Building a Simple Information Retrieval System using BM25 and GPT-3 and evaluated in the CISI collection

Introdução

Neste exercício, implementamos um sistema de buscas utilizando o algoritmo BM25 e avaliamos seu desempenho nos dados da CISI Collection. O algoritmo foi implementado usando linguagem Python num ambiente Jupyter Notebook (Google Colab) e os resultados foram avaliados utilizando o script trec-eval. Também foi criado um Makefile para fazer download e extração dos dados brutos do dataset.

Chat GPT foi usado em diversas etapas do projeto para buscar informações sobre IR e Python (praticamente eliminando a necessidade de usar o Google), desenvolvimento e formatação de código (doc strings e type hints). Também usei o GitHub Copilot em algumas partes, mais especificamente na geração dos gráficos finais reportados.

O ChatGPT também foi usado para melhorar o texto em algumas partes. Dado que usamos os assistentes de AI foram usados praticamente no projeto todo, tentei deixar o mais claro possível quando/como eles foram usados (por vezes, até os prompts usados são reportados). Nesse documento, texto formatado em vermelho foi escrito usando algum texto gerado pelo ChatGPT de forma quase integral.

Conceitos importantes / ferramentas

CISI Collection / Data Prep

Boa parte do tempo total do projeto foi gasto para fazer o processamento dos arquivos, dado o formato não usual (pelo menos para indivíduos não tão habituados com dataset sde IR, como eu). Um bom recurso para o entendimento desse dataset pode ser encontrado em [1].

Os arquivos CISI.ALL e CISI.QRY possuem uma estrutura semelhante e característica, com cada documento (arquivo CISI.ALL) ou query (CISI.QRY) sendo iniciados por uma linha contendo o marcador especial “.I” seguido de um número inteiro usado como identificador único, com os demais campos sendo iniciados pelo marcador especial, e as linhas seguintes representando o conteúdo dessa tag. Uma descrição detalhada de cada marcador especial pode ser encontrada em [1] e no notebook de entrega.

Os marcadores usados em cada arquivo e sua função estão na tabela abaixo:

Marcador	**Função CISI.ALL**	**Função CISI.QRY**
.I	id único	id único

.W	texto do documento	texto da query
.T	título do documento	título da query (não usado)

O arquivo CISI.REL contain os “relevance judgments”, com dados anotados relacionando cada query aos documentos relevantes. Cada linha no arquivo possui o ID da query, o ID do documento, e um score de relevância (0 para não relevante e 1 para relevante). Como todas linhas possuíam relevância igual a zero, para conseguir finalizar a tarefa e eu considerei que os documentos que fossem relacionados a uma query são relevantes para ela. Além disso, removemos queries sem documentos relevantes associados, pois não é possível quantificar os resultados retornados para elas; isso reduziu o número total de queries de 112 para 76.

Inicialmente eu tentei fazer o parse dos arquivos CISI.ALL e CISI.QRY usando regex, mas percebi que seria muito trabalhos (mesmo usando o ChatGPT para gerar as expressões) gerar boas expressões regulares. A solução final envolve a leitura incremental de cada arquivo, mapeamento de cada segmento de arquivo (ID, tag, conteúdo), e no final, para cada documento, mapeamos o conteúdo de cada tag a um dicionário.

O arquivo de qrels estava num formato mais trivial, então simplesmente lemos ele linha a linha. Além disso, esse arquivo também foi processado no formato TREC para ser usado na avaliação dos resultados.

BM25

BM25 é baseado em exact match entre os termos das queries e documentos, considerando o comprimento médio dos documentos do corpus inteiro, a dá mais peso a termos que aparecem em menos documentos. Para calcular o score entre um documento d e uma query q, foi usada a seguinte expressão [2]:

$$\mathrm{BM25}(q,d) = \sum_{t \in q \cap d} \log \frac{N - \mathrm{df}(t) + 0.5}{\mathrm{df}(t) + 0.5} \cdot \frac{\mathrm{tf}(t,d) \cdot (k_1 + 1)}{\mathrm{tf}(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{l_d}{L}\right)}$$

O termo contendo log é chamado de inverse document frequency (ID) e é composto por:

- df(t): quantos documentos contém o termo
- N: número total de documentos na coleção

Um valor de epsilon de 1e-8 foi adicionado ao denominador para evitar divisão por zero, que ocorre quando um documento aparece em todos documentos (isso foi uma sugestão do ChatGPT e não está incluído na fórmula acima).

O segundo termo da multiplicação é composto por:

- tf(t,d): quantas vezes o termo t aparece no documento d
- k1 e b são parâmetros configuráveis
- ld é o comprimento do documento d e L e o comprimento médio dos documentos no corpus, de forma que o score considera o comprimento normalizado do documento

O score de um par (query, documento) é calculado usando apenas a intersecção dos termos, o que torna o BM25 dependente do exact match entre os termos das queries e documentos.

Eu inicialmente tentei implementar o BM25 usando a solução mais eficiente possível, vetorizando códigos utilizando numpy, sklearn.feature_extraction.text.CountVectorizer para tokenização e matrizes esparsas SciPy para armazenar os dados; além de ser um tempo perdido otimizando algo sem necessidade, também incluiu bugs que não consegui identificar e resultou em métricas ruins de avaliação.

