INNOMATICS® RESEARCH LABS

INNOVATION. AUTOMATION. ANALYTICS

PROJECT ON

Building a Semantic Based Search Engine Relevance for Video Subtitles

Submitted By - Harshit Punera (IN1240881)
Abhiram Bussa (IN1240515)
Team Id- (T211095)

Objective

- Developing an advanced search engine algorithm
- Leveraging Natural language processing and machine learning
- Enhancing relevance and accuracy of search results for video subtitles

INNOMATICS RESEARCH LABS

Summary

- 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.

Core logic of this project

- Preprocessing of Data
- Cosine similarity calculation for relevance assessment
- Importance of document chunking for large documents

Step by Step Process

➢ Ingesting Documents

- Reading and decoding the database file
- Understand the encoding language
- Cleaning and preprocessing steps
- Experimenting with BOW/TFIDF and SentenceTransformers
- Implementing document chunking
- Storing embeddings in ChromaDB

- ➢ Retrieving Documents
- Take User query
- Preprocessing user queries
- Creating query embeddings
- Calculating cosine similarity scores
- Retrieving relevant candidate documents

INNOMATICS RESEARCH LABS

- Reading and Decoding the Database File:
  - The subtitle data is provided in a Database file format. Ingesting begins by reading this file and understanding its structure
  - Subtitles Data encoded with “Latin-1” and in a compressed format , so decoding is necessary to access the actual subtitle text
- Cleaning and Preprocessing Steps:
  - We **took a random sample 30% of the data for cleaning**
  - Before analysis The subtitles text require cleaning to remove irrerelevant elements include removing Timestamps,Special characters,stopwords ,apply lemmetaization

```python
def clean_subtitles(text):
    pattern = r'\d+\s*[\r\n]+\s*\d+:\d+:\d+,\d+\s*-->\s*\d+:\d+:\d+,\d+\s*[\r\n]+'
    cleaned_text = re.sub(pattern, " ",text)
    return cleaned_text

from tqdm import tqdm, tqdm_notebook
tqdm.pandas()
new_df['clean_subtitles'] = new_df['file_content'].progress_apply(lambda x:clean_subtitles(x))
```

Python

```
100%|██████████████████████████████████████████████████| 24749/24749 [01:47<00:00, 230.24it/s]
```

```python
from tqdm import tqdm, tqdm_notebook

def preprocess(raw_text, flag):
    # Removing special characters and digits
    sentence = re.sub("[^a-zA-Z]", " ", raw_text)

    # Change sentence to lowercase
    sentence = sentence.lower()

    # Tokenize into words
    tokens = sentence.split()

    # Lemmatization
    clean_tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return pd.Series([" ".join(clean_tokens)])

tqdm.pandas()
new_df['clean_data'] = new_df['clean_subtitles'].progress_apply(lambda x:preprocess(x,'lemma'))
```

Python

Document Chunking

- Document chunking involves dividing large subtitle documents into smaller,more manageable chunks
- This is important because embedding entire documents as single vectors may lead to information loss,especially with long documents
- Overlapping windows with a specified number of tokens can be used to ensure continuity and context preservation across chunks

```python
from tqdm import tqdm, tqdm_notebook
tqdm.pandas()
def chunk_text(text, chunk_size=500, overlap_size=50):
    chunks = []
    words = text.split()
    start_idx = 0
    while start_idx < len(words):
        end_idx = min(start_idx + chunk_size, len(words))
        chunk = ' '.join(words[start_idx:end_idx])
        chunks.append(chunk)
        start_idx += chunk_size - overlap_size
    return chunks

# Apply chunking function to the 'clean_subtitles' column
chunk_size = 500  # Number of tokens per chunk
overlap_size = 50  # Number of tokens to overlap between chunks

df['chunks'] = df['clean_data'].progress_apply(lambda x: chunk_text(x, chunk_size, overlap_size))
```

[6]

Python

```
100%|██████████| 24749/24749 [00:05<00:00, 4780.00it/s]
```

INNOMATICS
RESEARCH LABS

➢ Experimenting with BOW/TFIDF and SentenceTransformers

- After cleaning,the subtitle documents are prepared for analysis by converting them into numerical representations
- We used BERT-based “Sentence Transformers” vectorization technique.
- This Technique generate embeddings that encode semantic information,capturing the meaning and context of the text. They are suitable for building semantic search engines.

```python
from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer('all-MiniLM-L6-v2')
```

Python

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter a
  from .autonotebook import tqdm as notebook_tqdm
```

