INNOMATICS® RESEARCH LABS

INNOVATION. AUTOMATION. ANALYTICS

PROJECT ON

Enhancing Search Engine Relevance for Video Subtitles

Prepared By
Team_ID - T211172
Lingerkar Rithikha - IN1240029
Heli Pavashiya - IN1240112

Problem Statement:

In the rapidly evolving digital landscape, effective search engines are essential for connecting users with relevant information. However, existing search engines often struggle to provide accurate and meaningful results, especially for video subtitles, hindering the accessibility of video content. This project aims to address this challenge by developing an advanced search engine algorithm tailored specifically for video subtitles. The objective is to leverage natural language processing and machine learning techniques to enhance the relevance and accuracy of search results, thereby improving the accessibility of video content for users.

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.

INNOMATICS RESEARCH LABS

Introduction

Our Main ideology is to build an Semantic Search Engine so to proceed with it we must have some basics knowledge on the approach we use .

A Semantic Search Engine goes beyond traditional keyword matching to understand the context and intent behind a user's query. It utilizes natural language processing (NLP) and machine learning algorithms to comprehend the meaning of words and phrases within documents and queries. By analyzing the relationships between words and concepts, it can deliver more accurate and relevant search results. This approach enables the search engine to understand user intent, improve search accuracy, and provide more nuanced results, making it ideal for complex queries or domains where precise understanding is crucial, such as scientific research, legal documents, or technical manuals. Overall, a Semantic Search Engine enhances the user experience by delivering more contextually relevant information.

BERT based “Sentence Transformers” to generate embeddings which encode semantic information. BERT is the one of the most popular modern LLM model for Semantic problems .

BERT, or Bidirectional Encoder Representations from Transformers, is a state-of-the-art natural language processing (NLP) model developed by Google. It's designed to understand the context of words in a sentence by leveraging the bidirectional nature of Transformer models. Unlike previous NLP models, BERT can capture the meaning of a word by considering the words that come before and after it in a sentence. This allows it to grasp nuances and complexities of language more effectively, resulting in better performance on various NLP tasks such as sentiment analysis, text classification, named entity recognition, and question answering. BERT has significantly advanced the field of NLP and is widely used in various applications and frameworks due to its effectiveness and versatility.

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 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

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 "Sentence Transformers" 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.
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.

Approach

Our approach to solve the problem includes 4 major steps :