Após isso, tentei fazer o código mais simples e didático possível, implementando cada passo numa função separada (construção do inverted index, cálculo do IDF, score e retrieval dos documentos candidatos); além de obter melhores resultados, também foi bem mais fácil de achar bugs e aprender sobre o método BM25. E, provavelmente, ficou tão eficiente quanto a tentativa inicial de implementação.

Queries e documentos foram pré-processados apenas usando tokenização simples (por espaços) e remoção de caracteres non-word e non-space, com o texto sendo convertido para lowercase; a remoção ou não de stop words (uma lista fixa gerada pelo ChatGPT) foi opcional e explorada nos experimentos.

ChatGPT / GitHub Copilot

ChatGPT foi usado de forma constane no desenvolvimento do código. Nem sempre o texto gerado era o que eu queria, mas foi possível induzir melhores gerações mudando o prompt fornecido (em alguns casos parecia que eu estava conversando com o ChatGPT).

Detalhes mais profundos são mostrados no notebook, mas prompts comummente foram:

- “explain this code to me”
- “add google style doc strings to this code, respect PEP8, respect a maximum of 79 chars, use single quotes whenever possible”
- “why is this code wrong?” or “I am receiving the error <ERROR>”: com isso consegui ir corrigindo erros de códigos iterativamente
- “why did you use this?“: bom para entender o código, e também aprendi algumas coisas sobre Python e bibliotecas
- “is this code OK?”, “how would you rate this code?”, “how can this code be improved?”: com isso foi possível ir melhorando o código aos poucos

Métricas de avaliação

O script trec-evalfoi usado para avaliação dos resultados e pode ser baixado em [5]; a escolha de utilizá-lo foi dada por ele ser amplamente usado em problemas de IR e também porque julguei importante ter uma avaliação com o mínimo possível de inclusão de erros.

Também tentei automatizar o download e instalação usando o ChatGPT para gerar um Makefile, mas desisti porque estava gastando muito tempo; instruções de instalação e uso do trec-eval podem ser encontradas em [4].

As seguintes métricas foram consideradas:

- precision@k e recall@k: precision e recall considerando os primeiros k documentos retornados; essas métrica não consideram a ordem dos documentos retornados. Reportamos métricas para k em {1,5,10}
- NDCG@20 (average cumulative gain at top20 queries): métrica frequentemente usada em sistemas de busca e que calcula a efetividade de um ranking em fornecer scores altos para documentos relevantes e scores baixos para documentos irrelevantes.

Experimentos/Resultados

Para compreender melhor o efeito do processamento nos resultados, fiz alguns experimentos variando a preparação dos dados. As seguinte configurações foram variadas:

- texto usado para representar o documento: título, abstract e título+abstract
- remover ou manter stop words (lista fixa gerada pelo ChatGPT)
- remove or keep stop words (the list of stop words was generated with ChatGPT)

Todos experimentos usaram k1=1.2 e b=0.75 no BM25 (sugestão do ChatGPT) e retornaram o score de todos documentos, com um limite de documentos retornados na avaliação igual a 100 (usando a flag “-M” no script trec-eval).

O sistema teve pior desempenho quando stop words não foram removidas, o que é esperado já que a probabilidade de um documento conter elas é alta. Ao manter stop words, usar apenas o título teve melhor desempenho; o melhor resultado foi obtido quando representamos documentos com o título + abstract e removemos stop words do texto.

corpus	ndcg_cut_20	P_1	P_5	P_10	recall_1	recall_5	recall_10
title_abstract_remove_stopwords	0.3354	0.5395	0.3895	0.3079	0.0350	0.0856	0.1404
abstract_remove_stopwords	0.3146	0.5000	0.3816	0.3092	0.0216	0.0729	0.1347
title_remove_stopwords	0.2290	0.3026	0.2711	0.2250	0.0123	0.0503	0.0814
title_keep_stopwords	0.1703	0.2500	0.1974	0.1711	0.0101	0.0345	0.0562
title_abstract_keep_stopwords	0.0996	0.1053	0.1211	0.0961	0.0018	0.0152	0.0232
abstract_keep_stopwords	0.0915	0.1053	0.1158	0.1000	0.0019	0.0130	0.0226

Trabalhos futuros

Gostaria de testar algumas coisas caso tivesse tempo:

- usar também o título da quer
- mais pré-processamento nos dados (stemming, lemmatization etc)
- utilizar expansão nas queries (docT5query [7] ou o próprio ChatGPT)

Referências

[1] https://www.pragmalingu.de/docs/guides/data-comparison/#cisi
[2] [2010.06467] Pretrained Transformers for Text Ranking: BERT and Beyond

[3] GitHub Copilot · Your AI pair programmer
[4] https://github.com/usnistgov/trec_eval
[5] https://trec.nist.gov/trec_eval/
[6] https://www.elastic.co/guide/en/elasticsearch/guide/master/inverted-index.html
[7] From doc2query to docTTTTTquery