Limpeza_e_transformacao

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 sobre o codigo

Este código é uma extensão avançada para a criação, manipulação e pesquisa de um índice invertido, incluindo recursos de pré-processamento de texto, como remoção de stopwords, normalização de caracteres, tokenização e aplicação de stemming em português. Ele está integrado com operações de leitura e escrita JSON, e a implementação é orientada para uma análise de texto eficiente e recuperação de documentos.

2.0.1 Componentes Principais do Código

1. Preparação e Dependências

- Importações de módulos essenciais como `json`, `re` (para expressões regulares), `nltk` (biblioteca de processamento de linguagem natural), `os` (para manipulação de sistema de arquivos) e `time` (para medição de desempenho).
- Download de recursos do NLTK necessários para o processamento de texto, incluindo stopwords em português e o stemmer `RSLPStemmer`.

2. Limpe leitura e escrita de dados JSON

- **load_json** e **save_json**: Funções para carregar e salvar dados em formato JSON, garantindo que a codificação seja adequada para suportar caracteres especiais em UTF-8.

3. Limpeza e Análise Léxica

- **clean_text**: Remove caracteres especiais e stopwords, além de converter o texto para minúsculas, preparando-o para uma análise mais eficaz.
- **lexical_analysis_and_stemming**: Aplica tokenização, limpe you palavras desnecessárias e realiza stemming, transformando palavras em suas formas raiz para reduzir a complexidade do índice.

4. Transformação e Indexação

- **apply_lexical_analysis_and_stemming**: Transforma vocabulários e índices existentes com as operações de análise léxica e stemming para criar versões mais compactas e eficientes.
- **process_data_with_lexical_analysis_and_stemming**: Processa os dados lendo o índice geral e o vocabulário, aplicando as transformações e salvando os novos índices.

5. Funcionalidades de Busca

- **search_query**: Realiza uma busca efetiva utilizando o índice e vocabulário processados para encontrar documentos que contenham termos específicos, mostrando como palavras processadas podem ser rapidamente mapeadas aos documentos originais.

6. Avaliação de Desempenho

- **measure_performance**: Avalia o desempenho do sistema medindo o tempo de resposta para consultas e o tamanho dos arquivos de índice e vocabulário, permitindo uma avaliação objetiva da eficiência das otimizações implementadas.

2.0.2 Implementação e Uso

O código é cuidadosamente projetado para ser robusto e eficiente, utilizando práticas de codificação que garantem a integridade dos dados e a facilidade de manutenção. Ele oferece uma solução completa para a manipulação de grandes conjuntos de dados de texto em português, ideal para aplicações que requerem indexação e recuperação de informações textuais rápidas e precisas.

Este sistema é especialmente útil em contextos onde a precisão lexical e a rapidez na recuperação de informações são cruciais, como em motores de busca ou sistemas de gerenciamento de documentos digitais.

3 ————————————————————————————————————-

[ ]:
```
import json
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import RSLPStemmer
```

```
import os
import time

# Baixar recursos do NLTK
nltk.download("rslp")
nltk.download("punkt")
nltk.download("stopwords")

# Inicializar o Stemmer para português
stemmer = RSLPStemmer()
stop_words = set(stopwords.words("portuguese"))
```

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

4 Limpeza dos dados

Nessa etapa serão excluidas as stopworlds e transformadas os dados em UTF8 e retiradas os caracteres especias

