Henrique Araújo de Carvalho

Recuperação de Informação Baseada em Embeddings: Desenvolvendo um Recuperador Denso para o Repositório de Teses da Usp

São Paulo, Brasil

2023

Henrique Araújo de Carvalho

Recuperação de Informação Baseada em Embeddings: Desenvolvendo um Recuperador Denso para o Repositório de Teses da Usp

Universidade de São Paulo
Instituto de Matemática e Estatística
Graduação

Orientador Dr. Daniel Macêdo Batista

São Paulo, Brasil
2023

Agradecimentos

À Isabela, pela paciência, pelo incentivo, pela dedicação comigo, pelos inestimáveis conselhos, por acreditar mais em mim do que eu mesmo, por me aguentar, com curiosidade, falar sobre computação, e por todo o amor. Eu não poderia esperar mais. Com você eu aprendo todos os dias e eu quero, um dia, poder retribuir toda essa sabedoria que eu tomei emprestada.

Aos meus amigos do IME, pelas valiosas conversas e pelos momentos de companheirismo, pelas "idas à lousa", por me ensinarem sobre Computação, sobre Matemática e sobre a vida. O IME não teria sido o IME sem vocês.

Aos amigos em Brasília e aos amigos em São Paulo, que também escutaram pacientemente minhas aflições ao longo do curso de ciência da computação e me apoiaram incondicionalmente. Eu espero ter mais oportunidade de estar mais presente agora.

Ao meu orientador, Prof. Dr. Daniel Macêdo, pela competência, pela atenção e dedicação com este orientando, e pelas boas conversas. Você me faz ter orgulho de ter estudado na Universidade de São Paulo.

Resumo

Este trabalho investiga o desenvolvimento e a aplicação de um sistema de recuperação densa em buscas de literatura acadêmica, inspirado por trabalhos recentes que demonstram a eficácia desses sistemas em contextos de perguntas e respostas. A hipótese deste trabalho é que um sistema de recuperação baseado em representações vetoriais densas (*embeddings*) de passagens dos títulos, resumos e palavras-chave de trabalhos acadêmicos pode ser usada para aprimorar a recuperação desse tipo de literatura. Para testar essa hipótese, foram realizados experimentos com três sistemas distintos: dois implementados por mim, um baseado em vetores esparsos e outro baseado em vetores densos, e o sistema utilizado pelo repositório de teses da Universidade de São Paulo. A análise de desempenho foi realizada com 1800 avaliações de usuários em 45 consultas. Os resultados preliminares mostram que o sistema implementado com vetores densos (a) superou o desempenho do sistema utilizado pelo repositório de teses da Universidade de São Paulo em aproximadamente 12 pontos; e (b) superou o sistema baseado em vetores esparsos em 36 pontos. Estes resultados indicam um potencial avanço na eficácia de recuperação de informações usando modelos de linguagem.

Palavras-chaves: Recuperação Densa. Embeddings. Vetores Densos. Vetores Esparsos. Passage Retrieval.

Abstract

This work investigates the development and deployment of a dense retrieval system in searches of academic literature, inspired by recent works demonstrating these systems' effectiveness in Q&A contexts. The hypothesis of this work is that a retrieval system based on dense vector representations (*embeddings*) of passages from titles, abstracts, and keywords of academic papers can be used to enhance the retrieval of this type of literature. To test this hypothesis, experiments were conducted with three distinct systems: two implemented by me, one based on sparse vectors and the other based on dense vectors, and the system used by the thesis repository of the University of São Paulo. Performance analysis was carried out with 1800 user evaluations on 45 queries. Preliminary results show that the system implemented with dense vectors (a) outperformed the performance of the system used by the thesis repository of the University of São Paulo by approximately 12 points; and (b) surpassed the system based on sparse vectors by 36 points. These results indicate a potential advancement in the efficacy of information retrieval using language models.

Keywords: Dense Retrieval. Embeddings. Dense Vectors. Sparse Vectors. Passage Retrieval.

Sumário

Introdução

Em 2020, os autores Karpukhin et al. demonstraram que a recuperação de passagens em sistemas de Perguntas e Respostas (*Q&A - Question Answering*) utilizando recuperação densa superava os métodos tradicionais de busca baseados em palavras-chave, como o algoritmo Best Match 25 (BM25). Esse trabalho foi um dos primeiros a destacar o potencial de abordagens neurais em sistemas de busca usando representações vetoriais densas de texto [1].

Teses e dissertações produzidas em programas de pós-graduação são importantes na construção do conhecimento, o que ressalta a importância da recuperação de teses acadêmicas para disseminação de resultados de pesquisa, principalmente com o aumento no número de trabalhos acadêmicos desenvolvidos nos últimos anos, apesar da queda pontual ocorrida em 2022, provavelmente como efeito da pandemia de COVID-19 [2]. Essa necessidade crescente de uma recuperação eficiente de teses acadêmicas, particularmente evidenciada durante a pandemia [3] , levanta questões sobre a adequação dos sistemas de busca atuais. Em particular, analisando o sistema de busca do repositório de teses da Universidade de São Paulo, fica claro que existem limitações significativas a serem dirimidas.

Isso me levou à hipótese de que um sistema de recuperação baseado em vetores densos poderia oferecer alguma melhoria na busca de literatura acadêmica. O que poderia ser feito empregando simplesmente *tokens* criados com passagens dos títulos, resumos e palavras-chave dos documentos acadêmicos. Melhorar a busca significa, aqui, ser capaz de identificar e priorizar documentos que sejam mais relevantes e úteis para os pesquisadores em suas consultas.

Em 2020, os autores Cohan et al. propuseram o SPECTER, um modelo de linguagem capaz de gerar embeddings de documentos acadêmicos, treinado usando grafo de citações de trabalhos acadêmicos. Os autores concluíram, como resultado do trabalho, que a utilização de um modelo de linguagem *"off-the-shelf"* para gerar *embeddings* de passagens de títulos, resumos e palavras-chave não resultou em representações acuradas para os trabalhos acadêmicos; e, como consequência, essa não seria uma boa estratégia para a criação de um recuperador de literatura acadêmica [4].

---

[1] Representações vetoriais densas de texto também são chamadas, indistintamente, ao longo do texto, de vetores densos, ou de *embeddings*.

[2] <https://abori.com.br/relatorios/2022-um-ano-de-queda-na-producao-cientifica-para-23-paises-inclusive-o-brasil/>

[3] <https://www.kaggle.com/c/trec-covid-information-retrieval>

[4] Este problema é particularmente complexo devido à natureza diversificada e multidisciplinar da literatura acadêmica. Além disso, relevância neste trabalho tem uma definição simplista de similaridade de assunto (ou semântica), o que não necessariamente equivale à relevância em consultas acadêmicas.

Modelos de linguagem *"off-the-shelf"* avançaram bastante desde a publicação do SPECTER e, para este trabalho, proponho a criação de um recuperador de literatura acadêmica com a utilização unicamente de *embeddings* de texto, pré-treinados para tarefas de similaridade semântica.

Para avaliar minha hipótese, comparei o sistema utilizado pelo repositório de teses da Universidade de São Paulo com dois sistemas de ranqueamento distintos: (a) um baseado em vetores esparsos, que utiliza Apache Solr/Lucene, que funciona predominantemente por meio da correspondência de palavras-chave, mecanismo bastante comum em sistemas de busca tradicionais; e (b) outro baseado em vetores densos, que emprega a biblioteca Sentence Transformers e modelos de linguagem código aberto disponíveis publicamente. Esse último baseia-se em representações vetoriais que capturam a semântica do texto, permitindo uma correspondência mais rica, que vai além da correspondência exata.

O objetivo deste trabalho é duplo: (a) descrever o funcionamento e o desenvolvimento dos dois sistemas propostos, utilizando bibliotecas amplamente disponíveis; e (b) desenvolver experimentos para comparar a qualidade entre os sistemas implementados e o sistema do repositório de teses da Universidade de São Paulo.

Para avaliar efetivamente esses sistemas, coletei avaliações feitas por usuários e conduzi uma série de experimentos para análise de resultados [5]. Foram coletadas 1800 avaliações de um total de 45 consultas. Além disso, o estudo inclui minha análise sobre métodos de geração de embeddings, comparações entre formas de fazer consultas no repositório de teses da Universidade de São Paulo e entre diversos modelos baseados em configurações variadas.

Este estudo visou estabelecer bases para a criação de um novo sistema de recuperação de trabalhos acadêmicos nacionais, avaliando seu sucesso pela comparação com o sistema de busca da Biblioteca Digital de Teses e Dissertações da Universidade de São Paulo.

Os resultados preliminares, usando métricas de desempenho em recuperação de informação, nDCG@10 e Precisão@10, demonstram que o sistema com vetores densos superou o desempenho do sistema atual da Universidade de São Paulo em 12 pontos e o sistema baseado em vetores esparsos em 36 pontos.

Estes achados indicam um avanço potencial na eficiência de recuperação de informações em repositórios acadêmicos. No entanto, a continuação da pesquisa e a melhora da eficiência de ambos os sistemas descritos ainda é necessária de modo a garantir que um sistema performa melhor que o outro.

[5] Em geral, sistemas de recuperação são avaliados em datasets do Text REtrieval Conference (TREC), mas a intenção desse estudo não é estabelecer a eficácia de um sistema de recuperação para datasets genéricos, mas para consultas reais de usuários que usam um sistema de recuperação para literatura acadêmica.

O restante deste documento está organizado da seguinte forma: no Capítulo 1, apresento os fundamentos para compreensão dos sistemas implementados. Inicio com uma visão geral sobre a evolução e conceitos-chave da Recuperação de Informação, seguida por uma análise detalhada dos Sistemas e Modelos de Recuperação. Abordo o problema do ranqueamento de documentos, um subproblema da recuperação de informação, explorando tanto os modelos de recuperação esparsa, baseados em correspondência exata, quanto os de recuperação densa, baseados na busca em espaços vetoriais de vetores densos. Finalizo o capítulo com uma revisão das métricas fundamentais para avaliar a eficácia dos sistemas de recuperação de informação, estabelecendo assim a base teórica essencial para o desenvolvimento e avaliação dos sistemas que proponho em minha pesquisa. No Capítulo 2 descrevo a arquitetura geral de um sistema de recuperação de informação e especializo essa arquitetura para descrever os dois sistemas que foram implementados. No Capítulo 3, são relatados os experimentos que foram realizados para avaliar o desempenho dos três sitemas considerados. O Capítulo 4 finaliza a monografia apresentando a conclusão e sugestões para trabalhos futuros.

1 Referencial Teórico

Este capítulo estabelece as bases conceituais e metodológicas para a compreensão dos sistemas de recuperação de informação e para a análise de dois modelos distintos de recuperação de informação através de uma avaliação comparativa.

Na Seção 1.1 discuto os fundamentos e a natureza da recuperação de informação. Na Seção 1.2 discuto os sistemas de recuperação de informação como implementações práticas de modelos teóricos, destacando a importância da compreensão e definição da configuração desses modelos para a implementação do sistema. Na Seção 1.3 procuro definir o problema de ranqueamento e definir o objetivo e escopo do trabalho como sendo avaliar soluções para o problema de ranqueamento de coleção de literatura acadêmica. Na Seção 1.4, exploro as diferenças entre os modelos de correspondência exata e de busca semântica, ressaltando as particularidades de cada um na representação e processamento de informações. Na Seção 1.5, discuto métodos para avaliar a eficácia dos modelos de recuperação de informação, concentrando-me em métricas como Precisão e nDCG para medir o desempenho do ranqueamento.

