



INNOVATION. AUTOMATION. ANALYTICS

PROJECT ON

Building a **Semantic Based Search Engine**
Relevance For Video Subtitles

Submitted by: B Deepak Reddy,
IN124111

About me

- I am Deepak Reddy I have working as Process Executive at Nvidia Graphics Pvt. Ltd. I am Post Graduated with Specialization Mathematics.
- During my tenure I have observed without a data world wont move forward. So I decided to move on data driven skills. Data Science is skills are highly demanded across various industries. Who can extract insights from data. I am data science enthusiast who wants to apply my skills to build AI and ML models to solve real world problems.
- I have 2.5 years of experience on Autopilot: Engineering the Next Generation of Autonomous Cars. Working at Nvidia data factory with a history on image analysis programs of driver-less AI cars by using inbuilt tools which results in developing of deep learning and machine learning techniques.
- linkedin : <https://www.linkedin.com/in/deepak-reddy-bora/>
- Github:<https://github.com/deepakreddybora99>

Introduction & Objective of the Project:

- In today's rapidly evolving digital landscape, effective search engines serve as the linchpin connecting users with pertinent information.
- Google, a prime example, prioritizes delivering seamless and precise search experiences to its users.
- This project centers on elevating the search relevance for video subtitles, thereby augmenting the accessibility of video content.
- **Objective:** Develop an advanced search engine algorithm adept at retrieving subtitles based on user queries, with a specific emphasis on the content within subtitles.
- Our primary aim is to harness the power of natural language processing and machine learning techniques to enhance the relevance and accuracy of search results for video subtitles.

Keyword-based vs Semantic Search Engines

Keyword Based Search Engine:

- Relies heavily on exact keyword matches between user queries and indexed documents.
- Focuses primarily on matching specific keywords in documents to retrieve results.

Semantic Search Engines:

- Go beyond simple keyword matching to comprehend the meaning and context of user queries and documents.
- Aim to understand the deeper semantic meaning and context of user queries to deliver more relevant and meaningful search results.

Comparison:

Keyword-based: Emphasizes exact keyword matches for document retrieval.

Semantic-based: Focuses on understanding the deeper semantic meaning and context for more relevant search results.

Core Logic of this Project

1. Preprocessing of Data:

- For computational efficiency, randomly select 30% of the data if compute resources are limited.
- Cleaning step involves removing time-stamps to ensure data integrity before vectorization.

2. Cosine Similarity Calculation:

- Compute cosine similarity between the vector representations of subtitle documents and the user query.
- This similarity score quantifies the relevance of documents to the user's query.

3. Importance of Document Chunking for Large Documents:

- Document chunking is crucial for handling large documents efficiently.
- Helps mitigate information loss and ensures context continuity.
- Enables accurate representation of semantic meaning across document segments.

Step By Step Procedure followed:

Part 1: Ingesting Documents:

1. Read the given data
2. Understanding the Data
3. Data Sampling
4. Decoding and Data Cleaning
5. Implementing Document Chunker
6. Experimentation with Text Vectors
7. Creating Database and Storing the Encoded data

Step By Step Procedure followed:

Part 2: Retrieving Documents

1. Take the user's search query.
2. Preprocess the query (if required).
3. Create query embedding.
4. Creating Interface.

Part 1: Ingesting Documents:

1. **Read the given data:** Accessing the provided database file containing subtitle data.
2. **Understanding the Data:** Reviewing the README.txt file to comprehend the contents and structure of the database.
Need add pictures of data extract and decode

Table Name in Database:zipfiles

Column Names in Table :['num', 'name', 'content']

Length of zipfiles table: 82498

	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...

Part 1: Ingesting Documents:

3. Data Sampling :

Since compute resources are limited, we randomly selected 30% of the data for processing.

4. Decoding, Data Cleaning and saved to .csv File:

Ensured proper decoding of files within the database for accurate processing. Applied necessary cleaning steps to ensure data integrity, such as removing time-stamps.

```
1 import random
2
3 df=data.sample(frac=0.3,random_state=30)
4 df
```

	num	name	content
57586	9422324	doctor.who.confidential.s06.e11.heartbreak.hot...	b'PK\x03\x04\x14\x00\x00\x00\x08\x00\x974\x9aV...
4500	9200952	american.experience.s16.e06.tupperware.(2004)....	b'PK\x03\x04\x14\x00\x00\x00\x08\x00-\x8d\x99V...
73118	9481570	the.last.nosferatu.(2023).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00xe4d\x9aV...
59813	9431169	the.long.way.home.(1997).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00xe9:\x9aV...
11101	9227619	ncis.s06.e02.agent.afloat.(2008).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00xc3\x94V...
...
60579	9433130	steinheist.s01.e01.thoroughbred.(2022).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00xf7:\x9aV...
39740	9344223	puppet.master.(1989).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00\x0b0\xbcV...
1595	9188397	moment.of.truth.s01.e03.chapter.three.the.wron...	b'PK\x03\x04\x14\x00\x00\x00\x08\x00\x11\x80V...
68228	9462115	penny.dreadful.city.of.angels.s01.e03.wicked.o...	b'PK\x03\x04\x14\x00\x00\x00\x08\x00n5\x9aVxa...
34358	9318075	one.tree.hill.s07.e05.your.cheatin.heart.(2009...	b'PK\x03\x04\x14\x00\x00\x00\x08\x00<\xb1\x99V...

