Enhancing Search Engine Relevance for Video Subtitles (Cloning Shazam)

Background:
In the fast-evolving landscape of digital content, effective search engines play a pivotal role in connecting users with relevant information. For Google, providing a seamless and accurate search experience is paramount. This project focuses on improving the search relevance for video subtitles, enhancing the accessibility of video content.

Objective:
Develop an advanced search engine algorithm that efficiently retrieves subtitles based on user queries, with a specific emphasis on subtitle content. The primary goal is to leverage natural language processing and machine learning techniques to enhance the relevance and accuracy of search results.

Keyword based vs Semantic Search Engines:

- **Keyword Based Search Engine:** These search engines rely heavily on exact keyword matches between the user query and the indexed documents.
- **Semantic Search Engines:** Semantic search engines go beyond simple keyword matching to understand the meaning and context of user queries and documents.
- **Comparison:** While keyword-based search engines focus primarily on matching exact keywords in documents, semantic-based search engines aim to understand the deeper meaning and context of user queries to deliver more relevant and meaningful search results.

Core Logic:
To compare a user query against a video subtitle document, the core logic involves three key steps:

1. **Preprocessing of data:**
    a. If you have limited compute resources, you can take a random 10 to 30% of the data.
    b. Clean: A possible cleaning step can be to remove time-stamps (Note: Cleaning the Text data is crucial before vectorization)
    c. Vectorize the given Subtitle Documents
2. Take the user query and vectorize the User Query.
3. Cosine Similarity Calculation:
    a. Compute the cosine similarity between the vector of the documents and the vector of the user query.
    b. This similarity score determines the relevance of the documents to the user's query.
4. Return the most similar documents

Data:
Click here to download the subtitle data.

Step by Step Process

Part 1: Ingesting Documents

1. Read the given data.
    a. Observe that the given data is a database file.
    b. Go through the README.txt to understand what is there inside the database.
    c. Take care of decoding the files inside the database.
    d. If you have limited compute resources, you can take a random 30% of the data.
2. Apply appropriate cleaning steps on subtitle documents (whatever is required)
3. Experiment with the following to generate text vectors of subtitle documents:
    a. BOW / TFIDF to generate sparse vector representations. Note that this will only help you to build a **Keyword Based Search Engine**.
    b. BERT based "SentenceTransformers" to generate embeddings which encode semantic information. This can help us build a **Semantic Search Engine**.
4. **(Must Implement)** A very important step to improve the performance: Document Chunker.
    a. Consider the challenge of embedding large documents: Information Loss.
    b. It is often not practical to embed an entire document as a single vector, particularly when dealing with long documents.
    c. Solution: Divide a large document into smaller, more manageable chunks for embedding.
    d. Another Problem: Let's say we set the token window to be 500, then we'd expect each chunk to be just below 500 tokens. One common concern of this method is that we might accidentally cut off some important text between chunks, splitting up the context. To mitigate this, we can set overlapping windows with a specified amount of tokens to overlap so we have tokens shared between chunks.
5. Store embeddings in a **ChromaDB** database.

Part 2: Retrieving Documents

1. Take the user's search query in **audio format**. This audio must be a 2 minutes recording of a TV series or Movie which is a part of the database.
2. Preprocess the query (if required).
3. Create query embedding.
4. Using cosine distance, calculate the similarity score between embeddings of documents and user search query embedding.
5. These cosine similarity scores will help in returning the most relevant candidate documents as per user's search query.