jan. 2023.

BAEK, J.; CHUNG, K. **Multimedia recommendation using Word2Vec-based social relationship mining**. Multimedia Tools and Applications, 28 jan. 2020.

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Recuperação de Informação: conceitos e tecnologia das máquinas de busca**. 2. Ed. ed. [s.l.] Bookman Editora, 2013.

BENNACEUR, A.; MEINKE, K. **Machine learning for software analysis: Models, methods, and applications. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),** v. 11026 LNCS, n. 1, p. 3–49, 2018.

BRISSON, S.; NOEI, E.; LYONS, K. **We Are Family: Analyzing Communication in GitHub Software Repositories and Their Forks.** 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 2020, pp. 59-69, doi: 10.1109/SANER48275.2020.9054834.

CABOT, J. *et al*. **Exploring the use of labels to categorize issues in Open-Source Software projects.** 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, Canada, 2015, pp. 550-554, doi: 10.1109/SANER.2015.7081875.

COSH, K. **Current research themes in software engineering: An application of text mining.** 2016 8th International Conference on Knowledge and Smart Technology (KST), Chiang Mai, Thailand, 2016, pp. 125-129, doi: 10.1109/KST.2016.7440476.

DAS, D.; SAHOO, L.; DATTA, S. **A Survey on Recommendation System.**

Editora Ltda., 2014. 1056 p. ISBN 9788535251418. Disponível em: https://books.google.com.br/books?id=BsNeAwAAQBAJ. Acesso em: 02 nov. 2022.

EISENSTEIN, J. **Introduction to Natural Language Processing.** [s.l.] MIT Press, 2019. Disponível em: https://books.google.ca/books?hl=en&lr=&id=72yuDwAAQBAJ&oi=fnd&pg=PR5&dq=%20natural+language+processing+2019&ots=gVbNY2-%20kp2&sig=jf32aGmiIBbrU9C3ezcZ1dEnwlg#v=onepage&q=natural%20language%20processing%202019&f=false. Acesso em: 10 nov. 2022.

GENSIM. Gensim, 2022. Disponível em: https://radimrehurek.com/gensim/models/word2vec.html. Acesso em: 25 nov. 2022.

GIT. Git, 2022. Disponível em: https://git-scm.com/. Acesso em: 25 nov. 2022.

GITHUB. **About - GitHub**, 2022. Disponível em: https://github.com/about. Acesso em: 25 nov. 2022.

GITLAB. **About – GitLab**, 2020. Disponível em: https://about.gitlab.com/company. Acesso em: 25 nov. 2022.

GITHUB. **Sobre Problemas - Documentação do GitHub,** 2022. Disponível em: https://docs.github.com/pt/issues/tracking-your-work-with-issues/about-issues. Acesso em: 11 nov. 2022.

JATNIKAA, D.; BIJAKSANA, M.; SURYANI, A. **Word2Vec Model Analysis for Semantic Similarities in English Words.** 4th International Conference on Computer Science and Computational Intelligence 2019 (ICCSCI), 12-13 September 2019.

KALLIS, R. *et al*. **Predicting issue types on GitHub.** Science of Computer Programming, v. 205, p. 102598, 2021.

LAI, S. *et al*. **How to Generate a Good Word Embedding.** IEEE Intelligent Systems, vol. 31, no. 6, pp. 5-14, Nov.-Dec. 2016, doi: 10.1109/MIS.2016.45.

MIKOLOV, T. *et al*. **Efficient Estimation of Word Representations in Vector Space.** CoRR, v. abs/1301.3781, 2013. Disponível em: https://arxiv.org/abs/1301.3781, Acesso em: 31 out. 2022.

MACQUEEN, J. **Some methods for classification and analysis of multivariate observations.** Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, v. 5.1, p. 281–298, 1 jan. 1967.

PATEL, S. P.; UPADHYAY, S. H. **Euclidean distance based feature ranking and subset selection for bearing fault diagnosis.** Expert Systems with Applications, v. 154, p. 113400, set. 2020.

PORTUGAL, I.; ALENCAR, P.; COWAN, D. **The Use of Machine Learning Algorithms in Recommender Systems: A Systematic Review.** Expert Systems with Applications, Volume 97, 2018, Pages 205-227, ISSN 0957-4174. Disponível em: https://doi.org/10.1016/j.eswa.2017.12.020.

Python Software Foundation. **Python 3.7.1 documentation**. Python Software Foundation, 2023. Disponível em: https://docs.python.org/3. Acesso em: 12 jan. 2023.

Project Jupyter. **Jupyter Project Documentation**. Project Jupyter, 2023. Disponível em: https://docs.jupyter.org/en/latest/. Acesso em: 12 jan. 2023.

REIMERS, N.; GUREVYCH, I. **Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.** Disponível em: https://arxiv.org/abs/1908.10084, Acesso em: 02 dez. 2022.

ROBERT, P.; ESCOUFIER, Y. **A Unifying Tool for Linear Multivariate Statistical Methods: The RV- Coefficient.** Applied Statistics, v. 25, n. 3, p. 257, 1976.

ROBILLARD, M.; WALKER, R.; ZIMMERMANN, T. **Recommendation Systems for Software Engineering.** IEEE Software, v. 27, n. 4, p. 80–86, jul. 2010.

ROBILLARD, P. *et al*. (EDS.). **Recommendation Systems in Software Engineering.** Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

RUSSELL, S.; NORVIG, P. **Inteligência artificial: Tradução da 3a Edição**. Elsevier Editora Ltda., 2014. 1056 p. ISBN 9788535251418. Disponível em: https://books.google.com.br/books?id=BsNeAwAAQBAJ. Acesso em: 09 nov. 2022.

SHRESTHA, A.; MAHMOOD, A. **Review of Deep Learning Algorithms and Architectures.** IEEE Access, v. 7, p. 53040–53065, 2019.

SIMON, P. **Too Big to Ignore: The Business Case for Big Data**. [S.l.]: Wiley. 89 páginas. ISBN 978-1-118-63817-0, 2013.

SPINELLIS, D. **Git**. IEEE Software, vol. 29, no. 3, pp. 100-101, May-June 2012, doi: 10.1109/MS.2012.61.

WANG, J. *et al*. **Personalizing label prediction for GitHub issues**. Information and Software Technology, v. 145, p. 106845, 2022.

WILKS, S. S. **The Large-Sample Distribution of the Likelihood Ratio for Testing Composite Hypotheses.** The Annals of Mathematical Statistics, v. 9, n. 1, p. 60–62, 1938.

WITTEN, I. H. *et al*. **Data Mining Practical Machine Learning Tools and Techniques.** Fourth Edition ed. [s.l.] Elsevier, 2017.

ZHANG, S. *et al*. **Deep Learning Based Recommender System: A Survey and New Perspectives.** ACM Computing Surveys, v. 52, n. 1, p. 1–38, 25 fev. 2019.