1.1 Recuperação de Informação

Recuperação de informação ("IR – *Information Retrieval*") pode assumir diferentes definições. Para este trabalho, IR é definida como a disciplina que estuda como encontrar documentos relevantes, de natureza desestruturada, em meio a uma coleção de documentos, a partir de uma consulta (Manning; Raghavan; Schütze, 2008).

Documento, no contexto de IR, geralmente referencia um texto que se pretende recuperar [1]. No caso deste trabalho, documento se refere a um pedaço específico de texto usado para recuperação, ou ranqueamento [2]. Especificamente, documentos são os títulos, resumos e palavras-chave de teses acadêmicas, ou partes destes, como sentenças, já que planeja-se usar esses dados para encontrar os trabalhos relevantes. Presume-se que se um documento é relevante, o trabalho associado a este documento também é relevante.

Documentos relevantes são aqueles que satisfazem a necessidade do usuário (Manning; Raghavan; Schütze, 2008). A definição de relevância é específica de um problema e depende de fatores como características da busca, do usuário a que ela se destina, da consulta, dos documentos, e de tempo e espaço da busca, o que confere à relevância uma

[1] Documentos não são necessariamente texto, podem ser faixas de uma música ou um vídeo, no caso de recuperação desses formatos de mídia.

[2] Em Lin; Nogueira; Yates, uma definição para documento é "unidade 'atômica' de ranqueamento". Essa definição é ainda mais precisa.

natureza subjetiva e dinâmica. Como explicam Ceri et al., "a relevância é multifacetada, sendo determinada não só pelo conteúdo de um resultado recuperado, mas também por aspectos como autoridade, credibilidade, especificidade, exaustividade, atualidade e clareza de sua fonte" (Ceri et al., 2013).

Documentos são de natureza desestruturada quando não possuem um modelo de dados definido, ou que são difíceis para um computador estruturar, interpretar, analisar ou segmentar (Manning; Raghavan; Schütze, 2008). Exemplos de dados desestruturados são texto, imagem, áudio e vídeo, que são encontrados na forma de documentos como páginas da web, artigos científicos, imagens, filmes, dentre outros formatos. Em contraste, documentos estruturados são os que possuem uma estrutura relacional clara ou são fáceis para um computador estruturar. A disciplina de estudo de dados estruturados é a recuperação de dados, que se baseia em uma forma expressa e bem definida de recuperação, articulada de maneira formal, sobre informação que possui um modelo pré-definido de dados (Jurafsky; Martin, 2023).

A IR é tratada como um campo de conhecimento que abrange uma variedade de sistemas, aqui chamados de sistemas de recuperação de informação ("IRS – *Information Retrieval System*"), que se manifestam de diferentes formas. Por exemplo, IRSs são componentes centrais em motores de busca, sistemas de filtragem de informação, sistemas de sumarização de documentos, sistemas de perguntas e respostas, sistemas de recomendação, dentre outros (Ceri et al., 2013).

1.2 Sistema de Recuperação de Informação e Modelo de Recuperação de Informação

Defino o IRS como a implementação de um modelo de recuperação de informação ("IRM – *Information Retrieval Model*"). Um IRM descreve o funcionamento da recuperação de informação em um sistema e é formalmente definido pela quádrupla: (Ceri et al., 2013)

$$\text{IRM} = \{D, Q, F, R(q_k, d_j)\} \tag{1.1}$$

onde

- D é um conjunto de representações d_j de documentos;
- Q é um conjunto de representações q_k de consultas;
- F é a estratégia adotada para modelar a representação de documentos e consultas e suas relações;

- $R(q_k, d_j)$ é a função de ranqueamento que associa d_j a um número real que denota sua relevância para q_k.

Autores costumam categorizar modelos em diferentes abordagens, a depender de como são definidos os elementos D, Q, F e R do modelo. Em essência, modelos são categorizados a partir de seu entendimento sobre relevância e da estratégia adotada para recuperar documentos relevantes.

Modelos que se encaixam na abordagem de correspondência exata compreendem relevância como um valor binário e adotam uma estratégia para recuperar documentos fundamentalmente diferente da estratégia adotada por modelos que se encaixam ou na abordagem de espaço vetorial, ou de modelos que se encaixam na abordagem probabilística (Hiemstra, 2009).

Não é incomum, entretanto, encontrar descrições de modelos que se encaixam em diferentes abordagens, de modo que, tanto não parece existir um consenso de classificação dos diferentes modelos [3], quanto um modelo pode carregar características de diferentes abordagens sem que isso descaracterize sua abordagem principal. Por exemplo, modelos probabilísticos podem usar conceitos de modelos de correspondência exata para encontrar documentos relevantes e usar conceitos de modelo de espaço vetorial para representar seus documentos.

Cumpre observar, apenas, que a definição de um IRM é importante para delinear e conduzir experimentos e constatar sua eficácia para uma aplicação específica. A descrição formal do modelo é necessária para garantir consistência e para "garantir que o modelo pode ser implementado em um sistema real" (Hiemstra, 2009).

Ainda, ao mesmo tempo que um modelo é responsável por ser o esboço ou plano (*blueprint*) de um IRS, um IRS também pode implementar diferentes modelos. Como exemplo, o buscador do Google permite buscas por correspondências exatas de palavras-chave e booleanas [4] e, ao mesmo tempo, implementa outros modelos que permitem encontrar maior relevância em buscas cotidianas, como o PageRank ou modelos baseados em redes neurais [5].

Ao fim, a implementação de um IRS e os modelos escolhidos para serem implementados, dependerão das características dos documentos, das especificidades das consultas e da estratégia empregada na representação, recuperação e ranqueamento de um documento na coleção. Principalmente, a maneira que um IRM compreende relevância influencia diretamente na definição de seus elementos.

[3] Alguns autores descrevem modelos baseados em comparação de palavras-chave como modelos de correspondência exata () e outros os descrevem como modelos de espaço vetorial (Hiemstra, 2009).

[4] <https://support.google.com/websearch/answer/2466433>

[5] <https://searchengineland.com/how-google-uses-artificial-intelligence-in-google-search-379746>

E se, como argumentado, a definição de relevância é específica de um problema, então a escolha do IRM implementado por um IRS será primordialmente pautada pelo resultado que se pretende alcançar, considerando as características reais dos documentos, consultas e usuários. Ou seja, a eficácia do IRM é julgada pelos resultados obtidos após sua tradução em IRS, avaliados em um contexto específico, definido um objetivo específico.

Portanto, antes da implementação de qualquer IRS, defino o problema que pretendo estudar nesse trabalho, o objetivo que pretendo alcançar e os IRMs que pretendo implementar.

1.3 O Problema de Ranqueamento, Objetivo e Escopo do Trabalho

O problema de ranqueamento de documentos pode ser definido como, "dada uma necessidade de informação expressada por meio da consulta q, é a tarefa de retornar uma lista ranqueada de k textos $\{d_1, d_2...d_k\}$ de uma coleção de textos $C = \{d_i\}$ arbitrariamente grande, mas finita, que maximiza uma métrica de interesse, por exemplo, nDCG, AP, etc" (Lin; Nogueira; Yates, 2021).

Um IRM busca solucionar o problema de ranqueamento através de uma função de ranqueamento $R(q_i, d_j)$ que associa d_j a um número real s_j que denota sua relevância para q_i. A lista ranqueada de textos pode ser "mais explicitamente caracterizada como $\{(d_1, s_1), (d_2, s_2)...(d_k, s_k)\}$ com a restrição de que $s_1 > s_2 > ... > s_k$" (Lin; Nogueira; Yates, 2021).

O ranqueamento pode ser entendido como um problema específico dentro da recuperação de informação. Assim, ao dizer que, com este trabalho, proponho desenvolver um sistema de recuperação de literatura acadêmica brasileira, quero dizer que proponho desenvolver um sistema de ranqueamento de literatura acadêmica brasileira. A ideia deste trabalho não é esgotar todas as técnicas atuais para ranqueamento de documentos ou usar todas as melhores práticas para criar um sistema completo de busca. Pretendo estudar e comparar a eficácia da implementação de dois IRMs, usando ferramentas disponíveis ao público.

A motivação deste trabalho é compreender a eficácia e o uso de técnicas modernas de busca, baseadas em modelos de redes neurais, em comparação com técnicas tradicionais baseadas em correspondência de palavras. Ao longo deste texto, eu mostro como criar um simples sistema capaz de fornecer resultados que permitem avaliar a eficácia de diferentes modelos para os dados com que se trabalha. Tudo isso usando um conjunto de dados de tamanho mediano, aproximadamente 106 mil trabalhos acadêmicos, que resultam em aproximadamente 1,3 milhão de documentos.

O primeiro IRM, chamado aqui de **modelo de recuperação esparsa**, representa

documentos e consultas por meio de vetores esparsos e utiliza funções probabilísticas para comparar esses vetores e realizar ranqueamento. O segundo, chamado aqui de **modelo de recuperação densa**, representa documentos e consultas por meio de vetores densos e utiliza operações algébricas para ranqueamento.

Nesses dois IRMs, a função de ranqueamento busca estimar a relevância de um documento d em relação a uma consulta q, por meio de uma função S que calcula a similaridade entre as representações de q e d, φ_q e φ_d, respectivamente:

$$R(q, d) = S(\varphi_q(q), \varphi_d(d)) \tag{1.2}$$

Entendo que ambos os modelos podem ser considerados variações da abordagem de modelos de espaços vetoriais, uma vez que usam representações para documentos e consultas baseados em uma distribuição estatística dos termos na coleção de documentos. Entretanto, ambos são fundamentalmente diferentes na forma que criam essas representações dos documentos e consultas e na forma que calculam o ranqueamento, ou seja, na forma que atribuem relevância. No restante do capítulo justifico a estratégia usada por esses IRMs para representar documentos e consultas e, em seguida, explico suas diferenças na forma de abordar essa estratégia.

1.4 Os Modelos de Recuperação de Informação Estudados

Semântica vetorial é o nome que se dá à forma padrão de representar texto por meio de vetores. A base da atual semântica vetorial nasceu nos anos 50, quando foram propostas ideias de representar o significado de uma palavra por meio de sua distribuição no uso da linguagem e de representar essa distribuição por meio de um vetor (Jurafsky; Martin, 2023).

Essa abordagem é chamada de hipótese distribucional, que estipula que palavras que ocorrem em um mesmo contexto possuem significados similares (Pilehvar; Camacho-Collados, 2022). Aqui, possuir "significado similar" significa que seu uso na linguagem é similar, que ela ocorre nas mesmas posições em uma frase ou que mantém relação parecida com as demais palavras. Palavras que ocorrem em um mesmo contexto, teriam, assim, similaridade semântica. Talvez elas sejam palavras sinônimas, ou talvez elas carreguem algum sentido comum quando usadas em um contexto específico. [6] Sob a mesma premissa, documentos que possuem distribuição parecida de palavras são considerados similares.

A ideia por trás da semântica vetorial é representar palavras como um ponto em um espaço multidimensional derivado das distribuições de vizinhanças, de palavras e

[6] Cumpre notar que algumas palavras que carregam similaridade semântica em um contexto podem simplesmente não ter similaridade em outro contexto.

palavras que estão próximas a elas, possibilitando a medida de um grau de similaridade de termos ou documentos por meio de um cálculo de similaridade entre suas representações vetoriais (Jurafsky; Martin, 2023).

