

# Implementação de motores de busca – Busca e Recuperação de Informação

Davi Cardoso de Oliveira<sup>1</sup>, Rafael Souza de Almeida<sup>2</sup>, Maxwell William Severiano Chagas<sup>3</sup>

<sup>1</sup>Instituto de Informática – Universidade Federal Rural do Rio de Janeiro (UFRRJ) Nova Iguaçu – RJ – Brazil

<sup>2</sup>Departamento de Ciência da Computação – UFRRJ-IM

***Abstract.** This article aims to show the process of building the project of implementing search engines in thousands of documents, searching for specific words, and among other objectives.*

***Resumo.** Este artigo visa mostrar o processo de construção do projeto de implementação de motores de busca em milhares de documentos, procurando por palavras específicas, e entre outros objetivos.*

## 1. Introdução

A capacidade de recuperar informações relevantes de grandes conjuntos de dados é fundamental, especialmente no âmbito acadêmico e científico, onde a rápida localização de artigos é crucial para impulsionar a pesquisa. A criação de motores de busca eficazes desempenha um papel significativo nesse processo, sendo a escolha da ferramenta uma decisão crucial que afeta diretamente a precisão e eficiência dessa recuperação.

A criação dos Motores de Busca foi um desafio interessante para entender o funcionamento das buscas, os processos de como são feitas e como avaliar eles. Sendo assim foi preciso conhecer como funcionam as tecnologias que utilizamos para auxiliar no desenvolvimento. Sendo elas:

### 1.1. Whoosh

A Whoosh é uma ferramenta para buscar palavras em montes de textos. Imagina que se tem uma tonelada de documentos, como artigos ou páginas da web, e é necessário encontrar rapidamente onde determinadas palavras aparecem. A Whoosh facilitaria esse caso.

Ela faz duas coisas principais: cria um índice especial que acelera a busca, e depois realiza buscas nesse índice. A biblioteca é interessante porque é simples de usar e permite que seja ajustado às configurações de busca conforme necessário.

## 1.2. Elasticsearch

O Elasticsearch é uma ferramenta de busca e análise distribuída. Sua arquitetura distribuída possibilita escalabilidade, permitindo a adição de nós ao cluster para garantir alta disponibilidade e tolerância a falhas.

Em relação a indexação e pesquisa, o Elasticsearch se destaca pela eficiência. Ele suporta diversos tipos de dados e oferece uma indexação rápida e flexível. Além disso, sua capacidade de realizar pesquisas em tempo real torna-o ideal para casos de uso que necessitam de análise instantânea de grandes conjuntos de dados.

A API RESTful exposta pelo Elasticsearch simplifica a integração com diversas linguagens de programação, facilitando o desenvolvimento de aplicativos que interagem com o sistema. Isso, aliado à capacidade de análise e agregação avançada, proporciona uma experiência completa no gerenciamento e interpretação de dados.

## 1.3. Gráfico Inicial

Possuindo os “datasets” que serão utilizados como “matéria-prima” para as buscas de tamanho 1,5GB, começamos a primeira parte do nosso projeto implementando um código para gerar um gráfico de palavras com as métricas “Distribuição das palavras no campus x Frequência de palavras”, conformes vemos (Figura 1) abaixo:

