DESENVOLVIMENTO DE UMA FERRAMENTA DE PESQUISA UTILIZANDO ALGORITMOS DE LEVENSHTEIN E TF-IDF

Diego Ted Rodrigues Bogea[1]

Mirian Ferreira da Silva Bogea[2]

André da Silva Freitas[3]

RESUMO: O gerenciamento de informações em bases de dados tem se tornado cada vez mais desafiador com o aumento exponencial de documentos gerados online e offline. Este artigo explora a aplicação de técnicas de recuperação de informações na ferramenta de pesquisa *BugSearch*, utilizando TF-IDF (*Term Frequency-Inverse Document Frequency*) e a distância de Levenshtein para melhorar a precisão e relevância dos resultados. A *BugSearch* combina um web crawler para criar um dicionário de termos e uma interface desenvolvida em Flutter, processando consultas com algoritmos avançados e integração com a API do OpenAI para fornecer respostas geradas por Inteligência Artificial. A eficácia dessas técnicas foi avaliada, destacando a importância da combinação de TF-IDF e Levenshtein na recuperação robusta de informações.

Palavras-Chave: Recuperação de informações, TF-IDF, Distância de Levenshtein.

ABSTRACT: Managing information within databases has become increasingly challenging with the exponential growth of online and offline documents. This paper explores the application of information retrieval techniques in the BugSearch search tool, utilizing Term Frequency-Inverse Document Frequency (TF-IDF) and Levenshtein distance to enhance the accuracy and relevance of search results. BugSearch combines a web crawler to create a term dictionary and an interface developed in Flutter, processing queries with advanced algorithms and integrating with the OpenAI API to provide AI-generated responses. The effectiveness of these techniques was evaluated, highlighting the importance of combining TF-IDF and Levenshtein in robust information retrieval.

Keywords: Information retrieval, TF-IDF, Levenshtein distance

[1] Mestre em Educação. Professor de Metodologia Científica do Instituto Federal do Maranhão – IFMA Campus Imperatriz. diegobogea@ifma.edu.br

[2] Mestra em Educação. Professora de Metodologia Científica do Instituto Federal do Maranhão – IFMA Campus Imperatriz. mirian.bogea@ifma.edu.br

[3] Discente do Curso de Bacharelado em Ciência da Computação do Instituto Federal do Maranhão – IFMA Campus Imperatriz

1. INTRODUÇÃO

O gerenciamento de informações dentro de bases de dados vem se tornando um trabalho cada vez mais árduo e com uma linha de dificuldade em constante subida. O advento da era da informação faz com que a geração de documentos online e offline aumente de maneira exponencial, esse aumento por sua vez, gera a necessidade do desenvolvimento de metodologias para uma melhor organização e acesso a essas informações geradas, as informações armazenadas dentro de um Sistema de Gerenciamento de Banco de Dados (SGBD) por exemplo, necessitam de um alto nível de organização, já que que grandes bancos de dados contém informações importantes que permite empresas, por exemplo, a tomar decisões e gerar conhecimento (Ruberto et al, 2017).

As dificuldades no desenvolvimento de metodologias de organização surgem devido a problemas como a remoção de múltiplas versões do mesmo documento e a extração de conjuntos relevantes de grandes repositórios. Esses desafios requerem a criação de métodos que permitam acessar facilmente os valores de interesse especificados conforme a necessidade. (Bafna et al, 2016)

Nesse contexto, surge o tema da recuperação da informação e o desenvolvimento de Sistemas de Recuperação da Informação (SRI). Esses sistemas possibilitam diversos pontos de acesso à informação, como o planejamento de estratégias de busca mais complexas, que envolvem múltiplos conceitos na mesma consulta. Eles permitem a busca de palavras específicas em linguagens controladas nos campos de descritores, a pesquisa de termos compostos ou simples, além da possibilidade de truncagem de raízes de palavras e substituição de caracteres dentro dos termos, entre outros recursos de recuperação. (Lopes, 2002).

Ferramentas de busca na web, como o Google, beneficiam-se significativamente dos Sistemas de Recuperação da Informação (SRIs). Elas utilizam diversas técnicas para agrupar informações, permitindo que dados desestruturados sejam transformados em dados estruturados. Esses sistemas criam grupos de relevância para cada documento pesquisado, definindo seu nível de importância com base na consulta do usuário (Bafna et al., 2016).

