UE21CS342BA2 - Algorithms for Information Retrieval and Intelligence Web

Mini Project Report

"Youtube Comments Scraper"

Submitted by:

Sparsh BK **PES1UG21CS610**
Suhas Gowda Harish **PES1UG21CS633**

Prof. Nagegowda

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Project Report

Synopsis:

This project aims to create an integrated system that leverages advanced natural language processing (NLP) techniques and AI to enhance the user experience of engaging with YouTube content. The system consists of several components:

- `scrapper.py` scrapes YouTube for video IDs based on predefined criteria.
- `relc.py` processes the comments from these videos using BERT and SpaCy, extracting relevant comments in response to user queries.
- `qa1.py` enables interaction with the AI21 language model to generate intelligent, context-aware responses.
- `ui.py` ties these backend operations into a responsive Flask web application. The user interface is dynamically updated with relevant images sourced from Pexels API based on user-specified keywords. This system is designed for users who require deeper, AI-driven insights and interactions with video content, facilitating a richer online experience.

General Use Case:

The project can serve a wide range of applications, particularly in educational, entertainment, and research domains where video content plays a crucial role. For instance:

- Educational platforms can use it to extract meaningful discussions and questions from video lectures.
- Content creators and marketers can utilize the insights for better engagement strategies or to understand audience sentiment.
- Researchers analyzing social media and user interactions can deploy the tool to automate and refine data collection and analysis.

In-depth Usage of Models:

The application uses several advanced models to handle specific tasks:

BERT: Employed to understand the context within the YouTube comments. It excels in tasks that require an understanding of the entire context of a sentence or a paragraph, which makes it ideal for extracting relevant comments that are most likely to contain responses or opinions tied directly to user queries.

SpaCy: Used for its robust processing features, such as tokenization, part-of-speech tagging, and named entity recognition, which support BERT in isolating relevant textual elements from comments.

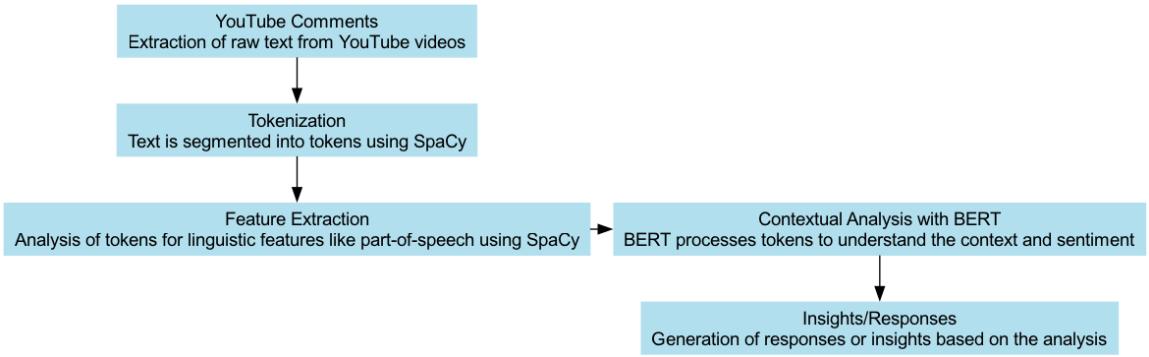
AI21: Leveraged to generate responses based on the data extracted and processed by BERT

and SpaCy. It's designed to produce text that is coherent and contextually appropriate, enriching the user's interactive experience with AI-driven narratives.