24749 rows × 3 columns

```
1 def preprocess_title(data):
2     #removing unnecessary characters in text
3     data= re.sub(r'\.eng\.1cd','',data).strip()
4     data = re.sub(r'episode\s\d+\s\d+', '',data).strip()
5     #lower the text
6     data = data.lower()
7     #removing special characters
8     data=re.sub(r'^\w\s.'],' ', data)
9     data=data.replace('.', ' ')
10    return data
11 df['title']=df['name'].progress_apply(preprocess_title)
```

100% | 24749/24749 [00:00<00:00, 116434.87it/s]

```
1 df1=df[['num','title','subtitles']]
2 df1.rename(columns={'num':'Ids'},inplace=True)
3 df1.reset_index(inplace=True)
4 df1.drop(columns=['index'],inplace=True)
5 df1.head()
6 df1.to_csv('cleand.csv',index=False)
```

Part 1: Ingesting Documents:

5. Implement Document Chunker :

- Divided large documents into smaller, manageable chunks to address the challenge of embedding large documents.
- Mitigated information loss and maintained context continuity by setting overlapping windows with a specified amount of tokens to overlap between chunks.

```
1 def document_chunker(data, chunk_size=500, overlap_size=50):
2     chunks = []
3     start_idx = 0
4     while start_idx < len(data):
5         end_idx = min(start_idx + chunk_size, len(data))
6         chunk = ' '.join(data[start_idx:end_idx])
7         chunks.append(chunk)
8         start_idx += chunk_size - overlap_size
9     return chunks
```

Python

```
1 chunked_texts = document_chunker(df1['subtitles'].tolist())
```

Python

Part 1: Ingesting Documents:

6. Experimentation with Text Vectors:

Explored different techniques for generating text vectors of subtitle documents:

Bag-of-Words (BOW) / TF-IDF: To Create sparse vector representations suitable for Keyword-Based Search Engine.

BERT-based "Sentence Transformers": To Generate embeddings encoding semantic information for Semantic Search Engine.

```
1 def generate_embeddings(texts):  
2     model = SentenceTransformer('paraphrase-MiniLM-L3-v2')  
3     embeddings = []  
4     for text_chunk in texts:  
5         chunk_embeddings = model.encode(text_chunk)  
6         embeddings.append(chunk_embeddings)  
7     return embeddings
```

```
1 chunk_embeddings=generate_embeddings(chunked_texts)
```

Part 1: Ingesting Documents:

7. Creating Database and Storing The encoded data:

- Creating the Database using ChromDB with Cosine distance metric
- stored generated data and embeddings in a ChromaDB database for efficient retrieval and further analysis.

```
1 import chromadb
2 client = chromadb.PersistentClient(path="vectordb")
3 from sentence_transformers import SentenceTransformer
4
5
6 model='paraphrase-MiniLM-L3-v2'
7
8
9 collection=client.get_or_create_collection(name='subtitle',
10 | | | | | | | | metadata={'hnsw':'cosine'},
11 | | | | | | | | )
[2]
... c:\Users\hp\Desktop\internship\search engine project\.env_subtitles\lib\site-packages\tqdm\auto.py:21: T
from .autonotebook import tqdm as notebook_tqdm
```

```
1 for index, row in df1.iterrows():
2     document_id = str(row['Ids']) # Use a unique identifier for each document
3     document_text = row['subtitles']
4     # document_embedding = corpus_embeddings[index % len(corpus_embeddings)] # Assuming embeddings are aligned with the rows
5
6     # -----
7     chunk_index = index % len(chunk_embeddings)
8     document_embedding = chunk_embeddings[chunk_index]
9     # -----
10    metadata = {'Title': row['title']}
11
12    # Insert document into ChromaDB collection
13    collection.add(ids=document_id, documents=[document_text], embeddings=[document_embedding.tolist()], metadatas=[metadata])
14
15
16 print("Insertion into ChromaDB collection complete")
```

Part 2 : Retrieving Documents

1. Take the user's search query.
2. Preprocess the query (if required).
3. Create query embedding.

```
1 query='the last nosferatu 2023'
2 result=collection.query(
3     query_texts = query,
4     include=["metadatas", 'distances'],
5     n_results=10
6 )
7
8 ids = result['ids'][0]
9 distances = result['distances'][0]
10 metadatas = result['metadatas'][0]
11 zipped_data = zip(ids, distances, metadatas)
12 sorted_data = sorted(zipped_data, key=lambda x: x[1], reverse=True)
13 for _, distance, metadata in sorted_data:
14     subtitle_name = metadata['Title']
15     print(f"Title: {subtitle_name.upper()}")
16
```

