Folder: api

Folder: __pycache__

File: app.py

Documented code for app.py:

```
from flask import Flask, request, request, render_template
from predict import predict_sentiments
from youtube import get_video_comments
from flask_cors import CORS
import requests
from urllib.parse import urlparse



app = Flask(__name__)
CORS(app)

def get_video(video_id):
    if not video_id:
        return {"error": "video_id is required"}

    comments = get_video_comments(video_id)
    predictions = predict_sentiments(comments)

    positive = predictions.count("Positive")
    negative = predictions.count("Negative")

    summary = {
        "positive": positive,
        "negative": negative,
        "num_comments": len(comments),
        "rating": (positive / len(comments)) * 100
    }

    return {"predictions": predictions, "comments": comments, "summary": summary}
```

```
def getvideo_id(value):
    """
    Examples:
    - http://youtu.be/SA2iWivDJiE
    - http://www.youtube.com/watch?v=_oPAwA_Udwc&feature=feedu
    - http://www.youtube.com/embed/SA2iWivDJiE
    - http://www.youtube.com/v/SA2iWivDJiE?version=3&amp;hl=en_US
    """
    query = urlparse(value)
    if query.hostname == 'youtu.be':
        return query.path[1:]
    if query.hostname in ('www.youtube.com', 'youtube.com'):
        if query.path == '/watch':
            p = urlparse(query.query)
            return str(p.path[2:]).split('&')[0]
        if query.path[:7] == '/embed/':
            return query.path.split('/')[2]
        if query.path[:3] == '/v/':
            return query.path.split('/')[2]
    # fail?
    return None


@app.route('/', methods=['GET', 'POST'])
def index():
    summary = None
    comments = []
    if request.method == 'POST':
        video_url = request.form.get('video_url')
        video_id = getvideo_id(video_url)
        print(video_id)
        data = get_video(video_id)
```

```
        summary = data['summary']
        comments = list(zip(data['comments'], data['predictions']))
    return render_template('index.html', summary=summary, comments=comments)

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

File: predict.py

Documented code for predict.py:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
import pickle

VOCAB_SIZE = 20000
MAX_LEN = 250
MODEL_PATH = "api\\sentiment_analysis_model.h5"

# Load the saved model
model = load_model(MODEL_PATH)

# Load the tokenizer
with open('api\\tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)


def encode_texts(text_list):
    encoded_texts = []
    for text in text_list:
```

```
        tokens = tf.keras.preprocessing.text.text_to_word_sequence(text)
        tokens = [tokenizer.word_index.get(word, 0) for word in tokens]
        encoded_texts.append(tokens)
                    return pad_sequences(encoded_texts, maxlen=MAX_LEN, padding='post', value=VOCAB_SIZE-1)


def predict_sentiments(text_list):
    encoded_inputs = encode_texts(text_list)
    predictions = np.argmax(model.predict(encoded_inputs), axis=-1)
    sentiments = []
    for prediction in predictions:
        if prediction == 0:
            sentiments.append("Negative")
        elif prediction == 1:
            sentiments.append("Neutral")
        else:
            sentiments.append("Positive")
    return sentiments
```

Folder: static

Folder: templates

File: index.html

Documented code for 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 Sentiment Analysis</title>
    <style>
```

```
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    text-align: center;
    margin: 50px 20px;
    background-color: #0f0e13;
    background-image:
      radial-gradient(at 0% 100%, hsla(253, 16%, 7%, 1) 0, transparent 50%),
      radial-gradient(at 50% 0%, hsla(225, 39%, 25%, 1) 0, transparent 50%);
    color: #333; /* Dark text for better visibility */
}

header {
    margin-bottom: 20px;
    position: relative;
}

.logo {
    width: 150px; /* Adjust the width of the logo */
    height: auto;
    margin-right: 10px;
}

.container {
    background-color: rgba(255, 255, 255, 0.9); /* Slightly transparent white background */
    background-image: radial-gradient(at 0% 0%, hsla(253, 16%, 7%, 0.2) 0, transparent 50%),
        radial-gradient(at 50% 100%, hsla(225, 39%, 25%, 0.2) 0, transparent 50%);
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    margin: 20px auto;
    max-width: 800px;
    overflow: hidden; /* Hide the vertical scrollbar for the main container */
}
```

