# Implementação de um Sistema de Buscas usando BM25

## Objetivo

O objetivo deste projeto é introduzir os conceitos básicos de Recuperação da Informação (Information Retrieval, IR) e o algoritmo de busca BM25. Será feita a construção de um sistema simples de IR que recupera informações relevantes de um corpus pré-determinado (o conjunto de dados CISI) baseado na query de um usuário. Será utilizado chatGPT para auxiliar nas etapas do projeto. Os resultados do projeto (como notebooks, relatório e outros arquivos) serão disponibilizados em uma conta no github.

É esperada a compreensão do BM25, familiaridade com sistemas de IR e suas métricas de avaliação, uso de Google Colab para geração de notebooks/apresentação, GitHub para hospedagem de repositórios e chatGPT para encontrar soluções e códigos de exemplo.

## Introdução

A área de Recuperação da Informação tem como objetivo desenvolver sistemas capazes de buscar e recuperar informações relevantes a partir de grandes coleções de dados. Esses sistemas são amplamente utilizados em diversas áreas, como e-commerce, mídia social, ciência da computação, biblioteconomia, medicina, entre outras. A busca e recuperação de informações é uma tarefa essencial para o processamento de dados em grandes volumes, tornando-se imprescindível para a tomada de decisões em diversas áreas.

A recuperação de informações é uma tarefa complexa, que envolve a análise de dados não estruturados e a seleção de informações relevantes a partir de um conjunto vasto de dados. Para realizar essa tarefa, é necessário o uso de técnicas de processamento de linguagem natural, mineração de dados e aprendizado de máquina, que permitem extrair informações úteis a partir dos dados brutos.

Nesse sentido, um projeto de IR envolve a análise de um conjunto de dados, a escolha de técnicas e algoritmos adequados para a busca e recuperação de informações, o desenvolvimento de um sistema de busca e a avaliação da sua eficácia. Uma vez implementado o sistema, é necessário avaliar seu desempenho e ajustar os parâmetros para melhorar sua eficiência.

Um projeto de IR pode envolver o desenvolvimento de sistemas para diferentes fins, como busca de informações em textos, imagens, áudio e vídeo. Esses sistemas são essenciais

para organizar e tornar acessível o grande volume de dados disponíveis na internet e em outras fontes de informação [5].
Neste projeto, será criado um sistema de IR baseado no algoritmo de busca BM25, que será testado no conjunto de dados CISI [1] e avaliado por um conjunto de métricas clássicas.

# Metodologia e Resultados

Toda a implementação do projeto foi feita usando a linguagem de programação Python em ambiente Google Colaboratory. Ao longo das sessões alguns snippets de código serão mostrados para exemplificar como o sistema de IR aqui apresentado pode ser utilizado. O código na íntegra se encontra no repositório do github anexo ao projeto.

As etapas do desenvolvimento desse projeto foram:

- Pesquisa e entendimento dos conceitos básicos de recuperação da informação;
- Entendimento do conjunto de dados CISI;
- Implementação do sistema de IR em python;
    - Processamento dos textos;
    - Algoritmo BM25;
    - Busca e recuperação baseado em uma query do usuário;
    - Extração de métricas de avaliação.

## Conceitos básicos de Recuperação da Informação

Sistemas de IR, de forma simples, são sistemas de busca. Em sistemas de IR é comum o usuário fornecer uma query (ou um texto a ser buscado) e o sistema deve ser capaz de, dado este texto, realizar a busca e recuperar as N melhores correspondências em um conjunto de dados (geralmente chamado de corpus).
Algumas métricas de avaliação, que são comumente utilizadas para avaliar a qualidade do sistema de IR, serão discutidas mais à frente.

## Conjunto de dados CISI

Neste projeto, o corpus utilizado para a busca foi o conjunto de dados CISI [1]. Este conjunto de dados contém 4 arquivos, 1460 textos, 112 queries de teste e 76 mapeamentos de relevância query -> texto.
Os 4 arquivos (aqui foram utilizados apenas 3), são:

- CISI.ALL: contém todos os 1460 textos para busca. O arquivo tem um formato específico indicando o índice do texto, autor, título e texto em si. Neste projeto foi feita a concatenação do autor, título e texto para criação de um texto expandido [2].
- CISI.QRY: arquivo contendo as 112 queries. O arquivo tem um formato indicando o índice da query e o texto da query em si.

