Python Internship - Coding Assignment #8: “Keerthi Challagundla”

1.Research

a. Keyword Research Detection

The technique of locating particular words or phrases within a broader text, audio, or speech discussion is known as keyword detection. It is also referred to as keyword spotting or keyword recognition. These pre-selected keywords work as indicators or start-ups for more research or analysis. The following are a few simple options for keyword detection:

i)Google Speech-To-Text:

Google’s Speech-to-Text API can be used for keyword detection in audio files. It provides speech recognition capabilities and can return transcribed text.

Pros-Easy to use, High accuracy.

Cons-It may come with associated costs.

ii)Mozilla DeepSpeech:

Mozilla DeepSpeech is an open-source automatic speech recognition (ASR)engine.

It can be used via the DeepSpeech API.

iii)Microsoft Azure Cognitive Services:

Microsoft’s Azure Cognitive Services includes a Speech Service that can be used for speech recognition and potentially keyword detection.

Pros-Integration with Azure services, easy to use.

Cons-Cost may vary based on usage.

b. Research Documentation

It is a challenging effort to create a comparative presentation that details the benefits and drawbacks of each keyword detection system, as well as expected accuracy and costs for a data volume of 100,000 minutes. You can use the condensed table style I've provided below as a jumping off point for your presentation. Please take note that depending on the precise project needs and dataset quality, anticipated accuracy and prices may differ dramatically. For accurate figures, you might need to carry out extensive testing and analysis.

Solution	Pros	Cons	Projected Accuracy	Projected Costs
Google Speech-to-Text	-High Accuracy -Easy to use -Good support and documentation	-Costs associated -Limited customization	Estimated high	Estimated cost
Mozilla DeepSpeech	-Open-source	-Requires customization	Estimated moderate	Estimated low

Microsoft Azure Cognitive Services	-Good support and documentation -Integration with Azure Services	-Costs associated -Microsoft cloud platform dependency	Estimated high	Estimated cost

- “pros” and “cons” provide a high-level overview of advantages and disadvantages of each solution.
- "Projected Accuracy" and "Projected Costs" are based on rough estimations and should be refined with actual testing and project specifics.

2.Prototype

a. Input

i) Implement a python script that takes audio file as input.

ii) Users should be able to upload their own audio files.

iii) Input a suitable dataset from these options-

https://huggingface.co/blog/audio-datasets

You may use Flask to build a web application that implements a Python script that lets users upload their own audio files and, optionally, access a dataset. An easy example of how to put up such a script is provided below:

Requirements-

- Python.
- Flask (web framework).
- Pandas (for dataset access, if needed).

Steps-

1.Install Flask and Pandas

```
Pip install flask pandas
```

2.Create the flask web application

```
from flask import Flask, render_template, request, redirect, url_for, flash
import os
import pandas as pd


app = Flask(__name__)
app.secret_key = 'your_secret_key'
```

```
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

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

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'audio' not in request.files:
        flash('No file part')
        return redirect(request.url)

    audio_file = request.files['audio']

    if audio_file.filename == '':
        flash('No selected file')
        return redirect(request.url)

    if audio_file:
        audio_file.save(os.path.join(app.config['UPLOAD_FOLDER'], audio_file.filename))

        flash('File uploaded successfully')
        return redirect(request.url)

@app.route('/access_dataset', methods=['GET'])
```

```
def access_dataset():
    dataset_path = "C:\Users\challagundla\Downloads\iris .csv"
    dataset = pd.read_csv(dataset_path)

    return dataset.to_html()

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

3.Create an HTML template for file upload

```
<!DOCTYPE html>
<html>
<head>
    <title>Upload Audio</title>
</head>
<body>
    <h2>Upload an audio file</h2>
    <form method="POST" action="/upload" enctype="multipart/form-data">
        <input type="file" name="audio">
        <input type="submit" value="Upload">
    </form>

    <h2>Access a Dataset</h2>
    <a href="/access_dataset">Access Dataset</a>
</body>
</html>
```

4.Run the flask application

```
Python app.py
```

5. Visit http://localhost:5000/ in your web browser. You will see a webpage that allows users to upload audio files and access the dataset (you can replace 'your_dataset.csv' with your preferred dataset).

b. Emotion Detection

You may use Python and well-known libraries like ‘Librosa’ for feature extraction and ‘scikit-learn’ for machine learning to carry out emotion detection on audio data and categorize it as either "happy" or "sad." The primary audio feature in this case is MFCC (Mel-frequency cepstral coefficients):

Requirements:

- Python
- Librosa
- Scikit-learn
- Other libraries: NumPy, Pandas, Matplotlib

Steps:

1)Install the required libraries:

```
pip install librosa scikit-learn numpy pandas matplotlib
```

2)Create a python script

```
import os
import librosa
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
def extract_audio_features(audio_file):
    y, sr = librosa.load(audio_file)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    return np.mean(mfccs, axis=1)
def load_and_process_dataset(dataset_path):
    dataset = pd.read_csv(dataset_path)
    audio_features = []
```

```
    for index, row in dataset.iterrows():
        audio_file = row[' "C:\Users\challagundla\Downloads\mixkit-game-show-suspense-waiting-667.wav" ']
        emotion = row['emotion']

        features = extract_audio_features(audio_file)
        audio_features.append(features)

    return np.array(audio_features), np.array(dataset['emotion'])
if __name__ == "__main__":
    dataset_path = 'C:\Users\challagundla\Downloads\iris .csv'
    X, y = load_and_process_dataset(dataset_path)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    classifier = RandomForestClassifier(n_estimators=100, random_state=42)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy}")
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)
    emotion_labels = ["happy", "sad"]
    predicted_emotions = [emotion_labels[y] for y in y_pred]
    plt.bar(emotion_labels, [predicted_emotions.count('happy'), predicted_emotions.count('sad')])
    plt.xlabel("Emotion")
```

```
plt.ylabel("Count")
plt.title("Predicted Emotions")
plt.show()
```

3) Dataset Format:

Make sure that the dataset (referred to in the script as "your_dataset.csv") has at least two columns: "audio_path" (which provides the file names to the audio samples), and "emotion" (which contains labels like "happy" or "sad").

4) Run the Script:

When the script is run, the dataset is loaded, audio features (in this case, MFCCs) are extracted, the dataset is divided into training and testing sets, a straightforward Random Forest classifier is trained, and finally the model's accuracy is assessed.

C. Keyword Detection

You must decide which approach is best for keyword identification depending on the demands of your particular project before incorporating it into your script. I'll describe how to use OpenAI's GPT-3 to incorporate keyword detection into your script. The GPT-3 language model is adaptable and can be used for keyword detection tasks.

Selecting GPT-3 for keyword detection:

1)Sign up for GPT-3 API:

If you haven't already sign up for the OpenAI GPT-3 API and obtain an API key.

2)Set up OpenAI Python SDK: Install the OpenAI Python SDK

```
Pip install openai
```

3)Integrate GPT-3 for Keyword Detection:

GPT-3 can be included into your Python script (such as app.py) to help you find keywords. Here is a straightforward illustration of how you could use it.

```
import openai
api_key = 'YOUR_API_KEY'
def detect_keywords(text, keywords):
    prompt = f"Find the following keywords in the text:\n\n{text}\n\nKeywords: {', '.join(keywords)}"
    response = openai.Completion.create(
        engine="davinci",
        prompt=prompt,
        max_tokens=150,  # Adjust as needed
```

```
        api_key=api_key
    )
    return response.choices[0].text.strip()
input_text = "This is a sample text where we want to find keywords like Python and AI."
target_keywords = ["Python", "AI"]
detected_keywords = detect_keywords(input_text, target_keywords)
print(f"Detected Keywords: {detected_keywords}")
```

4)Run keyword detection:

The detect_keywords function will use GPT-3 to locate the requested keywords in the input text when you run the script.

5)Integrate with Your Existing Script:

This keyword detection feature can be incorporated into your current script for emotional analysis. For instance, you can send the text you extracted from the dataset after transcribing the audio to the detect_keywords function.

Don't forget to substitute your actual GPT-3 API key for "YOUR_API_KEY". This is a simplistic example, and you can change the question, the maximum tokens, and other factors to suit your needs.

Due to GPT-3's adaptability, you can find a variety of keywords in text data. The content and the keywords can be customized to your project's needs.

d.Printing Results

Utilizing Python packages like Pandas for data management and Matplotlib for data visualization, you can publish the results of anticipated emotions and keyword recognition as tables and charts. How to do each of these tasks is provided below:

1)Print Predicted Emotions as a Table:

```
import pandas as pd
audio_files = ["audio1.wav", "audio2.wav", "audio3.wav"]
predicted_emotions = ["happy", "sad", "happy"]
data = {'Audio File': audio_files, 'Predicted Emotion': predicted_emotions}
df_emotions = pd.DataFrame(data)
print(df_emotions)
```

2)Show Predicted Emotions in a Smooth Line Chart:

You need time data and the anticipated feelings in order to generate a line chart showing the evolution of emotions. You may plot it as a line chart if you know the timestamps for each emotion that was predicted.

