UNIVERSIDADE FEDERAL DE SERGIPE

DEPARTAMENTO DE COMPUTAÇÃO

UNALDO SANTOS VASCONCELOS NETO

PROCESSAMENTO DE LINGUAGEM NATURAL - COMP0428

Relatório Técnico: Atividade 1 - Criação de Embeddings

São Cristóvão
2024

Sumário

1. Conjunto de Dados: Dinheiro em Circulação

O conjunto de dados selecionado para este projeto contém informações detalhadas sobre o dinheiro em circulação no Brasil, incluindo cédulas e moedas. As informações são disponibilizadas diariamente e estão categorizadas por espécie (cédula ou moeda), família (categoria) e denominação do Real (símbolos: R$, BRL). Este conjunto de dados é uma fonte rica para análise, pois oferece uma visão abrangente das variações nas quantidades de dinheiro em circulação ao longo do tempo.

O primeiro passo para a utilização deste conjunto de dados foi o pré-processamento, que incluiu a leitura dos dados em formato JSON, a seleção das colunas relevantes, e a criação de sentenças descritivas para cada registro. O objetivo era transformar os dados estruturados em texto, facilitando assim a criação de embeddings utilizando modelos de "sentence-transformers".

```python
import json
import pandas as pd

LOCAL_JSON_PATH = "inform_10000.json"

def fetch_data():
    with open(LOCAL_JSON_PATH, 'r', encoding='utf-8') as f:
        data = json.load(f)
    df = pd.DataFrame(data['value'])
    columns = ['Data', 'Categoria', 'Quantidade', 'Denominacao']
    df = df[columns]
    sentences = df.apply(create_sentences, axis=1).tolist()
    return sentences

def create_sentences(row):
    return f" Categoria: {row['Categoria']}, Data: {row['Data']}, Denominacao: {row['Denominacao']}, Quantidade: {row['Quantidade']}"
```

1. **Carregamento dos Dados**: O arquivo JSON contendo os dados foi carregado utilizando a biblioteca json. A função fetch_data abre o arquivo e carrega seu conteúdo na variável data.
2. **Criação do DataFrame**: Utilizando a biblioteca pandas, os dados foram convertidos para um DataFrame para facilitar a manipulação. Foram selecionadas as colunas Data, Categoria, Quantidade e Denominacao como as mais relevantes para o projeto.
3. **Criação das Sentenças**: A função create_sentences foi utilizada para transformar cada linha do DataFrame em uma sentença descritiva. Essa transformação é essencial para a criação de embeddings, pois modelos de "sentence-transformers" trabalham com texto natural. Cada sentença é composta pela categoria, data, denominação e quantidade, formando uma descrição textual de cada registro.

O resultado deste pré-processamento foi uma lista de sentenças que representam cada registro do conjunto de dados. Essas sentenças foram posteriormente utilizadas para gerar embeddings, permitindo análises e consultas semânticas mais eficientes.

2. Escolha do Modelo

Para a criação de embeddings das sentenças geradas a partir do conjunto de dados, foi utilizado o modelo "intfloat/multilingual-e5-small", disponível na plataforma Hugging Face. Este modelo é parte da família "E5" e é projetado para suportar múltiplos idiomas, incluindo o português, o que o torna ideal para o contexto deste projeto.

O modelo "intfloat/multilingual-e5-small" é uma variante compacta e eficiente, adequada para tarefas de processamento de linguagem natural em múltiplos idiomas, este modelo gera embeddings com 384 dimensões..

3. Utilizar o Modelo

O código a seguir foi utilizado para carregar o modelo e gerar embeddings para as sentenças derivadas do conjunto de dados:

```
from sentence_transformers import SentenceTransformer

def get_embeddings(sentences):
    model = SentenceTransformer('intfloat/multilingual-e5-small')
    return model.encode(sentences, convert_to_tensor=False)
```

1. **Carregamento do Modelo**: A função ‘get_embeddings’ inicia carregando o modelo "intfloat/multilingual-e5-small" utilizando a biblioteca ‘SentenceTransformer’, A biblioteca sentence-transformers é uma ferramenta poderosa para a geração de embeddings de sentenças e textos, é facilmente instalada usando o gerenciador de pacotes pip, o comando para instalação é: **pip install sentence-transformers**. Este modelo é especializado em transformar sentenças em embeddings, que são representações vetoriais de textos.
2. **Geração de Embeddings**: Após carregar o modelo, a função recebe uma lista de sentenças e as converte em embeddings.

Os embeddings gerados são representações numéricas que capturam as características semânticas das sentenças originais. Essas representações são cruciais para tarefas subsequentes.

4. Instalar um banco vetorial

Para o armazenamento e persistência dos embeddings gerados, foi utilizado o banco de dados vetorial Chroma. O Chroma é uma solução eficiente para armazenar e consultar grandes volumes de embeddings, facilitando a realização de operações baseadas em similaridade semântica.