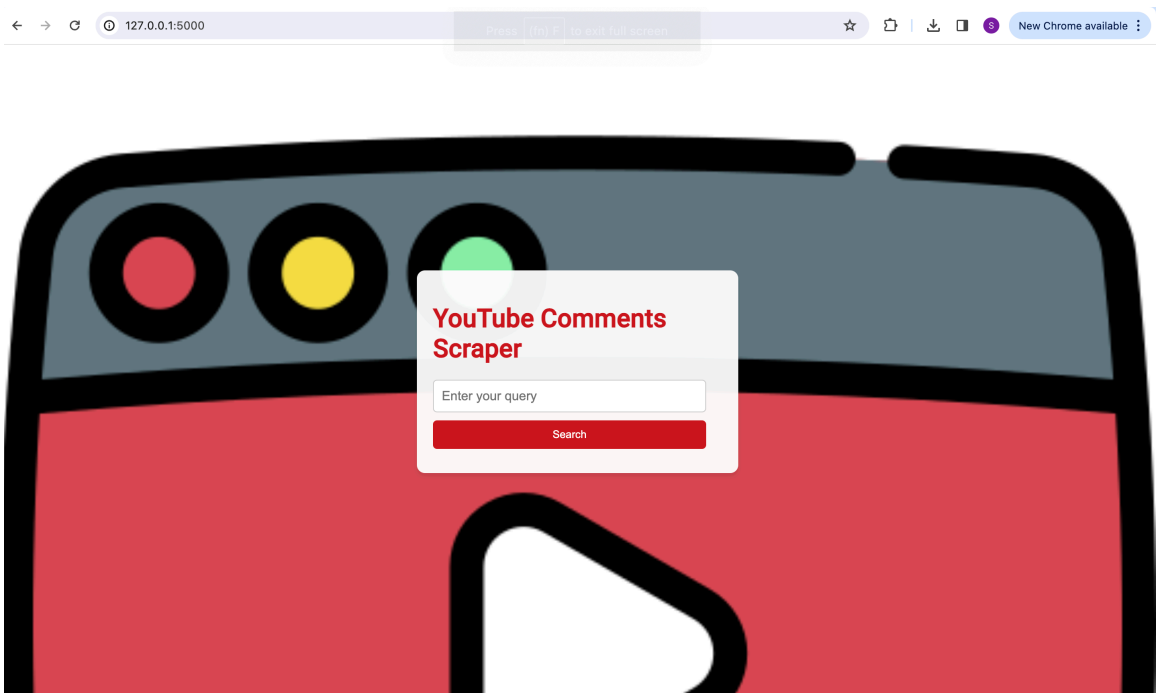
Dynamic Web UI:

The project incorporates a dynamic web user interface, built using Flask, which integrates with the Pexels API to fetch and display images relevant to the keywords detected in user interactions. This feature enhances the visual appeal of the application and provides a more engaging and immersive user experience. By dynamically updating the UI with contextually relevant visuals, the application not only delivers textual content but also enhances it with visual stimuli, making the information retrieval experience more comprehensive and enjoyable.

Working manner:



Working:



127.0.0.1:5000/display_comments

New Chrome available

Comments

("Uni blues are not under 20k but they are legit hottt🔥....I bought them at the resale price and I'm so", 'happy💙\t0')

('Best top five', '🔥🔥🔥🔥\t3')

('Bhai please tell us from where we can order jordans in', 'india\t0')

('Jordan 1 Low Bred Toe is also available for 20-21 right', 'now\t3')

('Karan bro pls do it for jordan 4s,5s and 6s too', 'pls\t1')

('Finally I got fearless Edison Chen Jordan mid', '.\t0')

('Can someone explain why these Jordans price more than 15k!!! They look like shoes of worth 5k', 'only!!!!\t0')

('Unc lows are', 'cool💯\t0')

("Racer blue's are best 💥 #kkfam", '❤️\t5')

('Hi bro pls make more vods like this under30k', '50k\t1')

('HEY BRO 😸 IS THERE ANY RETAIL STORE FOR JORDANS? IF YES PLEASE LET ME KNOW HERE OR YOU CAN MAKE A', 'VIDEO.\t0')

('Thx for providing giveaway to try our luck', '❤️😊\t0')

('Just can't believe you guyzz are giving away jordans omgg', '😎😎😎\t0')

('Nice approach to improve content in YouTube', 'India\t0')

('Happy bday magician 😇😇 exicted to see more videos with samay', '.\t0')

('Thnks for spreading the sneaker culture', '😇\t0')

('Jordans are becoming too mainstream now Can we expect some more high end fashion brand sneaker', 'videos?\t0')

('Unbox Therapy of india', '😊\t0')

('none of the links showed a price of under 20K..', 'weird\t0')

('10k ke shoes 20k me bechre ho kuch bhi bhai resale kro par itna double', 'price\t0')

