ranqueamento

June 11, 2024

1 Busca de Índice e Consulta de documento

Autor: Davi J. Leite Santos
Versão: 0.0.3
Data: 25 de Abril de 2024
Localização: Ribeirão das Neves, Minas Gerais - Brasil

1.1 Contato

- **Endereço:** Ribeirão das Neves, Minas Gerais - Brasil
- **Email:** davi.jls@outlook.com
- **LinkedIn:** davi-j-leite-santos
- **Website:** davijls.com.br

1.2 Principais Competências

- **Cibersegurança**
- **Segurança da Informação**
- **Operações de TI**

2 Função de ranqueamento

Para implementar um sistema de ranqueamento eficiente e matematicamente robusto, uma boa escolha é o modelo de Recuperação de Informação Vetorial, que utiliza o cálculo de similaridade de cosseno entre os vetores de termo-documento e o vetor de termos da consulta. Essa abordagem é eficiente e amplamente utilizada em sistemas de recuperação de informações.

2.1 Funções Matemáticas do Modelo Vetorial

TF-IDF (Term Frequency-Inverse Document Frequency):

- TF (Term Frequency): Frequência de um termo em um documento.
- IDF (Inverse Document Frequency): Medida de quanto um termo é raro em todos os documentos.

2.2 Similaridade de Cosseno:

Medida de similaridade entre dois vetores que quantifica a similaridade entre eles.

3 Sobre o codigo

Este código fornece uma implementação de um sistema de ranking de documentos usando técnicas de recuperação de informações baseadas em modelos vetoriais, como TF-IDF (Term Frequency-Inverse Document Frequency) e similaridade de cosseno. Essa abordagem é bastante utilizada para determinar a relevância de documentos com base em uma consulta fornecida pelo usuário. Aqui está uma descrição detalhada de cada componente e funcionalidade chave do código:

3.0.1 Componentes Principais do Código

1. Configurações e Bibliotecas

- O código utiliza bibliotecas como `re`, `nltk`, `sklearn`, e `json` para realizar limpeza de texto, processamento de linguagem natural e manipulação de dados.
- Configuração de stopwords e stemmer em português para preparar os dados textualmente.

2. Funções de Limpeza e Preprocessamento de Texto

- **clean_text**: Limpa o texto removendo caracteres especiais e convertendo tudo para minúsculas. Stopwords são também removidas para reduzir ruídos nos dados.
- **apply_stemming**: Aplica redução das palavras aos seus radicais, utilizando `RSLPStemmer`.
- **preprocess_query**: Combina as funções de limpeza e stemming para preparar consultas de busca.

3. Carregamento de Dados

- **load_data**: Carrega os arquivos de vocabulário e índice geral, que contêm informações preparadas para o processamento de textos e consulta.
- **load_document**: Carrega e processa documentos de um caminho especificado, aplicando limpeza e stemming.

4. Funções para Carregamento e Preparação de Documentos

- **get_documents**: Itera sobre o índice geral para extrair documentos e seus identificadores, que são essenciais para a recuperação de informações durante a busca.

5. Cálculo do TF-IDF e Busca

- **search_query_tfidf**: Utiliza o `TfidfVectorizer` da biblioteca `sklearn` para transformar documentos e consultas em representações de TF-IDF, e então calcula a similaridade de cosseno para determinar a relevância dos documentos em relação à consulta.

6. Exibição dos Resultados

- **display_results_with_scores**: Organiza e exibe os resultados da busca, mostrando os documentos com suas pontuações. Os documentos são exibidos em ordem de relevância com o tempo de busca registrado.

3.0.2 Uso e Benefícios

Este código é ideal para sistemas onde a precisão e a relevância da informação recuperada são cruciais, como em sistemas de busca internos de empresas ou aplicações acadêmicas onde documentos precisam ser recuperados com base em seu conteúdo semântico. A capacidade de processar e indexar grandes conjuntos de dados de texto e realizar buscas rápidas e eficientes são pontos-chave deste sistema.

3.0.3 Exemplo de Uso

No exemplo de código, uma consulta é pré-processada e uma busca TF-IDF é realizada nos documentos carregados. Os resultados são então exibidos com pontuações de relevância, demonstrando a eficácia do sistema em encontrar e classificar documentos com base na consulta do usuário.