```
.comments-container {
    max-height: 400px; /* Adjust the maximum height for the comments container */
    overflow-y: auto; /* Enable vertical scrolling for the comments container */
}

form {
    margin-bottom: 20px;
}

input[type="text"] {
    padding: 10px;
    width: 70%;
    border: 1px solid #3498db;
    border-radius: 5px;
    margin-right: 10px;
    box-sizing: border-box;
    color: #333;
    font-weight: bold; /* Bold text for better visibility */
}

input[type="submit"] {
    padding: 10px;
    background-color: #3498db;
    color: #fff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

table {
    margin: 20px auto;
    border-collapse: collapse;
```

```
            width: 100%;
        }

        th,
        td {
            padding: 10px;
            border: 1px solid #ddd;
            word-wrap: break-word;
            color: #333;
            font-weight: bold; /* Bold text for better visibility */
        }

        h2,
        h3 {
            color: #3498db;
            font-weight: bold; /* Bold text for better visibility */
        }
    </style>
</head>

<body>

    <header>
        <!-- Place your logo here -->
        <img src="{{url_for('static', filename='logo1.png')}}" alt="Logo" class="logo"/>
        <h1 style="color: #3498db;">YouTube Sentiment Analysis</h1>
    </header>

    <div class="container">
        <form id="analysisForm" action="/" method="post">
            <input type="text" name="video_url" placeholder="Enter YouTube URL" required>
            <input type="submit" value="Analyze">
        </form>
```

```
    {% if summary %}
    <div id="summarySection">
        <h2>Summary</h2>
        <p>Positive: {{ summary['positive'] }}</p>
        <p>Negative: {{ summary['negative'] }}</p>
        <p>Number of Comments: {{ summary['num_comments'] }}</p>
        <p>Rating: {{ summary['rating'] }}%</p>
    </div>
    <div class="comments-container" id="commentsContainer">
        <h3>Comments</h3>
        <table>
            <tr>
                <th>Comment</th>
                <th>Sentiment</th>
            </tr>
            {% for comment, sentiment in comments %}
            <tr>
                <td>{{ comment }}</td>
                <td>{{ sentiment }}</td>
            </tr>
            {% endfor %}
        </table>
    </div>
    {% endif %}
</div>

<script>
    // Adjust the height of the comments container based on the available space
    function adjustCommentsContainerHeight() {
        var container = document.getElementById('commentsContainer');
        var windowHeight = window.innerHeight;
        var headerHeight = document.querySelector('header').offsetHeight;
```

```
        var summaryHeight = document.getElementById('summarySection').offsetHeight;
            var containerHeight = windowHeight - headerHeight - summaryHeight - 40; // Adjust as needed
        container.style.maxHeight = containerHeight + 'px';
    }

    // Call the function on window resize
    window.onresize = adjustCommentsContainerHeight;

    // Call the function on page load
    window.onload = adjustCommentsContainerHeight;
  </script>

</body>

</html>
```

File: youtube.py

Documented code for youtube.py:

```
import os
import googleapiclient.discovery
import googleapiclient.errors
import os
from dotenv import load_dotenv

load_dotenv()
api_key = os.getenv("API_KEY")

def get_comments(youtube, **kwargs):
    comments = []
    results = youtube.commentThreads().list(**kwargs).execute()
```

```
    while results:
        for item in results['items']:
            comment = item['snippet']['topLevelComment']['snippet']['textDisplay']
            comments.append(comment)

        # check if there are more comments
        if 'nextPageToken' in results:
            kwargs['pageToken'] = results['nextPageToken']
            results = youtube.commentThreads().list(**kwargs).execute()
        else:
            break

    return comments

def main(video_id, api_key):
    # Disable OAuthlib's HTTPs verification when running locally.
    os.environ["OAUTHLIB_INSECURE_TRANSPORT"] = "1"

    youtube = googleapiclient.discovery.build(
        "youtube", "v3", developerKey=api_key)

    comments = get_comments(youtube, part="snippet", videoId=video_id, textFormat="plainText")
    return comments


def get_video_comments(video_id):
    return main(video_id, api_key)
```

File: training.py

Documented code for training.py:

```
import os
```

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding, GlobalAveragePooling1D
from keras.preprocessing.sequence import pad_sequences
import pickle
import pandas as pd

# Parameters
VOCAB_SIZE = 20000
MAX_LEN = 250
EMBEDDING_DIM = 16
MODEL_PATH = 'sentiment_analysis_model.h5'

file_path = 'data.csv'
data = pd.read_csv(file_path, encoding='ISO-8859-1')
df_shuffled = data.sample(frac=1).reset_index(drop=True)

texts = []
labels = []

for _, row in df_shuffled.iterrows():
    texts.append(row[-1])
    label = row[0]
    labels.append(0 if label == 0 else 1 if label == 2 else 2)

texts = np.array(texts)
labels = np.array(labels)

# Tokenize and pad the sequences
# Filter tokens in the tokenizer to keep only top VOCAB_SIZE-1 words
tokenizer = keras.preprocessing.text.Tokenizer(num_words=VOCAB_SIZE-1, oov_token='<OOV>')
```

```
tokenizer.fit_on_texts(texts)

# Reduce the vocabulary size by updating the tokenizer's word_index and word_counts
tokenizer.word_index = {word: idx for word, idx in tokenizer.word_index.items() if idx < VOCAB_SIZE - 1}
tokenizer.word_counts = {word: count for word, count in tokenizer.word_counts.items() if tokenizer.word_index.get(word)}

# Debug: Check max index value
print("Max index value:", max(tokenizer.word_index.values()))
print("VOCAB_SIZE:", VOCAB_SIZE)

sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=MAX_LEN, value=VOCAB_SIZE-1, padding='post')

# Debug: Check a sample padded sequence
print("Sample padded sequence:", padded_sequences[0])

# Save the tokenizer to a file
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

# Split data into training and test sets (you might want to do this in a more balanced way)
train_data = padded_sequences[:-5000]
test_data = padded_sequences[-5000:]
train_labels = labels[:-5000]
test_labels = labels[-5000:]

# Check if saved model exists
if os.path.exists(MODEL_PATH):
    print("Loading saved model...")
    model = load_model(MODEL_PATH)
```

```
else:
    print("Training a new model...")
    # Define the model
    model = Sequential([
        Embedding(VOCAB_SIZE, EMBEDDING_DIM, input_length=MAX_LEN),
        GlobalAveragePooling1D(),
        Dense(16, activation='relu'),
        Dense(3, activation='softmax')  # 3 classes: negative, neutral, positive
    ])

    # Compile the model
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    # Train the model
    model.fit(train_data, train_labels, epochs=10, batch_size=32, validation_split=0.2)

    # Save the trained model
    model.save(MODEL_PATH)

# Evaluate on test data
loss, accuracy = model.evaluate(test_data, test_labels)
print(f"Test accuracy: {accuracy * 100:.2f}%")

# Interactive loop for predictions
def encode_text(text):
    tokens = tf.keras.preprocessing.text.text_to_word_sequence(text)
    tokens = [tokenizer.word_index[word] if word in tokenizer.word_index else 0 for word in tokens]
    return pad_sequences([tokens], maxlen=MAX_LEN, padding='post', value=VOCAB_SIZE-1)

while True:
    user_input = input("Enter a sentence for sentiment analysis (or 'exit' to quit): ")
    if user_input.lower() == 'exit':
        break
```

```
    encoded_input = encode_text(user_input)
    prediction = np.argmax(model.predict(encoded_input))

    if prediction == 0:
        print("Sentiment: Negative")
    elif prediction == 1:
        print("Sentiment: Neutral")
    else:
        print("Sentiment: Positive")

# you need to download the training data from here: https://www.kaggle.com/datasets/kazanova/sentiment140?resource=download
```