As representacões vetoriais podem ser classificadas entre técnicas baseadas na contagem de frequência de ocorrência ou co-ocorrência das palavras, que resultam em representações esparsas de texto usadas no modelo de correspondência exata, e técnicas baseadas em redes neurais, que resultam em representações densas de texto usadas no modelo de recuperação densa.

Explicada a estratégia da representação vetorial de texto, que baseia os IRMs baseados no modelo de espaço vetorial, esclareço as diferenças entre as estratégias usadas pelo modelo de recuperação esparsa (Subseção 1.4.1) e pelo modelo de recuperação densa (Subseção 1.4.2).

1.4.1 Modelo de Recuperação Esparsa (ou por Correspondência Exata)

Para explicar o modelo de correspondência exata, explico primeiramente a forma usada por esse modelo para representar texto. Em seguida, explico a forma usada por esse modelo para calcular a similaridade entre textos.

Representações esparsas de texto são geralmente baseadas em uma matriz de co-ocorrência, uma forma de representar a ocorrência de um termo em relação a um documento. Essas são chamadas de matrizes de termo-documento. [7] (Jurafsky; Martin, 2023)

Nessas matrizes, um vetor coluna denota um documento, que é representado pelas frequências de seus termos. Cada linha i da matriz representa uma palavra e cada coluna j representa um documento. Ou seja, o valor $M_{i,j}$ representa a frequência da palavra P da linha i em um documento D da coluna j. Como exemplo, veja a coleção de documentos abaixo e parte de sua matriz de co-ocorrência termo-documento:

Documento	Texto
Documento 1	Esperou em um banco para pagar a conta. Depois de horas esperando, saiu do banco e foi ao parque.
Documento 2	Sentou em um banco da praça para aguardar sua chegada.
Documento 3	Eu banco a despesa.

Em uma matriz $m \times n$, cada vetor de palavra tem dimensão n e cada vetor de documento tem dimensão m. Uma consequência desse tipo de representação é que o número de linhas é igual ao tamanho do vocabulário e o número de colunas é igual ao

[7] Existem também as matrizes que relacionam termos a outros termos (matrizes de termo-termo), mas essas não possuem relevância para este trabalho.

Tabela 1 – Matriz Termo-Documento.

Termo	Documento 1	Documento 2	Documento 3
a	1	0	1
aguardar	0	1	0
ao	1	0	0
banco	2	1	1
chegada	0	1	0
⋮			

número de documentos. Essa representação ocupa um espaço proporcional a $m \times n$ e é considerada esparsa para a maior parte dos documentos, já que a distribuição de palavras por documento é pequena quando comparada ao número de documentos (Jurafsky; Martin, 2023).

A ideia é que a consulta também seja representada por um vetor esparso de frequências, da mesma forma que um documento, de modo que permita uma comparação entre as duas. Por esse motivo, a comparação entre uma consulta e um documento é também chamada, indinstintamente, de comparação entre documentos.

IRMs que usam representações esparsas de texto calculam a similaridade entre documentos por meio de uma análise da frequência dos termos presentes na consulta e que também estão presentes nos documentos da coleção. A similaridade entre documentos pode, assim, ser feita pela análise dos valores de frequência dos vetores de documento.

Entretanto, uma análise crua da frequência de termos por documento pode não ser representativa da similaridade entre documentos. Primeiro, a alta frequência de uma palavra em um documento pode afetar desproporcionalmente uma comparação entre documentos simplesmente por um documento ter mais palavras que o outro. Segundo, algumas palavras aparecem com muita frequência em documentos e não carregam grande significado semântico [8]. Necessita-se, portanto, de uma medida, baseada nos valores das frequências, que passe a representar a importância de uma palavra para um documento.

Por esses motivos, a análise de frequência é feita a partir da atribuição de pesos aos termos (*weighting*), seguida de alguma medida de associação entre os termos da consulta e os termos do documento. Assim, uma função de atribuição de valor (função de *scoring*) pode ser descrita como:

$$S(q, d) = \sum_{t \in q \cap d} w(t) \tag{1.3}$$

onde w é uma função que atribui um peso a um termo t, a partir de uma estatística associada ao termo no documento ou coleção.

[8] Como é o caso de algumas classes de palavras no Português, como conjunções, por exemplo.

As estatísticas associadas comumente usadas [9] são: (a) frequência do termo no documento (tf – *term frequency*) , (b) frequência do termo na coleção de documentos ou seu inverso (df – *document frequency* e idf – *inverse document frequency*) , e (c) tamanho do documento (dl – *document length*) (Lin; Nogueira; Yates, 2021).

$$\text{tf}(t, d) = count(t, d) \tag{1.4}$$

$$\text{df}(t) = |\{d \in D : t \in d\}| \tag{1.5}$$

$$\text{idf}(t) = \log \frac{|D|}{\text{df}(t)} \tag{1.6}$$

$$\text{dl}(d) = \sum_{t \in d} \text{tf}(t, d) \tag{1.7}$$

$$\text{avgdl} = \frac{1}{|D|} \sum_{d \in D} \text{dl}(d) \tag{1.8}$$

A função w mais usada em sistemas de busca por correspondência exata é a função BM25 (*Best Match 25*) (Lin; Nogueira; Yates, 2021):

$$\text{BM25}(d, q) = \sum_{t \in q \cap d} \frac{(k+1) \times \text{tf}(t, d)}{k \times ((1-b) + b \times \frac{\text{dl}(d)}{\text{avgdl}(d)}) + \text{tf}(t, d)} \times \log \frac{|D| - \text{df}(t) + 0.5}{\text{df}(t) + 0.5} \tag{1.9}$$

em que o segundo termo da equação é idf adaptado e k e b são hiperparâmetros.

Variações da BM25 são o estado da arte na representação vetorial esparsa de documentos e, por esse motivo, essa função ainda é largamente implementada em sistemas de recuperação de informações (Lin; Nogueira; Yates, 2021).

Uma vantagem desses IRMs é que eles são eficientes, já que o cálculo de relevância de uma consulta tem complexidade proporcional ao número de termos da consulta, e são simples de implementar, já que dependem apenas da criação de um índice invertido, uma estrutura de dados que, em essência, implementa a matriz termo-documento. Um ponto positivo é o fato de ser fácil visualizar o cálculo de similaridade nessa estrutura.

No entanto, o cálculo de similaridade baseado em análise de frequências apresenta desvantagens significativas. A matriz exemplificada na Tabela 1 ilustra as limitações dessa abordagem.

[9] Essas são comumente usadas para computar variações da família de funções chamadas de frequência de termo e inverso da frequência por documento (tf-idf – *Term Frequency Inverse Document Frequency*).

Primeiro, uma representação de um texto aqui descrita não consegue capturar a polissemia de um termo. No caso exemplo, o termo "banco" apresenta 3 significados diferentes e uma consulta "bancos de parque" provavelmente teria como retorno mais relevante o Documento 1 (independentemente da medida de associação a ser considerada), ao invés do Documento 2, que é factualmente mais relevante para a consulta.

Segundo, e mais importante, esse modelo sofre com a incompatibilidade de vocabulário (*vocabulary mismatch*). Esse é um problema conhecido em Processamento de Linguagem Natural (PLN) e se manifesta quando "pessoas diferentes nomeiam a mesma coisa ou conceito de formas diferentes" . No caso exemplo, a consulta "pagar os custos" provavelmente teria como documento mais relevante o Documento 1, o que não é necessariamente verdade, dado que o Documento 3 é "Eu banco a despesa", que contém uma ideia semanticamente mais parecida com a consulta.

Vetores esparsos têm uma boa capacidade de representar documentos para uma comparação por similaridade textual, mas não para similaridade semântica. Como se pode verificar pela explicação acima, para o caso em que o documento relevante define a uma mesma ideia com diferentes palavras, ou usa palavras sinônimas, mas não uma distribuição de palavras similar à da consulta, uma busca que usa BM25 torna-se insatisfatória. (Thakur et al., 2021b)

A necessidade de correspondência exata é amenizada por meio de algumas técnicas simples de PLN, como expansão da consulta, *stemming* e *lemmatization*, mas ainda assim, essas técnicas não são capazes de superar o problema da incompatibilidade de vocabulário.

Historicamente, três abordagens têm sido usadas para superar esse desafio (Lin; Nogueira; Yates, 2021):

(a) melhorar as representações da consulta;

(b) melhorar as representações dos documentos; e

(c) usar representações não dependentes de correspondência exata.

A primeira abordagem é chamada de expansão da consulta. Nessa técnica, a consulta é expandida através da adição de termos similares aos termos buscados. Essa é uma técnica eficiente que melhora consideravelmente o resultado da busca por correspondência exata (Lin; Nogueira; Yates, 2021).

Uma abordagem que deriva, de certa forma, da abordagem (c), é a utilização de representações textuais densas para calcular a similaridade semântica (Lin; Nogueira; Yates, 2021). De fato, para solucionar os problemas com o atual IRM, necessita-se uma nova forma de representar documentos e consultas, uma que considere o significado semântico

dos termos, com menor ênfase na contagem de termos léxicos e maior ênfase em adquirir intuição sobre seu significado no contexto (Mitra; Craswell, 2018).

1.4.2 Modelo de Recuperação Densa (ou por Correspondência Semântica)

Uma solução proposta para o problema observado em abordagens puramente léxicas foi a utilização de redes neurais para criar representações densas de texto (*embeddings*). Nessas representações, o texto é mapeado para um espaço vetorial de baixa dimensão, em que todas as dimensões têm valores diferentes de 0. Essas representações são chamadas de densas em oposição às representações esparsas observadas no modelo de correspondência exata, em que a dimensão era proporcional ao tamanho do vocabulário.

Modelos pré-treinados só passaram a ser usados com maior frequência em problemas de busca e ranqueamento de texto com modelos baseados na arquitetura Transformers [10] (Devlin et al., 2018) Uma grande vantagem dessa nova arquitetura é sua generalização para diferentes tarefas de IR e para diferentes conjuntos de dados. Com isso, é possível usar modelos pré-treinados para alcançar bons resultados em conjuntos de dados diferentes daqueles para os quais os modelos foram treinados (Jurafsky; Martin, 2023).

BERT (Reimers; Gurevych, 2019) é o mais representativo desta leva de modelos de linguagem. Como colocado pelos autores Lin; Nogueira; Yates, BERT se destacou por "reunir muitos ingredientes cruciais para produzir enormes saltos na eficácia de uma ampla gama de tarefas de processamento de linguagem natural". Em um movimento inspirado por BERT, vários outros modelos de linguagem surgiram a partir de 2019, avançando o estado da arte em tarefas de PLN.

Em particular, modelos de linguagem pré-treinados para tarefas gerais podem ser ajustados para outras tarefas, como geração de representações densas de documentos, permitindo novas técnicas em ranqueamento de texto.

A aplicação de redes neurais [11] para resolver o problema do ranqueamento tem se baseado em duas abordagens principais, a primeira usando modelos *cross-encoders* e a segunda usando modelos **bi-encoders**.

Nos modelos *cross-encoders*, são passados dois textos simultaneamente a uma rede com arquitetura Transformers e essa rede produz um valor de similaridade entre 0 e 1, indicando a similaridade entre as sentenças. Um *cross-encoder* não produz *embeddings*. Modelos *bi-encoders*, por sua vez, produzem um *embedding* de sentenças, que são passadas independentemente ao modelo e que podem ser comparadas usando medidas de agregação

[10] Como descrevem os autores Lin; Nogueira; Yates, a partir de Outubro de 2019 alguns artigos de blog do Google e Microsoft reportam a utilização de modelos de linguagem para melhorar a busca em seus serviços.