Title: SANTA EVITA S01 E04 UNA ESPOSA PERFECTA 2022
Title: A GUN FOR JENNIFER 1997
Title: BLUE BLOODS S13 E03 GHOSTED 2022
Title: KITA KITA 2017
Title: THE PINK CLOUD 2021
Title: COOPERS TREASURE S02 E03 WRECK IN THE SHALLOWS 2018
Title: SUDDEN DEATH 1995
Title: RESTAURANT TO ANOTHER WORLD S02 E06 RICE BURGERPIZZA 2021
Title: CALL ME FITZ S03 E09 TEETOTAL RECALL 2012
Title: BAD EDUCATION S04 E02 WHODUNNIT

Part 2 : Retrieving Documents

4. Creating Interface.

Creating the User interface using Streamlit

```
import streamlit as st
import chromadb
import time

# load the dataset
path = "vectordb"



client = chromadb.PersistentClient(path=path)
client.heartbeat()
collection = client.get_collection(name="subtitle")

def similar_title(query_text):
    result = collection.query(
        query_texts=query_text,
        include=["metadatas", "distances"],
        n_results=5
    )
    ids = result['ids'][0]
    distances = result['distances'][0]
    metadatas = result['metadatas'][0]
    zipped_data = zip(metadatas, ids, distances)
    sorted_data = sorted(zipped_data, key=lambda x: x[1], reverse=True)
    return sorted_data

st.title('🔍 Beyond Keywords: Subtitle Search Revolution')
st.subheader('Tired of video trailer roulette? 🤖')
st.subheader('Subtitle search: Your shortcut to the good stuff. 🎬')

query_text = st.text_input('Enter your search query:')
search = st.button("Search")
if search:
    result = collection.query(
        query_texts=query_text,
        include=["metadatas", "distances"],
        n_results=10
```

Results:


  **Beyond Keywords: Subtitle Search Revolution**

Tired of video trailer roulette? 🤔

Subtitle search: Your shortcut to the good stuff. 🤖

Enter your search query:

Search

 **Subtitle search: Your shortcut to the good stuff. 🤖**

Enter your search query:

Search

Here are the most relevant subtitle names:

[santa evita s01 e04 una esposa perfecta 2022]([https://www.opensubtitles.org/en/subtitles/santa evita s01 e04 una esposa perfecta 2022](https://www.opensubtitles.org/en/subtitles/santa%20evita%20s01%20e04%20una%20esposa%20perfecta%202022))

[a gun for jennifer 1997]([https://www.opensubtitles.org/en/subtitles/a gun for jennifer 1997](https://www.opensubtitles.org/en/subtitles/a%20gun%20for%20jennifer%201997))

[blue bloods s13 e03 ghosted 2022]([https://www.opensubtitles.org/en/subtitles/blue bloods s13 e03 ghosted 2022](https://www.opensubtitles.org/en/subtitles/blue%20bloods%20s13%20e03%20ghosted%202022))

[kita kita 2017]([https://www.opensubtitles.org/en/subtitles/kita kita 2017](https://www.opensubtitles.org/en/subtitles/kita%20kita%202017))

[the pink cloud 2021]([https://www.opensubtitles.org/en/subtitles/the pink cloud 2021](https://www.opensubtitles.org/en/subtitles/the%20pink%20cloud%202021))

[coopers treasure s02 e03 wreck in the shallows 2018]
([https://www.opensubtitles.org/en/subtitles/coopers treasure s02 e03 wreck in the shallows 2018](https://www.opensubtitles.org/en/subtitles/coopers%20treasure%20s02%20e03%20wreck%20in%20the%20shallows%202018))

[sudden death 1995]([https://www.opensubtitles.org/en/subtitles/sudden death 1995](https://www.opensubtitles.org/en/subtitles/sudden%20death%201995))

[restaurant to another world s02 e06 rice burgerpizza 2021]
([https://www.opensubtitles.org/en/subtitles/restaurant to another world s02 e06 rice burgerpizza 2021](https://www.opensubtitles.org/en/subtitles/restaurant%20to%20another%20world%20s02%20e06%20rice%20burgerpizza%202021))

Conclusion:

Key Takeaways:

Enhanced Search Relevance:

By leveraging **BERT**-based embeddings model '**paraphrase-MiniLM-L3-v2**' and cosine similarity, we improved the search relevance for video subtitles. The use of semantic search techniques allowed us to capture the meaning and context of user queries, resulting in more accurate search results compared to traditional keyword-based methods.

Efficient Storage and Retrieval:

ChromaDB provided a robust solution for storing and querying large collections of document embeddings. Its efficient indexing and retrieval mechanisms allowed for fast and scalable search operations, making it suitable for real-time applications with large datasets.

User-Friendly Interface:

The **Streamlit** frontend provided an intuitive and interactive interface for users to interact with the search engine. With features such as text input for queries, search button, and informative feedback messages, the app offered a seamless user experience.

THANK
YOU