- CISI.REL: arquivo contendo os 76  mapeamentos query -> textos. O arquivo tem um formato específico indicando o índice da query, e uma lista com os índices dos textos correspondentes (estes índices são os mesmos do arquivo CISI.ALL). Assim, este arquivo é o *ground truth*.
- CISI.BLN: não utilizado neste projeto.

Um exemplo de código utilizado para o carregamento dos dados encontra-se abaixo.

```
dataloader = DataLoader(PATH)
```

```
document_set, query_set, relevant_set = dataloader.load()
```

Figura 1: exemplo de código para carregamento dos dados. PATH indica o caminho dos arquivos, e os dados são carregados em forma de dicionário, com as chaves sendo os índices e os valores sendo os textos em questão.

## Processamento dos Textos

Os textos foram processados usando técnicas de processamento de linguagem natural (do inglês, *Natural Language Processing*, NLP). É importante destacar que o mesmo processamento se aplica tanto aos textos do corpus, quanto às queries.
O processamento feito seguiu as seguintes etapas:

- remoção de caracteres especiais (utilizado o módulo *string.punctuation*);
- tokenização dos textos (transformação do texto completo em uma lista de palavras ou tokens). É necessário tokenizar para utilizar como entrada nos algoritmos de matching como será mostrado;
- remoção de stopwords da língua inglesa (preposições, palavras comumente utilizadas no discurso e que não afetam o entendimento geral do texto). Essa etapa é realizada para remover parte do ruído causado por essas palavras;
- aplicação de stemmer (uma técnica utilizada para extração do núcleo das palavras, i.e. o núcleo das palavras loving e loved é love);

Este procedimento é aplicado para todos os textos do corpus e das queries, transformando-os em corpus e queries tokenizadas e já tratadas.

Um exemplo de código para o processamento dos textos encontra-se abaixo.

```
# Creating the stemmer and stopwords list
stemmer = nltk.stem.PorterStemmer()
stopwords = nltk.corpus.stopwords.words('english')

# Applying preprocessing class to documents and queries
preprocess = PreProcessing(stopwords = stopwords, stemmer = stemmer)
document_corpus = preprocess.preprocess_corpus(document_set)
query_corpus    = preprocess.preprocess_corpus(query_set)
```

Figura 2: exemplo de código de pré-processamento dos textos, transformando os textos originais em corpus limpos e tokenizados, tanto para os documentos de busca quanto para as queries.

## Algoritmo de busca BM25

BM25 (também conhecido como Okapi BM25) é um algoritmo de recuperação de informações usado em sistemas de busca para classificar documentos relevantes para uma determinada consulta ou query.

O BM25 foi desenvolvido para superar as limitações do modelo de espaço vetorial (VSM) na classificação de documentos. Em vez de atribuir pesos fixos a cada termo, o BM25 calcula uma pontuação ponderada para cada termo em um documento com base na frequência do termo na coleção de documentos e na frequência do termo no documento específico [4].

A pontuação do BM25 para um documento é calculada com base na frequência de cada termo na consulta, na frequência de cada termo no documento e no comprimento do documento. Os documentos com pontuações mais altas são considerados mais relevantes para a consulta de pesquisa.

O BM25 é amplamente utilizado em sistemas de busca na web e é um dos algoritmos mais eficazes disponíveis para classificar documentos relevantes para uma determinada consulta.

A função de score do BM25 é dada pela seguinte equação:

$$\text{BM25}(D, Q) = \sum_{i=1}^{n} IDF(q_i, D) \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i) + k_1 \cdot (1 - b + b \cdot |D|/d_{avg}))}$$

onde:

- Q: a query de interesse;
- qi: os tokens da query Q;
- D: o set de documentos do corpus;
- f(qi, D): o número de vezes que o termo qi aparece no documento D;
- |D|: o número de palavras existentes em D;
- davg: o número médio de tokens por documento;
- b e k1: hiperparâmetros.

O termo IDF (Inverse Document Frequency) é dado pela seguinte equação:

$$\text{IDF}(q_i) = \log\left(1 + \frac{N - N(q_i) + 0.5}{N(q_i) + 0.5}\right)$$

onde:

- N(qi): o número de documentos do corpus que contém o termo qi.

