Modelo Vetorial

Alunos: Brenno Muller, Daniel Jorge

Executando o projeto

- Baixar o arquivo compactado da pasta do drive: https://drive.google.com/drive/folders/1d9OxBOfdv2W7eALdyZkoGDWWoZYl6T_p?usp=sharing
- após baixar os arquivos, crie uma nova env em seu computador
- extraia os arquivos baixados para dentro da nova env, extraindo uma pasta chamada MV
- ative a nova env
- entre na pasta MV que foi descompactada
- baixe as bibliotecas do requirements.txt com “pip install -r requirements.txt”
- execute o busca.py na pasta MV com “python busca.py”

Sobre os arquivos

**Dentro da pasta do drive tera um arquivo comprimido .rar dentro dele tera 3 arquivos e 1 pasta que é o dataset, o arquivo modelo_vetorial é onde está a classe modelo vetorial contendo todas as funções e logica para o programa, o arquivo para ser executado é o busca.py que tem a chamada da classe e das funções do modelo vetorial

Sobre o projeto

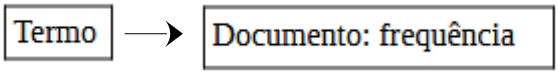
Gerando índice invertido

Para a criação do dicionário, os arquivos de texto foram pré-processados utilizando os seguintes métodos:

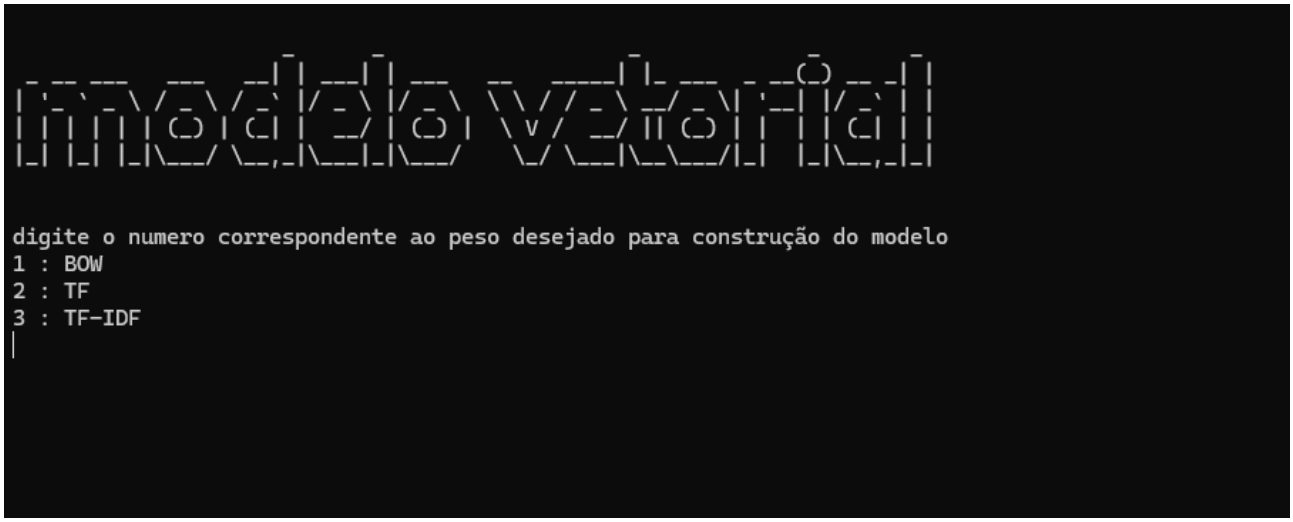
- Tokenização: Separando o arquivo de texto em sentenças, usando a biblioteca nltk
- Remover pontuação: usando esse método, junto a função **itsalnum,** nativa do python, os tokens serão apenas palavras, e nenhuma pontuação ou caractere especial separado do texto. O retorno desse método são tokens com apenas caracteres não especiais.
- Colocar os tokens em caixa baixa: Ao usar este método, garantimos que todos os tokens sigam o mesmo padrão do formato da letra. A função **lower**, nativa do python, é a ferramenta responsável por esse método. O retorno desse método são tokens em caixa baixa.
- Remover stopwords: Esse método é responsável por livrar os termos de stopwords. Essas stopwords são definidas pela biblioteca **stopwords** do NLTK. Stopwords são palavras que podem ser consideradas irrelevantes para nosso conjunto de resultados. Esse método retorna um conjunto de tokens livre de termos irrelevantes.
- Obter frequência de cada termo no documento: Método responsável por contabilizar a quantidade de vezes que o termo aparece em cada documento que ele está presente. Esse método retorna um conjunto de termos, e sua frequência em seu respectivo documento.
- Remover termos duplicados: Caso existam palavras repetidas no conjunto de tokens, esse método remove uma das duplicatas.
- Extraindo o radical dos termos: Esse método é responsável por extrair a raiz do termo, para uma maior generalização, e a redução da quantidade de termos, pois existem diversos tokens com o mesmo radical. O **PorterStemmer** do NLTK é o responsável por este método, retornando um conjunto de tokens com apenas o radical dos termos até aqui pré-processados