Com base nessas informações, este artigo busca explorar a aplicação de técnicas de recuperação de informações em uma ferramenta de pesquisa denominada Bug Search. A ferramenta utiliza a técnica TF-IDF (*Term Frequency-Inverse Document Frequency*) e o algoritmo de cálculo da Distância de Levenshtein para melhorar a precisão e relevância dos resultados de busca. O objetivo é avaliar a eficácia da combinação dessas técnicas na

recuperação de informações, proporcionando uma solução mais eficiente para a busca de dados em grandes repositórios.

2. Fundamentação Teórica

2.2. TF-IDF

Na classificação de textos, um documento pode ser relacionado a diversas categorias. Para encontrar a melhor categoria combinatória, utiliza-se a técnica TF-IDF (Term Frequency-Inverse Document Frequency). Essa técnica é eficaz na filtragem de palavras irrelevantes, conhecidas como palavras de parada. A premissa é que palavras que aparecem em quase todos os documentos são menos relevantes do que aquelas que aparecem concentradas em um conjunto reduzido de documentos. (Noda, 2020)

Como descrito por Trstenjak (2014), a frequência do termo (TF) é calculada como a razão entre o número de vezes que o termo *t* aparece no documento *d* e o número total de termos no documento *d*. Isso ajuda a normalizar a contagem de termos, permitindo comparações entre documentos de diferentes tamanhos. Já a frequência inversa do documento (IDF) é calculada usando a fórmula:

$$log(\frac{N}{1 + Nt})$$

Onde N é o número total de documentos no corpus e N_t é o número de documentos que contêm o termo *t*. A utilização do logaritmo serve para atenuar a diferença de frequências, enquanto a adição de 1 ao denominador evita divisões por zero. Sendo assim a fórmula completa pode ser representada da seguinte maneira:

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

O resultado da multiplicação do TF pelo IDF atribui um peso a cada termo em cada documento, conhecido como valor TF-IDF. Termos com alto valor TF-IDF são considerados mais importantes para o documento, pois são frequentes nesse documento específico, mas não comuns em todos os documentos do corpus. Quando se trata de pontuar expressões compostas por vários termos, o TF-IDF da expressão é calculado somando os valores TF-IDF individuais de cada termo (Manning, 2008).

Isso torna o TF-IDF uma ferramenta poderosa para diversas aplicações de processamento de linguagem natural, como a classificação de textos, a recuperação de

informações e a filtragem de spam, onde a relevância dos termos é crucial para a precisão dos resultados.

2.3 Distância de Levenshtein

A distância de Levenshtein (LD) mede a similaridade entre duas sequências de caracteres, sendo uma a string de origem (s) e a outra a string de destino (t). Essa distância é determinada pelo número de operações de deleção, inserção ou substituição necessárias para transformar a string de origem na string de destino. Quanto maior a distância de Levenshtein, maior a diferença entre as strings. No nosso caso, a string de origem é a entrada fornecida pelo usuário, enquanto a string de destino é uma das entradas presentes na base de dados. (Haldar, 2011)

Primeiro, define-se o comprimento das duas strings e inicia-se uma matriz com essas dimensões, onde a primeira linha e coluna representam as operações necessárias para transformar uma string vazia na outra. Em seguida, o algoritmo preenche a matriz comparando cada caractere das duas strings. Se os caracteres forem iguais, o custo é 0; se forem diferentes, o custo é 1. O valor de cada célula é determinado pelo menor custo entre a deleção (valor da célula acima mais 1), a inserção (valor da célula à esquerda mais 1) ou a substituição (valor da célula diagonalmente acima e à esquerda mais o custo). Esse processo continua até que a célula final da matriz contenha o valor da distância de Levenshtein, que representa o número mínimo de operações necessárias para transformar a string de origem na string de destino (Haldar, 2011).

O uso do cálculo da distância de Levenshtein oferece vantagens significativas na análise de textos, especialmente na correção ortográfica e na correspondência de padrões, ao quantificar a similaridade entre strings por meio de operações simples. Quando aplicado em conjunto com o TF-IDF, o algoritmo de Levenshtein pode aprimorar a precisão dos sistemas de recuperação de informação. O TF-IDF avalia a relevância dos termos em um corpo, enquanto a distância de Levenshtein permite identificar variações e erros de digitação nos termos de busca, tornando a recuperação de documentos mais robusta e eficiente. Essa combinação é particularmente útil em sistemas de busca e recomendação, onde a correção de pequenas divergências nos termos de consulta pode melhorar significativamente a qualidade dos resultados retornados ao usuário.