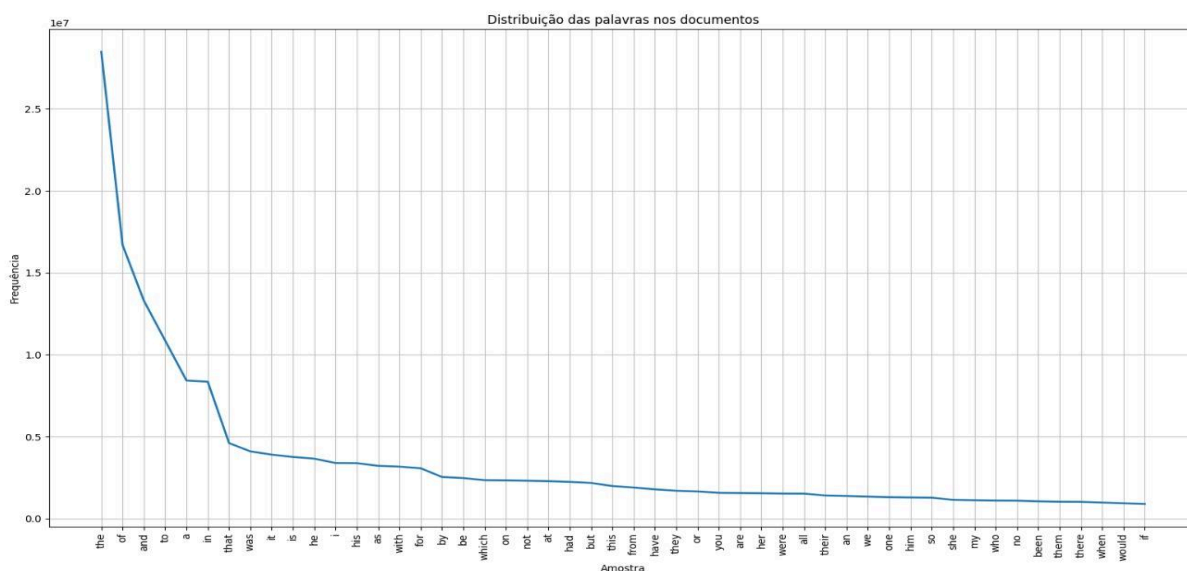


Figura 1: Gráfico Inicial

## 2. Primeira Parte

Utilizando esse gráfico (Figura 1), foram respondidas as seguintes perguntas:

1. Qual o tamanho do vocabulário?

R: 3278211

2. Quantas palavras a coleção tem no total?

R: 426063069

3. Quais as 10 palavras mais frequentes na coleção? Quais suas frequências?

R: (Figura 2)

```
> part9      Tamanho do vocabulário: 3278211
> part10     Total de palavras no arquivo: 426063069
> part11
> part12     10 palavras mais frequentes e suas frequências:
> part13     'the': 28480701
> part14     'of': 16690508
> part15     'and': 13296842
> part16     'to': 10864011
> part17     'a': 8428872
> part18     'in': 8355536
> part19     'that': 4612476
> part20     'was': 4106329
> part21     'it': 3907604
> part21     'is': 3765628
```

Figura 2: Resposta da 3

4. Quais as 10 palavras menos frequentes na coleção? Quais são as suas frequências?

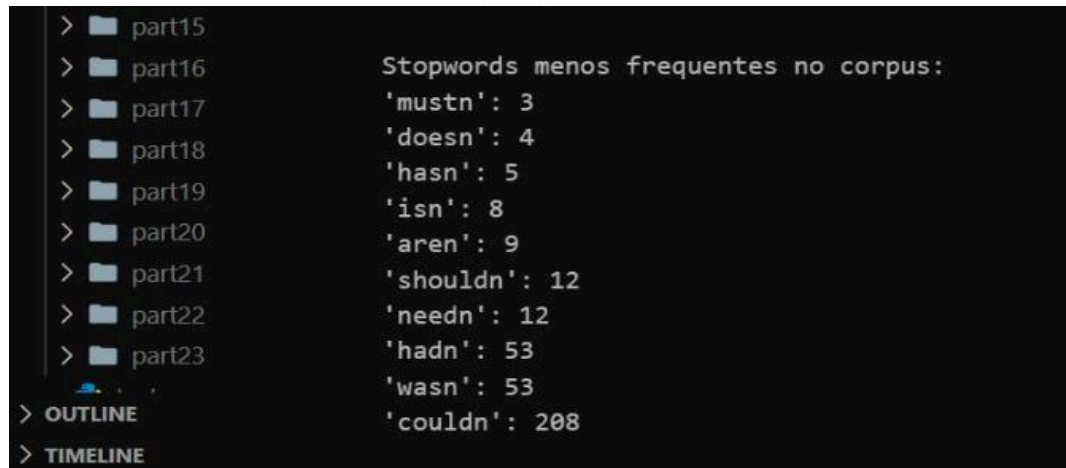
R: (Figura 3)

```
> part4
> part5     10 palavras menos frequentes e suas frequências:
> part6     'panfield': 1
> part7     'commandinginchiefs': 1
> part8     '8company': 1
> part8     'hallingbury': 1
> part9     'officerlieutcolonel': 1
> part10    'secondincommandmajor': 1
> part11    'officersurgcaptain': 1
> part12    'quartermastercapt': 1
> part13    'companyformerly': 1
> part14    'sergtmajore': 1
> part15
```

Figura 3: Resposta da 4

5. Avaliando as stopwords em inglês (usem a lista da NLTK), quais as mais frequentes? e as menos frequentes? Como elas podem influenciar o motor de busca?

