UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME DE MELO CARNEIRO

SISTEMA DE RECOMENDAÇÃO DE BUG REPORTS SIMILARES UTILIZANDO O BERT

CAMPINA GRANDE - PB

2023

GUILHERME DE MELO CARNEIRO

SISTEMA DE RECOMENDAÇÃO DE BUG REPORTS SIMILARES UTILIZANDO O BERT

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Franklin de Souza Ramalho

CAMPINA GRANDE - PB

2023

GUILHERME DE MELO CARNEIRO

SISTEMA DE RECOMENDAÇÃO DE BUG REPORTS SIMILARES UTILIZANDO O BERT

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Franklin de Souza Ramalho

Orientador – UASC/CEEI/UFCG

Professor Dr. Maxwell Guimarães de Oliveira

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em 14 de fevereiro de 2023.

CAMPINA GRANDE - PB

ABSTRACT

In the context of large-scale software projects, there is an increasing demand for fixes in their conception process that surpass the tests and quality filters of the Quality Assurance team and impact the end customers of the product. In order to document these behaviors so that they can be later analyzed and corrected, software engineering makes use of documents called Bug Reports (BR). As pointed out by Anvik et al [2], the frequency of new BRs being opened in large projects is high, exemplified by the Eclipse tool, which already had approximately 190 new BRs being opened daily in 2005. Motivated by this problem, this study proposes and evaluates a BR recommendation system based on textual similarity, with the differential use of the state-of-the-art text comprehension model BERT [3] as one of the factors in the similarity calculation. Its objective is to improve suggestions for BRs with a context close to that provided by the maintainer, which would supposedly increase their productivity and consequently the number of resolved BRs. As the results obtained attest, there were gains of approximately 14% in the frequency of relevant BRs for the first 20 recommendations, when compared to the technique that used only TF-IDF as a textual vectorization model. Finally, the BERT model added improvements to the evaluated metrics (precision, feedback, and likelihood) when used in a complementary manner to TF-IDF, but did not perform positively in an isolated manner.

Sistema de Recomendação de Bug Reports Similares utilizando o BERT

Guilherme Carneiro
guilherme.carneiro@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Orientador: Franklin de Souza Ramalho
franklin@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

No contexto de projetos de software de grande porte, há demanda crescente por correções de erros em sua concepção, que ultrapassam os filtros de testes e qualidade da equipe de *Quality Assurance* e impactam os clientes finais do produto. A fim de documentar estes comportamentos para que sejam posteriormente analisados e corrigidos, a engenharia de software faz uso de documentos chamados Bug Reports (BR). Como apontado por Anvik et. al [2], a frequência de novos BRs sendo abertos em grandes projetos é elevada, exemplificado pela ferramenta *Eclipse*, que, ainda em 2005, já contava com aproximadamente 190 novos BRs sendo abertos diariamente. Motivado por essa problemática, o presente estudo propõe e avalia um sistema de recomendação de BRs baseado em similaridade textual, com o diferencial de utilizar o modelo estado-da-arte de compreensão textual BERT [3] como um dos fatores no cálculo de similaridade. Este tem como objetivo aprimorar as sugestões de BRs de contexto próximo ao fornecido pelo mantenedor, o que supostamente aumentaria sua produtividade, e por consequência, a quantidade de BRs resolvidos. Como resultados obtidos, atestou-se ganhos de aproximadamente 14% na frequência de BRs relevantes para as 20 primeiras recomendações, quando comparado à técnica que utilizou apenas TF-IDF como modelo de vetorização textual. Por fim, o modelo BERT agregou melhoras às métricas avaliadas (precisão, *feedback* e *likelihood*) quando utilizado de maneira complementar ao TF-IDF, não desempenhando positivamente de maneira isolada.

1 INTRODUÇÃO

Os softwares, assim como produtos fabricados por outros segmentos do mercado, estão sujeitos a erros em sua concepção que, por vezes, não são localizados e corrigidos antes de chegar aos clientes finais. Tais erros, no contexto da computação, possuem características variadas, abrangendo problemas na interface de uso do produto, até lentidão ou quaisquer comportamentos não previstos durante a fase de idealização, e a estes erros atribuiu-se o nome de *bug*.

Dada a frequência de ocorrência de bugs nas mais diversas ferramentas computacionais, demandou-se da engenharia de software formas de documentação dos mesmos, a fim de tornar compreensível, e possivelmente replicável, tal comportamento. Para o mantenedor do software, uma descrição clara do problema associada a *logs* do sistema permite localizar o trecho de código incorreto, corrigí-lo e testá-lo até garantir o funcionamento esperado.

Como resultado da necessidade por descrições eficientes de bugs, foi criado o artefato de documentação de erros chamado de Bug Report (BR). Este é composto por campos variados, adaptando-se à necessidade do projeto em questão, e contém informações sobre a natureza do problema, categorizações, responsável, nível de urgência e também uma descrição textual sobre o problema ocorrido, dentre outros.

Em projetos de grande porte, o volume de BRs criados se eleva rapidamente, exemplificado pelo projeto da ferramenta *Eclipse*[1], que ainda em 2005, já contava com picos de aproximadamente 190 BRs abertos no mesmo dia, de acordo com Anvik [2]. Mesmo que uma parcela considerável dos BRs seja classificada como duplicada ou pouco útil [2], mantém-se um alto volume de informação a ser analisada pelos mantenedores do sistema que, não raramente, acaba se acumulando.

Tendo em vista o problema do alto volume de novos BRs sendo abertos diariamente, o presente trabalho apresenta um sistema de recomendação de BRs similares baseado em princípios de Processamento de Linguagem Natural (PLN). O objetivo desse sistema é tornar mais eficiente o trabalho dos mantenedores, sugerindo *Bug Reports* disponíveis para análise[2] que apresentem contexto semelhante ao BR utilizado como referência (*query*), o que evitaria grandes mudanças de contexto entre *Reports* sucessivos analisados pelo mesmo mantenedor, assim possivelmente aumentando a taxa de BRs resolvidos.

O objetivo do presente trabalho é analisar a aplicabilidade do modelo *Bidirectional Encoder Representations from Transformers* (BERT) [3] no contexto de recomendação de *Bug Reports* similares, aplicando-o em um sistema de recomendação da mesma natureza, buscando responder a seguinte questão de pesquisa: *(Q1) O BERT agrega melhoria de desempenho no contexto de recomendação de Bug Reports similares, considerando as métricas de Precisão, Likelihood e Feedback?*

O presente trabalho propõe o uso dos modelos de vetorização *Term Frequency - Inverse Document Frequency* (TF-IDF) [11] e *Bidirectional Encoder Representations from Transformers* (BERT) para geração dos vetores de contexto a partir dos campos de descrição dos BRs. A partir deles, calcula-se o nível de similaridade entre o BR de entrada (*query*) e os BRs candidatos através da similaridade de cosseno [12]. O cálculo desse valor, associado ao cruzamento de campos categóricos[3], obtém um score de similaridade para cada *report* candidato em relação à *query*, o que permite ao sistema ranqueá-los e retorná-los como recomendações ao usuário.

O documento está organizado como segue. A seção 2 expõe o embasamento teórico utilizado para fundamentar o presente estudo e explicar conceitos essenciais para melhor compreensão do mesmo. A seção 3 apresenta os trabalhos relacionados. A seção 4 explica detalhadamente a técnica proposta. A seção 5 explica a metodologia com a qual se realizou a avaliação da técnica proposta. A seção 6

[1] Disponível em: https://www.eclipse.org
[2] Bug Reports classificados como pendentes e ainda sem responsável definido.
[3] Campos que admitem um valor pertencente a um conjunto pré-definido de valores. Os campos *product* e *component* são exemplos desse tipo de campo.

desenvolve sobre os resultados e as discussões inerentes a eles, respondendo também o questionamento de pesquisa. Por fim, a seção 7 apresenta as últimas considerações e ressalvas relacionadas ao presente estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, expõe-se o embasamento teórico que fundamentou a execução do presente estudo, iniciando pela definição de contexto e sua relação com a vetorização textual (2.1), detalhamento do modelo de vetorização textual BERT (2.2), compreensão da métrica de similaridade entre textos no contexto da PLN (2.3) e, por fim, explicações relacionadas à sistemas de recomendação de informação (2.4).