Ask a Question

Enter your question

Ask Question

127.0.0.1:5000/generate_answer?question=top+airjordan+to+buy&theme=airjordan

New Chrome available

Code:

scrapper.py

```
import spacy
from googleapiclient.discovery import build
import operator
import argparse
import os

api_key = 'YOUR_API_KEY'
youtube = build('youtube', 'v3', developerKey=api_key)
def fetch_top_videos(query):
    search_response = youtube.search().list(
        q=query,
        part='snippet',
        order='relevance',
        type='video',
        maxResults=10,
    ).execute()
    video_ids = [item['id']['videoId'] for item in
search_response['items']]

    video_response = youtube.videos().list(
        part='statistics',
        id=','.join(video_ids)
    ).execute()

    videos_with_likes = {}
    for item in video_response['items']:
        video_id = item['id']
        like_count =
int(item['statistics'].get('likeCount', 0))
        videos_with_likes[video_id] = like_count

    sorted_videos = sorted(videos_with_likes.items(),
key=operator.itemgetter(1), reverse=True)
    # print(sorted_videos) to check out the video ids
    return [video[0] for video in sorted_videos[:3]]
```

```
def fetch_comments_from_video(video_id, max_results=100):
    comments_with_likes = []
    # Fetch comments from the video
    response = youtube.commentThreads().list(
        part='snippet,replies',
        videoId=video_id,
        textFormat='plainText',
        maxResults=max_results,
        order='relevance'  # This brings pertinent comments
to the top but not necessarily sorted by likes
    ).execute()
    # Collect all comments with their like counts
    for item in response['items']:
        comment_text = item['snippet']['topLevelComment']
['snippet']['textDisplay']
        like_count = item['snippet']['topLevelComment']
['snippet']['likeCount']
        comments_with_likes.append((comment_text,
like_count))
    return comments_with_likes

def store_comments_to_file(comments, file_path):
    with open(file_path, 'w', encoding='utf-8') as file:
        for comment_tuple in comments:
            comment_text = comment_tuple[0].replace('\n', '
')
            like_count = str(comment_tuple[1])
            file.write(f"{comment_text}\t{like_count}\n")
def run_scraper(query):

    top_videos = fetch_top_videos(query)
    return top_videos
```

Relc.py

```
from transformers import BertTokenizer, BertModel
import torch
from scipy.spatial.distance import cosine
import spacy



# Load models and tokenizers outside the function to avoid
reloading them on each call
tokenizer = BertTokenizer.from_pretrained('bert-base-
uncased')
```

```
model = BertModel.from_pretrained('bert-base-uncased')
nlp = spacy.load("en_core_web_sm")


def just_keyword(query):
    doc = nlp(query)
    noun_chunks = list(doc.noun_chunks)
    return noun_chunks[0].root.text


def generate_bert_embedding(text):
    input_ids = tokenizer.encode(text,
add_special_tokens=True)
    input_tensor = torch.tensor(input_ids).unsqueeze(0)
    with torch.no_grad():
        outputs = model(input_tensor)
        embeddings =
outputs[0].squeeze(0).mean(dim=0).numpy()
    return embeddings

def calculate_cosine_similarity(vec1, vec2):
    return 1 - cosine(vec1, vec2)

def process_file(file_path, query, output_file,
threshold=0.6):
    # Extract keywords from query
    doc = nlp(query)
    keywords = [token.text for token in doc if token.pos_
in ['NOUN', 'PROPN']]
    print(f"Processing new file for query: {query}")

    with open(file_path, 'r', encoding='utf-8') as file,
open(output_file, 'w', encoding='utf-8') as output:
        for line in file:

            parts = line.strip().split('\t')
            if len(parts) >= 2:
                comment, like_count = parts[0], parts[1]
            else:

                continue
            keyword_present = any(keyword.lower() in
comment.lower() for keyword in keywords)
            comment_embedding =
generate_bert_embedding(comment)
```

```
            query_embedding =
generate_bert_embedding(query)
            similarity =
calculate_cosine_similarity(query_embedding,
comment_embedding)

            if keyword_present or similarity >= threshold:
                output.write(f"{comment}\t{like_count}\n")


def calculate_similarity_score(query, comment):
    query_embedding = generate_bert_embedding(query)
    comment_embedding = generate_bert_embedding(comment)
    similarity =
calculate_cosine_similarity(query_embedding,
comment_embedding)
    return similarity
```

qa1.py

```
import requests

def load_comments(filepath):
    """Load comments from a file and return as a single
string."""
    comments = []
    with open(filepath, 'r', encoding='utf-8') as file:
        for line in file:
            comment, _ = line.strip().split('\t')
            comments.append(comment)
    return " ".join(comments)


def generate_text_with_jurassic(prompt, api_key):
    url = "https://api.ai21.com/studio/v1/j2-mid/complete"#
Ensure this is correct

    headers = {
        "Authorization": f"Bearer {api_key}",
        "Content-Type": "application/json",
    }

    data = {
        "prompt": prompt,
        "maxTokens": 100,
```

```
        "temperature": 0.7,
    }

    response = requests.post(url, headers=headers,
json=data)
    if response.status_code != 200:
        print(f"API Error: {response.status_code}")
        print(f"API Response: {response.json()}")
        return "An error occurred while processing the API
response."

    result = response.json()

    try:
        return result['completions'][0]['data']['text']
    except KeyError:
        print("KeyError: The expected keys were not found
in the response.")
        print("API Response:", result)
        return "An error occurred while processing the API
response."
```

