Speech Processing Dashboard Report

Table of Contents

---

1. Overview

The Speech Processing Dashboard is a Python-based web application built using the Dash framework. It allows users to upload audio or video files, process them, and analyze the content through several natural language processing (NLP) techniques.

The dashboard provides transcription services using **Wav2Vec2**, a state-of-the-art speech-to-text model by Facebook, along with additional features such as keyword analysis, topic modeling, language detection, sentiment analysis, and text summarization.

---

2. Application Workflow

1. **File Upload**: Users upload an audio or video file.
2. **Audio Extraction (for video files)**: For video files, the application extracts the audio track.
3. **Audio Processing**: The audio is transcribed to text using the **Wav2Vec2** model.

4. **Analysis**: The transcribed text undergoes further analysis to identify keyword frequencies, topics, and sentiment, and is summarized for quick reading.
5. **Visualization**: Key outputs, such as keyword frequency charts and LDA topic visualizations, are displayed.

3. Key Features

1. **Audio Transcription**: Converts spoken content into written text.
2. **Keyword Frequency Analysis**: Identifies frequently mentioned words.
3. **Topic Modeling**: Uses LDA to identify topics in the transcription.
4. **Language Detection**: Detects the language used in the transcribed text.
5. **Sentiment Analysis**: Analyzes the emotional tone (positive, negative, or neutral).
6. **Text Summarization**: Summarizes the main points of the transcription.