2.1 Contexto e Vetorização Textual

De acordo com o dicionário Houaiss, o significado do termo contexto remete ao "*conjunto de palavras, frases ou o texto que precede ou segue determinada palavra, frase ou texto e que contribuem para o seu significado*" [4]. Portanto, para que se permita estimar a similaridade entre conteúdos textuais, é necessário compreender a disposição das palavras no texto - como estão distribuídas e quais agregam mais significado ao conteúdo.

Dentro do arcabouço de ferramentas de Processamento de Linguagem Natural (PLN), há modelos responsáveis por expressar matematicamente o contexto de um texto através do processo de vetorização. Estes se baseiam na contabilização do vocabulário de textos, onde cada termo constituinte é expresso no espaço vetorial como uma dimensão separada e ortogonal às demais, originadas pelos outros termos [11]. A soma de cada vetor respectivo a cada termo fornece uma expressão matemática, i.e. vetor, convergente ao significado de contexto exposto acima. Exemplos de modelos que utilizam como base algoritmos dessa natureza são: *Term Frequency* (TF) e *Term Frequency - Inverse Document Frequency*(TF-IDF), e fazem parte do conjunto de ferramentas de vetorização textual da PLN, classificados como *bag of words* [11].

Porém, uma das desvantagens já conhecidas dos modelos *bag of words*, é que os mesmos não contabilizam a ordem dos termos em sua vetorização [11], limitando consequentemente a expressão do contexto de um texto.

Considerando as vetorizações textuais como expressões matemáticas do contexto semântico [11], a depender da complexidade do modelo vetorizador (i.e. dimensão do espaço vetorial), a vetorização gerada supostamente tende a ser mais fiel ao contexto semântico da linguagem natural, o que consequentemente tornaria o cálculo de similaridade entre conteúdos textuais mais assertivo.

Um passo adiante dos modelos *bag of words*, os modelos neurais (e.g *Skip-gram* [6], *BERT* [3]) assumem avanços na captura de contexto. Por adotarem algoritmos baseados em redes neurais, permitem que cada termo assuma um vetor de contexto próprio, a depender dos termos vizinhos, além do vetor geral do texto, criado a partir dos vetores relacionados aos termos, a exemplo do modelo *Word2Vec* [11]. Essa natureza de algoritmo se aproxima ainda mais do significado da palavra contexto, que assume importância plena no conceito de similaridade.

2.2 BERT

Pertencente ao grupo de modelos de vetorização textual baseado em redes neurais, o *Bidirectional Encoder Representations from Transformers* (BERT) é considerado o estado-da-arte no que se refere a vetorizadores de conteúdos textuais [8]. Este utiliza algoritmo baseado em redes neurais profundas a fim de pré-treinar um modelo de vetorização complexo, que gera representações de textos, nomeadas de *word embeddings* (WE) [3].

O modelo BERT foi pré-treinado de maneira não-supervisionada utilizando um volume massivo de texto, retirado de bancos de texto como *Wikipedia* e *OpenBooks* [8]. Através do uso de *transformers* profundamente bidirecionais [3], que consideram ambos os sentidos de leitura dos termos (direita-para-esquerda, esquerda-para-direita) como formadores do contexto de cada palavra, predisse palavras "mascaradas" do corpus textual (15% de todo o vocabulário), a fim de treiná-lo como um modelo de linguagem natural de propósito geral, que pode ser modificado para tarefas específicas facilmente [8], inclusive similaridade entre textos.

Múltiplos modelos foram disponibilizados pela *HuggingFace*[4], com variações de idiomas e destinados a ambientes com diferentes níveis de limitações computacionais. O repositório do BERT é *open-source*, e pode ser acessado através do seu *GitHub*[5].

2.3 Métricas de Similaridade

Os contextos dos textos são matematicamente expressos através de vetores de dimensões variáveis, a depender do modelo de vetorização textual utilizado. Com o objetivo de expressar o nível de similaridade entre textos-base, a distância entre as respectivas representações vetoriais pode ser utilizada [11]. Há várias métricas de distância vetorial elegíveis, como: distância euclidiana, correlação e distância de cosseno [11].

O conceito de similaridade de cosseno se trata do complementar matemático da distância de cosseno, visto que, no valor mínimo da distância, os vetores apontam para a mesma direção, o que torna a similaridade máxima, e quando o valor da distância for máximo, a similaridade consequentemente será mínima. Abaixo é apresentada a fórmula para a similaridade de cosseno [11], onde v e w são vetores n-dimensionais, e o resultado indica a orientação entre os vetores:

$$cos(v, w) = \frac{v \cdot w}{||v|| \cdot ||w||}$$

No sistema de recomendação *NextBug*, proposto por Rocha et. al. [9], o cálculo de similaridade entre BRs foi feito através da similaridade de cosseno. Nos trabalhos de Ye et. al. [13] e Yang et. al. [12], também relacionados ao escopo de recomendação de reports, utilizou-se a similaridade de cosseno para contabilizar semelhança textual, demonstrando ser uma técnica mais comumente usada para esse fim do que as demais (distância euclidiana e correlação).

Assim, para o presente trabalho, também foi utilizada a similaridade de cosseno, justificada pela sua eficiência para similaridade entre textos já atestada em estudos anteriores.

[4]Disponível em: https://huggingface.co
[5]Disponível em https://github.com/google-research/bert para mais detalhes.

2.4 Sistemas de Recomendação

De acordo com Isinkaye et. al. [5], recomendadores são sistemas de recuperação da informação que lidam com o problema de sobrecarga de informações através de algoritmos de filtragem de dados. Essa filtragem pode ser baseada em preferências ou interesses do usuário (colaborativa), na natureza do conteúdo (*content-based*), ou na mesclagem dos dois (híbrida), como indicado por Sharma e Gera [10]. Um sistema de recomendação ideal deve ter a capacidade de prever se um usuário prefere ou não um determinado item.

Recomendadores frequentemente utilizam formas de receber feedbacks relacionados à qualidade de suas recomendações, podendo utilizá-los para retroalimentar o modelo preditivo, caso modelos baseados em aprendizado sejam utilizados - como redes neurais, por exemplo -, e também para fins avaliativos. Tais feedbacks podem ser classificados em três tipos: implícito, explícito e híbrido. O primeiro se refere àqueles em que o sistema considera determinadas ações do usuário como respostas, como cliques ou compra de um produto. O segundo se refere a retornos explicitados diretamente pelo usuário, se o resultado trazido pelo sistema foi ou não útil. Já o terceiro propõe uma junção dos dois primeiros [5].

Além da natureza dos feedbacks, os sistemas também podem adotar diferentes técnicas de filtragem de recomendações, sendo classificadas principalmente em dois tipos: filtragem baseada em conteúdo, e filtragem colaborativa. A primeira é caracterizada por ser mais dependente do domínio do conteúdo, e assume ênfase maior nas características (i.e. propriedades) dos itens para gerar as predições, por isso é frequentemente utilizada para sugerir notícias ou publicações, por exemplo. Já a filtragem colaborativa é mais utilizada para conteúdos com tons de abstração, como músicas e filmes, que não são facilmente descritos pelas suas propriedades. Nela, o algoritmo cria um grupo chamado "vizinhança", no qual o perfil de um usuário é aproximado de perfis de outros usuários, e itens que receberam feedback positivo por usuários da vizinhança se tornam suas recomendações [5]. O presente trabalho faz uso do modo de filtragem por conteúdo.