Desta forma, o BM25 tenta representar o texto em um espaço vetorial para que possa ser feita a busca baseada em alguma medida de distância definida neste espaço.

Em termos gerais, quanto mais vezes os termos da query aparecerem em um documento, maior será seu score. Por outro lado, o termo |D|/davg é um penalizador indicando que quando um documento é maior do que o tamanho médio dos documentos, o score será menor. O termo IDF é uma forma de atribuir maior peso ao score baseado no quão raro são os termos da query. Maiores scores indicam melhor match [6].

Neste projeto foram usadas duas versões do algoritmo BM25, uma implementada pela biblioteca em python *rank_bm25* da *Okapi* [4], e outra implementada do zero [6]. Comparações entre os algoritmos serão apresentadas. Um exemplo de código para utilização dos algoritmos encontra-se abaixo

```
# BM25 from Okapi
# Index all documents using BM25
bm25 = BM25Okapi(document_corpus)

# Calculating score and top N matches
scores = bm25.get_scores(tokenized_query)


# BM25 from scratch
# Fitting BM25 and recovering scores
bm25 = BM25()
bm25.fit(document_corpus)
scores = bm25.get_scores(tokenized_query)
```

Figura 3: Exemplo de código para utilização do BM25, tanto da implementação da Okapi, quanto da implementação do zero.

## Busca e recuperação baseado em uma query do usuário

Com as classes encapsulando a lógica de carregamento, pré-processamento e busca já implementadas, a próxima etapa foi a implementação de uma classe para encapsular a lógica da busca através das queries e cálculo de métricas de performance. Essa classe usa como input todas as outras classes já implementadas, visando ser um centralizador e orquestrador da lógica do sistema de IR.
Assim, a lógica do sistema pode ser descrita da seguinte maneira:

- Definição do *DataLoader*:
    - depende do *PATH* com a localização dos dados;
- Definição do *PreProcessing*:
    - depende do *stemmer* e das *stopwords;*
- Definição dos algoritmos:
    - tanto a versão Okapi quanto a do zero;
- Definição do *trec_eval*:

    - Arquivo utilizado para cálculo de métricas. Aqui foi utilizada a versão do módulo *evaluate* da *HuggingFace* [8]. Esse componente é responsável por calcular as métricas mais comuns de sistemas de IR.
- Definição da classe *SearchEngine*:
    - depende do *DataLoader*, *PreProcessing*, algoritmos e *trec_eval.*

Um exemplo de código de como utilizar o sistema de IR está abaixo:

```
1 # DataLoader
2 PATH = '/content/drive/MyDrive/Colab Notebooks/IR/data/'
3 dataloader = DataLoader(PATH)
4
5 # PreProcessing
6 stemmer = nltk.stem.PorterStemmer()
7 stopwords = nltk.corpus.stopwords.words('english')
8 preprocessing = PreProcessing(stopwords = stopwords, stemmer = stemmer)
9
10 # Algorithm
11 algo1 = BM25Okapi
12 algo2 = BM25()
13
14 # trec eval
15 trec_eval = load("trec_eval")
16
17 # SearchEngine1
18 search_engine_1 = SearchEngine(dataloader, preprocessing, algo1, trec_eval)
19 most_relevant_docs_1, df_metrics_1 = search_engine_1.run()
20
21 # SearchEngine2
22 search_engine_2 = SearchEngine(dataloader, preprocessing, algo2, trec_eval)
23 most_relevant_docs_2, df_metrics_2 = search_engine_2.run()
```

Figura 4: Exemplo de código de utilização do sistema de IR. É importante destacar que aqui são testados os dois algoritmos de busca implementados (definidos nas linhas 11 e 12).

Esse sistema vai executar toda a lógica de busca, retornando uma lista com os documentos recuperados mais relevantes para cada query, e um *pandas DataFrame* com as métricas para cada uma das queries. Estatísticas descritivas podem ser extraídas desse *DataFrame* de métricas e assim os algoritmos podem ser comparados entre si. Mais sobre as métricas a seguir.