4. Code Overview

Main Application Code

The main dashboard application uses Python libraries such as **Dash** for building the web interface, **transformers** for Wav2Vec2 speech-to-text, and **gensim** for topic modeling.

```
import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import dash_bootstrap_components as dbc

import base64

import io

import tempfile

import os
import torch
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor, pipeline
import librosa
import jiwer
from collections import defaultdict
import spacy
import plotly.graph_objs as go
import gensim
from gensim.corpora.dictionary import Dictionary
from gensim.models import LdaMulticore
import en_core_web_md
import pyLDAvis.gensim_models
import pyLDAvis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from langdetect import detect
import moviepy.editor as mp

# Load models
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-large-960h")
model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-large-960h")
nlp = en_core_web_md.load()

# Sentiment Analysis Pipeline
sentiment_analyzer = pipeline("sentiment-analysis")
```

```
# Summarization Pipeline
summarizer = pipeline("summarization")

# Initialize Dash app
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])

# Layout for the dashboard
app.layout = dbc.Container([
    dbc.Row([dbc.Col(html.H1("Speech Processing Dashboard",
className="text-center mb-4"), width=12)]),

    dbc.Row([dbc.Col([dbc.Card([dbc.CardHeader("Upload Media File"),
                                dbc.CardBody(dcc.Upload(id='upload-media',
children=html.Button('Upload Media File'),
                                                       multiple=False,
accept=".mp4,.avi,.mov,.mp3"))]),], width=12)], className="mb-4"),

    dbc.Row([
        dbc.Col([dbc.Card([dbc.CardHeader("Transcription"),
                            dbc.CardBody(html.Div(id='transcription-output'))]),], width=6),

        dbc.Col([dbc.Card([dbc.CardHeader("Keyword Frequencies"),
                            dbc.CardBody(dcc.Graph(id='frequencies-barchart'))]),],
width=6)
    ]),
    dbc.Row([
        dbc.Col([dbc.Card([dbc.CardHeader("Word Error Rate (WER) and Character Error Rate (CER)"),
                            dbc.CardBody(html.Div(id='metrics-output'))]),], width=6),

        dbc.Col([dbc.Card([dbc.CardHeader("Word Count"),
                            dbc.CardBody(html.Div(id='word-count-output'))]),], width=6)
    ], className="mb-4"),

    dbc.Row([dbc.Col([dbc.Card([dbc.CardHeader("Topic Modeling"),
                                dbc.CardBody(html.Div(id='topics-output'))]),], width=12)], className="mb-4"),

    dbc.Row([dbc.Col([dbc.Card([dbc.CardHeader("Language Detection"),
                                dbc.CardBody(html.Div(id='language-output'))]),], width=6),

              dbc.Col([dbc.Card([dbc.CardHeader("Sentiment Analysis"),
                                dbc.CardBody(html.Div(id='sentiment-output'))]),], width=6)
    ], className="mb-4"),

    dbc.Row([dbc.Col([dbc.Card([dbc.CardHeader("Text Summarization"),
                                dbc.CardBody(html.Div(id='summary-output'))]),], width=12)], className="mb-4"),

    dbc.Row([dbc.Col([dbc.Card([dbc.CardHeader("LDA Visualization"),
                                dbc.CardBody(html.Iframe(id='lda-vis', srcDoc="", width='100%', height='600'))]),], width=12)])
```

```
])

# Helper functions
def save_uploaded_file(uploaded_file):
    if uploaded_file is None:
        raise ValueError("No file uploaded.")

    content_type, content_string = uploaded_file.split(',')
    decoded = base64.b64decode(content_string)

    with tempfile.NamedTemporaryFile(delete=False, suffix=".mp4") as tmp_file:
        tmp_file.write(decoded)
        tmp_file_path = tmp_file.name

    return tmp_file_path

def extract_audio_from_video(video_file_path):
    try:
        video = mp.VideoFileClip(video_file_path)
        audio = video.audio
        audio_file_path = video_file_path.replace(".mp4", ".wav")
        audio.write_audiofile(audio_file_path)
        return audio_file_path
    except Exception as e:
        print(f"Error in audio extraction: {e}")
        return None
```

```
def transcribe_audio(file_path):
    try:
        print("audio1")
        # Check if the file exists and is accessible
        if not os.path.isfile(file_path):
            raise FileNotFoundError(f"Audio file not found: {file_path}")

        # Load audio file
        speech, sr = librosa.load(file_path, sr=16000)
        print(f"Loaded audio file: {file_path} (Sample Rate: {sr}, Audio Length: {len(speech)})")

        # Process the audio file for transcription
        input_values = processor(speech, return_tensors="pt", sampling_rate=16000).input_values

        with torch.no_grad():
            logits = model(input_values).logits

        predicted_ids = torch.argmax(logits, dim=-1)
        transcription = processor.decode(predicted_ids[0])
        return transcription
    except Exception as e:
        print(f"Error in transcription: {e}")
        return None
```

```
def perform_lda(transcription, num_topics=2):
    removal = ['ADV', 'PRON', 'CCONJ', 'PUNCT', 'PART', 'DET', 'ADP', 'SPACE',
'NUM', 'SYM']
    tokens = []

    for summary in nlp.pipe(transcription.split()):
        proj_tok = [token.lemma_.lower() for token in summary if token.pos_ not
in removal and not token.is_stop and token.is_alpha]
        tokens.append(proj_tok)

    cleaned_nested_list = [inner_list for inner_list in tokens if inner_list]

    dictionary = Dictionary(cleaned_nested_list)
    corpus = [dictionary.doc2bow(text) for text in cleaned_nested_list]

    ldamodel = LdaMulticore(corpus, num_topics=num_topics,
id2word=dictionary, passes=15)
    topics = ldamodel.print_topics(num_words=3)
    return ldamodel, dictionary, corpus, topics

def generate_lda_vis(ldamodel, corpus, dictionary):
    lda_vis_data = pyLDAvis.gensim_models.prepare(ldamodel, corpus,
dictionary, sort_topics=False)
    html_data = pyLDAvis.prepared_data_to_html(lda_vis_data)
    return html_data

def keyword_frequencies(transcription, top_n=10):
    removal_tags = ['ADV', 'PRON', 'CCONJ', 'PUNCT', 'PART', 'DET', 'ADP', 'SPACE', 'NUM', 'SYM']
    tokens = []

    for summary in nlp.pipe(transcription.split()):
        proj_tok = [token.lemma_.lower() for token in summary if token.pos_ not in removal_tags and not token.is_stop and token.is_alpha]