[11] A explicação da arquitetura ou treinamento desses modelos está além do escopo dessa pesquisa. Aqui, entende-se esses modelos como uma função que tem texto como entrada e vetores que representam um texto como saída.

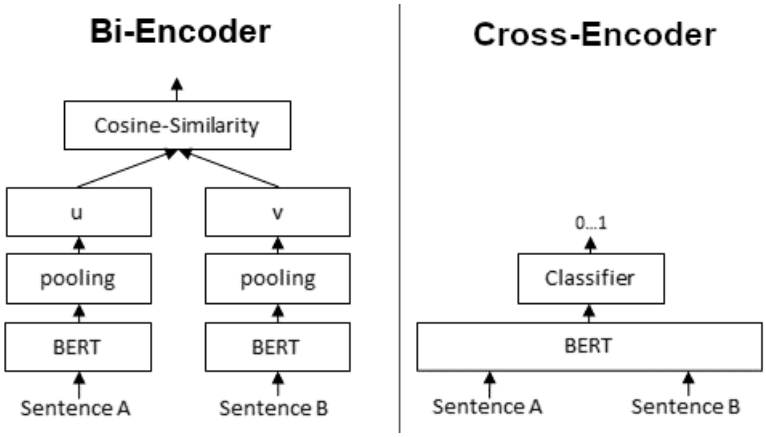

Figura 1 – Arquiteturas de Cross-Encoders vs Bi-Encoders(Sentence-BERT, 2023)

de vetores, como similaridade de cosseno.(Thakur et al., 2021a) Um diagrama dessas arquiteturas pode ser encontrado na Figura 1

Bi-encoders permitem o armazenamento de representações vetoriais para posterior comparação, o que não é feito com *cross-encoders*. Por esse motivo, *bi-encoders* são utilizados sempre que se faz necessária a comparação eficiente entre uma grande quantidade de documentos. Como explicado no website da biblioteca de sentence-transformers, para recuperação de informação de muitos documentos, "[c]ross-Encoders would be the wrong choice [...]: Clustering 10,000 sentence with CrossEncoders would require computing similarity scores for about 50 Million sentence combinations, which takes about 65 hours. With a Bi-Encoder, you compute the embedding for each sentence, which takes only 5 seconds. You can then perform the clustering.". (Sentence-BERT, 2023) *Cross-encoders* alcançam uma qualidade de comparação de texto maior, mas, como explicado, eles não escalam bem para datasets grandes. (Thakur et al., 2021a)

Portanto, a utilização desses modelos em ranqueamento de documentos usa arquiteturas diferentes. Abordagens que usam *cross-encoders*, os aplicam em uma segunda etapa de ranqueamento, depois de os documentos serem retornados por um IRM de correspondência exata, por exemplo, ou modelos neurais *bi-encoders*. Abordagens que usam *bi-encoders* computam o ranqueamento diretamente nas representações vetoriais. Esta última abordagem é chamada de recuperação densa (*dense retrieval*) (Lin; Nogueira; Yates, 2021).

Como notam os autores Lin; Nogueira; Yates, existem motivos para preferir a segunda abordagem. Primeiro, o custo computacional de rerranquear documentos é caro e é feito durante a consulta, ao invés de ser feito na etapa de pre-processamento como no modelo de *dense retrieval*, que calcula apenas a inferência da consulta e usa uma função de similaridade rápida de calcular em uma coleção de vetores densos usando soluções baseadas em *nearest neighbor search*. Segundo, arquiteturas de várias camadas são pouco elegantes e trazem uma série de problemas para o treinamento do modelo de rede neural

que se localiza na segunda camada.

A arquitetura proposta para o problema do ranqueamento neste trabalho pretende utilizar unicamente representações densas dos documentos obtidos por *bi-encoders* unido ao cálculo uma métrica simples de similaridade entre a representação da consulta e do documento. Essa métrica simples é geralmente obtida através de operações de álgebra linear que medem similaridade entre vetores.

Um método algébrico comum de medir a similaridade entre dois vetores é através do produto interno, definido por:

$$\text{dot}(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v} = \sum_i u_i v_i \tag{1.10}$$

Uma interpretação para o resultado do produto interno é a medição de quanto um vetor u está no mesmo sentido e direção de outro vetor v. Essa é uma interpretação conveniente para compreender a medida de similaridade entre duas representações vetoriais de texto.

Um problema de realizar medições com o produto interno é comparar similaridades, pois $dot(u, v)$ pode ser arbitrariamente grande ou pequeno a depender dos valores dos componentes u_i e v_i dos vetores. Nesse caso, pode parecer mais sensato calcular o produto interno com vetores normalizados, o que resulta no valor do cosseno do ângulo entre os vetores e leva todas as comparações a um intervalo entre 0 e 1, uma vez $v_i, u_i \geq 0, \forall i$:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|} \tag{1.11}$$

Assim, é possível comparar similaridades entre diferentes vetores de palavras. Quanto mais próximo de 1 for o resultado, mais similares são os termos comparados. Essa técnica funciona para qualquer modelo vetorial de texto.

No caso da similaridade entre um documento d e um conjunto de documentos V, tal que $V = \{\mathbf{v_1}, \mathbf{v_2}, ...\mathbf{v_n}\}$, pode-se calcular o cosseno entre a representação de d, denotada por $\mathbf{u}$, e o centróide $\mathbf{c}$ de V, dado por:

$$\mathbf{c} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{v_i} \tag{1.12}$$

Dessa forma, a similaridade entre d e V é dada por $\cos(\mathbf{u}, \mathbf{c})$.

A escolha da forma de comparação, entretanto, vai depender da arquitetura do modelo e de seu treinamento. Alguns modelos são treinados para serem usados com produto interno, outros com cálculo de cosseno e, para outros, não vai fazer qualquer diferença.

Desde a publicação do texto de referência usado para a elaboração dessa seção (Lin; Nogueira; Yates, 2021), diversos softwares que cuidam da *pipeline* de *dense retrieval*, popularmente chamadas bases de dados vetoriais, surgiram. Essas *pipelines* se baseiam na geração de representações densas de texto seguida da comparação de similaridade entre a representação densa de uma consulta e dos textos.

1.5 Avaliando a Qualidade de um IRM: Métricas de Ranqueamento

A ideia de implementar dois modelos diferentes para ranquear documentos a fim de encontrar o melhor modelo para uma tarefa exige o desenvolvimento de formas de comparar as implementações. Nesta seção é discutida a forma proposta para avaliação dos sistemas que se vai implementar.

A relevância é o principal critério para avaliar a qualidade do ranqueamento. Sua natureza subjetiva e dinâmica requer avaliação dentro do contexto específico do sistema de recuperação de informação, com base em métricas predefinidas. Portanto, necessita-se saber a priori a relevância de um documento para uma consulta. Isso é feito por meio de julgamentos de relevância.

Julgamentos de relevância (*relevance judgments*) são avaliações humanas da relevância de uma coleção de texto para uma consulta. Julgamentos de relevância servem a dois propósitos, podem ser usados para treinar um modelo de ranqueamento, ou podem ser usados para avaliar funções de ranqueamento (Lin; Nogueira; Yates, 2021).

Julgamentos de relevância são também chamados de qrels e representados por um conjunto de triplas (q, d, r), onde q é uma consulta, d é um documento e $rel(q, d) = r$ é um julgamento de relevância do documento d para uma consulta q. Em geral, $rel(q, d) = r \in \{0, 1, 2, 3, 4\}$, onde $r = 0$ significa que d não possui qualquer relevância para q e $r = 4$ significa que d é muito relevante para q.

Em um problema de ranqueamento, a ordem é relevante porque se presume que o usuário não consumirá muito mais do que 10 ou 20 resultados para sua busca. Esse argumento é apoiado por diversas pesquisas que afirmam que o usuário de um mecanismo de busca de páginas web dificilmente vai além da segunda página (Lin; Nogueira; Yates, 2021).

É comum, portanto, apresentar uma métrica com uma lista R cortada de 10 ou 20 resultados mais relevantes retornados pela consulta. Quando isso acontece, é comum representar com a notação Métrica@k, onde k é o tamanho da lista R truncada (Lin; Nogueira; Yates, 2021).

Quando a ordem dos documentos recuperados é irrerelevante, ou seja, quando a relevância de um documento é tratada como binária, métricas usadas na avaliação da

relevância da informação são precisão e recall, ou variações estatísticas dessas. Precisão avalia a "solidez" do sistema, provendo uma medida para a proporção de documentos relevantes dentre os recuperados. Precisão é definida por:

$$\text{Precisão}(R, q) = \frac{\sum_{d \in R} \text{rel}(q, d)}{|R|} \tag{1.13}$$

Para o problema de ranqueamento, a métrica mais relevante é o Ganho Acumulado com Desconto Normalizado (*nDCG – Normalized Discounted Cumulative Gain)*, métrica "especificamente desenvolvida para julgamentos de relevância graduados".(Lin; Nogueira; Yates, 2021)

O Ganho Acumulado com Desconto (*DCG – Discounted Cumulative Gain*) para uma lista ranqueada $R = (d_i, q)$ é definido como:

$$\text{DCG}(R, q) = \sum_{(i,d) \in R} \frac{2^{rel(q,d)} - 1}{\log_2(i + 1)} \tag{1.14}$$

A DCG é capaz de medir um ganho baseado na posição do documento recuperado por uma consulta, premiando quando um documento relevante aparece no topo da busca e penalizando quando aparece no fim da busca. Isso porque, quanto maior i, maior o desconto sobre o numerador que computa a relevância.

O nDCG é definido por:

$$\text{nDCG}(R, q) = \frac{\text{DCG}(R, q)}{\text{IDCG}(R, q)} \tag{1.15}$$

onde IDCG representa o valor DCG para R ideal para uma consulta q. Assim, nDCG é um valor no intervalo [0, 1], uma vez que DCG $\leq$ nDCG.

Entende-se que essas métricas são as que melhor se enquadram em uma avaliação do ranqueamento e são elas que serão utilizadas para avaliação dos IRSs considerados nesse trabalho.

Uma outra possibilidade seria, também, calcular a Cobertura (*Recall*). A Cobertura avalia a "completude" do sistema, provendo uma medida para a proporção de documentos relevantes que não foram recuperados. Cobertura é definida por:

$$\text{Cobertura}(R, q) = \frac{\sum_{(i,d) \in R} \text{rel}(q, d)}{\sum_{d \in C} \text{rel}(q, d)} \tag{1.16}$$

Entretanto, calcular a Cobertura exigiria saber a relevância de uma consulta para todos os documentos da coleção, o que demanda uma extensa coleta de dados de usuário, o que é virtualmente impossível para a pesquisa aqui proposta.

2 Implementação dos Sistemas e Configuração dos Experimentos

O problema inerente aos algoritmos clássicos de recuperação de informação que se baseiam no tf-idf ou no BM-25 é que a pesquisa deve ser conduzida utilizando as palavras exatas presentes no texto que se busca. Essa abordagem é problemática, pois o usuário do sistema pode compreender o que deseja buscar sem, necessariamente, conhecer as palavras específicas que constam no documento alvo.

Como constatado por Karpukhin et al., uma alternativa promissora a esse problema é a utilização de vetores densos (*embeddings*), em contraste com os vetores esparsos tradicionais. É possível oferecer resultados mais afinados com a intenção do usuário ao codificar a consulta e os documentos empregando modelos de redes neurais pré-treinados e calcular uma pontuação de similaridade entre essas duas codificações.