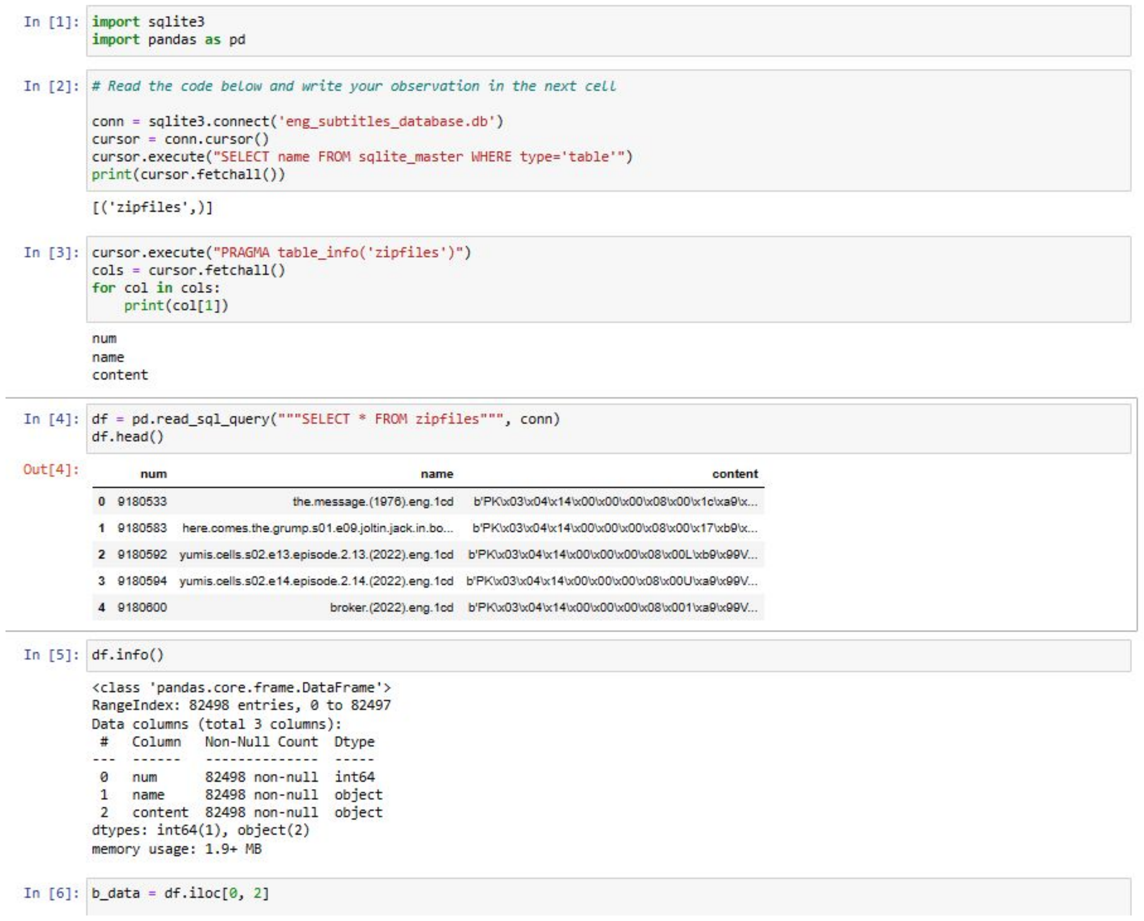
1. Data Cleaning:

The dataset is been downloaded . Then try to load the data by importing the necessary libraries need . As the data was in the form of database format which had zipfiles as the table-name . IT contain 82498 rows and 3 column.

The data was encoded in latin-1 model so we needed to decode the data for further process.

The cleaning steps were included for preprocessing the data where we clean the content columns. By removing time-stamp , special characters and any regex expression .

After the cleaning the data we have selected 30% of the data for our sentimental search approach . It due to the computation resources , we opted with 30% of the data . The 30% of the data is then saved to csv file format for easy processing of our task.



```
In [1]: import sqlite3
        import pandas as pd

In [2]: # Read the code below and write your observation in the next cell

        conn = sqlite3.connect('eng_subtitles_database.db')
        cursor = conn.cursor()
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
        print(cursor.fetchall())

        [('zipfiles',)]

In [3]: cursor.execute("PRAGMA table_info('zipfiles')")
        cols = cursor.fetchall()
        for col in cols:
            print(col[1])

        num
        name
        content

In [4]: df = pd.read_sql_query("""SELECT * FROM zipfiles""", conn)
        df.head()

Out[4]:
```

	num	name	content
0	9180533	the.message.(1976).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00\x1c\xa9\x...
1	9180583	here.comes.the.grump.s01.e09.joltin.jack.in.bo...	b'PK\x03\x04\x14\x00\x00\x00\x08\x00\x17\xb9\x...
2	9180592	yumis.cells.s02.e13.episode.2.13.(2022).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00L\xb9\x99V...
3	9180594	yumis.cells.s02.e14.episode.2.14.(2022).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00U\xa9\x99V...
4	9180600	broker.(2022).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x001\xa9\x99V...

```
In [5]: df.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 82498 entries, 0 to 82497
        Data columns (total 3 columns):
         #   Column   Non-Null Count  Dtype
        ---  ------   --------------  -----
         0   num      82498 non-null  int64
         1   name     82498 non-null  object
         2   content  82498 non-null  object
        dtypes: int64(1), object(2)
        memory usage: 1.9+ MB

In [6]: b_data = df.iloc[0, 2]
```

2. Data Preprocessing :

In this steps we experiment with various techniques like Bag-of-Words (BOW), Term Frequency-Inverse Document Frequency (TFIDF), and BERT-based embeddings to generate text vectors of subtitle documents.