        tokens.extend(proj_tok)

    word_freq = defaultdict(int)
    for word in tokens:
        word_freq[word] += 1

    sorted_freq = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)[:top_n]
    return dict(sorted_freq)

def plot_keyword_frequencies(frequencies):
    keywords = list(frequencies.keys())
    counts = list(frequencies.values())

    fig = go.Figure([go.Bar(x=keywords, y=counts)])
    fig.update_layout(
        title='Keyword Frequencies',
        xaxis_title='Keywords',
        yaxis_title='Frequency',
        template='plotly_dark'
    )
    return fig

def detect_language(transcription):
    try:
        language = detect(transcription)
        return f"Detected Language: {language}"
    except Exception as e:
        print(f"Language detection failed: {e}")
        return "Language detection failed."

def analyze_sentiment(transcription):
    try:
        sentiment = sentiment_analyzer(transcription)
        return f"Sentiment: {sentiment[0]['label']} (Score: {sentiment[0]['score']:.4f})"
    except Exception as e:
        print(f"Sentiment analysis failed: {e}")
        return "Sentiment analysis failed."

def summarize_text(text):
    try:
        summarized_text = summarizer(text, max_length=100, min_length=30, do_sample=False)
        return summarized_text[0]['summary_text']
    except Exception as e:
        print(f"Error in summarization: {e}")
        return f"Error in summarization: {e}"


@app.callback(
    [Output('transcription-output', 'children'),
     Output('metrics-output', 'children'),
     Output('frequencies-barchart', 'figure'),
     Output('word-count-output', 'children'),
     Output('topics-output', 'children'),
     Output('lda-vis', 'srcDoc'),
     Output('language-output', 'children'),
     Output('sentiment-output', 'children'),
     Output('summary-output', 'children')],
    [Input('upload-media', 'contents')]
)
def update_dashboard(uploaded_file):
    if uploaded_file is None:
        return "No file selected.", "", {}, "", "", "", "", "", ""


    try:
        media_file_path = save_uploaded_file(uploaded_file)
        file_extension = media_file_path.split('.')[-1].lower()
        print(media_file_path)
        print(file_extension)
        # Check file type and process accordingly
        if file_extension in ['mp4', 'avi', 'mov']:
            print("video")
            audio_file_path = extract_audio_from_video(media_file_path)
        elif file_extension == 'mp3':
            print("audio")
            audio_file_path = media_file_path
        else:
            return "Unsupported file format.", "", {}, "", "", "", "", "", ""

        if audio_file_path is None:
            return "Error extracting audio.", "", {}, "", "", "", "", "", ""

        # Transcription
        transcription = transcribe_audio(audio_file_path)
        if transcription is None:
            return "Error in transcription.", "", {}, "", "", "", "", "", ""

        # WER and CER Calculation
        ground_truth = "This is the actual transcription text."  # Replace with
actual ground truth
        wer_audio = jiwer.wer(ground_truth, transcription)
        cer_audio = jiwer.cer(ground_truth, transcription)
        metrics_output = f"WER: {wer_audio:.4f} | CER: {cer_audio:.4f}"

        # Word Count
        word_count = len(transcription.split())
        word_count_output = f"Total Words: {word_count}"

        # Keyword Frequencies and Bar Chart
```

```
        keyword_freq = keyword_frequencies(transcription)
        freq_chart = plot_keyword_frequencies(keyword_freq)

        # Topic Modeling
        ldamodel, dictionary, corpus, topics = perform_lda(transcription, num_topics=3)
        topics_output = html.Ul([html.Li(f"Topic {i + 1}: {topic}") for i, topic in enumerate(topics)])

        # LDA Visualization
        lda_vis_html = generate_lda_vis(ldamodel, corpus, dictionary)

        # Language Detection
        language_output = detect_language(transcription)

        # Sentiment Analysis
        sentiment_output = analyze_sentiment(transcription)

        # Summarization
        summary_output = summarize_text(transcription)

        return transcription, metrics_output, freq_chart, word_count_output, topics_output, lda_vis_html, language_output, sentiment_output, summary_output

    except Exception as e:
        print(f"An error occurred: {e}")
        return f"An error occurred: {e}", "", {}, "", "", "", "", "", ""


# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

5. Application Components

The application's layout is split into various sections:

1. **File Upload**: Users upload the audio or video files for processing.
2. **Transcription Display**: Shows the text version of the uploaded media's speech.
3. **Keyword Frequency Bar Chart**: Displays a bar chart with the most frequent keywords.
4. **Topic Modeling**: Lists topics identified in the transcription.
5. **Language Detection**: Shows the detected language of the transcription.
6. **Sentiment Analysis**: Displays the sentiment (positive, neutral, or negative).
7. **Summarization**: Provides a summary of the transcription.

6. Sample Usage and Results

Example 1: Uploading an Audio File

1. **Upload Audio**: Select an .mp3 file for upload.
2. **Transcription**: "This is a sample transcription."
3. **Keyword Frequencies**: {'sample': 1, 'transcription': 1}

4. **Topics**: ['sample', 'transcription']
5. **Sentiment**: Neutral (Score: 0.00)
6. **Summary**: "This is a sample transcription."

Example 2: Uploading a Video File

1. **Upload Video**: Select a .mp4 file.
2. **Transcription**: "This video shows a sample."
3. **Keyword Frequencies**: {'video': 1, 'sample': 1}
4. **Topics**: ['video', 'sample']
5. **Sentiment**: Positive (Score: 0.80)
6. **Summary**: "This video shows a sample."

---

7. Conclusion

The Speech Processing Dashboard offers a comprehensive tool for audio and video processing using state-of-the-art machine learning models. By leveraging models like **Wav2Vec2**, **LDA**, and natural language processing techniques, the dashboard allows users to extract meaningful insights from audio files quickly and efficiently. The code structure is modular, with each component focusing on a specific part of the analysis, ensuring flexibility and expandability.

This tool is particularly valuable for applications in media analysis, content monitoring, and audio-to-text data extraction.