[ ]:
```
def load_json(file_path):
    with open(file_path, "r", encoding="utf-8") as f:
        return json.load(f)


def save_json(data, file_path):
    with open(file_path, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=4)


def clean_text(text):
    text = re.sub(r"[^\w\s]", " ", text)  # Remove caracteres especiais
    text = text.lower()  # Converte para minúsculas
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]  # Remove␣
 ↪stopwords
    return " ".join(tokens)


def lexical_analysis_and_stemming(text):
```

```
    # Limpa o texto
    cleaned_text = clean_text(text)
    # Tokeniza o texto em palavras
    tokens = nltk.word_tokenize(cleaned_text, language="portuguese")
    # Remove stopwords e aplica stemming em cada token
    stemmed_tokens = [
        stemmer.stem(token) for token in tokens if token not in stop_words
    ]
    return stemmed_tokens


def apply_lexical_analysis_and_stemming(vocab, index):
    stemmed_vocab = {}
    stemmed_index = {}

    for word, word_id in vocab.items():
        # Análise léxica e stemming
        stemmed_word = " ".join(lexical_analysis_and_stemming(word))
        if stemmed_word not in stemmed_vocab:
            stemmed_vocab[stemmed_word] = word_id
        stemmed_word_id = stemmed_vocab[stemmed_word]

        if str(word_id) in index:
            for doc_id, positions in index[str(word_id)].items():
                if stemmed_word_id not in stemmed_index:
                    stemmed_index[stemmed_word_id] = {}
                if doc_id not in stemmed_index[stemmed_word_id]:
                    stemmed_index[stemmed_word_id][doc_id] = positions
                else:
                    if isinstance(stemmed_index[stemmed_word_id][doc_id], list):
                        stemmed_index[stemmed_word_id][doc_id].extend(positions)
                    else:
                        stemmed_index[stemmed_word_id][doc_id] = list(positions)

    return stemmed_vocab, stemmed_index


# Função principal para carregar, processar e salvar os dados com análise␣
 ↪léxica e stemming
def process_data_with_lexical_analysis_and_stemming(
    index_geral_file, vocab_geral_file, cleaned_index_file, cleaned_vocab_file
):
    # Carregar os arquivos
    index_geral = load_json(index_geral_file)
    vocab_geral = load_json(vocab_geral_file)

    # Aplicar análise léxica e stemming em vocabulário e índice
```

```
    stemmed_vocab, stemmed_index = apply_lexical_analysis_and_stemming(
        vocab_geral, index_geral
    )

    # Salvar os dados processados
    save_json(stemmed_index, cleaned_index_file)
    save_json(stemmed_vocab, cleaned_vocab_file)


# Função para realizar a pesquisa
def search_query(query, vocab_geral_file, index_geral_file, sites_dir):
    # Carregar vocabulário e índice geral
    vocab_geral = load_json(vocab_geral_file)
    index_geral = load_json(index_geral_file)

    # Processar a query
    query_terms = lexical_analysis_and_stemming(query.lower())

    # Obter IDs das palavras da query
    word_ids = []
    for term in query_terms:
        if term in vocab_geral:
            word_ids.append(vocab_geral[term])

    # Recuperar documentos correspondentes
    document_ids = set()
    for word_id in word_ids:
        if str(word_id) in index_geral:
            document_ids.update(index_geral[str(word_id)].keys())

    # Converter IDs dos documentos para nomes dos arquivos
    files = sorted(os.listdir(sites_dir))
    result_files = []
    for doc_id in document_ids:
        try:
            result_files.append(files[int(doc_id)])
        except (ValueError, IndexError):
            continue

    return result_files
```

[ ]:
```
# Defina os caminhos dos arquivos
index_geral_file = "index_geral.json"
vocab_geral_file = "vocab_geral.json"
cleaned_index_file = "cleaned_index_geral.json"
cleaned_vocab_file = "cleaned_vocab_geral.json"
```

[ ]:
```
# Processar os dados com análise léxica e stemming
process_data_with_lexical_analysis_and_stemming(
    index_geral_file, vocab_geral_file, cleaned_index_file, cleaned_vocab_file
)
```

[ ]:
```
# Exemplos de uso:
sites_dir = "../Motor_de_busca-WebCrawler/sites_visitados/"
query = "noticia"
```

[ ]:
```
result_files = search_query(query, cleaned_vocab_file, cleaned_index_file,␣
 ↪sites_dir)
```

[ ]:
```
for result in result_files:
    print(result)
```

5 Verificação

Instalando a biblioteca para fazer a analise Léxica e stemming

[ ]:
```
# Função para carregar o índice e vocabulário
def load_data(index_file, vocab_file):
    with open(index_file, "r", encoding="utf-8") as f:
        index = json.load(f)
    with open(vocab_file, "r", encoding="utf-8") as f:
        vocab = json.load(f)
    return index, vocab


# Função para buscar termos no índice
def search_term(index, vocab, term):
    cleaned_term = clean_text(term)
    stemmed_term = stemmer.stem(cleaned_term)
    if stemmed_term in vocab:
        word_id = vocab[stemmed_term]
        if str(word_id) in index:
            return index[str(word_id)]
    return {}


# Função para medir o tempo de pesquisa e o espaço de indexação
def measure_performance(index_file, vocab_file, search_terms, output_file):
    index, vocab = load_data(index_file, vocab_file)

    # Medir tempo de pesquisa
    search_times = []
    for term in search_terms:
        start_time = time.time()
```

```
        search_term(index, vocab, term)
        end_time = time.time()
        search_times.append(end_time - start_time)

    avg_search_time = sum(search_times) / len(search_times)

    # Medir tamanho dos arquivos
    index_size = os.path.getsize(index_file)
    vocab_size = os.path.getsize(vocab_file)
    total_size = index_size + vocab_size

    # Escrever resultados em um arquivo de texto
    with open(output_file, "w", encoding="utf-8") as f:
        f.write(f"Tempo médio de pesquisa: {avg_search_time:.6f} segundos\n")
        f.write(f"Tamanho do arquivo de índice: {index_size} bytes\n")
        f.write(f"Tamanho do arquivo de vocabulário: {vocab_size} bytes\n")
        f.write(f"Tamanho total dos arquivos: {total_size} bytes\n")
```

[ ]:
```
# Definir os caminhos dos novos arquivos
cleaned_index_geral_file = "cleaned_index_geral.json"
cleaned_vocab_geral_file = "cleaned_vocab_geral.json"
output_file = "performance_metrics.txt"

# Definir termos de pesquisa para teste
search_terms = ["educação", "linguagem", "noticia", "ideia", "politica"]
```

[ ]:
```
# Medir desempenho e salvar resultados
measure_performance(
    cleaned_index_geral_file, cleaned_vocab_geral_file, search_terms,␣
 ↪output_file
)
```

[ ]:
```

```