

Relatório de Busca e Recuperação de Informação

João Pedro Lacerda¹

¹Universidade Federal Rural do Rio de Janeiro (UFRRJ)

joaoplacerdas@ufrrj.br

Abstract. *This report is based on describing the process of creating two search engines, one using the Elastic Search API and one using a python library called Whoosh. After that, show the results obtained and the conclusions made in relation to each search engine. This report is based on describing the process of creating two search engines, one using the Elastic Search API and one using a python library called Whoosh . After that, show the results retrieved and the instructions made in relation to each search engine.*

Resumo. *Esse relatório se baseia em descrever o processo da criação de dois Motores de Busca, um utilizando a API Elastic Search e um utilizando a biblioteca de python chamado Whoosh. Após isso, mostrar os resultados retirados e as conclusões feitas com relação a cada motor de busca.*

1. Introdução

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 funciona as tecnologias que utilizamos para auxiliar no desenvolvimento. Sendo elas:

1.1. Whoosh

A Whoosh é como 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 as 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 dado.

O python contém uma biblioteca com funções que podemos chamar para acessar a API do Elasticsearch sem precisar fazer acesso direto com a API, e será essa biblioteca utilizada, para facilitar o desenvolvimento e o entendimento.

1.3. Entendendo o Conjunto de Dados

Para esse experimento utilizamos um Dataset chamado de **cranfield**, ele é dividido em 3 arquivos.

O primeiro arquivo chamado de **cran.all.1400**, ele é nosso arquivo em que realizaremos as buscas. São diversos artigos ou documentos que cada um é dividido em index, titulo, autor, bibliografia e corpo do documento. Para nosso caso foi apenas dividido em index, titulo e corpo, pelo fato do autor e bibliografia não ser importantes para a nossa busca. Então esses campos foram limpos como lixo afim de melhorar a legibilidade do documento.

O segundo arquivo chamado de **cran.qry**, ele será utilizado como valores de buscas. Dividido em index e pergunta, utilizaremos cada pergunta para achar os documentos que respondem a essa pergunta no primeiro arquivo.

Por ultimo o terceiro arquivo **cranqrel**, esse é o nosso gabarito. Esse arquivo tem os resultados de quais documentos do primeiro arquivo tem uma ligação com a pergunta do segundo arquivo. o primeiro elemento representa o index da pergunta, o segundo o index do documento relacionado e o terceiro o grau de relevancia do documento para a pergunta. sendo rankeados de 1 a 4, 1 sendo os melhores resultados e quarto os mais medianos.

2. Configurações

Para construir esse código foi necessário:

Python (3.10.8)

Whoosh(2.7.4)-Biblioteca

Elasticsearch(8.11.1)-Biblioteca

Matplotlib(3.8.0)-Biblioteca

dotenv(0.0.5)-Biblioteca

Além disso é necessário baixar o elasticsearch para conseguir realizar a busca na API. Após a instalação basta executar a pasta bin como é dito na documentação da ferramenta. Para conseguir realizar as buscas no código, é necessário criar um arquivo na pasta **Elasticsearch** chamado **.env** e copiar as variáveis do arquivo **.env.example** e alterar a senha e o nome para conseguir realizar a busca.

3. Entendendo pastas e arquivos

Em nosso projeto temos 4 pastas:

Whoosh: pasta que contém o código do motor de busca utilizando whoosh. Essa pasta também é referenciada para criar o index do whoosh.

ElasticSearch: pasta que contém o código do motor de busca utilizando elasticsearch. Nessa pasta também deve haver o arquivo **.env** para configurar o elasticsearch, segundo os dados que estão nele.

DataSets: pasta que contém os arquivos de dados utilizados pelos códigos tanto de Elasticsearch quanto do Whoosh.

Respostas: nessa pasta contém um arquivo texto, a fim de mostrar os resultados de cada busca. Esse arquivo traz 10 resultados para cada pergunta do arquivo **cran.qry**

No código de ambas as tecnologias, temos em comentários um método de realizar busca manual com interação com o usuário, caso deseje fazer uma busca específica. Porém para nosso experimento, iremos utilizar a busca automática para verificar como a busca se comporta em grande volume.

4. Desenvolvimento

Após entender as tecnologias que foram utilizadas e entender o conjunto de dados que vamos utilizar. Devemos entender como foram criados cada motor de Busca, passo a passo.

O processo de criação de ambas as tecnologias é o mesmo, primeiro deve-se criar ou abrir um index, pois será necessário para adicionar todos os documentos que estão no arquivo.