Uma outra funcionalidade do sistema de IR desenvolvido é a recuperação de documentos relevantes para uma query específica do usuário (sem necessariamente ser do arquivo de queries). Um exemplo de uso encontra-se a seguir.

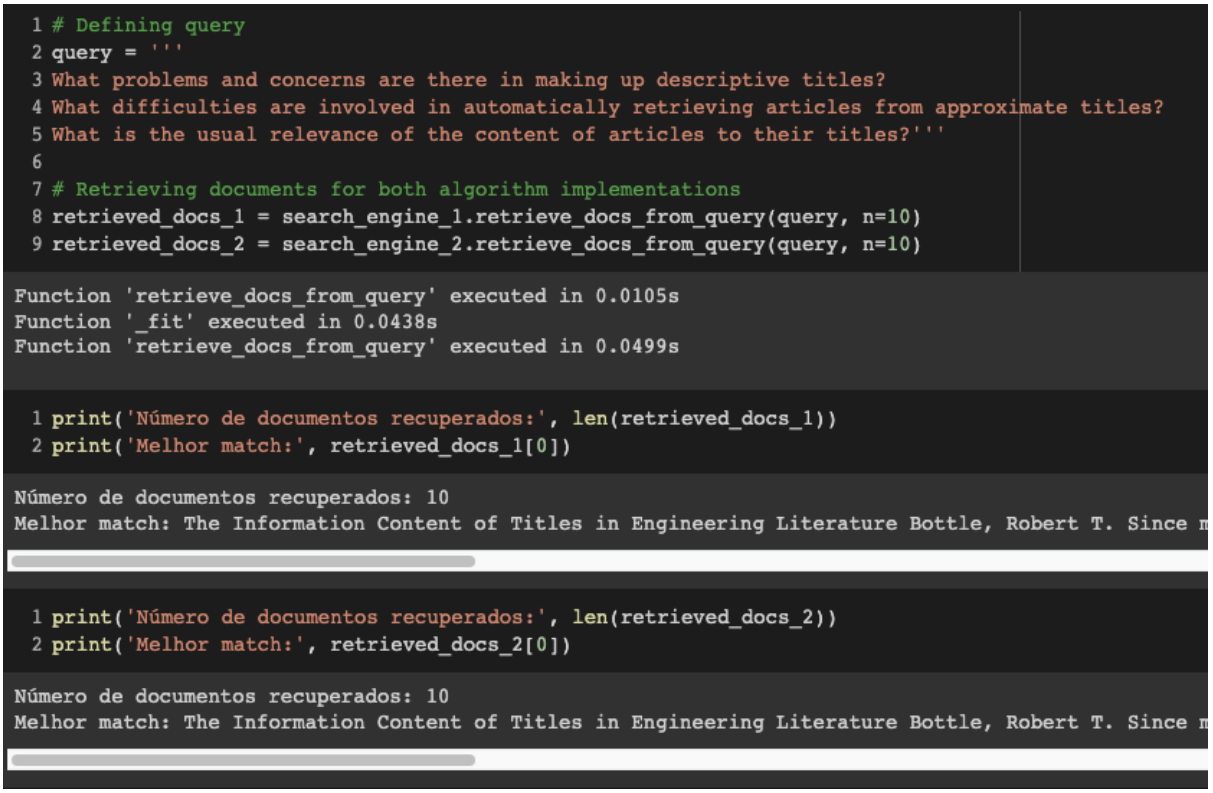

Figura 5: exemplo de código para o uso do sistema de IR recuperando melhores matches para uma query específica de usuário.

Aqui, na primeira célula é feita a definição da query como uma *string*, que então é passada para o método *retrieve_docs_from_query*, definindo um N de documentos a serem recuperados. O output desta função é uma lista com os top N documentos recuperados.

## Extração de métricas de avaliação

Um sistema de IR tem como objetivo extrair documentos relevantes baseados nas queries do usuário. Para saber se o sistema está realmente calibrado, é necessário utilizar algumas métricas para avaliação. O arquivo de documentos relevantes será utilizado como anotação (ou *ground truth),* para as queries submetidas (usualmente fornecido por um humano), desta forma é possível comparar os resultados do sistema com o resultado esperado. O conjunto (q, d, r), onde r é a anotação fornecida por um humano para a tupla (q(uery),d(ocument)) é chamado de *relevance judgments*. Uma forma comum de testar e extrair métricas do *relevance judgments* é através do *trec_eval* [3].