The techniques like BOW and TFIDF are mostly used for keyword based search engine . Bag-of-Words (BOW) is a simple text representation technique that converts text documents into a matrix of token counts, disregarding grammar and word order. Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure used to evaluate the importance of a word in a document relative to a corpus.

We selected BERT-based embedding technique for semantic search engine , as BERT is one of the popular modern LLMs . It is used for generate text vectors for subtitle documents i.e. content column data have been embedded into text vector form. By leveraging BERT-based embeddings, we can capture rich semantic information from subtitle documents, allowing for more accurate and contextually relevant representations that enhance the performance of the search engine algorithm.

3. Document Chunker:

Given the challenge of embedding large documents, we implement a document chunking strategy to mitigate information loss. This involves dividing large documents into smaller, more manageable chunks for embedding.

Why Chunkers?
Chunking is crucial for processing large documents in natural language processing tasks. It ensures memory efficiency by dividing documents into smaller segments, retains context through overlapping windows, and improves performance by utilizing computational resources more effectively. Ultimately, chunking enhances scalability and allows for faster processing of large datasets without sacrificing context or relevant information.

The chunk_size = 2500 and overlap = 50

INNOMATICS RESEARCH LABS

4. Retrieving Documents:

When a user enters a search query, we preprocess it and calculate its embedding. We then compute cosine similarity between the query embedding and document embeddings to identify the most relevant subtitle documents.

What is cosine similarity?
Cosine similarity is a measure used to determine the similarity between two vectors in a vector space. It calculates the cosine of the angle between the two vectors, indicating how closely they align in direction. In natural language processing tasks, cosine similarity is often employed to compare the similarity between document embeddings or word vectors. A cosine similarity score of 1 indicates perfect alignment, while a score of 0 indicates orthogonality (no similarity), and a score of -1 indicates perfect opposition. It is commonly used in information retrieval, recommendation systems, and clustering algorithms to assess similarity between documents or items based on their feature representations.

To retrieve the document we used cosine similarity for comparing the users query to search with embedded file cosine vectors and display the relevant documents . The embedded documents are stored in an database for easily access and for retrieving the documents in a faster way . For that we have install chromadb database is used for storing our data .