```python
from tqdm import tqdm
tqdm.pandas()

def encode_and_convert_to_list(text):
    encoded_vector = model.encode(text).tolist()
    return encoded_vector

# Apply the function to each element of 'chunks'
df_exploded['doc_vector_pretrained_bert'] = df_exploded['chunks'].progress_apply(encode_and_convert_to_list)
```

Python

```
100%|██████████| 311548/311548 [1:23:35<00:00, 62.12it/s]
```

INNOMATICS RESEARCH LABS

Storing Embeddings in a chromadb vectors database

- Once the subtitles documents are processed and converted into embeddings,they need to be stored for efficient retrieval during the search process.
- Chromadb is efficient in handling large-scale vector data and enabling fast retrieval
- Chroma gives you the tools to:
- store embeddings and their metadata
- embed documents and queries
- search embeddings on the basis of cosine similarity scores

INNOMATICS
RESEARCH LABS

```
from tqdm import tqdm

batch_size = 5000
total_batches = (len(documents) + batch_size - 1) // batch_size

for i in tqdm(range(total_batches), desc="Adding batches"):
    start_idx = i * batch_size
    end_idx = min((i + 1) * batch_size, len(documents))

    batch_doc = documents[start_idx:end_idx]
    batch_metadatas = metadatas[start_idx:end_idx]
    batch_ids = ids[start_idx:end_idx]
    embed = df_bert_pretrained[start_idx:end_idx]

    collection.add(
        embeddings=embed,
        documents=batch_doc,
        metadatas=batch_metadatas,
        ids=batch_ids
    )
```

```
Adding batches: 100%|██████████| 63/63 [08:16<00:00,  7.87s/it]
```

INNOMATICS RESEARCH LABS

Retrieving Documents

- ➢ Taking User query
- The process begins with the user inputting a query into the search engine . This query represents the information the user is seeking from the subtitle database.
- ➢ Preprocessing User queries:
- Before the query can be used for retrieval, it undergoes preprocessing steps to standardize and clean the text
- ➢ Creating Query Embeddings:
- After preprocessing, the user query is converted into a numerical representation,known as an embedding

INNOMATICS
RESEARCH LABS

- Bert-based Sentence Transformers is used to generate embeddings that encode the semantic information of the query.
- This embedding captures the semantic meaning and context of the query,allowing it to be compared to the embeddings of subtitle documents
- ➢ Calculating cosine similarity scores:
- Now Connection is created to the path of that chromadb database
- Once the query has been embedded, the next step is to compare it to the embeddings of the subtitle documents in the database.
- Cosine similarity scores is calculated by chromadb between the query embedding and the embeddings of all subtitle documents.

```python
# Query Point
query_point = "what happened last night when we were fully drunk"