Esse estudo se propõe a comparar a qualidade e a eficiência de implementações simples do modelo de recuperação densa e do modelo de recuperação esparsa. Minha intenção é saber se consigo reproduzir resultados obtidos em outros estudos semelhantes, usando unicamente modelos pré-treinados e ferramentas acessíveis, para a recuperação de trabalhos acadêmicos nacionais. O sucesso dos sistemas implementados é avaliado por sua comparação com o sistema de busca por palavras-chave da Biblioteca Digital de Teses e Dissertações da Universidade de São Paulo.

De nenhuma forma, esse estudo se propõe a esgotar técnicas utilizadas no desenvolvimento desses sistemas. Aqui se reconhece a existência de técnicas de IR de fácil implementação que poderia melhorar o ranqueamento de ambos os IRSs propostos [1] , mas que não foram implementadas nesse estudo por limitação de tempo. Inclusive, existem técnicas específicas de implementação de IRSs para recuperação de trabalhos acadêmicos, como grafos de citações (Cohan et al., 2020).

Reconhece-se também que existem discussões importantes que não foram abordadas aqui, como o valor de uma alta cobertura para recuperação de trabalhos acadêmicos, ou o fato de que o documento mais relevante não necessariamente é o trabalho acadêmico que mais se adequa ao que foi buscado (seja semanticamente ou pelo número de correspondências de palavras).

Ao fim, desejo apenas saber se é possível, com esforço mínimo e sem necessidade de um conhecimento profundo sobre técnicas de aprendizado de máquina, implementar um sistema que supere sistemas tradicionalmente usados em termos de qualidade de

[1] Algumas dessas técnicas são buscas por proximidade e buscas de expressões exatas.

ranqueamento, especificamente o implementado pelo repositório de teses da Universidade de São Paulo, segundo avaliações dos usuários.

Como muito da estratégia de implementação dos sistemas foi guiada pelo fato de que os sistemas passariam por experimentos, discuto, na Seção 2.1, os experimentos e a forma de avaliação dos sistemas, para em seguida se discutir a arquitetura geral de um IRS na Seção 2.2 e os elementos comuns na Seção 2.3 e particularidades de cada sistema na Seção 2.4.

2.1 Experimentos e Avaliação dos IRSs Considerados

Os experimentos foram projetados para permitir a comparação entre os IRSs denso e esparso e o implementado pelo repositório de teses da Universidade de São Paulo. Para que fosse possível o cálculo de medições nDCG@10 e Precisão@10, o experimento consistiu na coleta de consultas por usuários e na recuperação de um conjunto de documentos para anotação pelos usuários. A quantidade de documentos devolvida para anotação do usuário é mínima para o cálculo de medições @10, o que resulta, após excluídos os documentos duplicados, entre 10 e 30 documentos para os três IRSs que deseja comparar.

Para avaliação dos resultados do experimento, foi implementado um avaliador, cujo fluxo de informações está ilustrado na Figura 2. Esse módulo é responsável por estabelecer métricas e comparar diferentes implementações e configurações dos demais componentes, e ele não tem função direta na recuperação de documentos, apenas função na avaliação das implementações.

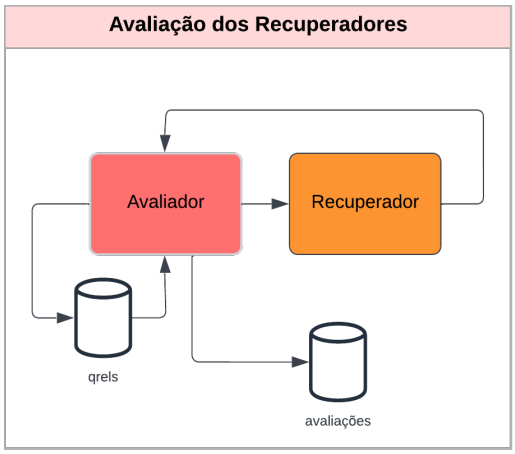

Figura 2 – Avaliador

O módulo avaliador realiza essa avaliação executando consultas armazenadas e comparando os resultados obtidos com um conjunto de anotações de relevância (qrels).

Para cada consulta, o avaliador calcula duas métricas padrão em sistemas de recuperação de informação, Precisão@10 e nDCG@10.

É possível executar o avaliador passando a ele um recuperador específico e rodando a avaliação, método que itera sobre cada consulta da base de dados, invoca o recuperador passado para recuperação dos documentos e calcula os resultados com base nessa recuperação. Os resultados são, ao final, inseridos no banco de dados para futura análise e comparação de desempenho entre diferentes sistemas de recuperação.

2.2 Arquitetura Geral de um IRS

No capítulo anterior, foram detalhadas a forma que cada um dos IRMs estudados compreende a representação e recuperação de documentos. Entretanto, nada foi dito sobre os diferentes componentes de um IRS e como esses componentes conversam entre si para resultar na recuperação da informação desejada. Nesta seção, descreve-se o funcionamento geral de um IRS, ao passo que são descritas as particularidades dos IRSs, cuja implementação será descrita nas seções seguintes.

Como discutido anteriormente, diferentes IRMs atendem a propósitos distintos. Eles variam conforme as características dos documentos, as especificidades das consultas feitas pelos usuários e a estratégia adotada para recuperar e ranquear documentos na coleção. Analogamente, a arquitetura de um IRS é moldada pelos IRMs que servem como fundamentação teórica, bem como pelos aspectos práticos considerados no desenvolvimento de software. Ainda assim, é possível identificar elementos comuns nas arquiteturas desses sistemas.

Os elementos comuns são discutidos a seguir, a nível de componentes do sistema, conforme demonstrado na Figura 3, em que as setas azuis representam o fluxo da coleção de documentos, as setas vermelhas representam o fluxo de uma consulta realizada por um usuário, os módulos são representados por uma caixa sem bordas e as engrenagens representam um passo de codificação do texto em vetores. É comum que outros componentes sejam mencionados em discussões sobre arquitetura de sistemas de recuperação. A escolha de ressaltar os componentes acima, da forma que foi feita, é uma escolha que faz sentido para este trabalho.

A coleta de dados e estruturação mínima desses é o primeiro passo no processo de construção de um IRS, realizada pelo Componente de Ingestão. Essa etapa pode envolver a coleta de dados de diversas fontes, como a *web*, bases de dados, sistemas de arquivos, entre outras. Esse componente é geralmente formado por *crawlers*, módulos utilizados para rastrear e coletar o conteúdo da *web*.

Uma vez coletados, os dados são levados ao Componente de Processamento para

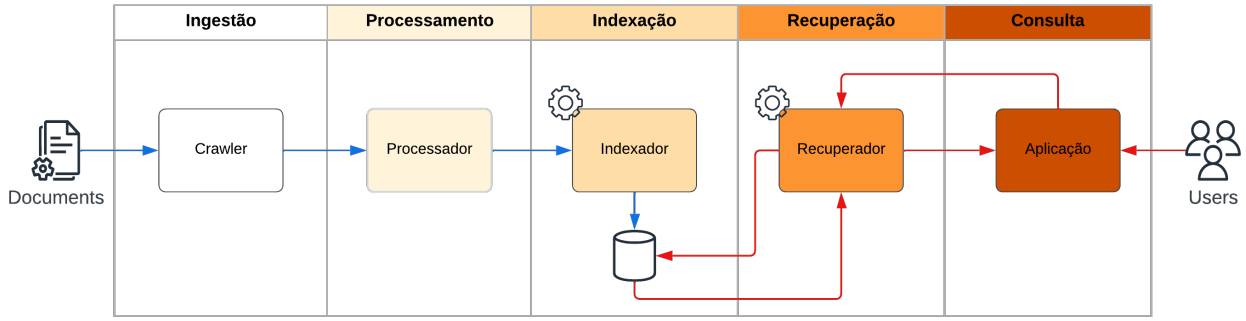

Figura 3 – Arquitetura Geral de um Sistema de Recuperação de Informação

limpeza e estruturação dos documentos. Esse procesamento é realizado para extração de informações relevantes e remoção de informações irrelevantes, como forma de preparar os documentos para indexação. Esse processamento envolve técnicas PLN, como limpeza, filtragem, estematização, lematização, tokenização, entre outras.

O processo de organização dos documentos de forma que possibilite sua recuperação eficiente é chamado de indexação, realizado pelo Componente de Indexação. Nesse componente, os documentos são codificados em vetores e esses são armazenados em uma estrutura de dados chamada índice, que permite ao sistema recuperar rapidamente os documentos relevantes.

Depois de recebida a consulta pelo Componente de Recuperação, este a envia ao índice para recuperação dos documentos mais relevantes. Neste componente podem ser implementados outros algoritmos de ranqueamento e filtragem para determinar os resultados mais relevantes para a consulta. Os resultados são, depois de recuperados, filtrados e ranqueados e então retornados ao usuário.

O Componente de Consulta é o responsável por receber a consulta do usuário por meio de uma interface gráfica, geralmente uma interface *web*, e passar essa consulta ao Componente de Recuperação.

Alguns componentes da arquitetura mostrada acima têm a mesma implementação em ambos os IRSs discutidos, como os componentes de Ingestão e de Consulta. Por outro lado, os componentes de Processamento, Indexação e de Recuperação são diferentes porque a forma que cada um dos modelos propõe resolver o problema do ranqueamento é fundamentalmente diferente. Esse problema não depende especificamente da forma que a coleta dos dados ou que a consulta é realizada, mas depende do tratamento, indexação e atribuição de relevância dos documentos, como será visto a seguir.

2.3 Elementos Comuns das Arquiteturas

O Componente de Ingestão implementado é composto basicamente por um *crawler*, uma ferramenta projetada para navegar e coletar dados de sitemaps do repositório de teses da Universidade de São Paulo. O código utiliza bibliotecas prontas para fazer requisições HTTP e BeautifulSoup para parsear XML, permitindo assim a extração de urls de documentos a serem processados. A classe implementada RawData lida especificamente com a coleta e estruturação dos metadados de teses de doutorado e dissertações de mestrado, usados como documentos para a recuperação das teses. Como documentos para indexação, coletou-se o título, resumo e palavras-chave. Além destes, apenas o nome do autor e a url são usados no sistema, para disponibilizar informações ao usuário. O restante dos metadados foi armazenado mas não foi utilizado.

Durante o processamento dos documentos, foram encontrados desafios significativos relacionados à consistência dos dados. Isso inclui a presença de strings indesejáveis que indicavam ausência de informação, como "Não disponível", a ocorrência de campos de resumos, títulos e palavras-chave vazios ou duplicados, e a falta de padronização na listagem de palavras-chave, com diferentes separadores.

Tais inconsistências podem afetar negativamente a eficiência da indexação e a precisão dos resultados de busca, sendo crucial implementar um processamento rigoroso para assegurar a uniformidade e a qualidade dos documentos. Além disso, foram registradas situações onde obras não foram encontradas, ou *urls* diferentes que continham o mesmo trabalho ou mesmo que não continham dados de resumo e título. Ao todo foram coletados 108.107 documentos, dos quais se aproveitou 106.781.

Sobre o Componente de Consulta, sua implementação serviu unicamente ao propósito de realizar o experimento. Ele é formado por uma aplicação web, cujo *website* contém um campo para consulta pelo usuário e uma explicação do experimento (Figura 5). Após realizar sua consulta, o cliente envia três solicitações que são processadas e respondidas separadamente pelo servidor, cada uma realizando um tipo de recuperação, como mostrado na Figura 4.