TREC é a sigla de *Text Retrieval Conferences*, uma série de eventos de IR organizados pelos EUA e pelo NIST com o objetivo de reunir praticantes de sistemas de IR, avaliar e rankear resultados de algoritmos de uma forma padronizada e rápida [3]. A implementação do *trec_eval* utilizada neste projeto foi a do módulo *evaluate* da *HuggingFace* [8].

As métricas que essa versão da *trec_eval* implementa são:

- map*: Mean average precision;*
- gm_map*: geometric mean average precision;*
- bpref*: binary preference score;*
- Rprec*: precision@R,* onde R é o número de documentos relevantes.
- recip_rank*: reciprocal rank*
- P@k*: precision@k* (k em [5, 10, 15, 20, 30, 100, 200, 500, 1000]).
- NDCG@k*: nDCG@k* (k em [5, 10, 15, 20, 30, 100, 200, 500, 1000]).

## Binary Preference Score

Considera apenas se o documento é relevante ou não. Pode ser calculada em cima da matriz de confusão, sendo a razão entre os documentos recuperados que de fato são relevantes e o número total de documentos recuperados. Não leva em consideração o rank e relevância relativa entre os documentos relevantes, por isso é recomendado usar em conjunto com outras métricas [7].

## Reciprocal Rank (RR)

É definido como o inverso do rank de um documento. Ou seja, se um documento aparece em primeiro no rank, RR=1, se aparece em segundo, RR=½, se aparece em terceiro, RR=⅓ e assim por diante [3].

## Precision

É definido como a fração dos documentos ranqueados na lista R que são relevantes

$$\text{Precision}(R, q) = \frac{\sum_{(i,d)\in R} \text{rel}(q, d)}{|R|},$$

onde rel(q,d) indica se o documento d é relevante para a query q, assumindo relevância binária. Muitas vezes Precision é avaliado em um corte k, com notação *precision@k* ou *P@k*, determinando o precision em k documentos recuperados [3].

## nDCG

*Normalized Discounted Cumulative Gain* é a métrica que é mais frequentemente utilizada para medir a qualidade de resultados de busca na web. Gain é usado com um senso de utilidade, isto é, quanto valor um usuário deriva de um resultado em particular. Dois fatores entram na conta: (1) a grade de relevância, resultados altamente relevantes valem mais que resultados relevantes; (2) o rank em que um resultado aparece, ou seja, resultados no topo valem mais.

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

Por fim aplica-se um fator de normalização IDCG que se refere ao rankeamento ideal da lista de resultados. Aqui também existem variantes como nDCG@k [3].

### Average Precision

A maneira mais intuitiva de entender average precision (e suas derivadas) é como uma média dos *Precision* em diferentes cortes k correspondendo a uma aparição de cada documento relevante.

$$\mathrm{AP}(R,q) = \frac{\sum_{(i,d)\in R} \mathrm{Precision@}i(R,q)\cdot \mathrm{rel}(q,d)}{\sum_{d\in C} \mathrm{rel}(q,d)},$$

Para outros tipos de cálculo de average precision, outros tipos de médias podem ser utilizadas, como média geométrica [3].

Algumas métricas geradas pelos algoritmos encontram-se abaixo

| | map_Okapi | gm_map_Okapi | bpref_Okapi | Rprec_Okapi | recip_rank_Okapi | map_Scratch | gm_map_Scratch | bpref_Scratch | Rprec_Scratch | recip_rank_Scratch |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 76.000000 | 76.000000 | 76.000000 | 76.000000 | 76.000000 | 76.000000 | 76.000000 | 76.000000 | 76.000000 | 76.000000 |
| mean | 0.225431 | 0.225431 | 0.202589 | 0.245756 | 0.661638 | 0.224083 | 0.224083 | 0.201311 | 0.246825 | 0.657520 |
| std | 0.157095 | 0.157095 | 0.154056 | 0.160005 | 0.372463 | 0.154845 | 0.154845 | 0.146955 | 0.152068 | 0.376336 |
| min | 0.011458 | 0.011458 | 0.000000 | 0.000000 | 0.013889 | 0.011614 | 0.011614 | 0.000000 | 0.000000 | 0.014286 |
| 25% | 0.109605 | 0.109605 | 0.083363 | 0.142143 | 0.333333 | 0.104506 | 0.104506 | 0.090604 | 0.145089 | 0.333333 |
| 50% | 0.196504 | 0.196504 | 0.187077 | 0.250000 | 1.000000 | 0.201266 | 0.201266 | 0.179778 | 0.250000 | 1.000000 |
| 75% | 0.308895 | 0.308895 | 0.276886 | 0.339112 | 1.000000 | 0.311406 | 0.311406 | 0.283473 | 0.351972 | 1.000000 |
| max | 0.833584 | 0.833584 | 0.833333 | 0.833333 | 1.000000 | 0.780052 | 0.780052 | 0.740741 | 0.722222 | 1.000000 |