2.4 *Web Crawling*

Um *web crawler* é um sistema projetado para o download em massa de páginas da web. Eles são utilizados por diversas razões, Olston (2010) descreve um dos usos principais: a construção de ferramentas de pesquisa e sistemas que agregam várias páginas da web, indexando-as para permitir que os usuários façam buscas eficientes dentro dessa coleção de páginas. Dessa forma, os *web crawlers* permitem a criação de um grande dicionário de informações da web, que pode ser facilmente filtrado com base em seus conteúdos. Quando combinados com técnicas de agrupamento de documentos, como a análise de distância de Levenshtein e o cálculo de TF-IDF, os web crawlers facilitam o estudo e desenvolvimento de ferramentas de pesquisa mais robustas e completas.

3. Desenvolvimento da Aplicação

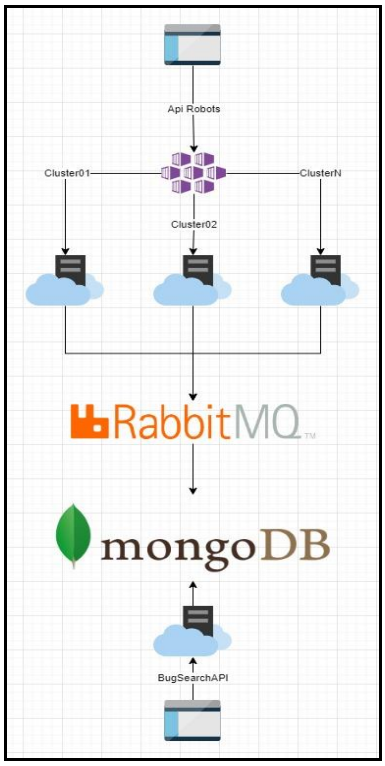
Para a confecção da ferramenta de pesquisa *BugSearch*, inicialmente foi criado um crawler chamado *BugSearch.Crawler* para realizar o *web crawling* necessário à construção do dicionário de páginas web utilizado pelo motor de busca. Embora as ferramentas tenham sido inicialmente desenvolvidas em JavaScript, posteriormente foram convertidas para a linguagem C# (CSharp) (Microsoft, 2024), para melhor desempenho e manutenção. A ferramenta é composta por duas partes principais: o lado do Bot, responsável pelo rastreamento e indexação das páginas, e o lado do Cliente, que lida com as consultas dos usuários e a apresentação dos resultados.

3.1 Lado do Bot

O lado do Bot é encarregado de operar o *crawler*, que navega pelas páginas da web e verifica recursivamente os links nelas contidos. Os dados são armazenados em duas coleções no banco de dados MongoDB (MongoDB, 2024): uma para o dicionário de termos e outra para os *EventCrawlers*. Um *EventCrawler* é um modelo que compreende informações detalhadas sobre metadados encontrados no dicionário montado, incluindo URL, favicon, título, descrição, conteúdo, termos relevantes e a pontuação associada pelo algoritmo.

A comunicação com o *RabbitMQ* (RabbitMQ, 2024), um sistema de fila, gerencia o armazenamento de dados para processamento e reprocessamento em caso de falhas. Posteriormente, esses dados são persistidos no banco MongoDB para armazenamento seguro e durável como representado pela Figura 1.

Figura 1: Representação gráfica do lado do bot no sistema



Fonte: Autores

3.2 Lado do Cliente

No lado do Cliente, foi desenvolvida uma interface gráfica utilizando a *framework Flutter* (Flutter, 2024), que possibilita aos usuários realizar consultas nos sites catalogados. A interface exibe um resumo da quantidade de sites e termos armazenados no banco de dados.

