Algoritmo de busca: Find my Pasta

Gustavo Ramalho
Gustavo Sanches Costa
Lucas Bryan Treuke
Rodrigo Gomes Hutz Pintucci
Vanessa Berwanger Wille

Abril de 2024

1 Introdução

O objetivo do projeto é desenvolver uma search engine capaz de encontrar arquivos ou trechos de conteúdo por nome do arquivo, conteúdo do arquivo e semântica.

2 Base de dados e escopo

Para restringir o escopo, e evitar problemas com falta de dados, optamos por selecionar uma base organizada que possa simular nossos arquivos e possibilitar o desenvolvimento. Nesse sentido, encontramos uma base de dados [1] que contém detalhes de receitas e suas avaliações. Ela consiste em mais de 220 mil receitas e mais de 700 mil avaliações de receitas cobrindo 18 anos de interações de usuários e uploads em Food.com (anteriormente GeniusKitchen)

Assim, a search engine envolveu a criação de um sistema de busca que permite aos usuários pesquisar eficientemente receitas por meio de seus atributos associados, como nome, descrição, ingredientes, instruções, categorias (tags), avaliações e informações nutricionais, considerando que essas informações estejam dispostas no conteúdo do arquivo, e não em forma tabelar ou de maneira dependente da estrutura.

3 Métricas de avaliação

Para avaliar os modelos da search engine, optamos pela avaliação offline, onde foram criadas queries que incluam pesquisas literais e semânticas, tanto por nome e conteúdo de aquivo.

Com isso, buscou-se usar as seguintes métricas para capturar diferentes aspectos da qualidade dos resultados recuperados:

- Discounted Cumulative Gain (DCG): Usada como métrica principal, o DCG considera tanto a relevância dos documentos retornados quanto sua posição na lista de resultados. Documentos relevantes que aparecem em posições mais altas contribuem mais para o DCG, refletindo a importância de retornar resultados relevantes de alta qualidade no topo da lista. Sendo sua fórmula:

$$DCG = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)}$$

onde, i é a posição do documento na lista de resultados e rel_i é a relevância do documento na posição i.

- Mean Average Precision (MAP): A MAP é uma métrica simples de implementar e fornece uma visão geral da precisão média dos resultados. Ela é calculada como a média das precisões para cada consulta, onde a precisão é a proporção de documentos relevantes entre os documentos retornados até um determinado ponto na lista de resultados. A MAP é útil para comparar diferentes sistemas de busca e entender sua capacidade de recuperar documentos relevantes em média.

4 Modelos

Definidas as métricas, decidimos testar três modelos diferentes para comparar os resultados. Dois deles utilizam medidas simples de avaliação *text-based*: **TF-IDF** e **BM25**. Testamos também um modelo semântico por **e5**, com a ideia de que seja capaz de melhor entender as relações entre as palavras para *queries* mais complexas. Os modelos foram alimentados com todos os dados de nosso conjunto.

4.1 TF-IDF

Nosso primeiro modelo calcula a relevância de um documento para uma determinada *query* a partir da métrica **TF-IDF**. Ela é composta de duas estatísticas: *Term frequency* e *Inverse distribution frequency*, tendo seu produto como resultado final.

O cálculo da *Term frequency* é dado por:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

onde $f(t,d)$ é a contagem de um termo t em um documento d, sendo o denominador a contagem total de termos. Isso faz com que um termo mais frequente retorne um maior resultado.
O cálculo da *Inverse distribution frquency* é dado por:

$$idf(t,D) = \log \frac{N}{|\ \{d : d \in D \text{ and } t \in D\}\ |},$$

onde $N = |D|$ é o número total de documentos e d o conjunto de documentos que contém o termo t. Dessa forma, é dado maior valor para termos que fornecem mais informação, com uma maior pontuação para documentos com termos raros (é válido informar que podemos adicionar 1 no denominador para não obtermos divisão por 0).

Por fim, chegamos no resultado final por $tfidf(t,d,D) = tf(t,d) * idf(t,D)$. Perceba que essa métrica não necessita que o modelo seja treinado, e que a relação entre termos e suas nuâncias não são levados em consideração, apenas suas frequências.