3 TRABALHO RELACIONADOS

De acordo com Anvik et al. [2], um dos principais problemas relatados por desenvolvedores envolvidos em projetos *open-source* é a dificuldade em encontrar Bug Reports duplicados. Para resolvê-lo, o mesmo propôs um modelo estatístico de classificação de BRs duplicados, baseado em vetorização textual aplicado ao campo de descrição. Foi utilizado como métrica de semelhança a similaridade de cosseno entre as vetorizações. Este foi capaz de acertar em 90% dos casos se um BR era único ou duplicado.

No escopo de recomendação de BRs similares a mantenedores, a ferramenta *NextBug*, proposta por Rocha et. al [9], foi desenvolvida como plug-in de navegador que tem por função recomendar reports similares em texto disponíveis na plataforma *Bugzilla*. O mesmo utiliza técnicas de PLN, como modelos de vetorização textual, especificamente TF-IDF, para criar uma expressão matemática de contexto e aproximar outros BRs em aberto similares à entrada, calculando a similaridade entre vetores também através da similaridade de cosseno. Como resultados, este obteve uma taxa de feedback de 65%, ou seja, a cada 10 BRs fornecidos como entrada, o *NextBug* obtém recomendações para pelo menos 6 deles, além de obter uma média de 3,2 recomendações por *query*. Obteve também uma precisão de 31% em suas recomendações, ou seja, aproximadamente 3 em cada 10 recomendações feitas pelo NextBug são de fato relevantes (considerando como relevante um BR que mais tarde é assumido pelo mesmo desenvolvedor do BR utilizado como entrada). Para obter tais métricas de avaliação, o *NextBug* utilizou o dataset de BRs do ecossistema *Mozilla*, abrangendo BRs marcados como resolvidos no período de 2009 à 2012.

Utilizando princípios de sistemas de recuperação da informação, Yang et. al [12] propôs um sistema de recomendação de BRs similares que pondera outras características além de apenas a representação vetorial da descrição de um *report*, a fim de retornar BRs ainda mais relevantes para o mantenedor. A pontuação de similaridade foi calculada através de três fatores: similaridade de cosseno entre representações TF-IDF, distância entre *word embeddings* gerados pelo modelo *Skip-Gram* [6], e a comparação dos campos de produto e componente, existentes nos *Bug Report*s relacionados ao *Mozilla*. Utilizando uma metodologia baseada na quantidade de arquivos modificados em comum entre soluções de BRs, obteve cobertura de aproximadamente 48% para top-10 resultados, isto é, a cada 10 BRs fornecidos como entrada, 5 deles obtêm pelo menos uma report relevante dentre as recomendações fornecidas.

O sistema de recomendação de BRs proposto no presente trabalho condensa decisões relevantes tomadas nos trabalhos relacionados, e propõe a utilização do modelo estado-da-arte BERT para a geração de *word embeddings*. A aplicação da comparação dos campos categóricos de produto e componente e a utilização de representações TF-IDF são partes comuns do presente sistema com os apresentados anteriormente, enquanto que o principal diferencial está na utilização do modelo BERT como gerador de *word embeddings*, não utilizado por nenhum outro estudo de mesmo escopo até o presente momento. Com isso, objetiva-se possível melhoria na obtenção de recomendações de BRs relevantes, dada a alta capacidade desse modelo em captar contextos textuais.

4 TÉCNICA PROPOSTA

O presente trabalho tem cunho quantitativo e exploratório, e foi realizado sob a base de dados *open-source* disponível na plataforma *Bugzilla*[6], abrangendo Bug Reports de diferentes projetos do ecossistema *Mozilla*. Aplicou-se o modelo de vetorização textual BERT para fins de recomendação de BRs, avaliando o seu desempenho dentro de um sistema de recomendação orientado à similaridade textual. Considerando experiências semelhantes já existentes, como executado por Rocha et al. [9], visou-se estudar a adição do BERT nesse contexto, dada a sua comprovada capacidade de compreender contextos de conteúdos textuais. Portanto, o presente trabalho visa responder a seguinte questão de pesquisa: *(Q1) O BERT agrega melhoria de desempenho no contexto de recomendação de Bug Reports similares, considerando as métricas de Precisão, Likelihood e Feedback?*

Para responder à questão acima levantada, foi implementada uma *lib* de um sistema de recomendação de *Bug Reports* utilizando a linguagem de programação *Python*[7], responsável por, a partir de um BR atribuído como *query*, gerar recomendações de BRs abertos com

[6]Disponível em: https://bugzilla.mozilla.org
[7]Disponível em: https://www.python.org

características textuais similares, a fim de permitir ao mantenedor receber sugestões de Reports pendentes[8] que possam ser de seu interesse. A Figura 1 explicita a arquitetura de seu funcionamento.

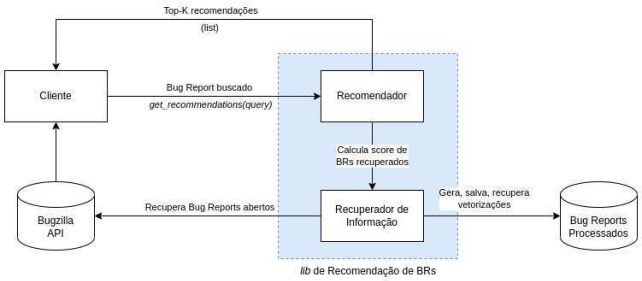

Figura 1: Arquitetura do Sistema de Recomendação

Uma vez que um BR é fornecido como *query*, o sistema de recomendação recupera do banco de dados *MongoDB*[9] todos os reports classificados como abertos no momento da busca, trazendo consigo também suas respectivas vetorizações textuais de seus campos de descrição (explicado em detalhes em 4.1). Após isso, é calculado o score de similaridade entre a *query* e todos os BRs candidatos (explicado em detalhes em 4.2), ordenam-se os resultados de maneira decrescente com base no score, retornando, no máximo, top-K recomendações, sendo K o número desejado de recomendações indicado pelo usuário.

Há a necessidade de buscar apenas por BR classificados como abertos (i.e. pendente e ainda sem responsável definido) devido ao objetivo do presente sistema ser recomendar o próximo BR a ser analisado pelo mantenedor, não devendo este BR já estar atribuído a outro responsável.

Para demonstração de uso e avaliação, utilizou-se a API do Bugzilla[10] para obter BRs reais de vários projetos *open-source* do ecossistema *Mozilla*. A Tabela 2 expõe um exemplo de resultado obtido através do sistema de recomendação desenvolvido quando aplicada a *query* exposta na Tabela 1.

Tabela 1: Bug Report ID=504249

Campo	Valor
ID	504249
Product	Penelope Graveyard
Component	Mail Window
Summary	*"copying messages from local to IMAP may result in incorrect message status and header"*

Ao inserir o BR de ID 504249 como entrada no sistema, percebe-se que as recomendações trazidas possuem *Summary* com grau de semelhança em relação à *query*, se relacionando também ao protocolo *IMAP*[11], no caso dos IDs 503682 e 478602, e o ID 503684 se relaciona ao contexto de mensagens. Percebe-se também que o *Score* obtido pelo ID 503682 foi maior, indicando maior similaridade em relação à *query*, decrescendo em caráter de semelhança de contexto para as posições seguintes da recomendação.

Tabela 2: Top-3 Recomendações para Bug Report ID=504249

ID	Score	*Summary*
503682	1.385	*"bogus characters when copying files from Local to IMAP folders"*
503684	1.361	*"bsent messages imported from Eudora 5 always showing "unread"status"*
478602	1.360	*"message corrupted upon Transfer from Local Folder to IMAP folder"*

É importante esclarecer que a pontuação de similaridade não necessariamente permanece entre 0 e 1, dada a forma como o mesmo é calculado, exposto em detalhes na seção 4.3.