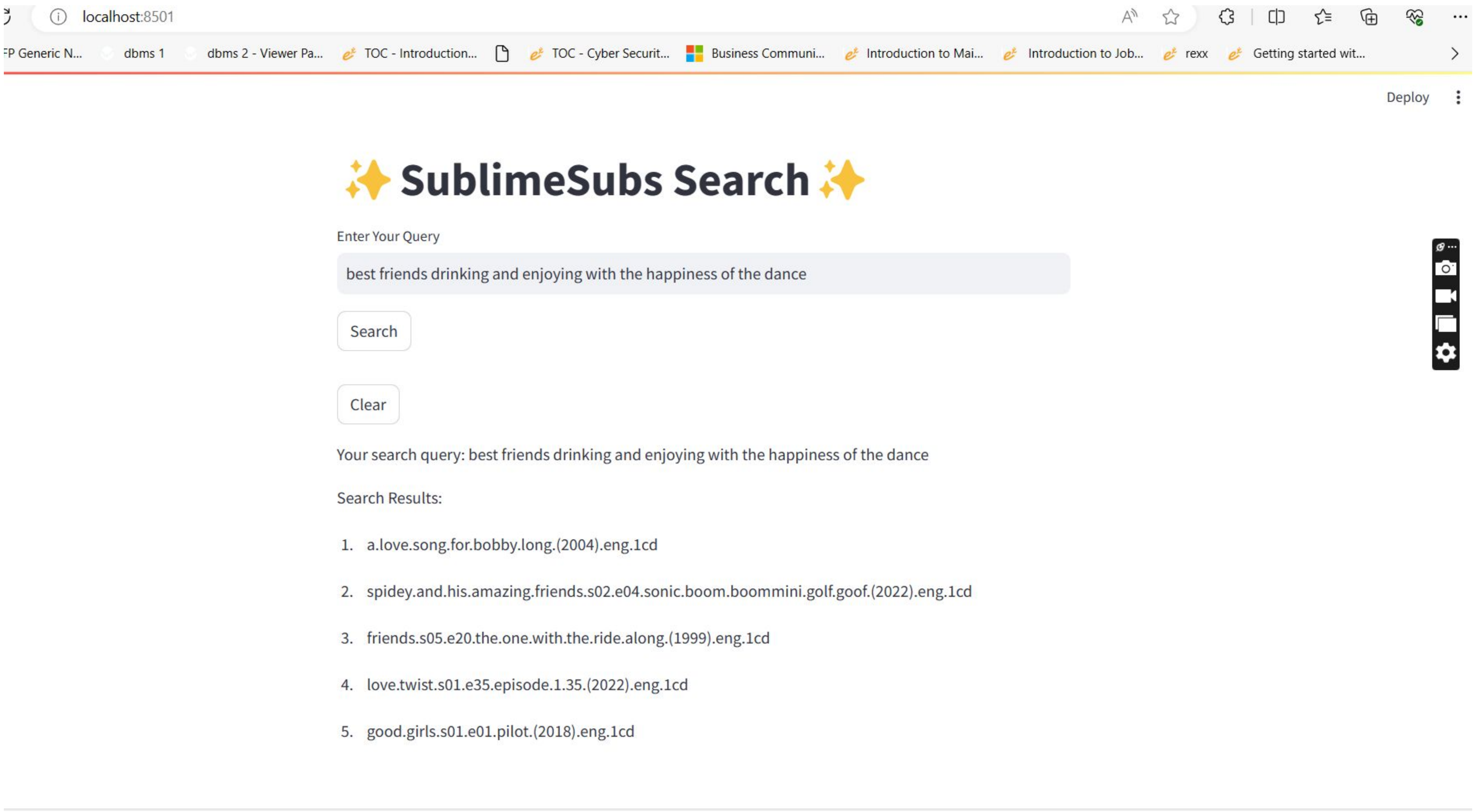
Chromadb ChromaDB is a database management system specifically designed for storing and querying high-dimensional vector embeddings efficiently. It is particularly well-suited for applications in natural language processing (NLP), machine learning, and information retrieval, where data often consists of embeddings representing text, images, or other high-dimensional features.

Implementation of Semantic Search Engine

A Streamlit web application is created for a search engine using ChromaDB to retrieve and display search results based on user queries. It utilizes Streamlit for the user interface, ChromaDB for storing and querying embeddings, and an embedding function to generate embeddings for subtitle documents. The application allows users to input search queries, retrieve relevant subtitle documents, and display them in the interface. It includes search and clear buttons for interaction, enabling users to refine their searches and reset the query input when needed. Overall, the application provides a user-friendly and interactive way to search for video subtitles based on user queries.

The Semantic Search Engine for movie subtitles - the name for this search engine is **“ SUBLIMESUBS SEARCH”** where it provide the most relevant documents from the database by processing the users query and finding its cosine similarity.

INNOMATICS RESEARCH LABS



✨ SublimeSubs Search ✨

Enter Your Query

best friends drinking and enjoying with the happiness of the dance

Search

Clear

Your search query: best friends drinking and enjoying with the happiness of the dance

Search Results:

1. a.love.song.for.bobby.long.(2004).eng.1cd
2. spidey.and.his.amazing.friends.s02.e04.sonic.boom.boommini.golf.goof.(2022).eng.1cd
3. friends.s05.e20.the.one.with.the.ride.along.(1999).eng.1cd
4. love.twist.s01.e35.episode.1.35.(2022).eng.1cd
5. good.girls.s01.e01.pilot.(2018).eng.1cd

INNOMATICS RESEARCH LABS

THANK YOU

INNOMATICS RESEARCH LABS