O servidor envia ao cliente os documentos recuperados sem ordem particular e o cliente confecciona cartões com os resultados da pesquisa e os consome em ordem aleatória. Cada cartão contém o título, autor, resumo e palavras-chave da tese, além de um pequeno formulário de relevância para a anotação pelo usuário (Figura 6). Ao usuário é solicitado escolher uma nota entre 1, menor relevância possível, e 5, maior relevância possível, para um documento recebido. Cada anotação de relevância é enviada ao servidor, contendo id do documento, id da consulta e relevância anotada.

Cumpre ressaltar que a necessidade de hardware especializado e caro resulta em desafios para a criação de uma aplicação *web* responsiva para um recuperador denso. No

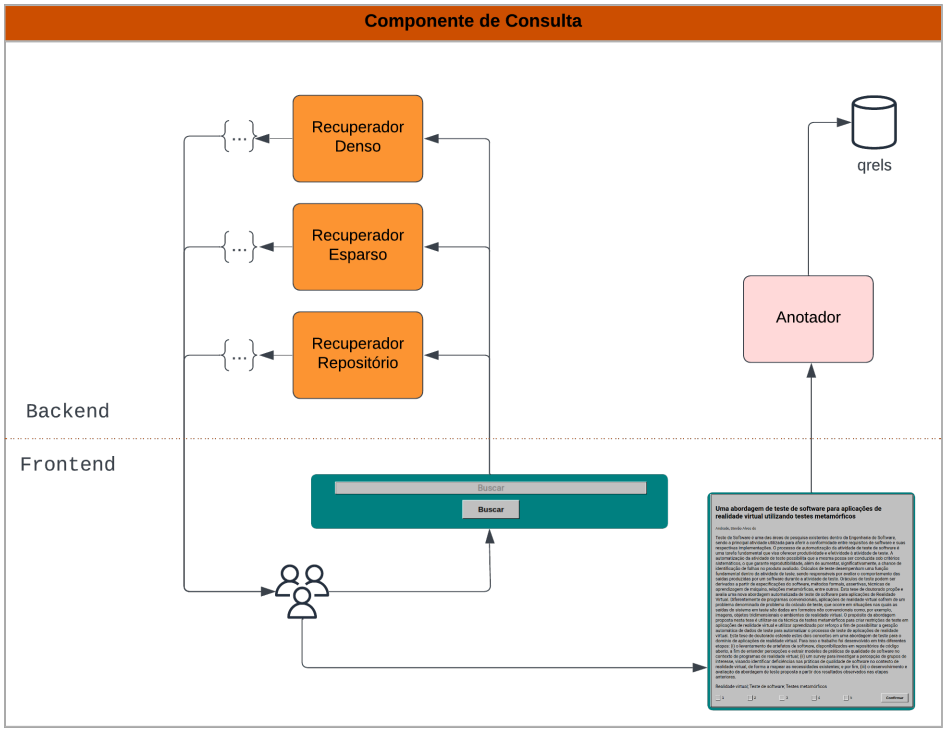

Figura 4 – Componente de Consulta

sistema desenvolvido, o modelo a ser carregado na memória da GPU ocupa um pouco menos que 0,5GB de memória, enquanto a placa usada no servidor tem 1,0GB no total. Isso significa que apenas uma classe de RecuperadorDenso pode ser instanciada por vez, caso contrário a tentativa de codificar um texto pode resultar em falta de memória. Ou seja, apenas um único *worker* pode ser lançado na criação do servidor. Para melhorar o uso de recursos, a arquitetura foi modificada para criar um microserviço separado para o codificador usado pelo RecuperadorDenso. O codificador, por sua vez, responde a pedidos realizados pelos *workers* por meio de HTTP [2], permitindo que vários deles funcionem ao mesmo tempo, solicitando o passo de codificação ao serviço externo.

2.4 Elementos Particulares de Cada Arquitetura

2.4.1 Arquitetura do IRS Esparso

Para implementação de um IRS baseado no IRM de vetores esparsos, primeiramente um documento coletado deve ser tratado de modo a mitigar erros derivados de uma inconsistência do vocabulário entre documento e consulta. Deve haver uma normalização dos termos, possivelmente retirando acentos, letras maiúsculas, realizando estematização, entre outros processamentos.

[2] Hypertext Transfer Protocol

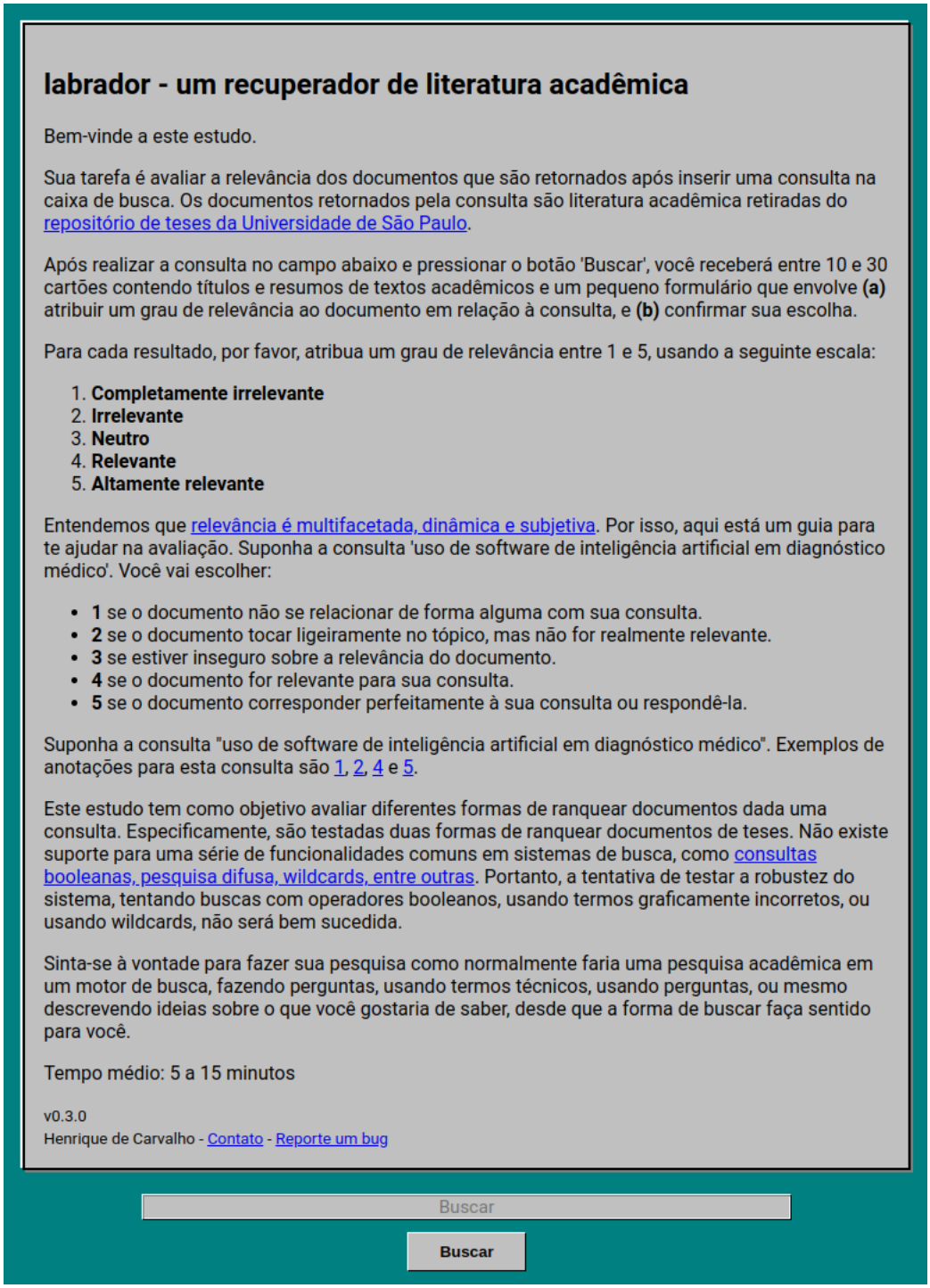

Figura 5 – Página Inicial do Componente de Consulta

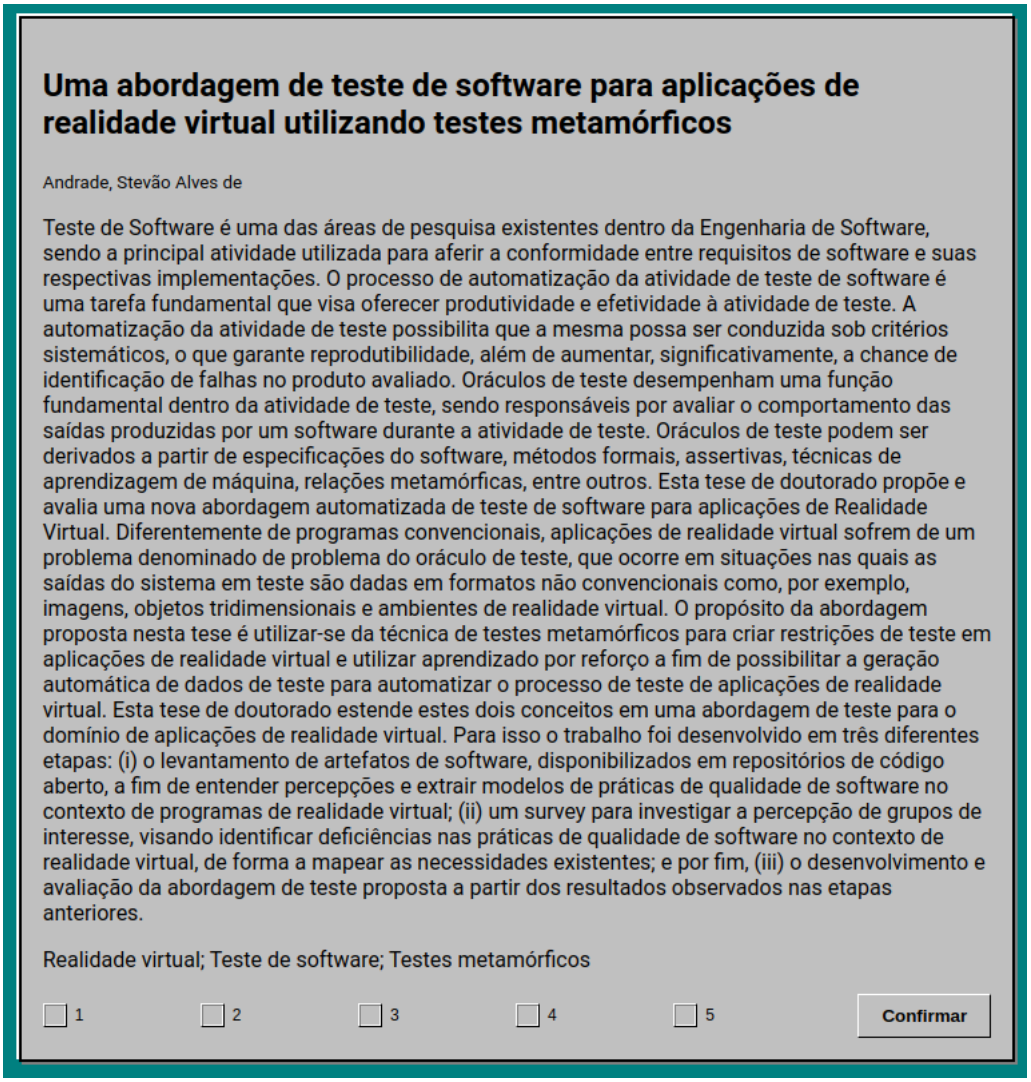

Figura 6 – Cartão do Experimento

A representação do documento é computada com base em uma matriz de incidência e armazenada em um índice invertido. Um índice invertido permite rápida recuperação de todos os documentos em que um termo está presente, através da busca do termo. Nesse sentido, um índice invertido se assemelha a um glossário, em que se buscam posições de uma palavra em um texto. No caso do índice invertido, frequência das palavras nos documentos.