```
import matplotlib.pyplot as plt

timestamps = [0, 10, 20]  # Replace with actual timestamps

emotions = ["happy", "sad", "happy"]  # Replace with actual emotions

emotions_numeric = [1 if e == "happy" else -1 for e in emotions]

plt.plot(timestamps, emotions_numeric, marker='o', linestyle='-')

plt.xlabel("Time")

plt.ylabel("Emotion")

plt.yticks([-1, 1], ["sad", "happy"])

plt.title("Emotions over Time")

plt.grid(True)

plt.show()
```

3)Print Predicted Keyword Detection as a Table:

Use pandas to create a table for keyword detection results:

```
audio_files = ["audio1.wav", "audio2.wav", "audio3.wav"]

detected_keywords = ["Python, AI", "Machine learning", "Data analysis"]

data = {'Audio File': audio_files, 'Detected Keywords': detected_keywords}

df_keywords = pd.DataFrame(data)

print(df_keywords)
```

4)Show predicted keyword detection in a smooth line chart:

The line chart for keyword detection is often not smooth. However, you can see how particular keywords appear or disappear over time:

```
import matplotlib.pyplot as plt

timestamps = [0, 10, 20]  # Replace with actual timestamps

keywords = ["Python, AI", "Machine learning", "Data analysis”]

for i, keyword in enumerate(keywords):
```

```
if keyword:
    plt.plot([timestamps[i]], [1], marker='o', label=keyword)
plt.xlabel("Time")
plt.yticks([1], ["Keywords Detected"])
plt.title("Keyword Detection over Time")
plt.grid(True)
plt.legend()
plt.show()
```

e. Prototype Documentation

i) Approach

In this prototype, we've created a Python script that analyses audio files for emotions, finds keywords in text, and displays the findings. An summary of the strategy is given below:

- **Emotion Detection**
  To evaluate the audio files, we employ methods for extracting audio features, such as MFCCs. We develop a straightforward Random Forest classifier to categorize the audio's emotions as "happy" or "sad."
- **Keyword Detection**
  To find specific terms inside the text data, we use OpenAI's GPT-3. A strong language model that can comprehend and respond to natural language cues is GPT-3.
- **Visualization**
  Users can analyse the data with the aid of visuals we offer, such as line charts that display emotions and keyword identification over time.

Features Chosen:

- We decided to extract MFCCs from audio files in order to discern emotions. For voice and audio analysis, these Mel-frequency cepstral coefficients are frequently employed. They record the audio's spectral properties.

- We make use of GPT-3's natural language comprehension abilities for keyword detection. The keywords that users wish GPT-3 to look for in the text data can be specified by the users.

Preprocessing Steps:

- The ‘librosa’ library is used to load the audio files for audio feature extraction. Following the audio processing, MFCCs are extracted and used for emotion analysis.

- We make use of the natural language comprehension capabilities of GPT-3 for keyword detection. GPT-3 is given the text data in a structured prompt that tells it to look for particular keywords.

Libraries and Tools Used:

- For this prototype, Python was chosen as the programming language.
    - We used the 'librosa' package to extract and analyse audio feature information.
    - We used'scikit-learn' for machine learning and emotion recognition.
    - Using the 'openai' Python SDK, we integrated OpenAI's GPT-3 for keyword identification.
    - We utilized packages like "Pandas" and "Matplotlib" for data management and visualization.

Challenges Faced:

- Preprocessing audio data can be difficult, especially when trying to extract important elements. It necessitates audio signal processing knowledge.

- Model selection: It's critical to pick the best machine learning model for emotion recognition. Although more complex models can be investigated, we choose to use a straightforward Random Forest classifier.

- Data format: It can be difficult to ensure that the audio and text data are in the right format for processing, especially when working with real-world data.

ii) User Manual:

Uploading Audio Files:

- Users ought to prepare their audio files in a format that is supported, like WAV, MP3, or another popular audio format.

- Put the audio files in the appropriate directory or folder.

- The path to the folder containing the audio files should be specified in the script. Make sure to adjust the course as necessary.

Running the Script:

- Execute the script to analyse the audio files, identify emotions, and find keywords.

- Based on the audio files that have been provided, the script will produce results.

Accepted Audio File Formats:

- WAV and MP3 audio files, among others, can be handled by the script. The script can be altered as necessary to support different formats.

Printing Result:

- The results of the anticipated emotions and keyword detection will be printed as tables in the console by the script.

- Additionally, line charts will be produced to show the evolution of emotions and keyword detection over time.

- The results can be viewed by users directly in the console or exported to a file for additional investigation or reporting.