Ui.py

```
from flask import Flask, request, render_template, url_for,
redirect
from scrapper import run_scraper,
fetch_comments_from_video, store_comments_to_file
from relc import process_file,just_keyword
from qa1 import generate_text_with_jurassic, load_comments
from flask import session, jsonify
import requests


app = Flask(__name__)
app.secret_key = 'sparsh'



@app.route('/')
def home():
    return render_template('index.html')
```

```
@app.route('/search', methods=['POST'])
def search():
    query = request.form.get('query')
    print(f"Comments stored for query: {query}")
    video_ids = run_scraper(query)  # Fetch video IDs based
on the query
    all_comments = []
    for video_id in video_ids:
        comments = fetch_comments_from_video(video_id)  #
Fetch comments for each video
        all_comments.extend(comments)

    # Store comments to a file
    comments_file_path = ''
    store_comments_to_file(all_comments,
comments_file_path)

    # Process and sort comments
    sorted_comments_file_path = ''
    process_file(comments_file_path, query,
sorted_comments_file_path)

    # Redirect to a new route to display comments
    return redirect(url_for('display_comments'))


def load_comments(file_path):
    comments_with_likes = []
    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:

            parts = line.rsplit(' ', 1)
            if len(parts) == 2:
                comment, likes = parts

comments_with_likes.append((comment.strip(),
likes.strip()))
    return comments_with_likes


def get_pexels_image_url(keyword):
    pexels_api_key = 'YOUR_PEXELS_API_KEY'
    headers = {
        'Authorization': pexels_api_key
    }
```

```
    url = f'https://api.pexels.com/v1/search?
query={keyword}&per_page=1'
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        data = response.json()
        if data['photos']:

            return data['photos'][0]['src']['original']
    return None

@app.route('/display_comments')
def display_comments():

    sorted_comments = load_comments('')#path of
sorted_comments
    return render_template('results.html',
comments=sorted_comments)



@app.route('/ask_question', methods=['POST'])
def ask_question():

    question = request.form['question']
    theme=just_keyword(question)
    return redirect(url_for('generate_answer',
question=question,theme=theme))

@app.route('/generate_answer')
def generate_answer():

    question = request.args.get('question')
    api_key = "YOUR_AI21_API_KEY"
    theme = request.args.get('theme')
    background_image_url = get_pexels_image_url(theme)

    print(theme)#check out the theme being sent to pexels
api
    context = load_comments('')#paste the cd
    prompt = f"Context: {context}\nQuestion: {question}"
    answer = generate_text_with_jurassic(prompt, api_key)
    return render_template('answer.html',
question=question,
answer=answer,background_image_url=background_image_url)
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

Templates:
Answer.html

```
<!-- <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Answer</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            padding: 20px;
            background-color: #f9f9f9;
        }
        .container {
            background-color: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }
        .question {
            color: #333;
            font-weight: bold;
        }
        .answer {
            margin-top: 10px;
            color: #555;
        }
    </style>
</head>
<body>
<div class="container">
    <div class="question">Question: {{ question }}</div>
    <div class="answer">Answer: {{ answer }}</div>
</div>
</body>
</html> -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Answer</title>
    <style>
```

```
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            padding: 20px;
            /* Use the background image URL from the Flask
route */
            background-image:
url("{{ background_image_url }}");
            background-size: cover;
            background-position: center;
        }
        .container {
            background-color: rgba(255, 255, 255, 0.8); /*
Slightly transparent white for readability */
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }
        .question {
            color: #333;
            font-weight: bold;
        }
        .answer {
            margin-top: 10px;
            color: #555;
        }

    </style>
</head>
<body>
<div class="container">
    <div class="question">Question: {{ question }}</div>
    <div class="answer">Answer: {{ answer }}</div>
</div>
</body>
</html>
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>YouTube Comments Scraper</title>
```