R: (Figura 4)



```
> part15
> part16
> part17
> part18
> part19
> part20
> part21
> part22
> part23
> OUTLINE
> TIMELINE

Stopwords menos frequentes no corpus:
'mustn': 3
'doesn': 4
'hasn': 5
'isn': 8
'aren': 9
'shouldn': 12
'needn': 12
'hadn': 53
'wasn': 53
'couldn': 208
```

Figura 4: Resposta da 5)

### 3. Segunda Parte

A qualidade dos resultados de busca depende diretamente das técnicas aplicadas em diferentes etapas do processamento de consultas. Tendo isso bem definido, vamos utilizar abordagens diferentes em motores de busca. Isto é, para cada uma das etapas, pensamos em duas abordagens para utilizar.

As técnicas propostas são organizadas em três etapas:

1. **Pré-processamento**: preparação inicial dos dados para reduzir ruído e aumentar a eficiência.
2. **Extração de subconsultas**: identificação dos termos mais significativos em uma consulta.
3. **Expansão de subconsultas**: enriquecimento semântico das consultas para abranger mais resultados relevantes.

Para cada etapa, duas abordagens são detalhadas.

#### 3.1 Pré-processamento

O pré-processamento é a etapa inicial no pipeline de um motor de busca, responsável por normalizar e reduzir a complexidade dos dados.

### **Abordagem 1: Remoção de Stop Words**

Stop words são palavras de alta frequência e baixo valor semântico (e.g., "e", "de", "o") que podem ser eliminadas para reduzir o ruído. Essa técnica visa simplificar a consulta e economizar recursos computacionais. No entanto, há risco de excluir palavras relevantes dependendo do contexto.

### **Abordagem 2: Radicalização com Stemming**

O stemming reduz palavras às suas raízes, agrupando variações morfológicas (e.g., "correr", "correndo", "corri" tornam-se "corr"). Essa técnica aumenta a precisão ao considerar todas as variações de uma palavra. Contudo, pode gerar ambiguidade quando diferentes termos compartilham o mesmo radical.

## **3.2 Extração de Subconsultas**

Após o pré-processamento, a consulta é analisada para determinar os elementos mais relevantes.

### **Abordagem 1: Aplicação de Ranking às Palavras ou Frases**

Nesta abordagem, termos ou frases recebem uma pontuação baseada em sua relevância para a consulta, utilizando critérios como frequência nos documentos ou posição na consulta. Essa técnica permite priorizar palavras significativas e ignorar termos genéricos. A principal dificuldade está na definição de critérios de ranking.

### **Abordagem 2: Atribuição de Pesos às Palavras Significativas**

As palavras identificadas como importantes recebem maior peso na formulação da consulta. Isso garante que esses termos influenciam mais os resultados, aumentando a precisão sem alterar substancialmente a consulta original.

## **3.3 Expansão de Subconsultas**

A expansão visa enriquecer a consulta original, incorporando termos adicionais para melhorar a abrangência sem perder relevância.

### **Abordagem 1: Pesquisa por Sinônimos**

A substituição ou inclusão de sinônimos amplia a consulta, cobrindo variações linguísticas nos documentos. Por exemplo, "feliz" e "contente" seriam tratados como equivalentes. Apesar dos benefícios, pode incluir termos irrelevantes ou redundantes.

## Abordagem 2: Expansão por Contexto

Essa abordagem utiliza bases de conhecimento ou modelos de aprendizado para incluir termos relacionados ao contexto semântico da consulta. Por exemplo, uma busca por "emergências médicas" poderia incluir termos como "hospital" ou "urgência". Essa técnica oferece maior alinhamento aos objetivos do usuário, mas exige processamento adicional.