Gerando as matrizes com pesos

Com os índices invertidos prontos onde cada termo tem sua lista de documentos com a frequência que o termo aparece naquele documento ex:

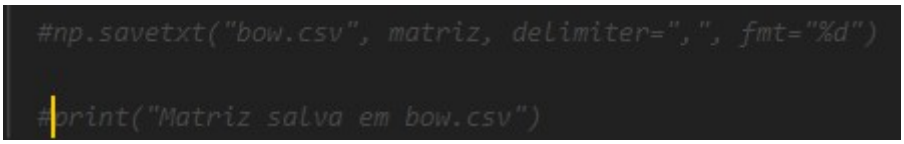


podemos agora montar as matrizes de acordo como o peso selecionado pelo usuário,



A matriz sera montada de acordo com a função de peso selecionada, bow utiliza as frequências já enviadas pelo índice invertido, TF fazemos utilizando a formula 1+log2(frequência) e TF-IDF utiliza além da frequência a quantidade de documentos que o termo aparece ou seja pelo índice invertido seria len(indice_invertido[termo]), para calcular o TF-IDF calculamos com a formula
1+log2(frequência)*(quantidade de documentos).

**Dentro do arquivo modelo_vetorial.py na função de montar matriz caso queira ver a matriz em csv descomente.



Com a matriz montada sera redirecionado para essa tela no terminal onde poderá inserir a consulta, com a string consulta podemos montar o vetor de buscar com os seguintes passos.

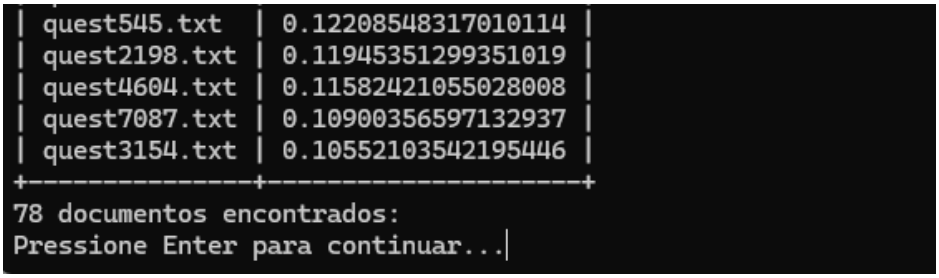
- Cria um vetor zerado com o mesmo tamanho da quantidade de linhas da matriz.
- Os termos da string enviada pelo usuário são processados pelas mesmas funções que os termos dos documentos foram processados.
- Cada posição do vetor representa um termo do vocabulário tanto no vetor documento quanto no vetor consulta, então os termos que o usuário inseriu será adicionado um valor referente a função peso escolhida na posição do termo no vetor

Agora com o vetor de consulta pronto será feito o calculo de similaridade cosseno do vetor consulta com os vetores de cada documento da matriz, ou seja as colunas da matriz.