Após isso é importante ler o conteúdo do nosso data set, e dividir em partes em que estão tratando-se de documentos diferentes, no nosso caso é quando encontramos a tag **.I**, então dividimos em diversos documentos diferentes.

Depois, temos que percorrer cada documento, e limpar o que temos de "lixo" semântico, nesse caso removemos palavras que juntam os textos, como "the", "an" e "a", e o nome dos autores e da bibliografia, pelo fato de não ser útil para a nossa pesquisa.

Após esse processo, dividimos as informações que temos em nosso texto, a fim de ficar mais claro em qual informação que devemos buscar. Neste caso foi separado em título, index e corpo, para então indexarmos cada documento em nosso índice.

Com o nosso índice pronto podemos então realizar nossas buscas, em nosso caso, como temos um arquivo de buscas pronto, teremos que ler o arquivo separando em perguntas como em nosso arquivo de dados (separando pelo **".I"**), e percorrendo a cada documento fazemos busca e adicionamos as respostas das buscas e as perguntas no arquivo, como falado antes.

A fim de avaliar a qualidade de nossa busca, fazemos a leitura do nosso arquivo de gabaritos, a fim de verificar se nossas buscas estão contidas nas possíveis respostas do nosso gabarito. Para fazer essa avaliação, utilizaremos o gráfico de Recall@k e gráfico de Precision@k.

Antes de explicar sobre como avaliamos, devemos ver as singularidades de cada ferramenta.

4.1. Whoosh

No Whoosh para indexar, precisamos criar um esquema (Class Schema), que basicamente cria um estrutura. Para isso, devemos declarar um novo schema e ao criar o nosso índice, devemos passar o schema existente para que nosso index tenha as partes igual ao nosso esquema. Sendo assim quando adicionamos o documento ao index, adicionamos valores de acordo com o schema.

O processo de adicionar o index é feito de uma forma interessante, ao chamar um **indexer.write()** estamos abrindo o index a adição de novos documentos. Porém a indexação só é feita de fato ao dar um **indexer.commit()**, antes disso estamos apenas adicionando os documentos na memória para enfim no commit adicionar de fato no index.

Ao realizar as buscas, temos no arquivo as perguntas que desejamos buscar os documentos equivalentes a essas perguntas. Infelizmente no whoosh a busca é feita pelo texto e não pelas palavras, sendo assim se realizássemos a busca passando a pergunta sem tratamento, os resultados seria apenas os resultados que tem o texto exatamente igual o da pergunta, tendo respostas equivocadas. Para concertar isso colocamos o operador OR entre as palavras, para a busca ter uma precisão mais correta.

Afim de nossa busca ser mais assertiva ainda, foi utilizada a classe **Multifield-Parser**, afim de fazer buscas tanto pelo título tanto pelo corpo(contéudo). Poderíamos utilizar a função **QueryParser**, que busca em apenas um campo do index, porém nossa busca seria mais limitada.

4.2. Elasticsearch

O elasticsearch começa em suas singularidades, quando deve-se ter um usuário e senha para acessar a API, mesmo que local. Além disso na parte de indexação temos algo interessante na forma de indexar. Caso utilizarmos a forma clássica de indexar então a cada documento eu chamar a função adicionar da biblioteca, iremos a cada documento do meu arquivo enviar uma requisição de API, fazendo nosso programa ter atrasos absurdos de funcionamento, para resolver esse problema utilizamos a função **bulk()** que basicamente envia diversos documentos para a API de uma vez só. Para isso armazenamos dentro de uma array os documentos devidamente separados e no final ao percorrer tudo enviamos na função bulk. Porém ainda temos um problema, o tempo de adicionar todos os índices na API, em alguns casos a aplicação roda tão rápida que fazemos a busca antes mesmo da API terminar de adicionar todos os documentos no index, havendo erros em alguns casos. Para resolver isso utilizamos a função **time.sleep()** para aguardar 0,5 segundos, que é o tempo mais que suficiente para a API terminar de indexar todos os documentos, e assim fazer a busca correta.

Além disso utilizamos também assim como no Whoosh a busca por mais de um campo, porém dessa vez com JSON passando na "multimatch" do JSON.