4.1 Otimizando Recuperação de Recomendações

A necessidade de ter que gerar vetorizações para os BRs, além de calcular as similaridades entre eles e ordená-las, impede que haja processamento sob demanda a cada pedido por recomendações, i.e, que o cálculo de similaridade entre a *query* e seus candidatos seja feito a cada requisição do usuário, por questões de performance. Como solução para esta dificuldade, optou-se por pré-calcular as vetorizações das descrições dos BRs, e salvar as métricas de similaridade obtidas como "arcos" de similaridade em um banco de dados *MongoDB*, permitindo maior velocidade à cada busca por recomendações.

4.2 Geração de Vetorizações Textuais

4.2.1 TF-IDF. Para que fosse possível obter uma vetorização TF-IDF otimizada dos conteúdos textuais, aplicou-se um conjunto de pré-processamentos frequentemente utilizados no segmento de PLN, sendo esses: tokenização e remoção de *stopwords*. O primeiro se trata da divisão das sentenças que compõem o texto em tokens individuais (no presente trabalho, em palavras), além da remoção das pontuações e caracteres especiais. O segundo se trata da remoção de conectivos, palavras muito curtas ou pouco significativas para o modelo vetorizador, como o artigo "*the*", o conectivo "*and*", entre outros [1].

Foi utilizado todo o corpus de BRs de avaliação para a criação do Modelo de Vetorização Espacial (MVS) [11] em que as vetorizações textuais do TF-IDF estão contidas. Há essa necessidade devido à natureza do modelo TF-IDF em gerar vetores nos quais cada dimensão se relaciona a um token pertencente ao vocabulário total [11], por isso, utilizou-se todos os tokens válidos de todos os Bug Reports do corpus (aproximadamente 106.000 BRs) como constituintes desse vocabulário.

Ao utilizar o modelo TF-IDF, o sistema de recomendação visa usufruir da capacidade de comparação entre documentos textuais pertencentes ao mesmo corpus, já que são analisados sob o mesmo MVS, mais do que compreender o contexto inerente a cada BR individualmente, seguindo o que foi proposto por Yang et al. [12].

[8] Que ainda não possui um responsável, nem foi resolvido, permanecendo disponível para que algum mantenedor o assuma.
[9] Disponível em: https://www.mongodb.com
[10] Disponível em: https://bugzilla.mozilla.org
[11] Internet Message Access Protocol

4.2.2 Word Embeddings. Para gerar os *word embeddings* de cada BR, utilizou-se o modelo *sentence-BERT* (sBERT), variação do BERT baseado em redes neurais siamesas [7]. Frequentemente utilizado para fins de cálculo de similaridade entre sentenças, nenhum ajuste de parâmetros ou *fine-tuning* foi realizado no modelo. Todos os parâmetros foram mantidos com seus valores padrão.

Ao utilizar o modelo sBERT para gerar *word embeddings*, torna-se possível compreender o contexto inerente às palavras e a ordem em que elas se apresentam, permitindo expressar de maneira mais eficiente o contexto do documento textual, independentemente da existência de um corpus. Diferentemente de técnicas *bag of words* como TF-IDF, que encontram mais aplicação na relação entre textos pertencentes ao mesmo corpus.

4.3 Score de Similaridade

O objetivo do score de similaridade é calcular o quão similares são dois BRs, e para isso, propõe-se uma função de similaridade que engloba características de duas naturezas: categórica e contexto textual. Através da utilização de diferentes naturezas de características, visa-se afunilar o escopo das buscas por candidatos, já que mantenedores naturalmente não tendem a trabalhar em *reports* de contextos muito distantes entre si, por consequência fazendo sentido levar menos em consideração aqueles candidatos categoricamente mais distantes da *query*, explicado em mais detalhes a seguir.

Sobre o conceito de similaridade categórica, os BRs utilizados no presente estudo possuem os seguintes campos[12]:

- ***product***: Referencia o produto do ecossistema *Mozilla* ao qual o BR se refere. Exemplo: *Core*
- ***component***: Componente pertencente ao produto definido no campo ***product***. Um ***product*** pode possuir vários componentes. Exemplo: *Javascript Engine*

Para otimização de buscas por candidatos e priorização de recomendações de contexto próximo, considera-se que BRs que não possuam pelo menos um dos dois campos (*product* ou *component*) em comum com a *query* não farão parte das recomendações. Tal medida se baseia na hipótese de que um mantenedor assumirá BRs posteriores que tenham em comum pelo menos um desses dois campos, apresentando assim algum nível de semelhança entre escopos. Tal melhoria foi proposta por Yang et. al., tendo se mostrado útil [12]. Como forma de expressar matematicamente a semelhança categórica, define-se a seguinte função, onde x e y são Bug Reports, $x.prod$ e $y.prod$ são seus respectivos valores do campo *product*, e $x.comp$ e $y.comp$ são os respectivos valores do campo *component*:

$$Score_{cat}(x,y) = \begin{cases} 1 \text{ se } (x.prod = y.prod) \wedge (x.comp = y.comp) \\ 0.5 \text{ se } x = (x.prod = y.prod) \vee (x.comp = y.comp) \\ 0 \text{ caso contrário} \end{cases}$$

Em relação às características de semântica textual, admite-se como expressão de contexto textual as vetorizações TF-IDF e *word embeddings* dos campos de descrição dos BRs. Para calcular a similaridade entre contextos, propomos a similaridade de cosseno, calculada tanto entre pares de vetores TF-IDF ($Score_{TFIDF}$), quanto entre pares de *word embeddings* ($Score_{we}$). Por meio destas duas métricas de similaridade, se deseja expressar semelhança entre textos através de modelos de diferentes níveis de complexidade - TF-IDF mais simples, utilizando o conceito de distribuição de frequência de palavras, e o BERT, mais complexo, utilizando *transformers* para gerar contextos profundos. Supostamente, ao adotar ambas formas de vetorização, elas devem se complementar. Abaixo se encontram as fórmulas que definem $Score_{TFIDF}$ e $Score_{we}$, onde x e y são BRs, e as funções $TFIDF$ e we retornam a vetorização TF-IDF e os *word embeddings* do campo de descrição do BR passado como parâmetro, respectivamente:

$$Score_{TFIDF}(x,y) = cos(TFIDF(x), TFIDF(y))$$

$$Score_{we}(x,y) = cos(we(x), we(y))$$

Por fim, para englobar todas os métodos de expressão de similaridade propostos, a função que calcula o score final de similaridade ($Score_{TP}$) é apresentada abaixo:

$$Score_{TP} = (Score_{TFIDF} + Score_{we}) * Score_{cat}$$

4.4 Coleta de Dados

Como base de dados de demonstração e testes, utilizou-se os Bug Reports disponíveis no repositório *open-source Bugzilla*, compreendendo documentos criados entre janeiro de 2009 e outubro de 2012, sempre com estado definido como *resolved*. Esse filtro foi aplicado devido à necessidade de avaliar BRs que foram de fato resolvidos, se alinhando com a proposta de avaliar os ganhos de *feedback*, *likelihood* e precisão ao utilizar o sistema de recomendação proposto. No total, foram obtidos aproximadamente 141.000 BRs, porém, após a remoção daqueles que continham o e-mail "*nobody@mozilla.org*"[13] como responsável, restaram 106.605 BRs para fins de avaliação.

Devido à ausência de uma feature que indicasse quando o status de um report foi alterado para resolvido - e que consequentemente não deveria ser utilizado como recomendação, foi criado sinteticamente o campo "*when_changed_to_resolved*". Essa informação foi recuperada por meio do histórico de alterações de cada BR, também disponível na API do *Bugzilla*, e foi útil para compreender os candidatos válidos para cada *query*, de acordo com a definição explicada em detalhes na seção 5.2.