### 3.4. Combinações diferentes das abordagens

Com essas abordagens pensamos em 8 configurações diferentes para serem implementadas no projeto visando a melhor performance da busca conforme está sendo mostrado a seguir (Tabela 1):

Configuração	Pré-processamento	Extração de Subconsultas	Expansão de Subconsultas	Descrição
1	Remoção de Stop words	Aplicar um ranking às palavras ou frases da consulta para identificar as mais significativas	Pesquisa por Sinônimos	Otimizado para consultas onde o foco é maximizar a precisão, eliminando palavras irrelevantes e expandindo com sinônimos das palavras mais significativas
2	Remoção de Stop words	Dar peso às palavras mais significativas	Expansão por contexto	Ideal para consultas onde o contexto do termo é importante. O motor foca em palavras-chave e as relaciona com termos encontrados frequentemente no mesmo contexto
3	Radicalização por stemming	Aplicar um ranking às palavras ou frases da consulta para identificar as mais significativas	Pesquisa por sinônimos	Reduz as palavras à sua raiz (radicalização) para aumentar a abrangência e combina isso com um ranking e expansão semântica para cobrir sinônimos

4	Radicalização por stemming	Dar peso às palavras mais significativas	Expansão por contexto	Prioriza palavras-chave importantes, expande o significado usando contexto e simplifica a consulta através de stemming.
5	Remoção de stop words	Aplicar um ranking às palavras ou frases da consulta para identificar as mais significativas	Expansão por contexto	Foco é em consultas otimizadas pela remoção de palavras irrelevantes e expansão que captura a relação contextual entre os termos.
6	Remoção de stop words	Dar peso às palavras mais significativas	Pesquisa por sinônimos	Essa configuração equilibra simplicidade e relevância. Remove stop words, prioriza palavras-chave por peso e adiciona sinônimos para expandir o escopo
7	Radicalização por stemming	Aplicar um ranking às palavras ou frases da consulta para identificar as mais significativas	Expansão por contexto	Esta configuração é ideal para consultas onde a generalização dos termos (via stemming) e o contexto dos termos-chave são cruciais para encontrar resultados relevantes
8	Radicalização por stemming	Dar peso às palavras mais significativas	Pesquisa por sinônimos	Focada em consultas mais específicas, essa configuração combina stemming, priorização por peso e sinônimos para atingir alta precisão

				em um escopo mais amplo
--	--	--	--	-------------------------

Tabela 1: 8 configurações diferentes para abordagem no motor de busca

## 4. Terceira Parte

Dessas 8 configurações escolhemos a 1ª e a 6ª, pois no processo da indexação dos arquivos (preparo), tivemos o objetivo de criar um sistema de indexação de documentos que utiliza remoção de stopwords como único passo de pré-processamento, armazenando o índice em disco para realizar buscas eficientes. Stop words são palavras muito frequentes, como "e", "de", "para", que não agregam significado relevante às consultas. O índice é armazenado em disco, garantindo eficiência nas buscas subsequentes. Ao eliminarmos essas palavras sem significância, conseguimos ter no nosso arquivo após o pré-processamento, somente as palavras importantes, e com isso, fica mais fácil assim fazer um ranking das mais importantes que aparecem e também do peso de cada uma na camada de dados, podendo variar esse peso e ranking entre frequência e importância da palavra para o contexto do documento

### 4.1. Etapas realizadas

#### 4.1.1. Etapa de Definição do Processo

As etapas do processo incluem pré-processamento, criação do índice, carregamento dos documentos e indexação. O uso das bibliotecas Whoosh e NLTK, combinadas com a linguagem Python, permitiu implementar uma solução eficiente para gestão de informações textuais. Sua remoção visa reduzir o tamanho do índice e melhorar a relevância dos resultados de busca.

#### 4.1.2. Etapa de Estrutura do Código

Pré-processamento:

1. **Tokenização:** O texto foi segmentado em palavras individuais.
2. **Remoção de StopWords:** As palavras correspondentes à lista do NLTK foram eliminadas.
3. **Reconstrução do Texto:** O texto foi formatado sem as palavras removidas.



Criação do Índice:

1. **Esquema de Índice:** Um esquema Whoosh foi definido com campos para título, conteúdo e caminho do arquivo.
2. **Diretório de Armazenamento:** O índice foi criado e armazenado no diretório especificado (“indexdir”).