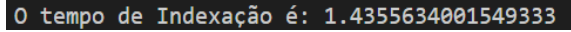
5. Resultados

Por fim temos aqui os resultados de código, para isso precisamos entender como foram feitos esses:

5.1. Tempo de indexação

Tempo que foi utilizado para todo o processo explicado de indexação, e calculado diminuindo o tempo final pelo tempo inicial:

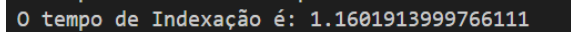
WHOOSH:



```
O tempo de Indexação é: 1.4355634001549333
```

Figure 1. Tempo indexação Whoosh

ELASTICSEARCH:



```
O tempo de Indexação é: 1.1601913999766111
```


Figure 2. Tempo indexação Elasticsearch

Aqui como podemos ver o tempo de indexação do elasticsearch é muito superior ao do whoosh. Não podemos esquecer que ainda incrementamos 0.5 de tempo para que se conclua com tranquilidade a indexação, porém nem sempre irá demorar esse tempo para se concluir, tendo então uma margem de erro no valor desse tempo no elasticsearch, podendo ser entre o valor passado e o valor passado menos 0,5 segundos.

5.2. Tempo de Busca

Tempo que foi utilizado para todo o processo explicado de Busca, e calculado diminuindo o tempo final pelo tempo inicial:

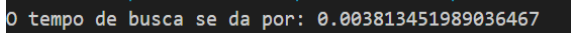
WHOOSH:



```
O tempo de busca se da por: 0.009352211556914779
```

Figure 3. Tempo de Busca Whoosh

ELASTICSEARCH:



```
O tempo de busca se da por: 0.003813451989036467
```

Figure 4. Tempo de Busca Whoosh

É claro observar que o Elasticsearch novamente é superior em relação ao tempo.

5.3. Recall@K

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

$$recall@k = \frac{número\ de\ itens\ relevantes\ no\ top\ k}{número\ total\ de\ itens\ relevantes\ no\ conjunto\ de\ dados}$$

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.

Como são diversos documentos fez-se a média para cada K , então foi feito um gráfico afim ver a média de K em todos os documentos:

WHOOSH:

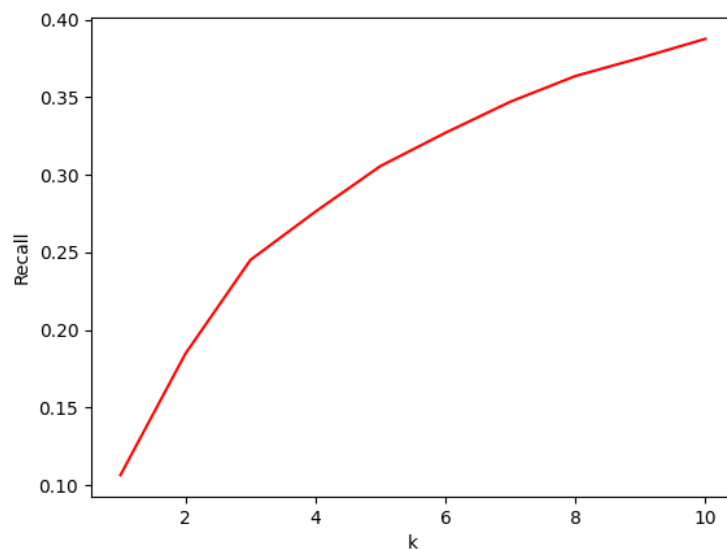


Figure 5. Recall@k Whoosh

ElasticSearch:

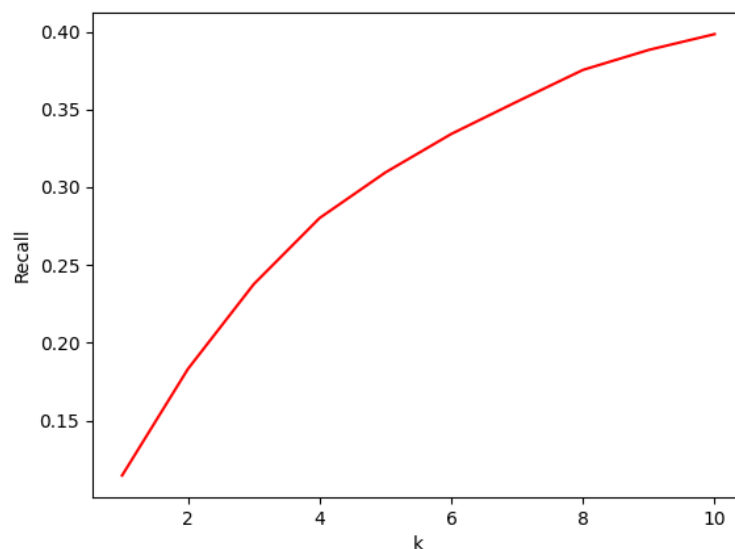


Figure 6. Recall@k Elasticsearch

Podemos ver que por mais que sejam muito proximos o Elasticsearch tem uma porcentagem de acerto um pouco melhor, mesmo quando usamos comparando com todos os valores possiveis.