Por fim, tratamentos para casos excepcionais de BRs foram aplicados: reports contendo como responsável o seguinte e-mail "*nobody@mozilla.org*" foram desconsiderados, visto que não se tratava de um mantenedor válido, e BRs com campo de descrição nulo tiveram seu título replicado para a descrição, permitindo que pudessem ser vetorizados.

Optamos por utilizar esse subconjunto na tentativa de aproximar a base de dados de validação daquela utilizada pelo *NextBug* [9], com objetivo de permitir mostrar semelhanças e diferenças de resultados válidas entre o ambos os trabalhos.

[12] Documentação da API Bugzilla disponível em: https://bmo.readthedocs.io/en/latest/using/understanding.html

[13] e-mail utilizado como *placeholder* para *Bug Reports* ainda sem responsável definido.

5 AVALIAÇÃO

5.1 Construção de Oráculo

A fim de checar a relevância dos BRs recomendados, utilizou-se o princípio proposto por Rocha et. al [9], assumindo como recomendações candidatas apenas BRs pendentes na data de criação da *query*, isto é, com data de criação anterior à *query* e ainda não resolvidos até a mesma data. Esse método tem como objetivo simular o uso normal de um sistema de recomendação de BRs de um repositório *open-source*, tal qual o *NextBug*, no qual apenas BRs já existentes na base de dados e ainda não resolvidos integrariam os resultados.

Assume-se que o sistema de recomendação agregará valor sempre que recomendar BRs que foram assumidos pelo mesmo responsável, o que indicaria ganhos de produtividade, pois se assume que a sugestão acelerou o processo de correspondência entre o BR pendente e o mantenedor capaz de resolvê-lo.

Devido ao alto custo computacional requerido para gerar os *word embeddings* de todos os 106.605 BRs, e posteriormente calcular a similaridade entre eles, uma amostra aleatória de 10.000 BRs (do universo de aproximadamente 106.000) foi utilizada para avaliar o presente sistema. Cada BR da amostra foi inserido como *query*, e suas recomendações foram recuperadas e avaliadas de acordo com a técnica descrita no início desta seção.

Para fins de performance, foram pré-calculadas as similaridades de cosseno entre os BRs que compõem a amostra e os seus possíveis candidatos, otimizando a execução dos testes e a extração das métricas de avaliação.

Toda a implementação supracitada se encontra disponível publicamente no *GitHub*[14].

5.2 Métricas de Avaliação

Para avaliar o desempenho das recomendações obtidas a partir dos BRs que compõem a amostra, as seguintes métricas foram utilizadas: *Feedback* [9], *Precisão* [9] e *Likelihood* [9]. As métricas são calculadas a partir da presença de BRs relevantes (de acordo com a técnica proposta no oráculo) dentre as recomendações retornadas. A seguir são apresentadas cada uma das métricas, e sua interpretação no contexto do sistema de recomendação.

$$Feedback(k) = \frac{|Z_k|}{|Z|}$$

O *Feedback* mede a porcentagem de *queries* pertencentes à amostra que retornaram pelo menos k recomendações. Na fórmula, Zk se refere à quantidade de *queries* que retornaram pelo menos k recomendações, e Z é o total de queries avaliadas, assim, para $k = 5$, se para 100 *queries* testadas, apenas 50 retornaram pelo menos 5 recomendações, o *Feedback* será de 50%.

$$Precisao(k) = \frac{Recomendados(k) \bigcap Relevantes}{Recomendados(k)}$$

A *Precisão*, no contexto de sistemas de recomendação, mede a frequência de itens relevantes nos resultados, isto é, qual a proporção de BRs relevantes dentre k recomendações. Na fórmula, $Recomendados(k)$ se refere às recomendações retornadas pela *query* e *Relevantes* se refere àqueles BRs úteis no contexto do oráculo. Logo, para $k = 5$, se forem retornados 4 resultados, dos quais apenas 2 são relevantes, atinge-se uma *Precisão* de 50%. Devido à necessidade de avaliar o sistema de recomendação baseado em múltiplas execuções, será referenciado como *Precisão* a média aritmética das precisões geradas por cada *query*, podendo a partir dela avaliar a precisão geral do protótipo.

$$Likelihood(k) = \begin{cases} 1 \text{ se } (Recomendados(k) \bigcap Relevantes) \neq \emptyset \\ 0 \text{ caso contrário} \end{cases}$$

O *Likelihood* é uma métrica binária frequentemente utilizada para avaliar a utilidade das recomendações fornecidas, bastando que pelo menos um item recomendado, dentre k, seja relevante para que seu valor seja 1, e 0 caso contrário. Assim, sempre que pelo menos um item relevante for retornado, a recomendação será considerada útil. Como o *Likelihood* é calculado sob o resultado de cada *query*, será referenciado como *Likelihood* a média aritmética dos *Likelihoods* obtidos por cada query da amostra, permitindo avaliar de maneira geral a utilidade das recomendações do protótipo.

5.3 Metodologia

Para permitir que o sistema de recomendação proposto fosse avaliado quantitativamente, o conteúdo textual dos campos de descrição de todos os 106.605 BRs foi vetorizado utilizando dois modelos: TF-IDF e sBERT. A implementação, em Python, utilizada do modelo TF-IDF está disponível no arcabouço do *scikit-learn*[15], enquanto a implementação do sBERT utilizada para a geração dos *word embeddings* foi a *all-MiniLM-L6-v2*, disponibilizada pela empresa *HuggingFace*[16]. Para suprir a alta demanda computacional necessária para o processamento dos *word embeddings*, fez-se uso de VMs pagas disponibilizadas pelo *Google Colab*[17] do seguinte tipo: "(GPU) de back-end do Google Compute Engine em Python 3".

A execução do processamento das vetorizações durou aproximadamente 30 horas ininterruptas, tendo sido paralelizado entre quatro VMs. O tempo de processamento gasto por BR permaneceu entre 1 e 6 segundos.

Com todos os vetores gerados, calculou-se as similaridades entre campos categóricos e de cosseno entre os vetores de cada BR pertencente à amostra de avaliação, e seus possíveis candidatos a recomendações - de acordo com o oráculo. Cada conjunto de métricas de similaridade calculado entre um par de BRs foi salvo em entidades nomeadas arcos de similaridade, em um banco de dados *MongoDB*. No total, foram calculados 11.560.074 arcos, que foram indexados com intuito de acelerar consultas.

Obtidos todos os arcos de similaridade entre os BRs da amostra e suas possíveis recomendações, foram executadas três baterias de avaliação, aplicando cada BR da amostra como *query* na *lib* de recomendação apresentada na Seção 4, variando, entre cada bateria, os fatores envolvidos no cálculo de similaridade, a fim de checar a diferença entre cada combinação de fatores e a técnica proposta (disponível na seção 4). As seguintes variações para o cálculo de similaridade foram testadas:

- ***Técnica Proposta (TP)***: Método de score proposto no presente trabalho, exposto na seção 4.3. Utiliza como fatores

[14]Disponível em: https://github.com/guimcarneiro/similar-bug-reports-recommender

[15]Disponível em: https://scikit-learn.org

[16]Mais detalhes em: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[17]Disponível em: https://colab.research.google.com

no cálculo do score: as similaridades de cosseno entre os vetores TF-IDF ($Score_{TFIDF}$), entre os *word embeddings* ($Score_{we}$) e a pontuação do cruzamento dos campos categóricos ($Score_{cat}$). Se apresenta com a seguinte fórmula:

$$Score_{TP} = (Score_{TFIDF} + Score_{we}) * Score_{cat}$$

- ***Técnica I (T1)***: Utiliza como fatores no cálculo do score: a similaridade de cosseno entre os vetores TF-IDF ($Score_{TFIDF}$) e a pontuação do cruzamento dos campos categóricos ($Score_{cat}$), excluindo-se a influência dos *word embeddings*. Segue a fórmula abaixo:

$$Score_{T1} = Score_{TFIDF} * Score_{cat}$$

- ***Técnica II (T2)***: Utiliza como fatores no cálculo do score: a similaridade de cosseno entre os *word embeddings* ($Score_{we}$) e a pontuação do cruzamento dos campos categóricos ($Score_{cat}$), excluindo-se a influência do TF-IDF. Segue a fórmula a seguir:

$$Score_{T2} = Score_{we} * Score_{cat}$$

Cada uma das baterias testou os seguintes valores para $k = [1, 2, ..., 20]$, sendo k a quantidade desejada de recomendações por *query*. A comparação entre os resultados das três baterias deve permitir responder a questão de pesquisa Q1 e avaliar se houve melhora no desempenho do sistema de recomendação ao utilizar *word embeddings* gerados pelo sBERT.