# Encode query point
doc_vector = clean_and_encode_query(query_point, model)

# Query the collection
result = collection.query(
    query_embeddings=doc_vector,
    n_results=10,
)

# Print movie names
print_movie_names(result)
```

[7] Python

```
Movie Names:
csi.miami.s03.e03.under.the.influence.(2004).eng.1cd
friends.s07.e16.the.one.with.the.truth.about.london.(2001).eng.1cd
working.class.heroes.(2022).eng.1cd
murdoch.mysteries.s09.e06.the.local.option.(2015).eng.1cd
revenge.for.daddy.(2020).eng.1cd
workin.moms.s07.e07.its.all.gone.(2023).eng.1cd
mister8.s01.e05.komediaperjantai.(2021).eng.1cd
brutal.bridesmaids.(2021).eng.1cd
50.first.dates.(2004).eng.1cd
according.to.jim.s08.e07.the.ego.boost.(2009).eng.1cd
```

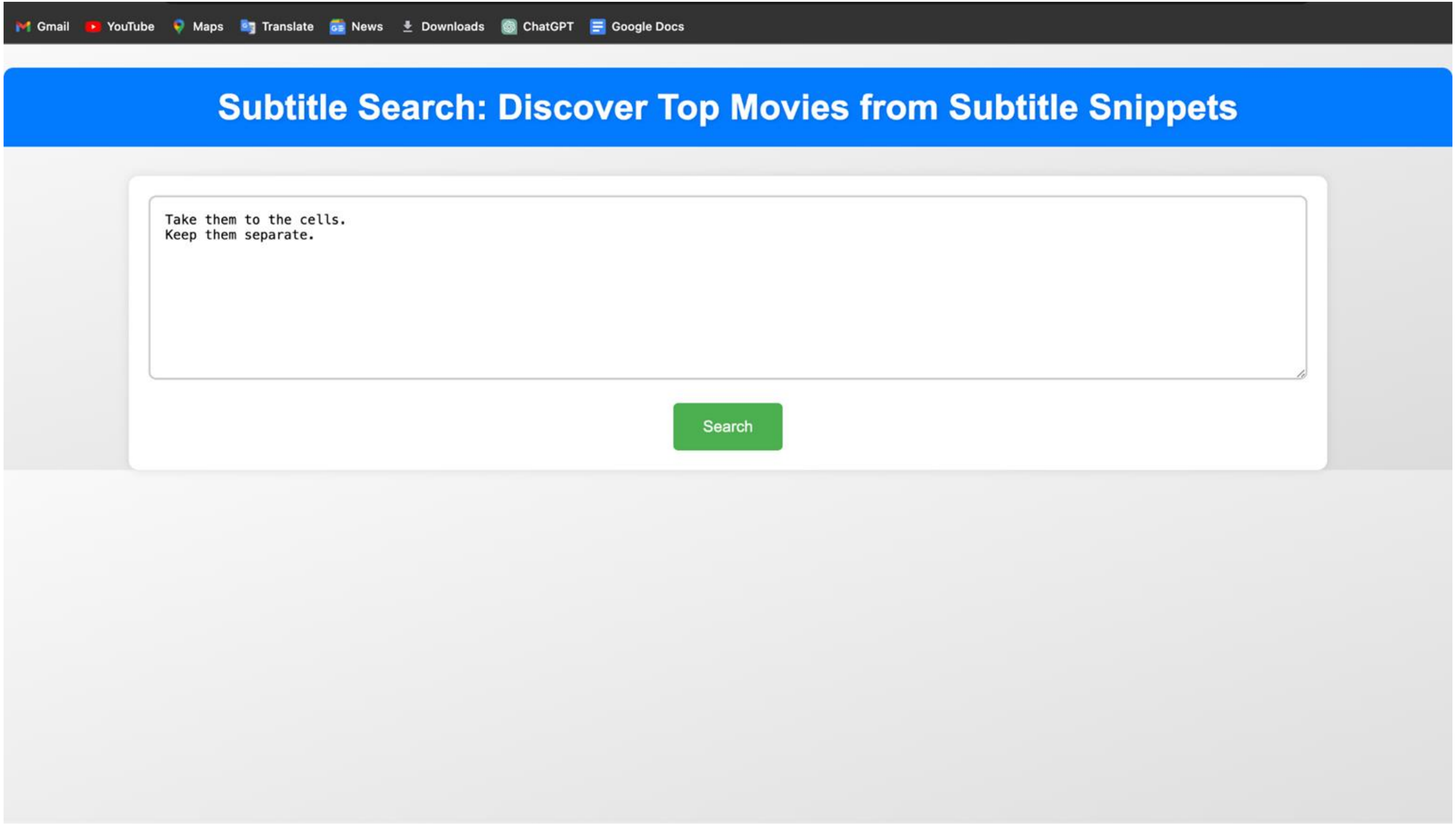
Retrieving Relevant candidate Documents

- Finally ,based on the calculated cosine similarity scores , the search engine retrieves the most relevant Top N candidate documents from the stored data in Chromadb database

- **Web application.**

  We used Flask to build a web application that searches for a specific part of a subtitle and returns the top ten results. Linking each filename to the OpenSubtitles page allows for easy download of the subtitles.

INNOMATICS RESEARCH LABS


Gmail
YouTube
Maps
Translate
News
Downloads
ChatGPT
Google Docs
Subtitle Search: Discover Top Movies from Subtitle Snippets
Take them to the cells.
Keep them separate.
Search

Gmail YouTube Maps Translate News Downloads ChatGPT Google Docs

Top Movies Related to Subtitle Snippet:

magino.story.raising.silkworms.(1977).eng.1cd

View Subtitles

csi.crime.scene.investigation.s04.e16.getting.off.(2004).eng.1cd

View Subtitles

csi.crime.scene.investigation.s04.e16.getting.off.(2004).eng.1cd

View Subtitles

magino.story.raising.silkworms.(1977).eng.1cd

View Subtitles

the.makery.s01.e08.wakey.wakey.(2023).eng.1cd

View Subtitles

the.repair.shop.s08.e01.charles.ii.portrait.(2021).eng.1cd

THANK YOU