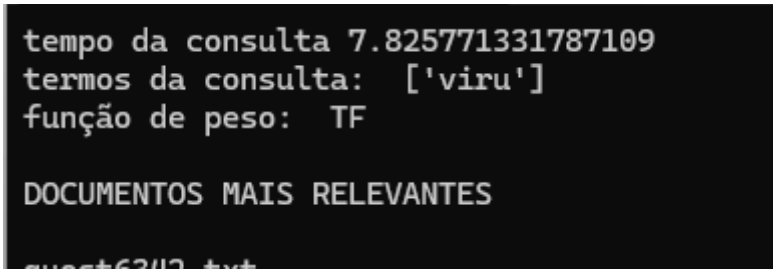
Se a função similaridade retornar um numero diferente de 0 então o documento e sua similaridade com o vetor busca será armazenado em uma nova lista, após os cálculos com todos vetores a lista é ordenada de forma decrescente e retornada de uma forma mais visual para o usuário.

Na resposta do programa o programa exibira pra o usuário

- Quantidade de documentos encontrados



- No topo scrollando para cima, vera detalhes como termos da consulta do usuário, tempo para executar a consulta, a função escolhida



- Em seguida os 3 documentos mais relevantes que tiveram um rank similaridade maior.

```
quest6342.txt

answerKey: D;
fact1: Some viruses can cause cancer.;
fact2: Cervical cancer is caused by a virus, the human papaloma virus.;
combinedfact: A virus is responsible for cervical cancer;
formatted_question: What is responsible for cervical cancer? ;
(A) lmestone ;
(B) Hemoglobin ;
(C) alleles ;
(D) A virus ;
(E) chemicals ;
(F) cigarettes ;
(G) ovaries ;
(H) smoking;
```

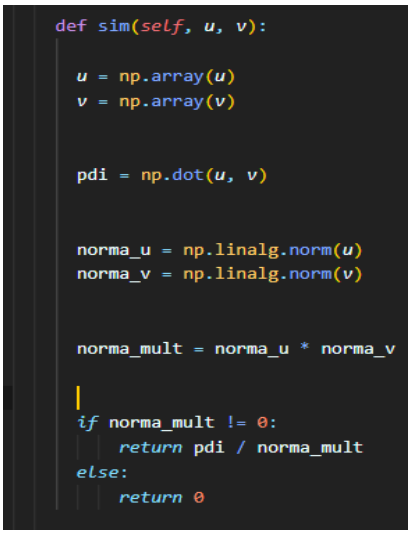
- E por último uma tabela com todos os documentos com seus ranks ordenados

TODOS DOCUMENTOS

Arquivo	Similaridade
quest6342.txt	0.459196654797090 65
quest8021.txt	0.41537563125677013
quest2857.txt	0.3741683620318 97
quest4927.txt	0.33928741365500237
quest4240.txt	0.32424624911023 74
quest5754.txt	0.32371941711963464
quest7301.txt	0.31217209660110473
quest5916.txt	0.30596593765241653
quest3042.txt	0.2937238793374718
quest41.txt	0.1889822365046136
quest6100.txt	0.17766156842794503
quest3698.txt	0.17395620999283506
quest2384.txt	0.1723061836673371
quest7547.txt	0.1723061836673371
quest3256.txt	0.17138245392452653
quest6027.txt	0.16815853950394474
quest6868.txt	0.1674076176658694
quest7157.txt	0.1665600713540171
quest3444.txt	0.1650063371841598
quest5097.txt	0.1614498725198318
quest7555.txt	0.160784938902862 88
quest2918.txt	0.15938595353868187
quest226.txt	0.15874609587406607
quest2798.txt	0.15874609587406607

Detalhes

** A função similaridade está sendo calculada com nparray pois com vetores normal do python demora muito 27s uma consulta simples, a biblioteca scipy também demora muito.



```python
def sim(self, u, v):

    u = np.array(u)
    v = np.array(v)


    pdi = np.dot(u, v)


    norma_u = np.linalg.norm(u)
    norma_v = np.linalg.norm(v)


    norma_mult = norma_u * norma_v

    if norma_mult != 0:
        return pdi / norma_mult
    else:
        return 0
```