Os testes aplicados sob a *lib* de recomendação foram executados em um computador com processador *Intel Core i5-8250U*, 8gb de memória RAM e placa gráfica integrada *Intel UHD Graphics 620*. Cada bateria de testes com a amostra de 10.000 BRs durou aproximadamente de 15 minutos.

As recomendações obtidas por cada *query* e suas respectivas métricas de avaliação estão disponíveis publicamente no repositório[18] oficial do presente trabalho, assim como todos os *scripts*, códigos-fonte e vetorizações utilizados no decorrer deste estudo.

6 DISCUSSÃO E RESULTADOS

Com a realização da avaliação, percebeu-se inicialmente a rapidez na recuperação das recomendações, atestando a importância da geração prévia das vetorizações e dos arcos de similaridade, buscando-os através de um banco de dados, ao invés de calculá-los sempre que for feito um pedido.

Os resultados obtidos na avaliação da técnica proposta estão disponíveis na Tabela 3:

Tabela 3: Métricas Obtidas pela Técnica Proposta

K	Precisão	Likelihood	Feedback
1	25,07%	25,07%	99,81%
3	22,09%	41,73%	99,51%
5	20,49%	49,97%	99,22%
10	18,11%	60,34%	98,29%
20	15,35%	69,79%	96,32%

Em relação aos altos valores de *feedback* obtidos, estes se justificam devido ao fato de não haver um limite mínimo de score de similaridade para as recomendações retornadas, que por consequência sempre irá trazer resultados, mesmo que nem sempre muito similares ou relevantes. Há um valor decrescente de *feedback* à medida em que o K aumenta, isso se deve à natureza da métrica, que avalia a frequência de resultados que trouxeram pelo menos K recomendações, logo, para valores de K mais altos, nem sempre há candidatos válidos suficientes.

De acordo com a Tabela 3, percebe-se que a *precisão* para $K = 1$ é de aproximadamente 25%, indicando que a cada 4 pedidos por recomendações, há ocorrência de BR relevante na primeira posição da lista de recomendações. A medida em que o o valor de K aumenta, a precisão geral diminui, já que mais resultados são trazidos, diminuindo a proporção entre relevantes e não relevantes, porém, também há aumento na frequência de recomendações relevantes por pedido, expresso pela métrica de *Likelihood*, na Tabela 3.

Compreende-se que há aumento gradual dos valores do *Likelihood* à medida em que K aumenta, visto que ao trazer mais BRs recomendados, há uma maior chance ser recuperada pelo menos uma recomendação relevante. Assim, para $K = 10$, a técnica proposta trouxe pelo menos uma recomendação relevante em aproximadamente 60% das *queries* que constituíram a amostra.

Para o contexto do presente trabalho, foram considerados relevantes também os resultados obtidos por valores de K maiores, pois compreende-se que os potenciais usuários de um sistema de recomendação dessa natureza navegarão por recomendações além do topo, justificando assim o fato de se analisar as 20 primeiras recomendações para cada *query*. A partir disso, percebe-se que o sistema atingiu desempenho expressivo, já que nas primeiras 20 recomendações obtidas, há uma chance de aproximadamente 70% de ser recuperado pelo menos um BR relevante, i.e, que foi assumido posteriormente pelo mesmo mantenedor do BR de entrada.

6.1 Comparação com Técnicas Alternativas

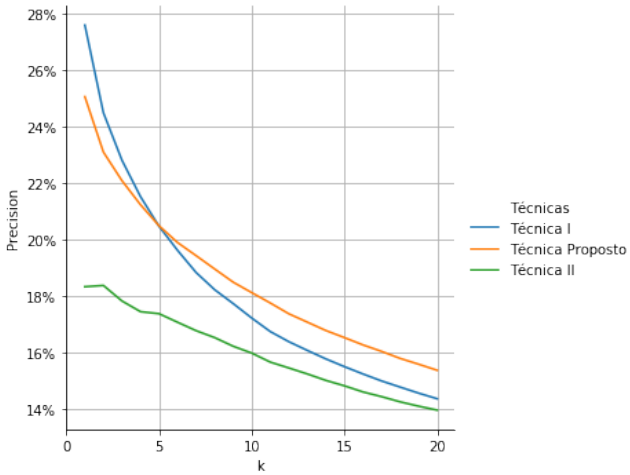

Figura 2: Comparação entre *Precisões* Obtidas para Valores de K

Em relação às metricas obtidas nas baterias de avaliação, percebeu-se uma diferença considerável em relação à *Precisão* entre TP, T1 e

[18]Disponível em: https://github.com/guimcarneiro/similar-bug-reports-recommender

T2, explicitada na Figura 2. Para o topo das recomendações, i.e, para valores de K menores, a T1 obteve precisão aproximadamente 3% superior à TP, decrescendo à medida em que K aumenta, indicando que, para as recomendações de "primeira página", a técnica composta apenas por TF-IDF e similaridade categórica atingiu melhores resultados, enquanto que para valores de $K \geq 5$, TP obteve uma frequência maior de resultados relevantes. Sobre as precisões obtidas por T2, foram consideravelmente inferiores às técnicas que têm em suas composições a vetorização TF-IDF, ressaltando a eficiência desse modelo para o contexto de recomendações textuais apesar de sua simplicidade, e também a ineficiência dos *word embeddings* se utilizados de maneira isolada no mesmo contexto.

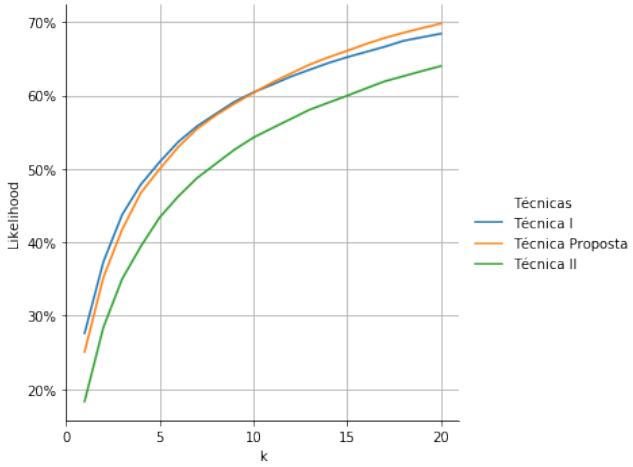

Figura 3: Comparação entre *Likelihoods* Obtidos para Valores de K

Em relação à frequência de pelo menos um item relevante dentre K, medido pelo *likelihood*, T1 e TP atingiram desempenhos bastante semelhantes, como mostrado pelo gráfico da Figura 3. Apesar disso, percebe-se novamente um leve prevalecimento de T1 para valores de K menores, e de TP para valores de K maiores. O desempenho de T2 se mantém alinhado ao gráfico da Figura 2, sendo consistentemente inferior para todos os valores de K, com diferença de aproximadamente 6% em relação às demais.