5.4. Precision@K

Outra avaliação do desempenho do modelo foi realizada utilizando a métrica *precision@k*, que mede a proporção de itens relevantes entre os top *k* itens classificados. A fórmula para *precision@k* é dada por:

$$precision@k = \frac{\text{número de itens relevantes nos top } k}{k}$$

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 media para cada K, então foi feito um gráfico afim ver a média de K em todos os documentos: WHOOSH:

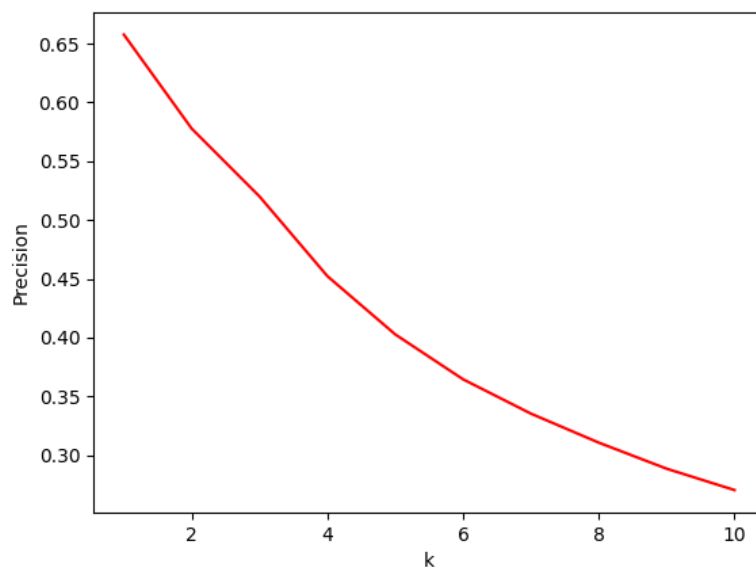


Figure 7. Precision@k Whoosh

ELASTICSEARCH:

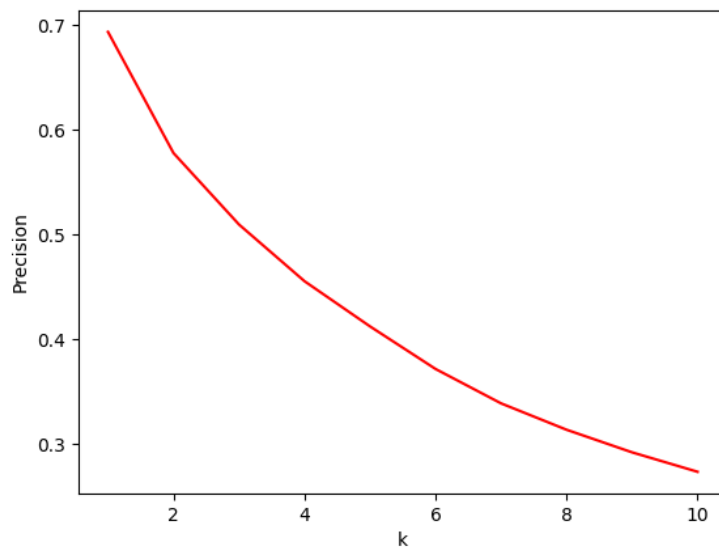


Figure 8. Precision@k Elasticsearch

Podemos ver que por mais que sejam muito proximos o Elasticsearch tem uma porcentagem de acerto um pouco melhor no incio, nos K mais altos sao basicamente parecidos.

6. Conclusão

Posso concluir que O elastic Search é uma ferramenta de pesquisa mais eficiente e mais rapida que o Whoosh, apesar da eficiência do Whoosh ser impressionante. Apesar do Elasticsearch ser mais complexo de configurar, a manuseabilidade é muito mais escalável que o whoosh. Whoosh ainda sim é uma ferramenta singular para buscas apesar de não observar tanto o contexto, apenas a igualdade. Podemos dizer que são duas ferramentas bem interessantes de se usar no dia a dia .

7. References

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

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

Duarte, F. (2023). The Problem of Search and Information Retrieval.Lecture,
Department of Computer Science, UFRRJ.