4.2 BM25

Nosso segundo modelo é o **BM25**, que aprimora a fórmula vista no **TF-IDF** levando dois quesitos em consideração: penalização de termos saturados e tamanho do documento.
Para penalizar termos saturados, utilizamos $\frac{TF}{TF+k_1}$, onde k_1 controla a curva de saturação. Isso faz com que a partir de certo ponto, o acréscimo de um termo não influencie tanto. Essa atualização também faz com que resultados que utilizem todos os termos da *query* tenham mais visibilidade do que resultados que utilizam apenas um dos termos com maior frequência.

Para levar o tamanho do documento em consideração, podemos multiplicar k_1 por $\frac{|D|}{avgdl}$, sendo *avgdl* o tamanho médio dos documentos. Mais especificamente, multiplicamos por $(1 - b + b\frac{|D|}{avgdl})$, onde podemos alterar b para determinar quanto isso é relevante.

A função *IDF* é levemente alterada, mas não causa muitas alterações no resultado final:

$$IDF(q_i) = \log(1 + \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}),$$

onde q_i é uma *keyword* da *query*. Mais informações sobre essas transformações podem ser vistas aqui: Understanding TF-IDF and BM25 No fim, temos a seguinte fórmula para o **BM25**:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) * \frac{f(q_i,D) * (k_1+1)}{f(q_i,D) + k_1(1 - b + b\frac{|D|}{avgdl})}$$

Novamente, não é necessário treinar o modelo, e a relação entre os termos não influencia em seus resultados.

4.3 E5

Por fim, utilizamos um modelo semântico com o objetivo de que entender a relação entre os *tokens* retorne resultados mais coerentes - como por exemplo, caso a *query* traga a ideia de ausência de um ingrediente ou método de preparo. Nossa escolha foi o modelo **E5**, que significa *EmbEddings from bidirEctional Encoder rEpresentations*. Mais especificamente, utilizamos o E5-Small-V2.

Trata-se de um *LLM* que utiliza *Transformers* em sua estrutura para gerar os *embeddings* dos *tokens*. O diferencial desse modelo está na forma de treinar os *embeddings*: utiliza a técnica de *Contrastive Learning* no dataset *CCPairs - Colossal Clean Text Pairs*.

Esse conjunto de dados se destaca devido à sua grande quantidade de textos diversos e de alta qualidade. Junto com *fine-tuning* supervisionado, esse modelo consegue ser eficiente e versátil, parecendo assim ser uma boa escolha para nosso teste. O modelo foi treinado com o conjunto inteiro.

5 Resultados e avaliação

5.1 Avaliação

Para avaliar os resultados obtidos pelos modelos supracitados, foram criadas doze *queries* de diferentes tipos, podendo variar de *queries* com o foco em *Keywords* e com o foco na semântica da *query*.

As *queries* por *keyword* são, no geral, mais fáceis e é esperado que os modelos performem melhor nelas. Enquanto as semânticas podem possui alguma palavra que inverta o sentido, por exemplo uma negativa "*no*"ou a preposição "*without*".

Além disso, também anotou-se a dificuldade de cada *query* para futuras explorações.

Query	Tipo	Dificuldade
chocolate cake recipe	Keywords	Simples
strogonoff with rice	Keywords	Simples
fresh lemonade	Keywords	Simples
apple pie	Keywords	Simples
Brûlée Cream	Keywords	Simples
pasta without eggs	Semantic	Média
how to make a pizza without an oven	Semantic	Média
pancake without flour and milk	Semantic	Média
healthy recipe for quick lunch	Semantic	Difícil
what can I make for a romantic dinner	Semantic	Difícil
I'm vegan. How can I make a bolognese?	Semantic	Difícil

Uma vez com as *queries* definidas, foram utilizadas nos modelos e capturamos os cinco primeiro textos do ranking, considerando os três modelos, montamos uma base de avaliação de 180 textos (alguns deles podendo ser repetidos).

Foi então atribuído uma nota de 0 a 5 para avaliação por quatro pessoas, totalizando 720 avaliações para serem utilizadas no cálculo das métricas. Para o cálculo da métrica MAP, documentos avaliados com notas de 0 a 2 foram considerados irrelevantes e de 3 a 5 relevantes.