A recuperação nesse tipo de sistema é realizada após o recebimento da consulta de um usuário, processamento dessa consulta, criação de uma representação vetorial esparsa, de forma idêntica à representação de um documento, e uma medida de associação, como o BM25, é computada para todos os documentos em que estão presentes os termos da consulta. Em seguida, é realizado o ranqueamento por meio da comparação dos resultados obtidos pela medida de associação e os resultados ranqueados são retornados ao usuário.

Para criação desses componentes, decidiu-se pela utilização exclusivamente do Solr,

um "plataforma de pesquisa corporativa de código aberto construída com Apache Lucene" [3]. O Solr permite a indexação eficiente de grandes volumes de texto e a realização de pesquisas complexas com alto desempenho, oferecendo uma vasta gama de funcionalidades como faceting, busca por proximidade, e suporte a diversas linguagens.

Uma instância do Solr é executada localmente e se comunica com o cliente por meio de requisições HTTP, oferecendo uma interface flexível para a manipulação e recuperação dos dados indexados. Dessa forma, a recuperação é abstraída em uma simples requisição a um serviço. Esta abordagem facilita a integração com outras aplicações e serviços, permitindo que o sistema de recuperação de informação seja facilmente escalável e adaptável a diferentes contextos de uso.

Com o Solr, o processamento dos documentos é realizado pelo próprio serviço, de acordo com a configuração passada ao serviço usando-se especificações de `Analyzers`, `Tokenizers` e `Filters`. Esses são responsáveis por transformações realizadas no texto durante a indexação e durante a busca, a depender da forma que forem configurados. Enquanto `Tokenizers` e `Filters` executam trabalhos específicos de geração de diferentes tipos de tokens [4] e de filtragem de tokens [5], `Analyzers` são uma forma de agrupar e implementar diversos `Tokenizers` e `Filters` em uma linha de execução que comporta análise e transformação do texto e retorno de uma sequência de tokens. [6]

O `Analyzer` empregado neste trabalho usa:

- `solr.StandardTokenizerFactory`;
- `solr.StandardFilterFactory`;
- `solr.LowerCaseFilterFactory`;
- `solr.StopFilterFactory`; e
- `solr.BrazilianStemFilterFactory`.

Esse é um analisador padrão demonstrado na documentação. A única alteração realizada em comparação com a documentação, para configuração da instância usada no componente de processamento, foi o idioma usado no estematizador. Entende-se que é necessária uma análise mais aprofundada para entender os benefícios de cada módulo do analisador acima no conjunto de documentos testado, o que não foi realizado e consiste em um trabalho futuro.

[3] <https://solr.apache.org/>
[4] Exemplos de tokenizadores: <https://solr.apache.org/guide/6_6/tokenizers.html>
[5] Exemplos de filtros: <https://solr.apache.org/guide/6_6/about-filters.html>
[6] Exemplos de analisadores: <https://solr.apache.org/guide/6_6/analyzers.html>

Por fim, testei também um sistema com filtro de sinônimos para expansão de consulta (`SynonymGraphFilterFactory` [7]). Com esse filtro, é possível expandir um termo para todos os seus sinônimos, antes de executar a consulta. O dicionário de sinônimos foi encontrado em <https://github.com/stavarengo/portuguese-brazilian-synonyms>.

2.4.2 Arquitetura do IRS Denso

Nesta subseção, exploro a arquitetura e os aspectos fundamentais de um sistema de recuperação baseado em recuperação densa, aqui chamado de IRS Denso. O sistema é estruturado em três componentes principais que garantem o processo de consulta.

Decidiu-se por uma arquitetura modular, em que cada componente comunica-se com uma base de dados relacional (no caso, optou-se por Postgresql), e a base de dados é responsável por orquestrar esses componentes, que tentam ou armazenar ou recuperar dados de forma paralelizável.

Cada um dos processos que rodam nos componentes persistem os dados em uma base de dados relacional. Dessa forma, eu garanto que todas as etapas são persistidas em disco e não são perdidas durante a execução. Essa abordagem também é uma forma de orquestrar a execução dos processos paralelamente, sem precisar me preocupar em concorrência, deixando que a base de dados cuide disso.

O **Componente Processador** desta arquitetura tem a principal finalidade de definir passagens para codificação, ou seja, para geração de vetores densos. Dentro do componente Processador de Texto, funciona um módulo de tokenização, que desempenha uma função crítica no pré-processamento de texto, segmentando documentos em unidades lógicas - tokens - que são posteriormente utilizadas para codificação por um modelo de linguagem de rede neural. Enquanto os tokenizadores da biblioteca SentenceTransformers têm a finalidade de transformar texto bruto em estruturas padronizadas que um modelo de linguagem pode interpretar eficientemente, o tokenizador desenvolvido aqui tem como objetivo avaliar qual a melhor forma de criar tokens e o melhor tamanho de cada token, antes do passo de tokenização específico realizado pela biblioteca.

Minha hipótese é que quanto mais passagens de um documento eu consigo gerar vetores densos, maior a quantidade de detalhes de um só documento e, portanto, melhor a qualidade da consulta. Essa hipótese é apoiada em resultados de pesquisa (Lin; Nogueira; Yates, 2021), que levantam evidências de que a qualidade dos *embeddings* gerados por modelos de linguagem tende a diminuir com segmentos de texto maiores.

Portanto, o componente Processador será responsável por eliminar strings irrelevantes, campos vazios, duplicatas e formatação inconsistente de palavras-chave, e consertar

[7] <https://solr.apache.org/guide/6_6/filter-descriptions.html#FilterDescriptions-SynonymGraphFilter>

ao máximo problemas de formatação para viabilizar a criação de diferentes combinações de textos que devem ser passados ao codificador.

Durante experimentos preliminares, avaliei a qualidade dos *embeddings* gerados por três tipos de tokens diferentes. Tokens formados por parágrafos inteiros, ou seja, título, resumo e palavras-chave como um só token (P), sentenças completas separadas de palavras-chave isoladas (S), e sentenças completas apensadas a palavras chave (SK). Minha hipótese aqui é que, no caso de SK, a sentença pode ser enriquecida com o contexto quando apensadas as palavras-chave, mesmo que elas não contenham informações relevantes sobre todo o documento da qual foi retirada. Esse procedimento permite que, mesmo sentenças com carga semântica limitada, quando isoladas, possam ser relacionadas ao conteúdo maior do documento, potencializando a criação de embeddings mais ricos e significativos no espaço semântico do modelo.

Eu pude verificar que as sentenças completas apensadas a palavras-chave estão mais próximas umas das outras que as sentenças sem apensar. Isso significa que elas ganham maior similaridade entre si, mas não quer necessariamente dizer que elas se tornam melhores para busca. Por outro lado, verifiquei que, ao mesmo tempo que estão mais próximos dos demais tokens para um mesmo documento, estão mais distantes de outros documentos.

Supus que isso pode melhorar a busca, então, para os experimentos, decidi usar o recuperador denso com vetores criados com sentenças apensadas a palavras-chave.

O **Componente de Indexação** foi implementado com uma base de dados vetorial [8] chamada Qdrant. Qdrant é um software de código aberto otimizado para armazenamento eficiente e recuperação rápida de vetores densos. É projetada para lidar com grandes conjuntos de dados e oferece suporte a operações de busca por similaridade e ranqueamento. Para utilizar o Qdrant, basta instalar a instância do serviço e comunicar-se com ela via API REST ou gRPC, de forma semelhante ao Solr. Isso proporciona uma interface acessível para a inserção e recuperação de vetores densos. A principal tarefa do Qdrant é armazenar os vetores densos gerados pelo componente processador. Embora o Qdrant ofereça a capacidade de gerar vetores densos por meio de seus próprios codificadores, neste sistema optou-se por enviar vetores pré-codificados. O Qdrant opera tanto na memória quanto com índices em disco, permitindo equilibrar entre desempenho e uso de recursos. Isso o torna adequado para aplicações que exigem respostas rápidas, bem como para sistemas com grandes volumes de dados.

O Qdrant pode ser configurado para utilizar diferentes algoritmos de indexação, como Hierarchical Navigable Small World (HNSW) ou Product Quantization (PQ). Essas

[8] Apesar de serem chamados de bases de dados vetoriais, esses softwares se assemelham muito mais a um motor de busca do que a uma base de dados. Neles são implementadas formas prontas para criar, indexar e recuperar *embeddings* textuais.

configurações influenciam diretamente o desempenho da busca, permitindo adaptar o sistema às necessidades específicas de recuperação de informação. Por fim, o componente de recuperação é abstraído por meio de uma consulta ao índice Qdrant, que retorna uma lista de documentos baseada na similaridade dos vetores densos com a consulta do usuário.

O Componente de Recuperação funciona com o recebimento de uma consulta, consulta ao índice, filtragem dos resultados e ranqueamento da consulta. Decidiu-se por não ranquear ou filtrar os resultados, de modo que os resultados diretos do índice são os únicos passados. Tudo é realizado por meio do Qdrant.

3 Resultados dos Experimentos

Neste capítulo eu comparo o IRS do Repositório de teses da USP usando o campo de consulta por palavras chave (RSK) e por título, resumo e palavras chave (RSTAK), o IRS Denso (DS) e o IRS Esparso (SS).

3.1 Comparação Entre os Recuperadores

Recuperador	**nDCG@10**	**Precisão@10**	**Consultas**
RSK	0.490725	0.266667	45
RSTAK	0.208768	0.150000	45
SS	0.247658	0.101111	45
DS	0.610483	0.344444	45

A maior média de NDCG@10 pertence ao DS[1]. Isso pode indicar que o DS é o mais eficaz em retornar documentos relevantes no topo da lista de resultados. Talvez isso seja devido a um entendimento mais sofisticado do conteúdo dos documentos. A maior média de Precisão@10 também pertence ao DS, o que sugere que ele também é o melhor na recuperação de uma proporção maior de documentos relevantes.

Por outro lado, o RSK é o segundo maior em ambas as métricas, com uma média de NDCG@10 de 0.500813 e uma Média de Precisão de 0.269565. Embora não tão eficaz quanto o DS, esse recuperador apresenta um bom desempenho em relação aos demais, indicando que a busca por palavras-chaves das teses é bastante eficiente quando comparada com a busca por palavras em todo o resumo ou título.

Os demais obtiveram desempenho inferior em ambas as métricas. A média próxima de NDCG@10 e de Precisão@10 entre os dois últimos sugere que esses buscadores são parecidos em seu funcionamento e que eles têm dificuldades em ranquear eficazmente os documentos relevantes e recuperar uma alta proporção de documentos relevantes. A razão disso pode ser porque estão utilizando algoritmos de busca mais simples, que podem não capturar as nuances dos documentos tão eficazmente quanto o DS e também desconsideram a eficácia de buscar só pelas palavras-chave como o RSK.