Figura 6: alguma das métricas extraídas pelo *trec_eval*

Como os resultados ficaram pequenos na Figura 6, os resultados na íntegra podem ser encontrados no notebook *search_engine_bm25*, onde existe uma tabela comparativa de todas as métricas para ambos os modelos.

# Conclusão

Foi criado um sistema de recuperação da informação simples usando como algoritmo de busca o BM25. O sistema foi testado no conjunto de dados da CISI. Foram utilizadas técnicas de NLP para limpeza de texto e tokenização. Foram feitas duas versões do BM25, uma implementação da Okapi e outra do zero. Métricas de avaliação foram utilizadas para comparar os algoritmos.

# Repositório

A estrutura do repositório segue o seguinte formato:

**notebooks/**

- **exploration.ipynb**: notebook inicial de exploração. A primeira versão do código foi implementada neste notebook, onde toda a lógica inicial foi montada, desde o carregamento e processamento dos dados, rodagem dos algoritmos e extração das métricas de avaliação.
- **search_engine_bm25.ipynb**: este notebook é a versão final do projeto. Contém toda a lógica encapsulada em classes e funções, facilitando todas as etapas do sistema de IR. Contém uma tabela com as métricas comparando os algoritmos.

**data/**

- CISI.ALL
- CISI.REL
- CISI.QRY
- CISI.BLN

# Utilização do chatGPT

O chatGPT foi uma ferramenta que me auxiliou em praticamente todas as etapas do projeto:

- entendimento do que é um sistema de recuperação da informação;
- entendimento do algoritmo BM25;
- como avaliar um sistema de recuperação da informação;
- como entender e obter métricas no formato do trec_eval.

Obtive snippets de código em python nas seguintes etapas:

- carregamento dos dados do CISI;
- cálculo de métricas e formatação para gerar os formatos do *trec_eval*;

Os códigos dados pelo chatGPT continham erros, mas foram ótimos pontos de partida. Aprendi que a ferramenta em si é muito boa, desde que exista um senso crítico com o que é gerado por ela. Os exemplos de código são um exemplo disso: servem como template, como ponto de partida, e depois é necessário realizar ajustes e correção de erros.

# Referências


[1] CISI collection - http://ir.dcs.gla.ac.uk/resources/test_collections/cisi/

[2] Free-to-use datasets - https://www.pragmalingu.de/docs/guides/data-comparison#cisi

[3] Lin, J., Nogueira, R., Yates,A. Pretrained Transformers for Text Ranking: BERT and Beyond - https://arxiv.org/pdf/2010.06467.pdf

[4] Documentação rank_bm25 - https://pypi.org/project/rank-bm25/#description

[5] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. *An Introduction to Information Retrieval*, Cambridge University Press, 2009, p. 233.

[6] Stephen Robertson & Hugo Zaragoza (2009). "The Probabilistic Relevance Framework: BM25 and Beyond". *Foundations and Trends in Information Retrieval*. **3** (4): 333–389. CiteSeerX 10.1.1.156.5282. doi:10.1561/1500000019

[7] Sakai, T., Kando, N. On information retrieval metrics designed for evaluation with incomplete relevance assessments. *Inf Retrieval* 11, 447–470 (2008). https://doi.org/10.1007/s10791-008-9059-7

[8] HuggingFace Evaluate - https://huggingface.co/spaces/evaluate-metric/trec_eval
[9] trectools - https://github.com/joaopalotti/trectools