Comparando mais especificamente as técnicas TP e T1 a fim de atestar o efeito dos *word embeddings* sob o cálculo de similaridade, os gráficos presentes nas Figuras 4 e 5 explicitam o volume de recomendações por posição obtidos por cada técnica.

Para T1, houve a recuperação de aproximadamente 250 BRs relevantes a mais que TP para $K = 1$, comportamento que se mantém para K=2 com menor força, indicando que a Técnica I recupera mais relevantes para o topo das recomendações, porém esse comportamento se altera para valores de $K \geq 3$, onde TP atinge um volume consistentemente maior de BRs relevantes para todas as posições posteriores. Portanto, TP recupera um volume maior de relevantes considerando maiores intervalos de K, em consonância ao comportamento da precisão exposta na Figura 2, onde o desempenho de TP prevalece sob o de T1 à medida em que o K aumenta. Tal comportamento fica mais claro na Figura 5, que expressa a diferença

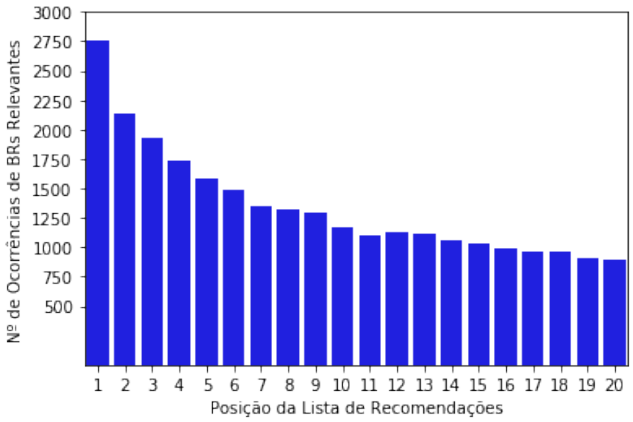

Figura 4: Ocorrências de BRs Relevantes por Posição (T1)

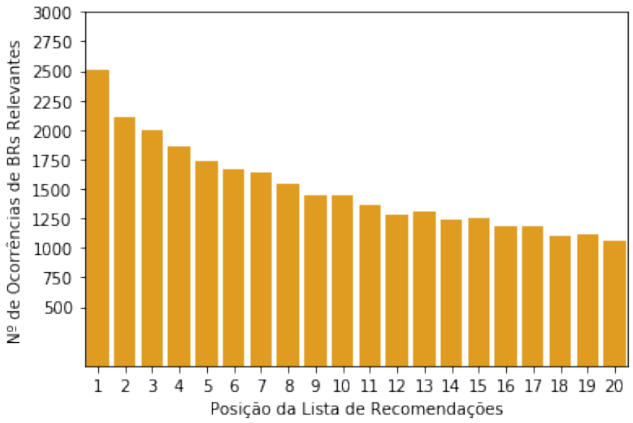

Figura 5: Ocorrências de BRs Relevantes por Posição (TP)

entre as técnicas TP e T1 em relação à quantidade de BRs relevantes recuperados em cada posição.

Aparentemente os *word embeddings* ajudam a capturar características de contexto textual nos quais as vetorizações TF-IDF não são eficientes, sendo útil na recuperação de similares em caráter de contexto - ao invés de frequência de palavras, que não seriam recomendados se não fosse utilizado um modelo mais complexo de compreensão textual. Argumenta-se que, para os BRs relevantes que foram recuperados por TP e não por T1, a parcela do score obtida pela similaridade de cosseno entre os *word embeddings* ajudou a recuperar BRs de similaridade menos óbvia, i.e, menos palavras diretamente em comum, mas com contexto qualitativamente mais próximo, que foram assumidos pelo mesmo responsável da *query* posteriormente, o que atesta possíveis ganhos de produtividade.

O ganho de produtividade pode ser compreendido através do seguinte exemplo: Um mantenedor economizaria tempo ao receber recomendações de BRs menos obviamente similares, visto que o sistema de recomendação recuperaria BRs não apenas semelhantes em questão de frequência de palavras, mas também semelhantes

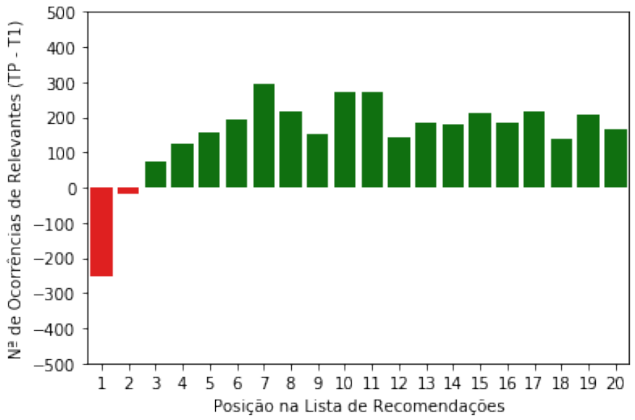

Figura 6: Diferença de Nº de Ocorrências de Relevantes entre TP e T1 (TP - T1)

em contexto, aumentando as chances de o próximo BR a ser assumido pelo mesmo mantenedor ser solucionado mais rapidamente, já que a mudança de contexto entre o BR anterior (resolvido pelo mantenedor) e o posterior (recomendado pelo sistema) seria menor.

6.2 Relação com o *NextBug*

A ferramenta proposta por Rocha et. al [9] chamada de *NextBug* possui objetivo bastante próximo do presente estudo: Recomendar BRs relevantes para mantenedores de repositórios *open-source*, baseando o cálculo dessas recomendações em conceitos de similaridade textual. Esta seção explicita sucintamente algumas diferenças entre os resultados obtidos pelo *NextBug* e pelo sistema de recomendação proposto, assim como algumas ressalvas relativas a essas diferenças. Ressaltamos ainda que não foi utilizada exatamente a mesma base de dados para avaliar ambos os trabalhos, e no presente estudo também fez-se uso de amostragem, diferentemente do *NextBug*, logo, não é possível comparar diretamente os resultados de ambos, apenas ressaltar pontos relevantes.

Para valores de *feedback*, o *NextBug* obteve uma taxa de 64% para K=1 [9], i. e. 6 em cada 10 *queries* retornarão pelo menos uma recomendação, não necessariamente relevante. Já o sistema proposto, para o mesmo valor de K, obteve taxa de *feedback* de aproximadamente 99,81%, mostrando uma frequência bem mais alta de *queries* que de fato obtêm alguma recomendação, independentemente da sua relevância.

A diferença considerável entre *feedbacks* se justifica devido à técnica proposta não implementar *threshold* para filtrar recomendações por um valor mínimo aceitável de similaridade, quase sempre retornando alguma recomendação, mesmo que pouco similar.

Em relação às métricas de precisão, o *NextBug* obteve valores de aproximadamente 30% para os valores de K=[1, 2, 3, 4, 5], enquanto que, para o mesmo intervalo de K, o presente trabalho obteve precisões decrescentes à medida em que o K aumenta, abrangendo valores de aproximadamente 25% para $K = 1$, até 20% para $K = 5$.

Sobre a frequência de ocorrência de pelo menos um BR relevante dentre K, relativo à métrica de *likelihood*, novamente houve disparidade entre ambos os trabalhos. O *NextBug* apresentou *likelihood* de 57% para K=3, enquanto o presente trabalho 41% para o mesmo valor de K.

Apesar de o *NextBug* apresentar métricas para precisão e *likelihood* superiores à técnica proposta na seção 4.2, deve-se ressaltar a forma como essas métricas são calculadas, explicitadas na seção 5.3, onde apenas as *queries* que obtém pelo menos K recomendações são consideradas no cálculo. O fato de o presente trabalho apresentar taxas de *feedback* consideravelmente altas provoca a diminuição das outras duas taxas, já que há uma maior quantidade de *queries* a serem avaliadas, inclusive aquelas com recomendações pouco ou nada relevantes, diminuindo a *Precisão* e o *Likelihood*.