Carregamento de Documentos:

1. **Leitura Recursiva:** Os documentos .txt foram lidos a partir de uma estrutura de diretórios fornecida (/part1, /part2, etc.).
2. **Aplicação do Pré-processamento:** O conteúdo de cada documento foi processado para remoção de stop words.

Indexação:

1. **Adição ao Índice:** Os documentos processados foram inseridos no índice.
2. **Armazenamento em Disco:** O índice final foi salvo no diretório “indexdir”.

#### 4.1.3. Etapa de Execução

1. **Caminho dos Documentos:** Substituído por path\_to\_parts, representando o local onde os arquivos .txt estão armazenados.
2. **Caminho do Índice:** Definido como “indexdir”.

Ferramentas Utilizadas:

1. **Whoosh:** Para criação e gerenciamento do índice.
2. **NLTK:** Para tokenização e remoção de stop words.
3. **Python:** Linguagem principal utilizada para implementação.

#### 4.2. Resultados

A remoção de stopwords realizada com sucesso nos documentos. Stop words eliminadas com base na lista do NLTK para português.

Os documentos foram processados e adicionados ao índice. O índice foi armazenado em “indexdir” e pode ser reutilizado para buscas sem necessidade de nova indexação.

Houve a redução do tamanho do índice ao eliminar palavras irrelevantes e um aumento da precisão das buscas, já que palavras sem relevância foram removidas.

### 4.3. Precision at K

Mostra a precisão em diferentes valores de k. Outra avaliação do desempenho do modelo foi realizada utilizando a métrica  $\text{precision@k}$ , que mede a proporção de itens relevantes entre os top k itens classificados. A fórmula para  $\text{precision@k}$  é dada por:

$$\text{precision@k} = (\text{número de itens relevantes nos top k}) / k$$

Onde:

Número de itens relevantes nos top k: É o número de exemplos relevantes que foram classificados corretamente dentro dos k primeiros itens.

Essa métrica é crucial em cenários onde a minimização de falsos positivos é importante. Como são diversos documentos, fez-se a média para cada k, então foi feito um gráfico a fim de ver a média de k em todos os documentos.

### 4.4. Recall at K

Mede a proporção de documentos relevantes recuperados nos top-K resultados. Gráficos mostram a eficácia do sistema em encontrar documentos relevantes

Uma das avaliações do desempenho do nosso modelo foi realizada utilizando a métrica  $\text{recall@k}$ , que mede a capacidade do modelo em identificar exemplos relevantes nos top k itens classificados. A fórmula para o  $\text{recall@k}$  é dada por:

$$\text{recall@k} = (\text{número de itens relevantes nos top k}) / (\text{número total de itens relevantes no conjunto de dados})$$

Onde:

Número de itens relevantes nos top k: É o número de exemplos relevantes que foram classificados corretamente dentro dos k primeiros itens.

Número total de itens relevantes no conjunto de dados: Representa a quantidade total de exemplos relevantes disponíveis no conjunto de dados.

## 5. Quarta Parte

### 5.1. Detectando Plágio com Whoosh

Agora vamos descrever o processo de detecção de plágio utilizando a ferramenta Whoosh, focando na comparação de documentos suspeitos com um índice previamente criado. O

método inclui etapas de pré-processamento, extração e expansão de subconsultas, busca por similaridade e análise de métricas de precisão e revocação para avaliação de desempenho.

### **5.1.1. Metodologia**

#### **5.1.1.1. Pré-processamento dos Documentos**

O processo iniciou-se pela remoção de stopwords por meio da função `preprocessamento(contéudo)`, utilizando a biblioteca NLTK. As palavras comuns e pouco relevantes foram excluídas após a tokenização do texto. Este pré-processamento visou simplificar e melhorar a relevância das consultas.

#### **5.1.1.2. Extração de Subconsultas**

A função `extrair_subconsultas(tokens)` calcula a frequência das palavras nos documentos e as divide em categorias baseadas em faixas de frequência (como 100, 50, 20 e 5 ocorrências). Isso foi feito para identificar palavras mais relevantes, com maior peso dado àquelas que ocorrem com mais frequência no documento.