5.2 Resultados

Os resultados seguiram o esperado, havendo uma melhora considerável do modelo semântico para os outros dois modelos. O ocorrido é consequência direta do teor das *queries* feitas, que em metade delas, é esperado que apenas o modelo semântico performe bem.

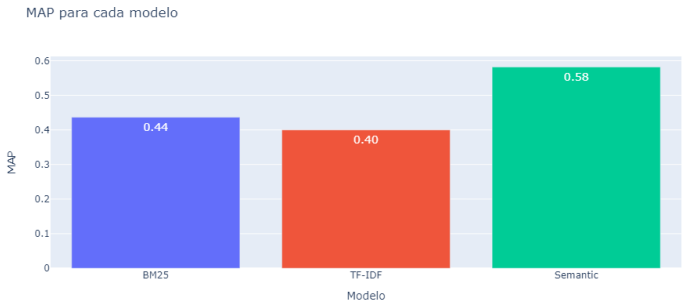

Figura 1: Avaliação dos modelos: MAP

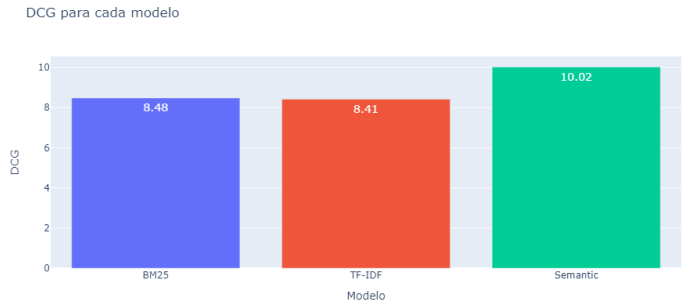

Figura 2: Avaliação dos modelos: DCG

Nos gráficos acima podemos perceber que a rede e5 possuí *scores* consideravelmente mais altos, em quanto o modelo TF-IDF fica com os piores. Sendo que o melhor desempenho do modelo é justificado pelo melhor resultado nas perguntas também semânticas.

5.3 Implementação

Para implementar um mecanismo de busca que utiliza os modelos descritos, foi utilizado a API pyvespa, com uma estrutura que já implementa as buscas com Bm25 e utilizando o transformer e5. Os códigos utilizados para o deploy do aplicativo com o pyvespa estão no repostorio Find my pasta. Para interação com o app Vespa, uma interface simples foi criada para permitir melhor visualização dos resultados das pesquisas, para isso foi utilizado o framework Flask.

Figura 3: Visualização da página

6 Dificuldades e trabalho futuro

6.1 Dificuldades

Uma dificuldade que tivemos foi lidar com o grande volume de dados envolvidos em nosso conjunto de dados. Para avaliar adequadamente os modelos, era necessário processar toda a base de dados, o que resultou em tempos de execução longos e exigiu bastante de recursos computacionais.

Outra dificuldade foi a de utilizar as Ranking Expressions do Vespa para montar um calculo para fazer o ranking dos resultados conforme o modelo TF-IDF, então para fazer uma abordagem mais rápida foi utilizado um código separado para fazer isso, o que não deixa essa parte integrada com o restante.

6.2 Trabalho futuro

Para o trabalho futuro, queremos buscar e testar soluções para aumento da relevância nos resultados e melhor entendimento de restrições/filtros em queries, pois ainda ocorreram falhas significativas relacionadas as queries semânticas onde o sentido da frase era invertido por algumas palavras.

Ainda, planejamos explorar a implementação de um chatbot para permitir uma melhor interação com os usuários. Isso proporcionaria a oportunidade de fazer perguntas adicionais ou solicitar mais informações sobre os arquivos selecionados, melhorando assim a experiência geral do usuário.

Por fim, explorar a possibilidade de utilizar redes que aprendam especificam o ranking feito pelos algoritmos de *retrieval* utilizando MAP e DCG como função de perda. Em uma pesquisa inicial [2], encontramos redes como LambdaMART e similares para a tarefa.

Referências

[1] Generating Personalized Recipes from Historical User Preferences. Bodhisattwa Prasad Majumder*, Shuyang Li*, Jianmo Ni, Julian McAuley EMNLP, 2019.

[2] From RankNet to LambdaRank to LambdaMART: An Overview. Christopher J.C. Burges. Microsoft Research Technical Report MSR-TR-2010-82.