Como o *NextBug* adota um *threshold* de similaridade de 10% em sua avaliação, *queries* que não obtiveram recomendações não foram refletidas no cálculo da precisão e *likelihood*, fazendo essas métricas aumentarem, ao mesmo tempo em que o *feedback* diminui.

Percebe-se que houve diferentes *trade-offs* entre o *NextBug* e o sistema proposto: O primeiro "sacrificou"*feedback* em busca de precisão e *likelihood* por meio da implementação do *threshold* de similaridade, enquanto que o segundo obteve menor precisão e *likelihood*, mas com *feedback* próximo de 100% para todos os valores de K, por não implementar o mesmo *threshold*.

Por fim, a decisão de não implementar *threshold* de similaridade no presente trabalho impossibilitou a comparação deste com o *NextBug*, além do fato de o presente trabalho utilizar apenas uma amostra do *dataset* de avaliação utilizado pelo *NextBug*. Comparação esta que pode ser mais explorada em trabalhos futuros.

6.3 *Q1: O BERT agrega melhoria de desempenho no contexto de recomendação de Bug Reports similares, considerando as métricas de Precisão, Likelihood e Feedback?*

De acordo com os resultados já expostos nos parágrafos anteriores, houve melhora das métricas de *Likelihood* e precisão para valores de $K \geq 3$, quando utilizada a técnica proposta (TP) para cálculo do score de similaridade. Comparando-se esta com T1, que utiliza apenas TF-IDF como vetorizador textual, fica mais visível o efeito positivo dos *word embeddings* na relevância das recomendações, como apresentado na Figura 6. No gráfico da Figura 7 é possível perceber, em porcentagem, o ganho na frequência de relevantes por posição, comparando-se TP, que utiliza TF-IDF e *word embeddings*, com T1, que utiliza apenas TF-IDF.

Com o uso do BERT associado ao TF-IDF, atesta-se um ganho de, em média, 14% na frequência de recomendações relevantes, i.e, de BRs que foram assumidos posteriormente pelo mantenedor responsável pelo BR de entrada. Tal fato não necessariamente indica ganhos de produtividade, mas sugere a presença de melhores resultados, já que as sugestões trazidas pelo sistema seriam mais interessantes para o usuário final (mantenedor). Tal fato aceleraria o processo de encontrar um responsável capaz de resolver um BR pendente a partir de outro resolvido antes, de contexto próximo, contribuindo assim para o aumento das taxas de Bug Reports resolvidos.

Como ressalva à utilização do BERT no contexto de recomendações de Bug Reports similares, o custo computacional necessário para gerar os *word embeddings* é alto, principalmente considerando a realidade na qual há grande volume de BRs novos sendo abertos

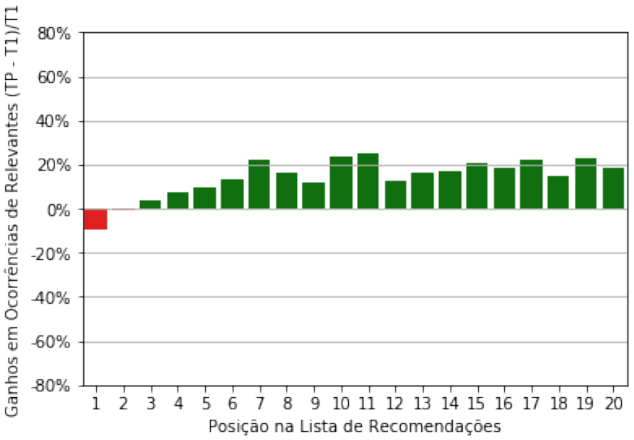

Figura 7: Ganhos em Ocorrências de Relevantes

diariamente, o que exigiria poder computacional elevado na tentativa de manter como candidatos à recomendações até os BRs mais recentes.

Além disso, os *word embeddings* gerados pelo BERT não mostraram resultados expressivos quando utilizados sem a cooperação das vetorizações TF-IDF, explicitado por T2 nas Figuras 2 e 3. Isso atesta que o BERT foi útil na recomendação de BRs similares, mas como método complementar no cálculo do score de similaridade, ainda necessitando do TF-IDF para atingir resultados positivos.

7 CONCLUSÃO

Por fim, através do presente estudo, propomos uma aplicação do modelo BERT no escopo de recomendação de Bug Reports similares, buscando identificar seu desempenho, e associando-o a técnicas já conhecidas, como a vetorização TF-IDF e similaridade categórica. Neste processo, confirmou-se o desempenho positivo do modelo BERT no contexto de recomendações quando utilizado de maneira complementar ao modelo TF-IDF, tendo atingido aumento de aproximadamente 14% no volume de Bug Reports relevantes recomendados, se comparado à técnica que utiliza apenas vetorização TF-IDF, para as 20 primeiras recomendações ($K = 20$).

Há ressalvas quanto à alta demanda computacional requerida pelo BERT no cálculo dos *word embeddings*, e se o ganho de desempenho "compensa"tal custo, já que técnicas computacionalmente mais simples, com apenas similaridade categórica e TF-IDF (T1), por exemplo, obtiveram métricas de desempenho razoáveis com custo bastante menor. Para concluir, para trabalhos futuros pode-se utilizar outras variações do modelo BERT, inclusive implementações que tenham capacidade de *tokens* maior que 256 e estudar qualitativamente os resultados obtidos.

8 AGRADECIMENTOS

Agradeço ao meu orientador, prof. Franklin Ramalho, pelo acompanhamento desde projetos anteriores, e de ter assumido tal responsabilidade com empenho. Agradeço também a Manoel Ferreira por toda ajuda e paciência, desdes os tempos do e-Pol. Por fim, sou grato a minha namorada, família e amigos que sempre me acompanharam nesse percurso.

REFERÊNCIAS

[1] Alura. 2021. "Guia de NLP - Conceitos e Técnicas". Disponivel em: https://www.alura.com.br/artigos/guia-nlp-conceitos-tecnicas.
[2] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2005. Coping with an Open Bug Repository. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology EXchange (eclipse '05)*. Association for Computing Machinery, New York, NY, USA, 35–39. https://doi.org/10.1145/1117696.1117704
[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/ARXIV.1810.04805
[4] Houaiss. [n. d.]. Definição da palavra "Contexto". Disponivel em: https://houaiss.uol.com.br.
[5] Folasade Isinkaye, Yetunde Folajimi, and Bolanle Ojokoh. 2015. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal* 16 (08 2015). https://doi.org/10.1016/j.eij.2015.06.005
[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. https://doi.org/10.48550/ARXIV.1310.4546
[7] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://doi.org/10.48550/ARXIV.1908.10084
[8] Google Research. 2023. "What is BERT?". Disponivel em: https://github.com/google-research/bert.
[9] Henrique Rocha, Guilherme de Oliveira, Humberto Marques-Neto, and Marco Tulio Valente. 2015. NextBug: a Bugzilla extension for recommending similar bugs. *Journal of Software Engineering Research and Development* 3 (Mar. 2015), 3:1 – 3:14. https://sol.sbc.org.br/journals/index.php/jserd/article/view/419
[10] Lalita Sharma and Anju Gera. 2013. A Survey of Recommendation System: Research Challenges.
[11] LiHong Xu, ShuTao Sun, and Qi Wang. 2016. Text similarity algorithm based on semantic vector space model. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. 1–4. https://doi.org/10.1109/ICIS.2016.7550928
[12] Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. 2016. Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 127–137. https://doi.org/10.1109/ISSRE.2016.33
[13] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From Word Embeddings to Document Similarities for Improved Information Retrieval in Software Engineering. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 404–415. https://doi.org/10.1145/2884781.2884862