O lado do Cliente conta com três principais pontos de acesso – ou *endpoints* – em sua estrutura: o ponto de acesso de Pesquisa que permite pesquisar sites no *MongoDB* com base em uma consulta e um limite/profundidade. Ele utiliza os algoritmos TF-IDF e Levenshtein para calcular uma pontuação que reorganiza os links a serem exibidos na resposta. O ponto de acesso do Resumo que retorna um resumo dos sites catalogados e a quantidade de termos registrados. Por fim, há o ponto de acesso de *Prompt*, criado como uma função extra dentro da ferramenta, que integra o algoritmo da OpenAI (OpenAI, 2024). Este ponto de acesso pode ser chamado em paralelo com a pesquisa, retornando uma resposta gerada por uma inteligência artificial (IA) com base na consulta fornecida. A integração de motores de busca com IA se tornou comum e, como afirma Ashraf (2024), essa combinação pode revolucionar os motores de busca, gerando resultados mais personalizados por meio de interfaces fáceis de utilizar.

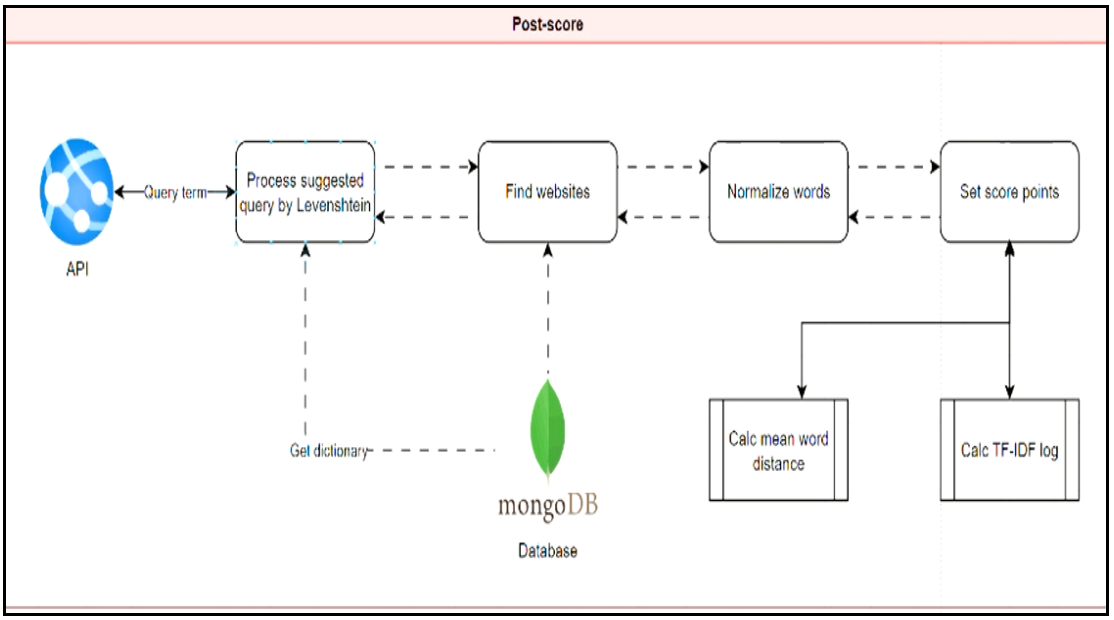
O lado do Cliente da aplicação é armazenado em um *Docker*, uma tecnologia que permite criar, implantar e executar aplicações em contêineres. Um contêiner é uma unidade de

software que empacota o código da aplicação e todas as suas dependências, garantindo que ela seja executada de forma consistente em diferentes ambientes. Isso facilita o desenvolvimento, teste e implantação da aplicação, eliminando problemas relacionados a diferenças de configuração entre os ambientes. A utilização do *Docker* garante que o lado do Cliente da aplicação *BugSearch* funcione de maneira previsível e eficiente, independentemente do sistema onde está sendo executado (Docker, 2024) .

3.3 Cálculo da Pontuação

A Figura 2 representa o fluxo lógico de cálculo da pontuação pelo sistema. A interface programável da aplicação – Application Programming Interface (API) – recebe o termo de consulta, que é processado utilizando o algoritmo de Levenshtein, sugerindo correções ou aproximações com base no dicionário de termos armazenado no MongoDB. Em seguida, o sistema busca sites relevantes no banco de dados do MongoDB, que contém um índice ou dicionário de termos e outros metadados. As palavras encontradas são normalizadas, um processo que pode incluir a conversão para letras minúsculas, remoção de pontuações e outras transformações para garantir consistência na análise. Posteriormente, é calculada a distância de Levenshtein média entre as palavras da consulta e as palavras nos documentos encontrados. Além disso, o valor de TF-IDF é calculado para as palavras dos documentos encontrados, gerando assim a pontuação atribuída aos websites, indicando sua relevância em relação à consulta do usuário.