O Chroma pode ser instalado utilizando o gerenciador de pacotes pip. O comando para instalação é: **pip install chromadb**

O código utilizado para inicializar o banco de dados Chroma, adicionar embeddings e garantir a persistência dos dados.:

```python
from chromadb import PersistentClient

def init_chromadb(db_path, collection_name):
    client = PersistentClient(path=db_path)
    try:
        print("Concectando ao banco...")
        collection = client.get_collection(name=collection_name)
    except:
        print("Criando banco...")
        collection = client.create_collection(name=collection_name)
    return collection

def add_embeddings_to_chromadb(collection, sentences, embeddings):
    for i, (sentence, embedding) in enumerate(zip(sentences, embeddings)):
        embedding_list = embedding.tolist()
        collection.add(documents=[sentence], embeddings=[embedding_list], ids=[f"id_{i}"])
```

1. **PersistentClient**: O PersistentClient é utilizado para criar ou conectar a um banco de dados Chroma persistente localizado no caminho especificado por db_path.
2. **get_collection** e **create_collection**: O código tenta recuperar uma coleção existente com o nome fornecido. Se a coleção não existir, ela é criada.
3. Adição de Documentos e Embeddings: O método add é utilizado para adicionar documentos e seus embeddings ao banco de dados. Cada embedding é convertido para uma lista e associado a um ID único.
4. IDs: IDs únicos são gerados para cada entrada para facilitar a recuperação e a gestão dos dados.

Os dados são persistidos no disco pelo PersistentClient, garantindo que os embeddings e documentos adicionados ao banco de dados sejam salvos e possam ser recuperados em execuções futuras.

5. Consultas, Similaridade e Cénario

O banco de dados vetorial Chroma é utilizado para consultas com base em similaridade semântica. A similaridade semântica permite encontrar documentos que são semanticamente semelhantes a uma consulta, o que é mais eficaz do que consultas tradicionais baseadas em palavras-chave

```python
from data_preprocessing import fetch_data
from embedding_utils import get_embeddings
from db_utils import init_chromadb, add_embeddings_to_chromadb

def main(reload_data=False):

    db_path = "/path/to/save/to"
    collection_name = "banco"

    collection = init_chromadb(db_path, collection_name)

    if reload_data:
        print("Carregando dados...")
        sentences = fetch_data()
        print("Gerando embeddings")
        embeddings = get_embeddings(sentences)
        add_embeddings_to_chromadb(collection, sentences, embeddings)
    else:
        print("A coleção já contém dados recentes. Você pode pular a etapa de carregamento de dados.")

    query_sentence = "Futebol"
    query_embedding_list = get_embeddings([query_sentence])[0]
    query_embedding = query_embedding_list.squeeze().tolist()

    results = collection.query(
        query_embeddings=[query_embedding],
        n_results=3
    )


    if results and 'documents' and 'distances' in results and results['distances'] and results['documents']:
        for i, (document, distance) in enumerate(zip(results['documents'][0], results["distances"][0])):
            print(f"|{i+1}°| Distância: {distance:.4f} |{document}")
    else:
        print("Nenhum documento encontrado nos resultados.")

if __name__ == "__main__":
    main(reload_data=False)
```

.

1. **Inicialização do Banco de Dados:** O banco de dados Chroma é inicializado ou carregado, dependendo se reload_data é True ou False.

2. **Geração e Armazenamento de Embeddings:** Se os dados são recarregados, o código carrega dados, gera embeddings e os adiciona ao banco de dados.

3. **Geração do Embedding da Consulta:**O embedding para a sentença de consulta "Futebol" é gerado utilizando o modelo sentence-transformers.

4. **Execução da Consulta:** A consulta é feita ao banco de dados para encontrar os documentos mais semelhantes. A função query do Chroma retorna os documentos mais próximos e suas distâncias em relação ao embedding da consulta.

5. **Exibição dos Resultados:** Os documentos e suas distâncias são exibidos, mostrando os resultados mais relevantes da consulta.

Cenário

Você está realizando uma pesquisa sobre moedas comemorativas, especificamente sobre a moeda comemorativa do futebol lançada durante as Olimpíadas. O objetivo é encontrar todos os registros que mencionam e descrevem essa moeda específica em um banco de dados que contém informações sobre diversas moedas em circulação.

Uso da Similaridade Semântica

Em um banco de dados tradicional, uma consulta típica pode envolver palavras-chave específicas e informações precisas. No entanto, isso pode ser limitado para capturar todos os documentos relevantes, especialmente se a terminologia utilizada variar.

Com Similaridade Semântica:

- **Consulta Baseada em Contexto**: A similaridade semântica permite que a instituição busque documentos que são semanticamente semelhantes a uma consulta. Mesmo que um documento não contenha exatamente essas palavras-chave, ele pode ser encontrado se discutir conceitos relacionados.
- **Exemplos de Consultas**: A consulta pode ser algo como "Moeda Futebol" ou "Cédulas de 20 reais", e o banco de dados retornará documentos que tratam de temas semelhantes, proporcionando uma visão mais abrangente.