```
    <link href="https://fonts.googleapis.com/css2?
family=Roboto:wght@400;500&display=swap" rel="stylesheet">
    <style>
        body {
            background-image: url("{{ url_for('static',
filename='youtube.png') }}");
            background-size: cover;
            background-position: center top;
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-repeat: no-repeat;
        }

        .container {
            background-color: rgba(255, 255, 255, 0.95);
            border-radius: 10px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            padding: 20px;
            width: 90%;
            max-width: 400px;
            margin: auto;
            box-sizing: border-box;
        }

        h1 {
            color: #CC181E; /* YouTube's red color for
branding */
            margin-bottom: 20px;
        }

        input[type="text"], input[type="submit"] {
            width: calc(100% - 20px);
            padding: 10px;
            margin-bottom: 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
            box-sizing: border-box;
            transition: all 0.3s ease;
        }

        input[type="text"] {
```

```
            font-size: 16px;
        }

        input[type="text"]:focus {
            outline: none;
            border-color: #CC181E;
            box-shadow: 0 0 0 2px rgba(204,24,30,0.2);
        }

        input[type="submit"] {
            background-color: #CC181E;
            color: #fff;
            font-weight: 500;
            border: none;
            cursor: pointer;
        }

        input[type="submit"]:hover {
            background-color: #A3161A;
        }

        @media (max-width: 768px) {
            .container {
                width: calc(100% - 40px);
                box-sizing: border-box;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>YouTube Comments Scraper</h1>
        <form action="/search" method="post">
            <input type="text" name="query"
placeholder="Enter your query" required>
            <input type="submit" value="Search">
        </form>
    </div>
</body>
</html>
```

Results.html

```
<!-- <!DOCTYPE html>
<html>
<head>
    <title>Results</title>
```

```
</head>
<body>
    <h1>Comments</h1>
    <ul>
    {% for comment, like_count in comments %}
        <li>{{ comment }} - Likes: {{ like_count }}</li>
    {% endfor %}
    </ul>
</body>
</html> -->
<!-- results.html -->
<!-- ...
{% if comments %}
<ul>
    {% for comment in comments %}
    <li>{{ comment }}</li>
    {% endfor %}
</ul>
{% endif %}
<form action="/ask_question" method="post">
    <input type="text" name="question" placeholder="Enter
your question">
    <input type="submit" value="Ask Question">
</form>
```

```
{% if answer %}
<div>{{ answer }}</div>
{% endif %} -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Results</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            margin: 0;
            padding: 20px;
        }
        .container {
            background-color: #fff;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
            padding: 20px;
            margin-bottom: 20px;
```

```
        }
        h1 {
            color: #333;
        }
        ul {
            list-style-type: none;
            padding: 0;
        }
        li {
            background-color: #e9e9e9;
            margin-bottom: 8px;
            padding: 10px;
            border-radius: 4px;
        }
        form {
            display: flex;
            flex-direction: column;
        }
        input[type="text"] {
            margin-bottom: 10px;
            padding: 10px;
            border: 1px solid #ddd;
            border-radius: 4px;
        }
        input[type="submit"] {
            cursor: pointer;
            padding: 10px 20px;
            background-color: #5cb85c;
            color: white;
            border: none;
            border-radius: 4px;
        }
        input[type="submit"]:hover {
            background-color: #4cae4c;
        }
        .answer {
            background-color: #d9edf7;
            border-color: #bce8f1;
            color: #31708f;
            padding: 10px;
            border-radius: 4px;
        }
    </style>
</head>
<body>
    <div class="container">
```

```
        <h1>Comments</h1>
        {% if comments %}
        <ul>
            {% for comment in comments %}
            <li>{{ comment }}</li>
            {% endfor %}
        </ul>
        {% endif %}
    </div>
    <div class="container">
        <h1>Ask a Question</h1>
        <form action="/ask_question" method="post">
            <input type="text" name="question"
placeholder="Enter your question">
            <input type="submit" value="Ask Question">
        </form>
    </div>
    {% if answer %}
    <div class="container answer">
        <h1>Answer</h1>
        <p>{{ answer }}</p>
    </div>
    {% endif %}
</body>
</html>
```

show_button.html

```
<form action="{{ url_for('generate_answer') }}"
method="get">
    <input type="hidden" name="question" value="{{ question
}}">
    <input type="submit" value="Show Answer">
</form>
```