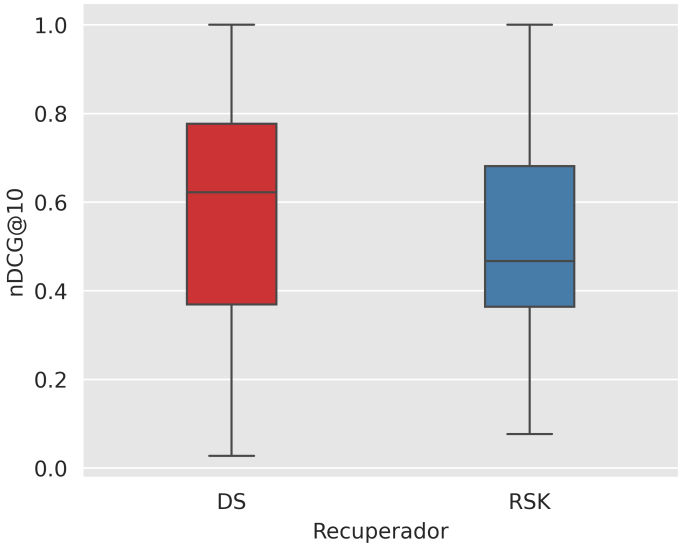

Figura 7 – Distribuição dos Valores de nDCG@10 para cada Consulta

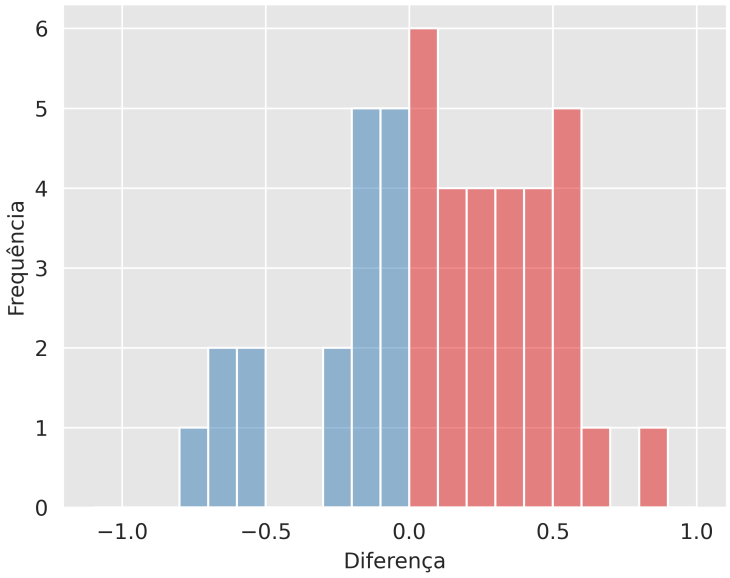

Figura 8 – Distribuição da Diferença dos Valores de nDCG@10 (DS - RSK) para cada Consulta

3.2 Lições Sobre o Recuperador Denso e sobre o Recuperador da USP

Os dados apresentados mostram um desempenho variável entre dois métodos de busca, dependendo da consulta específica. O DS superou o RSK em 63,0% das consultas, o que significa um melhor desempenho geral entre os dois. Na maior parte das vezes, a diferença é marginal, a média das diferenças de nDCG@10 para cada consulta é 0,097; mas em outras vezes, existe uma variação notável de desempenho.

Em algumas consultas, há um ganho notável do DS, já para outras, existe um ganho notável do RSK. Essa variação de desempenho sugere que não existe uma solução única para os algoritmos de busca, sendo o desempenho altamente dependente da consulta específica.

Dada a variabilidade, uma abordagem híbrida que escolha dinamicamente entre os dois métodos com base na natureza da consulta pode ser mais eficaz. Além disso, compreender por que certas consultas têm um desempenho melhor com um método pode fornecer insights para melhorar a estratégia geral de busca, envolvendo a análise das características das consultas e dos documentos correspondentes.

Avaliando os casos em que um supera o outro por um valor maior que o desvio-padrão (0,373), (isso acontece em 35,4% das consultas), observo que o DS supera a qualidade do RSK em 70,588% das consultas.

Consulta	**Diferença**
a língua galega	0.686184
aprendizado de máquina para sistemas de recome...	0.623620
Influência da teledramaturgia mexicana	0.589530
busca por vulnerabilidades em software	0.579821

Tabela 2 – Diferença entre nDCG@10 de RSK e DS

Consulta	**Diferença**
estruturas de dados persistentes	0.886358
Arbitragem icsid brasil	0.615511
Papel da contemplacao na educacao segundo os e...	0.596484
classificadores de análise de sentimento em po...	0.572825
...	

Tabela 3 – Diferença entre nDCG@10 de DS e RSK

Olhando apenas para essas consultas, em que um dos dois superou em muito o desempenho do outro, não consigo observar qualquer relação entre as buscas por palavras-chave e os resultados do RSK.

[1] Embora os dados mostrem diferenças de desempenho, avaliar se essas diferenças são estatisticamente significativas exigiria uma análise estatística mais aprofundada.

Por exemplo, a consulta em que DS performou melhor na comparação da Tabela 3 considerada foi "estruturas de dados persistentes". Considerando o documento mais relevante avaliado, suas palavras-chave são "Árvores rubro-negras; **Estruturas de dados**; Localização de ponto; **Persistência**". Não faz sentido que um recuperador que usa unicamente palavras-chave não consiga capturar esses termos, de modo que o RSK deve usar algo mais complexo em sua lógica e não considerar puramente alguma medida de ranqueamento como palavras-chave.

Caso a consulta fosse "estruturas de dados persistência", ainda assim a busca ranqueia o documento mais relevante em décimo lugar. Por outro lado, uma consulta por "árvores rubro-negras" ranquearia o documento mais relevante em primeiro lugar. Isso faz RSK um sistema imprevisível.

3.3 Melhorando o Recuperador Denso

Analisei os casos em que o DS foi pior que o RSK e concluí que, em algumas consultas, palavras-chave que mereciam ênfase foram ignoradas. Um exemplo é a consulta "aprendizado de máquina para sistemas de recomendação", para a qual o DS retornou apenas documentos relacionados a Aprendizado de Máquina, negligenciando a importância de "sistemas de recomendação".

Portanto, concluo que uma certa medida de processamento da consulta deve existir para que seja considerada a correspondência exata e inexata para um termo consultado. A medida certa do processamento e a forma que deve ser realizada a consulta depende do contexto onde ela se insere.

Proponho, como pesquisa futura, que termos da consulta sejam segmentados e sejam buscadas de forma conjunta e também separados, permitindo uma filtragem posterior dos resultados mais inteligente que meramente usar o ranqueamento do recuperador denso.

4 Conclusão

Este trabalho apresentou uma compreensiva análise comparativa entre diferentes sistemas de recuperação de informação (IRS) aplicados ao repositório de teses da Universidade de São Paulo. O foco principal foi avaliar o desempenho dos recuperadores por meio de métricas como nDCG@10 e Precisão@10, comparando sistemas que utilizam diferentes abordagens de busca: palavras-chave (RSK), títulos, resumos e palavras-chave (RSTAK), um sistema denso (DS) e um sistema esparso (SS).

Através dos experimentos, observou-se que o DS apresentou um desempenho superior, tanto em termos de relevância quanto precisão. Isso sugere uma maior eficácia deste sistema na compreensão e no tratamento do conteúdo dos documentos, provavelmente devido a uma abordagem mais sofisticada na interpretação do texto. Já o RSK, embora não tão eficiente quanto o DS, mostrou-se relativamente eficaz, indicando que a busca por palavras-chave específicas é uma estratégia razoavelmente boa.

A análise detalhada revelou que o desempenho dos sistemas varia significativamente dependendo da consulta específica. Isso sugere que a eficácia de um IRS pode ser altamente dependente do contexto e da natureza das consultas. Uma descoberta relevante foi que a performance dos sistemas não é uniforme, com alguns casos mostrando diferenças marcantes na eficácia entre o DS e o RSK. Essas variações indicam que não existe uma solução única ideal para todas as consultas e que um sistema híbrido, capaz de adaptar-se dinamicamente à natureza da consulta, pode ser mais eficiente.

Outro aspecto observado foi a imprevisibilidade do RSK em relação à relevância dos termos na consulta. Por exemplo, o RSK mostrou resultados inconsistentes ao lidar com consultas que incluíam termos presentes nas palavras-chave dos documentos, sugerindo que sua lógica de busca vai além de uma simples correspondência de palavras-chave.

Finalmente, a pesquisa apontou que o DS, embora geralmente superior, falha em algumas consultas específicas, especialmente quando termos importantes são ignorados. Isso indica a necessidade de um tratamento mais refinado das consultas, possivelmente incorporando análises de correspondência exata e inexata dos termos. Como trabalho futuro, sugere-se a investigação de métodos que combinem diferentes estratégias de busca, tanto conjunta quanto separadamente, para uma filtragem mais eficiente dos resultados.

Referências

Ceri, S. et al. *Web Information Retrieval*. 1. ed. : Springer Berlin, Heidelberg, 2013. (Data-Centric Systems and Applications). Number of Pages: XIV, 284. ISBN 978-3-642-39314-3.

Cohan, A.; Feldman, S.; Beltagy, I.; Downey, D.; Weld, D. S. Specter: Document-level representation learning using citation-informed transformers. *ArXiv*, abs/2004.07180, 2020. Available on: <https://api.semanticscholar.org/CorpusID:215768677>.

Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. Available on: <http://arxiv.org/abs/1810.04805>.

Hiemstra, D. Information retrieval models. *Information Retrieval: Searching in the 21st Century*, John Wiley & Sons, Ltd, p. 1–19, 2009. [PDF] from psu.edu.

Jurafsky, D.; Martin, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2023. Available on: <https://web.stanford.edu/~jurafsky/slp3/ed3book_jan72023.pdf>.

Karpukhin, V. et al. *Dense Passage Retrieval for Open-Domain Question Answering*. 2020.

Lin, J.; Nogueira, R.; Yates, A. *Pretrained Transformers for Text Ranking: BERT and Beyond*. 2021.

Manning, C. D.; Raghavan, P.; Schütze, H. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. ISBN 978-0-521-86571-5. Available on: <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>.

Mitra, B.; Craswell, N. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, v. 13, n. 1, p. 1–126, 2018. ISSN 1554-0669. Available on: <http://dx.doi.org/10.1561/1500000061>.

Pilehvar, M. T.; Camacho-Collados, J. *Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning*. 1. ed. : Springer Cham, 2022. (Synthesis Lectures on Human Language Technologies). Softcover ISBN: 978-1-636-39021-5, Published: 13 November 2020; eBook Published: 31 May 2022. ISSN 1947-4040. ISBN 978-1-636-39022-2.

Reimers, N.; Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019. Available on: <http://arxiv.org/abs/1908.10084>.

Sentence-BERT. *Cross-Encoder Applications - Sentence-BERT Documentation*. 2023. Accessed: 2023-12-01. Available on: <https://www.sbert.net/examples/applications/cross-encoder/README.html>.

Thakur, N.; Reimers, N.; Daxenberger, J.; Gurevych, I. Augmented SBERT: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, 2021. p. 296–310. Available on: <https://www.aclweb.org/anthology/2021.naacl-main.28>.

Thakur, N.; Reimers, N.; Rücklé, A.; Srivastava, A.; Gurevych, I. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. In: *Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS 2021) - Datasets and Benchmarks Track (Round 2)*. 2021. Available on: <https://arxiv.org/abs/2104.08663>.