#### **5.1.1.3. Expansão das Subconsultas**

Para melhorar as buscas, uma etapa de expansão de subconsultas foi implementada. A função `expandir_subconsulta(subconsulta)` utiliza o WordNet (um banco de dados lexical de palavras) para procurar sinônimos das palavras mais significativas nas subconsultas extraídas. Isso amplia as possibilidades de correspondência ao procurar por sinônimos e termos relacionados.

#### **5.1.1.4. Busca no Índice**

O índice foi construído previamente a partir dos documentos relevantes utilizando Whoosh. Os documentos foram indexados e armazenados em um diretório chamado “indexdir”.

A função `buscar_no_indice` realiza a busca no índice, utilizando a consulta que foi criada a partir das subconsultas extraídas e expandidas. Os resultados de busca são ordenados de acordo com o score de similaridade (pontuação de relevância) dos documentos encontrados. A consulta foi estruturada para levar em consideração diferentes pesos para as subconsultas, de acordo com a sua importância, o que é refletido no uso do operador  $\wedge$  para atribuição de pesos nas consultas do Whoosh.

### **5.1.2. Detecção de Plágio**

A função `detectar_plagio` analisou documentos suspeitos, ela percorre os documentos suspeitos no diretório indicado (`diretorio_suspeito`), e para cada documento, o conteúdo é pré-processado, subconsultas são extraídas e expandidas, e então a busca é realizada no índice.

Os documentos encontrados que possuam um score de similaridade maior que um limite (750 neste caso, embora tenha sido modificado para sempre retornar True) são considerados como fontes potenciais de plágio.

Para cada documento suspeito, são exibidos os 10 documentos mais relevantes com suas respectivas pontuações e caminhos.

Os 10 documentos mais relevantes foram exibidos para cada documento suspeito, com detalhes sobre pontuação e localização.

### 5.1.3. Métricas de desempenho

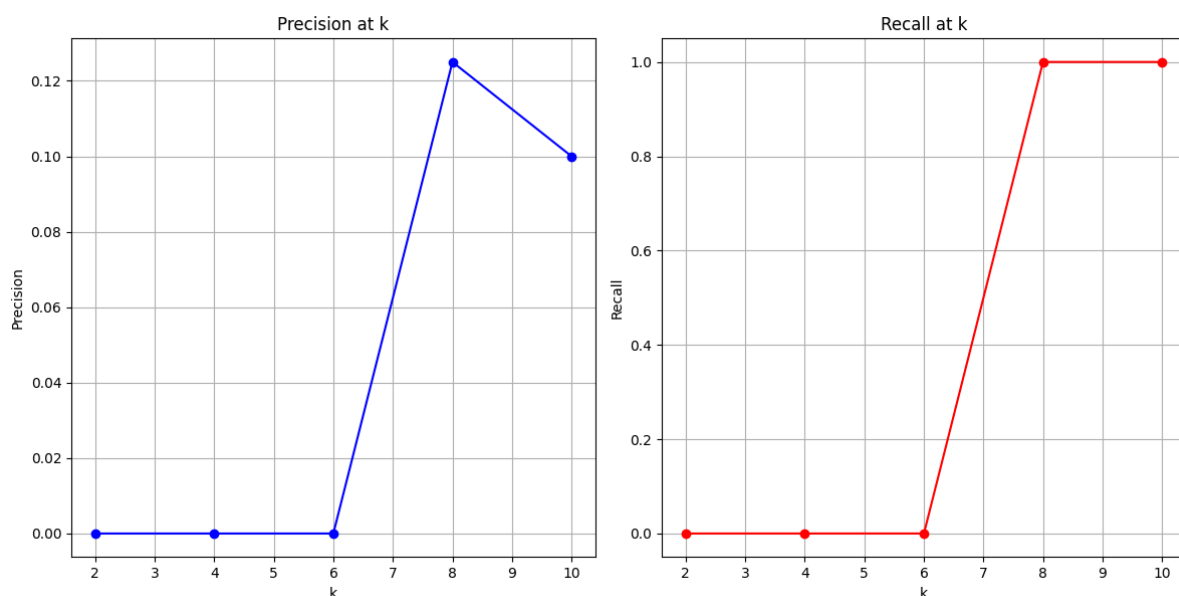
#### 5.1.3.1. Cálculo de Precision e Recall

1. Precision ( $P@k$ ): Avaliada como a proporção de documentos relevantes entre os  $k$  documentos retornados.
2. Recall ( $R@k$ ): Determinada pela proporção de documentos relevantes encontrados entre os disponíveis.