Figura 2: Fluxo do funcionamento da ferramenta de pesquisa



Fonte: Autores

4. Conclusão

Durante as etapas iniciais de desenvolvimento do *BugSearch*, diversos conceitos relacionados à recuperação de informação foram explorados, proporcionando uma compreensão mais profunda das metodologias aplicadas no desenvolvimento de motores de busca e seus algoritmos. O foco principal foi nas aplicações dos cálculos de TF-IDF e LD, que se mostraram extremamente eficazes quando utilizados no algoritmo criado. Além disso, foi possível identificar outros métodos potenciais, como os algoritmos de Soundex e Megaphone, conforme discutido por Ruberto (2017), que poderiam ser aplicados para aprimorar ainda mais o sistema.

A implementação da API do OpenAI, que utiliza prompts criados por uma IA generativa, conferiu à aplicação uma abordagem mais moderna, permitindo a geração de análises automáticas baseadas no dicionário de termos. Em um ambiente controlado, essa funcionalidade oferece um nível adicional de acesso à informação. No entanto, como descrito no estudo de Zhao (2023), a precisão dos dados gerados pode ser comprometida dependendo dos filtros aplicados pela IA, o que pode resultar na criação de dados falsos ou enganosos.

Caso o sistema continue a ser desenvolvido, mais testes poderiam ser realizados analisando a capacidade de uso da ferramenta e o quanto poderia ser utilizada. Atualmente a ferramenta ser armazenada em um container de Docker, permite que ela seja utilizada em qualquer sistema operacional, no entanto, por ser um sistema muito grande, acaba por gastar muita memória, neste caso mais estudos seriam necessários para otimizar seu funcionamento.

REFERÊNCIAS

ASHRAF, Amir Reza; MACKEY, Tim Ken; FITTLER, András. Search Engines and Generative Artificial Intelligence Integration: Public Health Risks and Recommendations to Safeguard Consumers Online. **JMIR Public Health and Surveillance**, v. 10, n. 1, p. e53086, 2024.

BAFNA, Prafulla; PRAMOD, Dhanya; VAIDYA, Anagha. Document clustering: TF-IDF approach. In: 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT). IEEE, 2016. p. 61-66.

DOCKER. Docker. Disponível em: https://www.docker.com. Acesso em: 15 jun. 2024.

FLUTTER. Disponível em: <https://flutter.dev/>. Acesso em: 14 jun. 2024.

Haldar, Rishin, and Debajyoti Mukhopadhyay. "Levenshtein distance technique in dictionary lookup methods: An improved approach." arXiv preprint arXiv:1101.1232 (2011).

LOPES, Ilza Leite. Estratégia de busca na recuperação da informação: revisão da literatura. Ciência da Informação, v. 31, p. 60-71, 2002. MongoDB. Disponível em: <https://www.mongodb.com/>. Acesso em: 14 jun. 2024.

MICROSOFT. C#. Disponível em: <https://dotnet.microsoft.com/pt-br/languages/csharp>. Acesso em: 14 jun. 2024.

MANNING, Christopher D. **Introduction to information retrieval**. Syngress Publishing,, 2008.

NODA, Mauricio. **Levantamento de indicadores através de data mining, Latent Dirichlet Allocation e TF-IDF**. 2020. Tese de Doutorado.

OLSTON, Christopher et al. Web crawling. **Foundations and Trends® in Information Retrieval**, v. 4, n. 3, p. 175-246, 2010.

RABBITMQ**.** RabbitMQ. Disponível em: https://www.rabbitmq.com/. Acesso em: 15 jun. 2024.

RUBERTO, Diogo Luis Von Grafen; ANTONIAZZI, Rodrigo Luiz. Análise e comparaçao de algoritmos de similaridade e distância entre strings adaptados ao português brasileiro. In: **Anais da XIII Escola Regional de Banco de Dados**. SBC, 2017.

TRSTENJAK, Bruno; MIKAC, Sasa; DONKO, Dzenana. KNN with TF-IDF based framework for text categorization. **Procedia Engineering**, v. 69, p. 1356-1364, 2014.

ZHAO, Ruochen et al. Can chatgpt-like generative models guarantee factual accuracy? on the mistakes of new generation search engines. arXiv preprint arXiv:2304.11076, 2023.