SEMANTIC BASED SEARCH ENGINE

PROJECT REPORT

TEAM : T211160

1. Setting up the environment

The initial section installs necessary libraries and sets up the environment for data preprocessing and analysis. It includes installations of packages such as **sentence-transformers**, **tqdm**, **beautifulsoup4**, **nltk**, and **torch**. Additionally, it imports essential modules and defines necessary configurations.

---

2. Part 1: Ingesting Documents

This part focuses on reading and decoding an SQLite database containing subtitle documents. It defines a function **read_and_decode_database** to read the database, decode binary data, and preprocess the documents. The documents are stored in a DataFrame for further analysis and processing.

---

3. Selecting the 30% of the given data randomly

Here, a 30% random sample of the ingested subtitle documents is selected for subsequent analysis and experimentation. The sampled data is stored in a DataFrame for further processing.

---

4. Applying the appropriate cleaning steps on subtitle documents

This section defines a function **clean_text** to preprocess the subtitle documents. It removes timestamps, line numbers, HTML tags, special characters, punctuation, and stopwords from the documents. The cleaned content is stored in a new column in the DataFrame.

---

5. Experiment with the following to generate text vectors of subtitle documents

Generating Text Embeddings with the TfidfVectorizer for keyword-based search engine experimentation

The code utilizes the **TfidfVectorizer** from scikit-learn to transform the cleaned subtitle text data into TF-IDF vectors. These vectors represent the importance of words in the context of the entire document collection. The TF-IDF vectors, along with feature names, are stored for further analysis.

Generating Text Embeddings with the multi-qa-MiniLM-L6-cos-v1 model

The code utilizes the **multi-qa-MiniLM-L6-cos-v1** model from Sentence Transformers to generate sentence embeddings for the subtitle documents. These embeddings map sentences and paragraphs to a 384-dimensional dense vector space, facilitating semantic search. The generated embeddings are stored for further analysis.

---

6. Saving the generated embeddings to numpy array and PyTorch file

The generated embeddings from both the TfidfVectorizer and the Sentence Transformers model are saved to files for future use. These files include a NumPy array file (**subtitle_embeddings.npy**) and a PyTorch file (**subtitle_embeddings.pt**).

---

7. Storing the generated embeddings in a ChromaDB database

This section sets up and populates a ChromaDB collection named **subtitle_multi-qa-MiniLM-L6-cos-v1_bert_embeddings_final** with the generated subtitle embeddings. The collection is configured to use cosine similarity for embedding space and includes metadata for each document.

---

8. Part 2: Retrieving Documents

This part defines functions for preprocessing user search queries and performing document retrieval based on semantic similarity. The **preprocess_text** function cleans and preprocesses the query, while the **search_query** function generates query embeddings, calculates cosine similarity scores, and returns the most relevant documents.

Conclusion

The provided code demonstrates a comprehensive workflow for ingesting, preprocessing, embedding, and retrieving subtitle documents. It leverages various techniques and libraries for text processing, embedding generation, and semantic search. The generated embeddings are stored in a database for efficient retrieval and analysis.

This report summarizes the functionality and workflow of the provided code snippets. It outlines the key steps involved in ingesting, preprocessing, embedding, and retrieving subtitle documents, along with explanations for each part of the process.