Essas métricas foram calculadas para valores de  $k$  (2, 4, 6, 8, 10) e serviram como indicadores da eficácia do modelo.

#### 5.1.3.2. Visualização com Gráficos

Após calcular as métricas de precisão e recall, foram gerados gráficos para visualização das métricas para os valores de  $k$  2, 4, 6, 8 e 10. Precision at  $k$  ( $P@k$ ) e Recall at  $k$  ( $R@k$ ) foram plotados para visualizar o desempenho do modelo em relação a cada valor de  $k$ .



Os 10 documentos mais relevantes, com os maiores scores, foram extraídos e exibidos após a busca, juntamente com seus títulos, caminhos e pontuação de similaridade. Esses resultados podem ser usados para identificar as fontes mais prováveis de plágio.

## **5.2 Sistema de Busca com Elasticsearch**

Fizemos a implementação de um sistema baseado no Elasticsearch para detectar documentos relevantes em uma base previamente indexada. A abordagem combina técnicas de pré-processamento, extração de palavras-chave, expansão de consultas e análise de métricas de precisão e revocação. O objetivo é otimizar a recuperação de informações, equilibrando relevância e abrangência.

### **5.2.1. Metodologia**

#### **5.2.1.1. Configuração do Sistema**

1. Elasticsearch: Utilizado para buscas eficientes em grandes repositórios.
2. Pré-processamento: Remoção de stopwords e tokenização para refinar o conteúdo textual.
3. Expansão de Consultas: Planejada para incorporar sinônimos e ampliar correspondências.

#### **5.2.2. Etapas do Processo**

##### **5.2.2.1. Leitura e Pré-processamento**

A função `ler_txt(caminho_arquivo)` extrai o conteúdo de arquivos `.txt` no repositório. Em seguida, a função `extrair_palavras_chave(texto, idioma="english", top_n=200)` remove stopwords e palavras não alfanuméricas, tokeniza o texto e retorna as 200 palavras mais frequentes como palavras-chave.

##### **5.2.2.2. Criação de Consultas**

A função `buscar_com_palavras_chave(palavras_chave)` constrói uma consulta do tipo `bool` no Elasticsearch: cada palavra-chave é adicionada a uma cláusula `should`. O parâmetro `minimum_should_match` exige a presença de todas as palavras. Os 200 documentos mais relevantes são recuperados e classificados por score.

##### **5.2.2.3. Cálculo de Métricas: Precision e Recall**

A função `calcular_precision_recall_at_k(results, k_values)` avalia o desempenho com base em  $k$  (2, 4, 6, 8, 10):

1. **Precision@K**: Proporção de documentos relevantes nos top- $k$  resultados.
2. **Recall@K**: Proporção de documentos relevantes recuperados em relação ao total disponível.

#### 5.2.2.4... Processamento Completo

A função `processar_arquivo(caminho_arquivo, k_values)` aplica o pipeline completo a cada documento:

1. Lê o conteúdo.
2. Extrai palavras-chave.
3. Realiza buscas.
4. Calcula métricas para valores de  $k$ .
5. Retorna os valores de precisão e revocação para análise.

#### 5.2.2.5. Visualização com Gráficos

A função `gerar_grafico(folder_path)` executa o pipeline para todos os documentos em um diretório, calcula as médias de  $P@K$  e  $R@K$ , e gera gráficos:

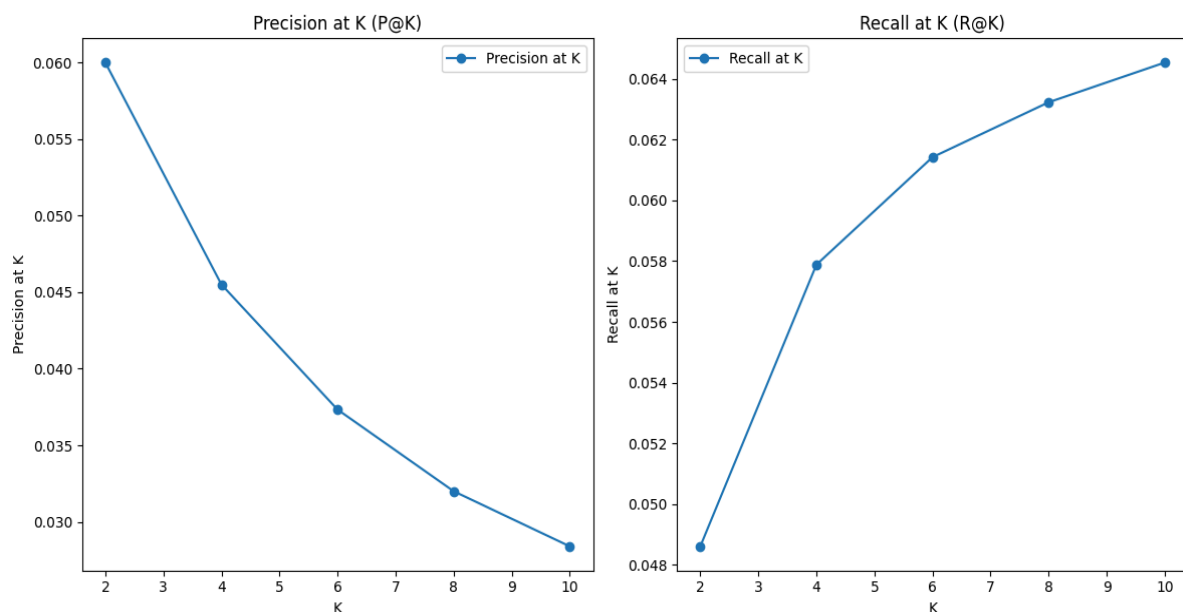


Figura 6: Gráfico ElasticSearch

### **5.2.3.. Resultados e Discussão**

#### **5.2.3.1.. Desempenho Esperado**

1. Precisão ( $P@K$ ): Avalia a relevância dos resultados nos k primeiros documentos.
2. Revocação ( $R@K$ ): Mede a proporção de documentos relevantes encontrados.

Gráficos permitem visualizar como precisão e revocação se comportam em diferentes configurações de k.

#### **5.2.3.2.. Interpretação Gráfica**

Um sistema ideal apresenta alta precisão e revocação para valores baixos de k. Este equilíbrio é essencial para garantir relevância sem comprometer a cobertura.

#### **5.2.3.3.. Limitações Identificadas**

1. Configuração Rígida de `minimum_should_match`: Exigir todas as palavras-chave pode reduzir a flexibilidade.
2. Peso das Palavras-Chave: Não diferencia entre palavras mais ou menos importantes.

### **5.3.. Tempo de Indexação Whoosh e ElasticSearch**

Whoosh:

Tempo médio de pre-processamento: 0.19973275184631348

Tempo medio de extracao: 0.2554649353027344

ElasticSearch:

Tempo médio de pré-processamento: 0.13181961521856688

Tempo médio de extração: 0.1686078572998047

Diferença: ElasticSearch é 34% mais rápido na extração em relação ao Whoosh.

## **6. Conclusão**

Um aspecto relevante na comparação entre os motores é o nível de complexidade envolvido na utilização do Elasticsearch em relação ao Whoosh. O Elasticsearch demanda a instalação, configuração e gerenciamento de um servidor dedicado, o que pode representar uma barreira inicial para usuários que não possuem familiaridade com a infraestrutura necessária. Essa complexidade, embora recompensada pela escalabilidade e desempenho do Elasticsearch em cenários de grandes volumes de dados, contrasta com a simplicidade do Whoosh.

Por ser uma biblioteca escrita em Python, o Whoosh pode ser integrado diretamente a projetos sem a necessidade de uma infraestrutura adicional, tornando-o ideal para aplicações menores ou em que a facilidade de uso seja prioritária. Em contrapartida, o Elasticsearch é mais adequado para aplicações que exigem alto desempenho, busca distribuída ou operações em larga escala, justificando seu maior esforço inicial de configuração.



## **7. References**

<https://whoosh.readthedocs.io/en/latest/intro.html>

<https://www.elastic.co/guide/index.html>

[Precision, Recall and F1-Score using R - GeeksforGeeks](#)