4 ————————————————————————————————————-

[ ]:
```
import re
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from rank_bm25 import BM25Okapi
from nltk.corpus import stopwords
from nltk.stem import RSLPStemmer
import pandas as pd
import json
import time

nltk.download("stopwords")
nltk.download("rslp")
nltk.download("punkt")

stop_words = set(stopwords.words("portuguese"))
stemmer = RSLPStemmer()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\davim\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package rslp to
[nltk_data]     C:\Users\davim\AppData\Roaming\nltk_data…
[nltk_data]   Package rslp is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\davim\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

4.0.1 Funções de Limpeza e Preprocessamento

[ ]:
```
def clean_text(text):
    text = re.sub(r"[^\w\s]", " ", text)  # Remove special characters
    text = text.lower()  # Convert to lowercase
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]  # Remove␣
 ↪stopwords
    return " ".join(tokens)


def apply_stemming(text):
    tokens = nltk.word_tokenize(text, language="portuguese")
    stemmed_tokens = [stemmer.stem(token) for token in tokens]
    return " ".join(stemmed_tokens)


def preprocess_query(query):
    cleaned_query = clean_text(query)
    stemmed_query = apply_stemming(cleaned_query)
    return stemmed_query
```

[ ]:
```
import os


def load_document(file_path):
    with open(file_path, "r", encoding="utf-8") as file:
        text = file.read()
    cleaned_text = clean_text(text)
    stemmed_text = apply_stemming(cleaned_text)
    return stemmed_text
```

4.0.2 Carregar Dados e Preparar Documentos

[ ]:
```
def load_data(vocab_geral_file, index_geral_file):
    with open(vocab_geral_file, "r", encoding="utf-8") as f:
        vocab_geral = json.load(f)
    with open(index_geral_file, "r", encoding="utf-8") as f:
        index_geral = json.load(f)
    return vocab_geral, index_geral


vocab_geral_file = "cleaned_vocab_geral.json"
index_geral_file = "cleaned_index_geral.json"

vocab_geral, index_geral = load_data(vocab_geral_file, index_geral_file)
```

[ ]:
```python
from tqdm import tqdm


def get_documents(index_geral):
    documents = []
    doc_ids = []
    for word_id, data in tqdm(index_geral.items(), desc="Loading documents"):
        for doc_id in data["doc_info"]:
            doc_ids.append(doc_id)
            documents.append(data["file_names"][0])
    return documents, doc_ids


documents, doc_ids = get_documents(index_geral)
```

```
Loading documents: 100%|      | 6781155/6781155 [00:12<00:00, 526978.31it/s]
```

4.0.3 Funções para o calculo do TF-IDF

[ ]:
```python
def search_query_tfidf(query, documents):
    start_time = time.time()

    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(documents)
    query_vector = vectorizer.transform([query])

    scores = (tfidf_matrix * query_vector.T).toarray().flatten()

    # Adicionar barra de progresso
    sorted_scores_idx = tqdm(
        scores.argsort()[::-1], desc="Calculating Scores", unit="doc"
    )

    sorted_scores = [
        (documents[i], scores[i]) for i in sorted_scores_idx if scores[i] > 0
    ]

    end_time = time.time()
    search_time = end_time - start_time

    return list(set(sorted_scores)), search_time
```

5 Usando todas as funções

[ ]:
```
def display_results_with_scores(sorted_scores, search_time):
    print(f"{'Site':<200}{'Score':<110}")
    print("=" * 210)
    for file_name, score in sorted_scores:
        print(f"{file_name:<200}{score:<110.4f}")

    print(f"\nSearch completed in {search_time:.2f} seconds")
```

[ ]:
```
# Exemplo de uso
query = "buceta cu vagina anal sexo toy pussy pinto dick desgraça tnc fuder␣
 ↪sefuder fdp bunda"
```

[ ]:
```
# TF-IDF Search
preprocessed_query = preprocess_query(query)
sorted_scores_tfidf, search_time_tfidf = search_query_tfidf(
    preprocessed_query, documents
)

print("TF-IDF Results:")
display_results_with_scores(sorted_scores_tfidf, search_time_tfidf)
```

```
Calculating Scores: 100%|     | 9048738/9048738 [00:05<00:00,
1553536.13doc/s]

TF-IDF Results:
Site
Score
================================================================================
================================================================================
================================================
www_metropoles_com]celebridades]deborah-secco-admite-que-ama-sexo-anal-e-nunca-
cuspiu-gozo-veja-video.txt
0.2045
www_metropoles_com]colunas]pouca-vergonha]bifobia-e-falocentrismo-o-que-
envelheceu-mal-em-sex-and-the-city.txt
0.2454
www_metropoles_com]colunas]pouca-vergonha]beijo-grego-44-das-pessoas-ja-gozaram-
com-o-carinho-anal-veja-dicas.txt
0.2063

Search completed in 107.22 seconds
```

[ ]: