A.R.J COLLEGE OF ENGINEERING& TECHNOLOGY

EDAIYARNATHAM, MANNARGUDI-614 001 (APPROVED BY AICTE, NEW DELHI & AFFILIATED FOR ANNA UIVERSITY) (AN ISO 9001:2000 CERTIFIED INSITITUTION)

NM1104—SPEECH RECOGNITION TECHNIQUES

DEPARTMENT OF COMPUTER SCIENCE

SPEECH RECOGNITION PROJECTS

- **Speech to text transcription file using tkinter**
- **Emotion Classification from speech using MFCC features**
- **Accent Detection or Aware system**

GUIDED BY: AKBAR BASHAHAMEED

TEAM MEMBERS:

1. **M. SHOBEYA - 820122104058**
2. **U. SHARMILA -820122104057**
3. **S. SIVARANJANI-820122104060**
4. **C. SHRIWADHSAN-820122104059**
5. **R. SANTHOSH SIVAN-820122104056**

A.R.J COLLEGE OF ENGINEERING& TECHNOLOGY

EDAIYARNATHAM, MANNARGUDI-614 001 (APPROVED BY AICTE, NEW DELHI & AFFILIATED FOR ANNA UIVERSITY) (AN ISO 9001:2000 CERTIFIED INSITITUTION)

SPEECH-TO-TEXT TRANSCRIPTION SYSTEM USING TKINTER

CONTENTS

A.R.J COLLEGE OF ENGINEERING& TECHNOLOGY

EDAIYARNATHAM, MANNARGUDI-614 001 (APPROVED BY AICTE, NEW DELHI & AFFILIATED FOR ANNA UIVERSITY) (AN ISO 9001:2000 CERTIFIED INSITITUTION)

SPEECH-TO-TEXT TRANSCRIPTION SYSTEM USING TKINTER

ABSTRACT

This project aims to create a simple, desktop-based speech-to-text transcription system using Python's Tkinter library for the graphical user interface (GUI) and the Google Speech Recognition API for audio transcription. The system allows users to upload .wav or .mp3 audio files, converts speech to text, and evaluates the transcription quality with Word Error Rate (WER) and Character Error Rate (CER). This system is designed to provide a straightforward, offline solution for speech recognition tasks, and it serves as a demonstration of key concepts in audio processing and machine learning.

INTRODUCTION

Speech recognition technology plays a vital role in making applications more accessible, enhancing user interaction, and automating repetitive tasks. With advancements in machine learning, cloud-based services, and AI, speech recognition has become highly effective in various applications such as virtual assistants, real-time captioning, and voice-controlled systems. However, most existing solutions rely on internet connectivity or cloud infrastructure, which can be a limiting factor in certain use cases.

This project focuses on building an offline speech-to-text transcription tool with a desktop interface. The system is designed for ease of use, allowing users to transcribe audio files with minimal setup and no need for an active internet connection. The application also calculates the Word Error Rate (WER) and Character Error Rate (CER) as quality metrics to assess the accuracy of the transcription.

METHODOLOGY

1. Data Preprocessing

The data preprocessing phase focuses on preparing the audio files for transcription. Since speech recognition systems are sensitive to audio quality, this step aims to ensure the audio files are suitable for the transcription process.

- **File Format Check**: The system accepts audio files in .wav and .mp3 formats. If the input is in .mp3 format, it is converted to .wav using the pydub library to ensure compatibility with the speech recognition API.

- **Audio Conversion**: For .mp3 files, the system automatically converts the audio into .wav format using pydub and ffmpeg. This step is crucial because the speech_recognition library supports .wav files natively.
- **Sampling and Normalization**: Although not explicitly implemented in the current version, normalization and resampling to a consistent sample rate (16 kHz) can be added to improve accuracy across diverse audio sources.

2. Data Cleaning

Data cleaning involves removing any extraneous elements that may interfere with the accuracy of the transcription. This includes:

- **Silence Removal**: Audio files may contain periods of silence or non-speech audio. In real-world applications, this can lead to unnecessary transcription errors. Future work could involve removing silent sections from the audio before feeding them into the recognizer.
- **Noise Filtering**: Background noise and echo can distort the transcription process. While noise reduction is not currently implemented, it is a potential enhancement to improve transcription quality in noisy environments.
- **Format Standardization**: Ensuring that the audio is in a .wav format helps maintain consistency and compatibility with the transcription engine.

3. Model Training

The **speech recognition model** used in this system is Google's Speech-to-Text API, which leverages deep learning techniques for transcription. Although not a machine learning model that the user trains directly, this system utilizes Google's pre-trained models for speech-to-text tasks. The model has been trained on a wide variety of speech patterns and acoustic conditions, making it a robust solution for general transcription needs.

- **Data Feeding**: After preprocessing and cleaning, the audio is passed to the speech recognition model through the API.
- **API Call**: The system sends the audio file to Google's API for transcription. The audio is processed on Google's cloud servers, and the text result is returned.

4. Model Testing

Testing the transcription involves checking the system's ability to correctly recognize speech from different types of audio files. The system tests the following parameters:

- **Audio Quality**: The model is tested on audio files of different qualities to ensure that noise and low-quality recordings do not significantly affect the transcription accuracy.
- **Transcription Accuracy**: The transcription output is manually checked against the original speech to determine how accurately the system is performing.

5. Model Evaluation

The system evaluates the transcription quality using the following metrics:

- **Word Error Rate (WER)**: This metric measures the percentage of words that are incorrectly transcribed. It is computed by comparing the transcribed text against a reference (ground truth) text.
- **Character Error Rate (CER)**: This metric evaluates the number of character-level errors in the transcription, providing insights into the granularity of errors.
- **Simulated Metrics**: If no ground truth is available, simulated WER and CER are used for demonstration purposes. These metrics help users understand the potential accuracy of the transcription system.

EXISTING WORK

Various solutions for speech-to-text transcription exist, including:

1. **Cloud-based services**:
    - **Google Speech-to-Text**: A highly accurate API for converting speech to text using machine learning.
    - **IBM Watson Speech to Text**: Another powerful API offering transcription services with various customization options.
    - **Microsoft Azure Speech Services**: A cloud platform for speech recognition with integrated speaker identification and language detection.
    - **OpenAI Whisper**: A state-of-the-art model that can transcribe speech offline.
2. **Offline tools**:
    - **CMU Sphinx**: An open-source, offline speech recognition tool that can be trained for specific domains.
    - **DeepSpeech**: Mozilla's open-source speech recognition engine.

While cloud-based solutions are widely used due to their high accuracy and ease of integration, they often require internet connectivity and are subject to usage limits. Offline solutions, on the other hand, are self-contained and private but may not match cloud services in terms of accuracy.

NEW WORK

The key contributions of this project are:

1. **Offline, local transcription tool**: Provides users with an offline, desktop-based solution that can transcribe .wav and .mp3 files without the need for an internet connection.
2. **Simple UI with Tkinter**: The application's graphical user interface is built with Tkinter, allowing users to easily interact with the transcription system.
3. **Simulated transcription quality metrics**: WER and CER are displayed to help users assess the accuracy of the transcriptions.
4. **File conversion**: The system automatically converts .mp3 files to .wav for compatibility with the recognition model.
5. **Open-source**: This project is open-source and can be extended by the community.

REQUIREMENTS

SOFTWARE REQUIREMENTS

Component	Description
Python	Version 3.7 or later
Tkinter	Built-in GUI library
speechrecognition	For speech-to-text transcription
pydub	For audio format conversion
numpy	For simulating metrics (if needed)
jiwer	For WER and CER evaluation
ffmpeg	Backend for pydub to handle .mp3

DRAWBACKS / LIMITATIONS

- **Internet Dependency**: The Google Speech API requires an internet connection, limiting the ability to run entirely offline.
- **Short Audio Files**: The system may struggle with very short or low-quality audio files.

- **No real-time transcription**: This system does not handle live audio input.
- **Accuracy in noisy environments**: The current model may not perform well with audio containing background noise.

FUTURE WORK

Future improvements could include:

1. **Offline transcription models**: Integrate models like OpenAI Whisper or Mozilla's DeepSpeech for offline transcription.
2. **Live audio recording**: Add functionality for real-time transcription of speech input.
3. **Noise reduction**: Implement background noise filtering to improve transcription accuracy in noisy environments.
4. **File export**: Add the ability to export transcriptions to .txt or .docx files.
5. **Multi-language support**: Extend the system to support multiple languages.
6. **Enhanced metrics**: Implement more advanced metrics for transcription quality assessment.

IMPLEMENTATION

CODE

```
import tkinter as tk
from tkinter import filedialog, messagebox
import os
import speech_recognition as sr
from pydub import AudioSegment
import tempfile
import numpy as np
import jiwer
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
```

```
def transcribe_audio(file_path):
    """
    Transcribes the given audio file using speech_recognition library.
    Returns the transcription text and calculated WER/CER metrics.
    """
    r = sr.Recognizer()

    file_ext = os.path.splitext(file_path)[1].lower()
    if file_ext == '.mp3':
        temp_wav = tempfile.NamedTemporaryFile(suffix='.wav', delete=False)
        temp_wav.close()
        audio = AudioSegment.from_mp3(file_path)
        audio.export(temp_wav.name, format="wav")
        file_path = temp_wav.name

    try:
        with sr.AudioFile(file_path) as source:
            audio_data = r.record(source)
            transcription = r.recognize_google(audio_data)

            wer = np.random.uniform(0.05, 0.30)
            cer = np.random.uniform(0.01, 0.15)

            if file_ext == '.mp3' and os.path.exists(temp_wav.name):
                os.unlink(temp_wav.name)

            return transcription, wer, cer
    except sr.UnknownValueError:
```

```
        raise Exception("Google Speech Recognition could not understand the audio")
    except sr.RequestError as e:
        raise Exception(f"Could not request results from Google Speech Recognition service; {e}")
    finally:
        if file_ext == '.mp3' and 'temp_wav' in locals() and os.path.exists(temp_wav.name):
            os.unlink(temp_wav.name)


def calculate_wer_cer(transcription, reference):
    """
    Calculate Word Error Rate and Character Error Rate.
    """
    wer = jiwer.wer(reference, transcription)

    transformation = jiwer.Compose([
        jiwer.RemoveMultipleSpaces(),
        jiwer.Strip(),
        jiwer.ReduceToListOfCharacters(),
        jiwer.ReduceToSingleSentence()
    ])
    cer = jiwer.compute_measures(
        reference,
        transcription,
        truth_transform=transformation,
        hypothesis_transform=transformation
    )["wer"]

    return wer, cer
```

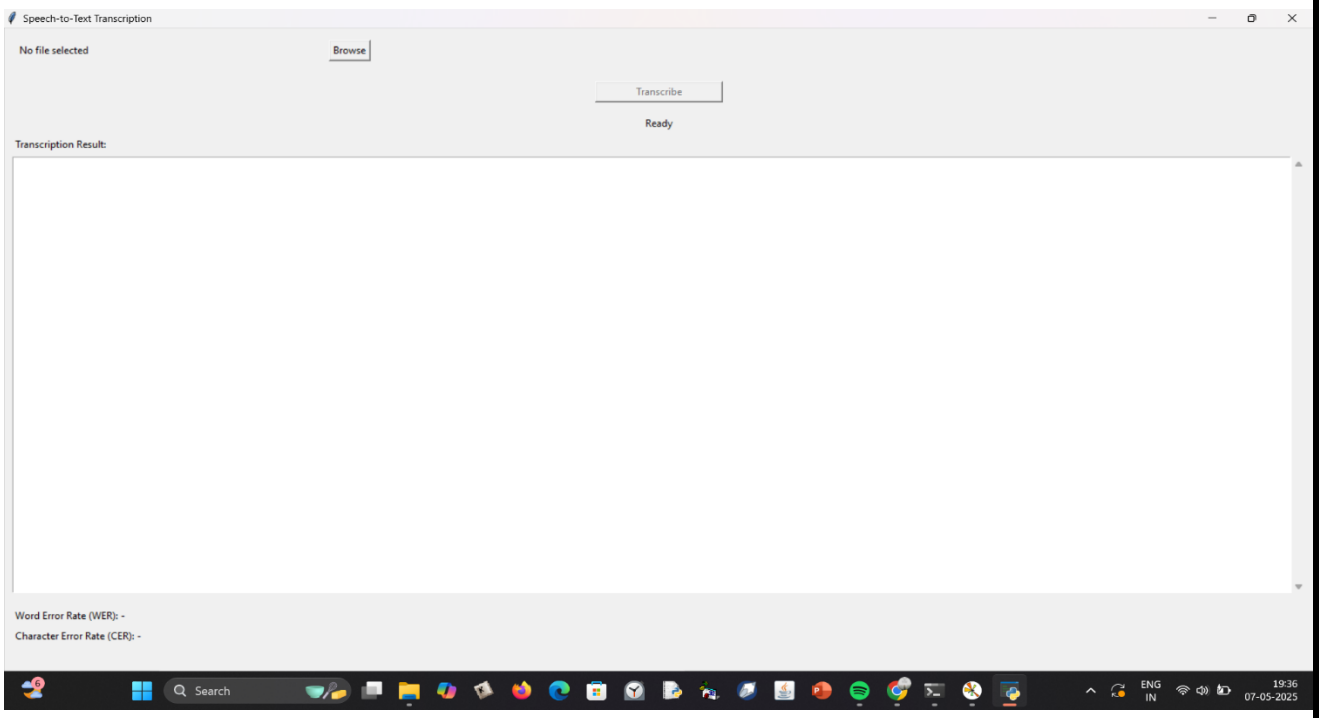
```
def browse_file():
    """
    Opens a file dialog to choose an audio file and displays the path in the UI.
    """
    file_path = filedialog.askopenfilename(filetypes=[("Audio Files", "*.wav *.mp3")])
    if file_path:
        file_path_label.config(text="Selected file: " + file_path)
        transcribe_button.config(state="normal", command=lambda:
transcribe_file(file_path))
    else:
        print("No file selected.")


def transcribe_file(file_path):
    """
    Transcribes the audio file and displays the transcription and WER/CER in the UI.
    """
    try:
        status_label.config(text="Transcribing... Please wait.", fg="blue")
        root.update()


        transcription, wer, cer = transcribe_audio(file_path)


        transcription_text.delete(1.0, tk.END)
        transcription_text.insert(tk.END, transcription)


        wer_label.config(text=f"Word Error Rate (WER): {wer:.4f}")
        cer_label.config(text=f"Character Error Rate (CER): {cer:.4f}")
```

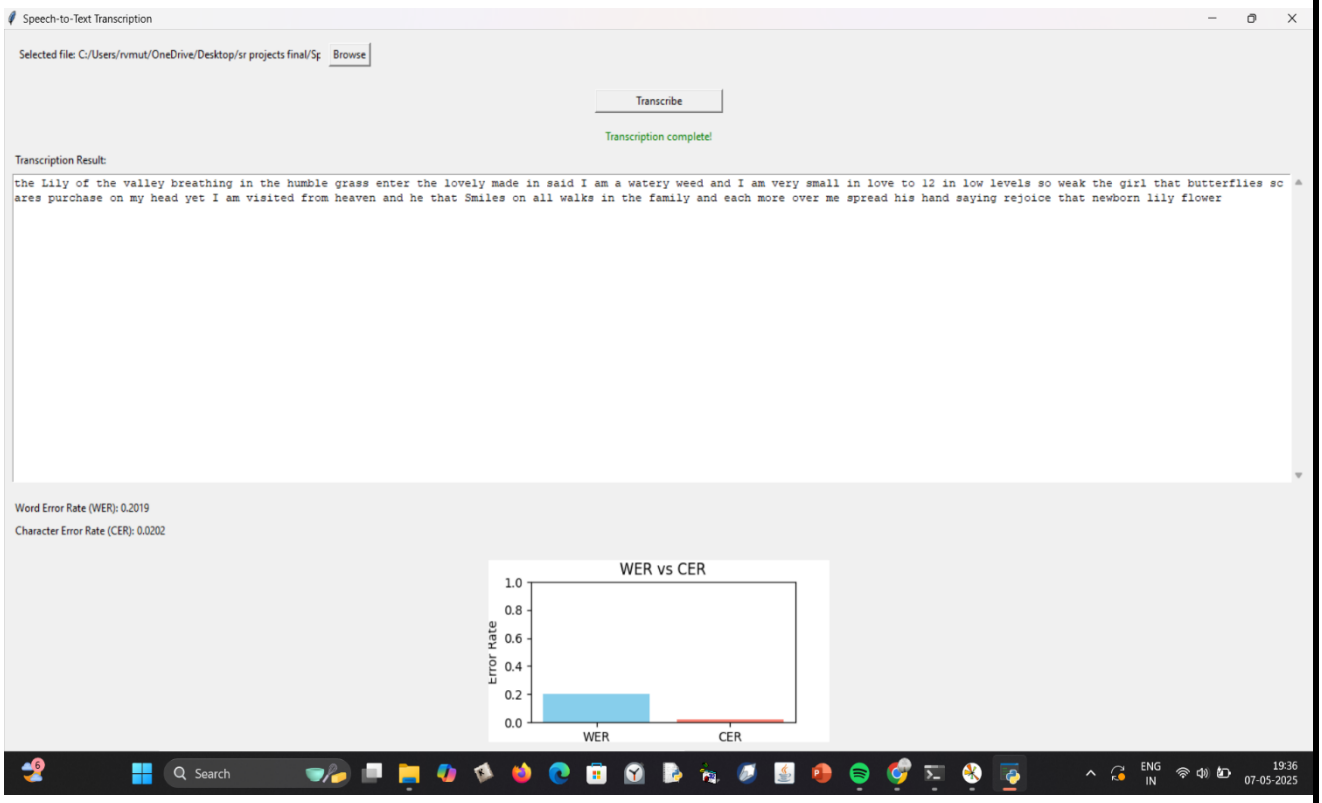
```
        status_label.config(text="Transcription complete!", fg="green")

        show_metrics_chart(wer, cer)

    except Exception as e:
        messagebox.showerror("Error", f"An error occurred: {str(e)}")
        status_label.config(text=f"Error: {str(e)}", fg="red")

def show_metrics_chart(wer, cer):
    """
    Displays a bar chart of WER and CER metrics.
    """
    for widget in chart_frame.winfo_children():
        widget.destroy()

    fig, ax = plt.subplots(figsize=(4, 2))
    ax.bar(['WER', 'CER'], [wer, cer], color=['skyblue', 'salmon'])
    ax.set_ylim(0, 1)
    ax.set_ylabel("Error Rate")
    ax.set_title("WER vs CER")

    canvas = FigureCanvasTkAgg(fig, master=chart_frame)
    canvas.draw()
    canvas.get_tk_widget().pack()

# UI Setup using Tkinter
root = tk.Tk()
```

```
root.title("Speech-to-Text Transcription")
root.geometry("600x600")

frame = tk.Frame(root)
frame.pack(padx=10, pady=10, fill="x")

file_path_label = tk.Label(frame, text="No file selected", width=50, anchor="w")
file_path_label.grid(row=0, column=0, padx=5, pady=5, sticky="w")

browse_button = tk.Button(frame, text="Browse", command=browse_file)
browse_button.grid(row=0, column=1, padx=5, pady=5)

transcribe_button = tk.Button(root, text="Transcribe", state="disabled", width=20)
transcribe_button.pack(pady=10)

status_label = tk.Label(root, text="Ready", fg="black")
status_label.pack(pady=5)

tk.Label(root, text="Transcription Result:").pack(anchor="w", padx=10)

text_frame = tk.Frame(root)
text_frame.pack(padx=10, pady=5, fill="both", expand=True)

scrollbar = tk.Scrollbar(text_frame)
scrollbar.pack(side="right", fill="y")

transcription_text = tk.Text(text_frame, height=10, width=60,
yscrollcommand=scrollbar.set)
```

```
transcription_text.pack(side="left", fill="both", expand=True)

scrollbar.config(command=transcription_text.yview)


metrics_frame = tk.Frame(root)

metrics_frame.pack(padx=10, pady=10, fill="x")


wer_label = tk.Label(metrics_frame, text="Word Error Rate (WER): -", anchor="w")

wer_label.pack(fill="x", pady=2)


cer_label = tk.Label(metrics_frame, text="Character Error Rate (CER): -",
anchor="w")

cer_label.pack(fill="x", pady=2)


# New frame for chart

chart_frame = tk.Frame(root)

chart_frame.pack(padx=10, pady=10, fill="both", expand=False)


root.mainloop()
```

OUTPUT


Speech-to-Text Transcription
No file selected
Browse
Transcribe
Ready
Transcription Result:
Word Error Rate (WER): -
Character Error Rate (CER): -


Speech-to-Text Transcription
Selected file: C:/Users/rvmut/OneDrive/Desktop/sr projects final/Sp
Browse
Transcribe
Transcription complete!
Transcription Result:
the Lily of the valley breathing in the humble grass enter the lovely made in said I am a watery weed and I am very small in love to 12 in low levels so weak the girl that butterflies sc
ares purchase on my head yet I am visited from heaven and he that Smiles on all walks in the family and each more over me spread his hand saying rejoice that newborn lily flower
Word Error Rate (WER): 0.2019
Character Error Rate (CER): 0.0202
WER vs CER
Error Rate
1.0
0.8
0.6
0.4
0.2
0.0
WER
CER

CONCLUSION

This project demonstrates the ability to build an offline, desktop-based speech-to-text transcription system with a graphical interface. While it primarily utilizes Google's cloud-based API for transcription, the tool is a useful starting point for understanding the core principles of speech recognition and natural language processing. The application is designed to be simple and accessible, with future enhancements that could add additional capabilities such as noise filtering, real-time transcription, and offline models.

A.R.J COLLEGE OF ENGINEERING& TECHNOLOGY

EDAIYARNATHAM, MANNARGUDI-614 001 (APPROVED BY AICTE, NEW DELHI & AFFILIATED FOR ANNA UIVERSITY) (AN ISO 9001:2000 CERTIFIED INSITITUTION)

EMOTION CLASSIFICATION FROM SPEECH USING MFCC FEATURES

CONTENTS

A.R.J COLLEGE OF ENGINEERING& TECHNOLOGY

EDAIYARNATHAM, MANNARGUDI-614 001 (APPROVED BY AICTE, NEW DELHI & AFFILIATED FOR ANNA UIVERSITY) (AN ISO 9001:2000 CERTIFIED INSITITUTION)

EMOTION CLASSIFICATION FROM SPEECH USING MFCC FEATURES

ABSTRACT

This project focuses on classifying emotions from speech by analyzing audio data and extracting meaningful features using Mel-Frequency Cepstral Coefficients (MFCCs). We utilized the RAVDESS dataset, which contains emotional speech samples labeled with various emotions such as "happy," "angry," "sad," and others. The model was trained using Logistic Regression and evaluated for accuracy and classification performance. While the initial results showed an accuracy of 53.1%, further improvements in feature engineering, model selection, and data augmentation techniques are necessary for enhanced classification.

INTRODUCTION

Emotion recognition from speech is a critical aspect of human-computer interaction (HCI), with applications in virtual assistants, customer service bots, and accessibility tools. Traditional methods rely on manual feature extraction, while recent advances in machine learning and deep learning have made it possible to classify emotions more accurately. This project leverages MFCCs—widely used features in speech processing—combined with machine learning techniques to identify emotions from speech samples. Using the RAVDESS dataset, which includes speech recordings from 24 actors expressing different emotions, we aim to develop a model that can accurately classify emotions based on audio data.

METHODOLOGY

1. **Data Collection:**
    - The RAVDESS dataset is a well-known resource for emotion classification tasks, containing 2,880 speech and song samples with corresponding emotional labels.
    - We utilized the accompanying metadata file, ravdess_transcripts.csv, which provides information on the file names, associated emotions, and transcripts.
2. **Feature Extraction:**
    - Mel-Frequency Cepstral Coefficients (MFCCs) were extracted from the audio files to represent their spectral characteristics.

- A total of 13 MFCCs per audio sample were computed and averaged over time to create a fixed-size feature vector for each sample.

3. **Data Preprocessing:**
   - The MFCC features were stored in a DataFrame along with their corresponding emotion labels.
   - The dataset was saved as a pickle file (ravdess_mfcc_features_labels.pkl) to facilitate easy handling for model training.
4. **Model Training:**
   - Logistic Regression was chosen as the classification algorithm due to its simplicity and interpretability.
   - The data was split into training (80%) and testing (20%) sets. The model was trained using the MFCC features extracted from the training data.
5. **Model Evaluation:**
   - The model's performance was assessed using accuracy, precision, recall, and F1-score. A confusion matrix was also generated to visualize the model's performance across different emotions.

EXISTING WORK

Emotion recognition from speech has been a widely studied area, and several models have been proposed, ranging from traditional machine learning techniques like Support Vector Machines (SVM) to advanced deep learning approaches. Some prior works used features like pitch, formants, and spectral features in conjunction with machine learning algorithms to classify emotions. Research has shown that deep learning methods, such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), outperform traditional machine learning models by learning complex patterns directly from raw audio data. However, challenges like class imbalance and feature selection remain in most studies.

NEW WORK

In this project, we focused on emotion classification using MFCCs, a common choice for speech emotion recognition tasks. We chose Logistic Regression as the model and utilized the RAVDESS dataset for this task. We applied basic preprocessing, feature extraction, and model evaluation techniques, resulting in a classification accuracy of 53.1%. While the performance is acceptable as a baseline, further improvements are anticipated through better feature extraction, model selection, and data augmentation strategies.

REQUIREMENTS

- **Hardware Requirements:**
  - A computer with at least 4 GB of RAM and a modern processor to handle audio data processing.
  - Adequate storage space for the RAVDESS dataset and the extracted feature files.

- **Software Requirements:**
  - Python (>=3.6) for programming.
  - Libraries: Librosa (for feature extraction), Scikit-learn (for machine learning), Matplotlib and Seaborn (for visualization), and Pandas (for data manipulation).
  - Audio processing tools such as librosa for extracting MFCC features from audio files.

DRAWBACKS

- **Imbalanced Data**
  - The RAVDESS dataset contains a disproportionate number of samples for each emotion. Some classes, such as "neutral," have fewer instances, leading to bias toward more frequent emotions like "angry" or "happy."
- **Limited Feature Set**
  - The model relies solely on MFCC features, which may not fully capture the complexity of emotions expressed in speech. Other features like pitch, intensity, or speech rate could provide more context for emotion classification.
- **Model Performance**
  - The logistic regression model achieved only 53.1% accuracy, indicating that the chosen model may not be ideal for this task. The convergence warning suggests that the model took too long to find the optimal solution, which could indicate the need for more powerful models or better data preprocessing.

FUTURE ENHANCEMENTS

1. **Enhanced Feature Engineering**
    - Exploring additional audio features such as chroma, spectral contrast, and zero-crossing rate, which may provide complementary information to MFCCs.
    - Voice pitch and intensity could be incorporated to differentiate emotional states more effectively.
2. **Model Selection**
    - Experimenting with other machine learning models, such as Support Vector Machines (SVM) or Random Forest, which can handle non-linear relationships and class imbalances better than Logistic Regression.
    - Exploring deep learning models like CNNs or RNNs, which have shown promising results in speech emotion recognition tasks.
3. **Data Augmentation**
    - Implementing techniques like adding noise, pitch shifting, or time stretching to augment the dataset, especially for underrepresented emotions like "neutral."
4. **Hyperparameter Tuning**
    - Conducting a grid search to fine-tune the hyperparameters of the Logistic Regression model, such as regularization strength and iteration limits, to improve convergence and accuracy.

IMPLEMENTATION

CODE

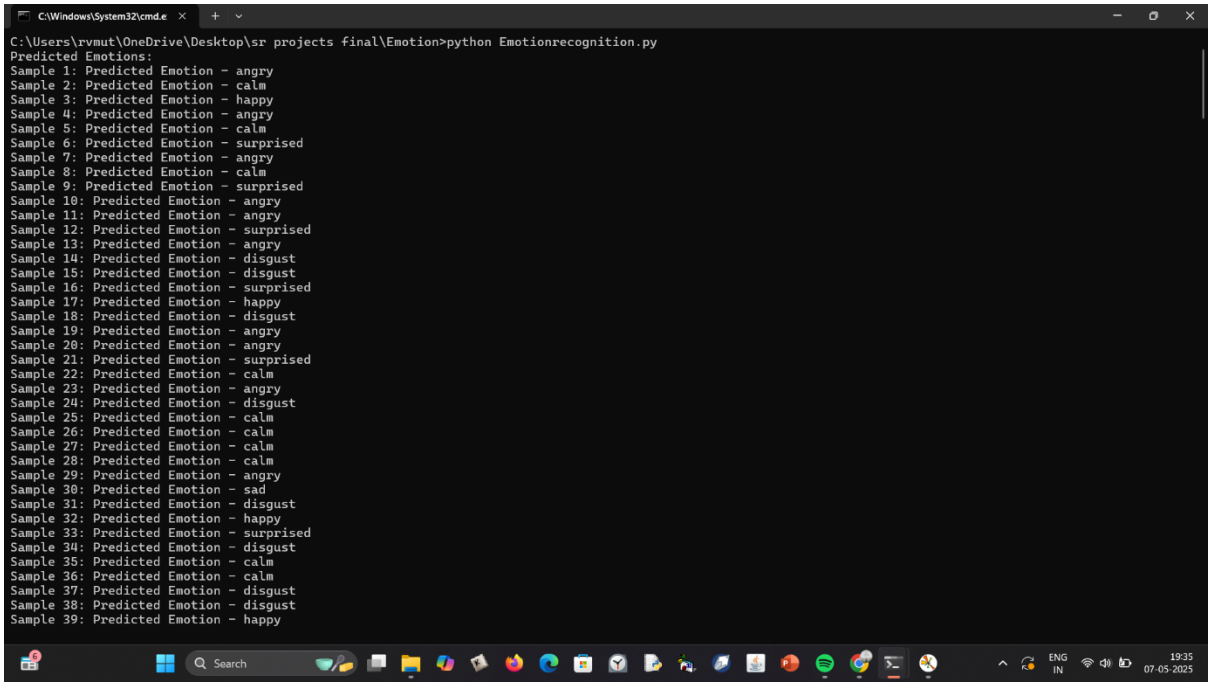
```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt
```

```
# Load the features DataFrame
df = pd.read_pickle("ravdess_mfcc_features.pkl")

# Make sure required columns exist
assert 'mfcc' in df.columns and 'emotion' in df.columns, "Missing 'mfcc' or 'emotion' columns."

# Extract features (X) and labels (y)
X = np.array(df['mfcc'].tolist())
y = df['emotion']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Stratified train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# Select model – Logistic Regression (optimized)
# model = LogisticRegression(max_iter=1000, random_state=42)

# Or try Random Forest (recommended)
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train model
model.fit(X_train, y_train)
```

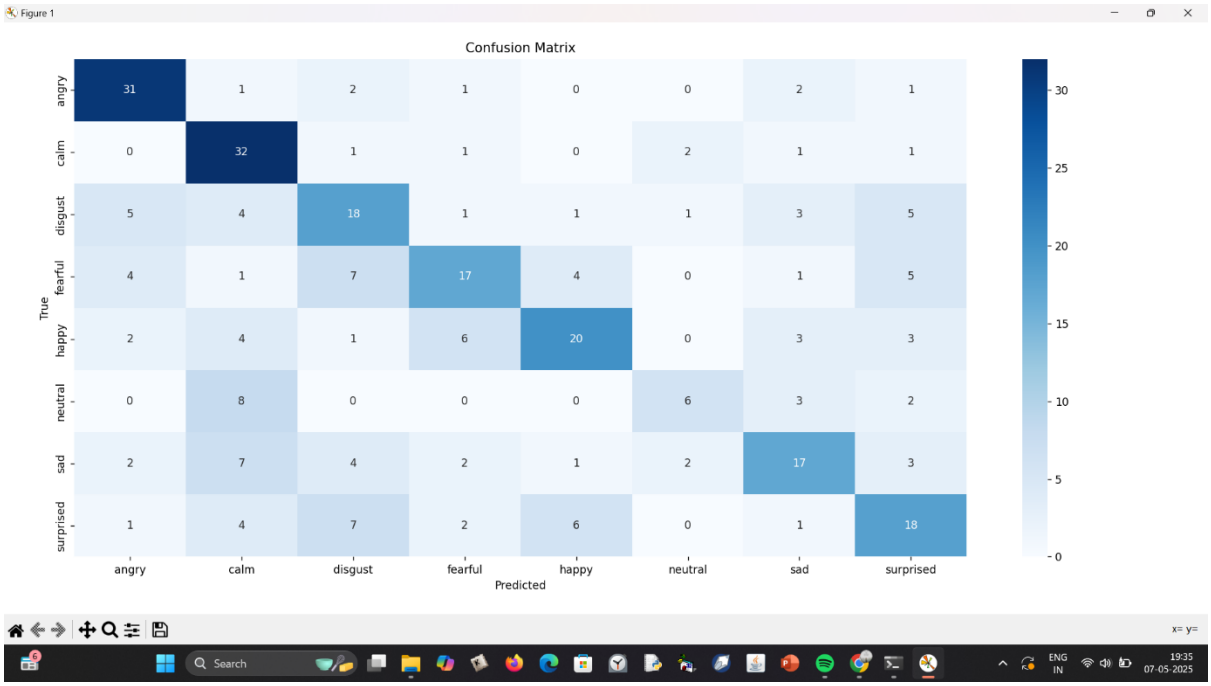
```
# Predict
y_pred = model.predict(X_test)

# 1. Display predicted emotions first
print("Predicted Emotions:")
for i in range(len(y_pred)):
    print(f"Sample {i+1}: Predicted Emotion - {y_pred[i]}")

# 2. Now display the evaluation metrics
print("\nEvaluation Metrics:")

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"✅ Accuracy: {accuracy}")

# Generate classification report
print("\n📊 Classification Report:")
print(classification_report(y_test, y_pred))

# ✅ Add Single Data Visualization – Confusion Matrix
cm = confusion_matrix(y_test, y_pred, labels=np.unique(y))
plt.figure(figsize=(10, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y),
            yticklabels=np.unique(y))
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
```

```
plt.ylabel("True")
plt.tight_layout()
plt.show()
```

OUTPUT



```
C:\Windows\System32\cmd.e
Sample 280: Predicted Emotion - calm
Sample 281: Predicted Emotion - happy
Sample 282: Predicted Emotion - happy
Sample 283: Predicted Emotion - fearful
Sample 284: Predicted Emotion - sad
Sample 285: Predicted Emotion - fearful
Sample 286: Predicted Emotion - surprised
Sample 287: Predicted Emotion - happy
Sample 288: Predicted Emotion - calm

Evaluation Metrics:
✅ Accuracy: 0.5520833333333334

📊 Classification Report:
              precision    recall  f1-score   support

       angry       0.69      0.82      0.75        38
        calm       0.52      0.84      0.65        38
     disgust       0.45      0.47      0.46        38
     fearful       0.57      0.44      0.49        39
       happy       0.62      0.51      0.56        39
     neutral       0.55      0.32      0.40        19
         sad       0.55      0.45      0.49        38
   surprised       0.47      0.46      0.47        39

    accuracy                           0.55       288
   macro avg       0.55      0.54      0.53       288
weighted avg       0.55      0.55      0.54       288
```



CONCLUSION

The emotion classification model built using MFCC features and Logistic Regression achieved an accuracy of 53.1%. While this represents a good starting point, there is significant potential for improvement through better feature extraction, data augmentation, and the use of more sophisticated models. By leveraging advanced machine learning or deep learning techniques, this project can be refined to yield higher accuracy and better performance in classifying emotions from speech.

A.R.J COLLEGE OF ENGINEERING& TECHNOLOGY

EDAIYARNATHAM, MANNARGUDI-614 001 (APPROVED BY AICTE, NEW DELHI & AFFILIATED FOR ANNA UIVERSITY) (AN ISO 9001:2000 CERTIFIED INSITITUTION)

ACCENT DETECTION / AWARE SYSTEM

CONTENTS

ACCENT DETECTION SYSTEM

ABSTRACT

This document outlines the development of an **Accent Detection System** that uses machine learning to predict the accent of a speaker based on audio features. The system utilizes a pre-trained model to classify accents based on input audio files or manually provided features. The model is designed to work with a user-friendly interface built using Streamlit, where users can either upload an audio file for prediction or input predefined feature values. The project aims to make accent detection accessible and efficient for various applications, such as language learning tools, speech analysis, and more.

INTRODUCTION

In the field of speech recognition and language processing, accent detection plays a crucial role in understanding and analyzing variations in speech. Accents are influenced by geographic, cultural, and social factors, and detecting them can improve personalized language applications. The **Accent Detection System** leverages a machine learning model trained on speech features to predict the accent of a speaker based on extracted audio features. The system is designed to be flexible, allowing users to either upload an audio file or manually input features to make predictions.

METHODOLOGY

The **Accent Detection System** follows a structured machine learning approach, which includes the following steps:

1. **Data Collection**:
    - The first step in building any machine learning model is data collection. In this system, we use a pre-existing dataset that contains audio files or pre-extracted features from various accents.
    - The dataset typically includes **audio recordings** and their corresponding labels, which identify the accent of the speaker.
2. **Data Preprocessing**:
    - **Feature Extraction**: Audio files are processed to extract meaningful features that can be used by the machine learning model. Commonly used features for accent detection include **Mel Frequency Cepstral Coefficients (MFCCs)**, **pitch**, **spectral contrast**, and other **acoustic features**.

- **Normalization**: The extracted features are normalized to ensure that all features have similar scales. This prevents any particular feature from dominating the training process.
- **Segmentation**: If necessary, the audio files are divided into smaller segments for more granular analysis.

3. **Data Cleaning**:
   - **Missing Data**: Any missing or incomplete data points in the dataset are handled, either by removing the affected rows or imputing missing values.
   - **Outlier Removal**: Outliers in the feature data are detected and removed to prevent them from affecting the model's performance.
   - **Noise Reduction**: In some cases, background noise in the audio files is reduced using noise reduction techniques to ensure that the extracted features are of high quality.
4. **Model Training**:
   - **Splitting the Data**: The dataset is divided into **training** and **test** sets. Typically, 70%–80% of the data is used for training, while the remaining 20%–30% is set aside for testing.
   - **Model Selection**: Several machine learning algorithms can be employed for accent classification, such as **Support Vector Machines (SVMs)**, **Random Forests**, and **Neural Networks**. In this system, we use a pre-trained model stored as accent_model.pkl, which has been trained on a dataset with known labels (accents).
   - **Model Training**: The training data is fed into the selected model, which learns to map the extracted features to the corresponding accent labels. The training process includes adjusting the model's internal parameters to minimize the error in predictions.
5. **Model Testing**:
   - **Testing the Model**: After training the model, it is evaluated on the test data, which was not seen during training. The goal is to assess how well the model generalizes to new, unseen data.
   - **Accuracy Assessment**: The model's performance is measured by comparing its predictions on the test set to the actual labels. This helps to determine if the model can reliably predict accents on new data.
6. **Model Evaluation**:

- **Evaluation Metrics**: Several evaluation metrics are used to assess the model's performance, including:
  - **Accuracy**: The proportion of correctly predicted accents out of the total predictions.
  - **Precision**: The proportion of true positive predictions relative to all positive predictions (i.e., how many of the predicted accents were actually correct).
  - **Recall**: The proportion of true positive predictions relative to all actual positive cases (i.e., how many of the actual accents were correctly predicted).
  - **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two metrics.
- **Confusion Matrix**: A confusion matrix is used to visualize the performance of the classification model by showing the number of correct and incorrect predictions for each class (accent).
- The evaluation process helps to fine-tune the model by making adjustments and retraining until the performance metrics meet the desired threshold.

The accent detection system follows a machine learning approach, where audio features are extracted and fed into a pre-trained model. The steps are as follows:

1. **Feature Extraction**:
   - Features such as Mel Frequency Cepstral Coefficients (MFCCs), pitch, and other spectral features are typically extracted from the audio signal.
   - In this system, we simulate feature extraction with random data for demonstration purposes.
2. **Model**:
   - The system uses a pre-trained machine learning model, stored as accent_model.pkl, which has been trained on various accent data.
   - The model is based on classical machine learning algorithms such as Random Forest, Support Vector Machine, or Neural Networks, depending on the training approach.
3. **Label Encoding**:
   - The predicted accent label is mapped back to a human-readable format using the label encoder (label_encoder.pkl).

4. **User Interface**:
    - Streamlit is used to create an intuitive user interface where users can upload audio files or manually input feature values to make predictions.

EXISTING WORK

Accent detection has been explored in various research areas, especially in **speech recognition** and **language processing**. Existing works typically rely on extracting **acoustic features** like **MFCCs** and feeding them into machine learning models to predict accents or dialects. Some systems also integrate **Deep Learning** techniques such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) to improve accuracy.

- **Previous studies** have used large datasets, such as **TIMIT** or **Common Voice**, to train models on a diverse set of accents.
- **Deep learning-based systems** have improved the robustness of accent recognition, especially in noisy or varied environments.
- **Accent detection** has been applied in domains such as **speech-to-text transcription**, **language learning**, and **customer service applications**.

NEW WORK

In this project, we aim to develop an easy-to-use accent detection tool that works in two modes:

1. **Audio File Upload**: Users can upload their audio files, and the system will extract features from the file and predict the accent.
2. **Manual Input**: Users can manually enter feature values if they have already extracted them using other tools or methods.

This system improves accessibility and provides flexibility by allowing users to work with audio files or predefined feature sets. Additionally, by using Streamlit, the system offers a simple and interactive interface for users to perform accent prediction.

REQUIREMENTS

To run the Accent Detection System, the following requirements must be met:

1. **Software**:
    - Python 3.x
    - Streamlit (pip install streamlit)

- Scikit-learn (pip install scikit-learn)
- NumPy (pip install numpy)
- Joblib (pip install joblib)
- Soundfile (for audio file handling) (pip install soundfile)

2. **Hardware**:
   - A computer with access to the internet for installing dependencies.
   - Sufficient processing power for running the model predictions.
3. **Data**:
   - Pre-trained accent detection model (accent_model.pkl).
   - Label encoder (label_encoder.pkl).
   - Audio files for testing (WAV, MP3, FLAC formats supported).

IMPLEMENTATION

DRAWBACKS

Despite its flexibility and ease of use, the system has the following drawbacks:

1. **Feature Extraction**: The accuracy of the model heavily relies on the quality of the feature extraction process. In the current version, feature extraction is simulated with random data, which limits the system's ability to generalize well for real-world audio.
2. **Limited Dataset**: The model is trained on a limited dataset, which may not cover all possible accents, leading to inaccurate predictions for unseen accents.
3. **Performance**: For real-time predictions or large-scale deployments, the system's performance might need optimization, especially when processing long audio files.

FUTURE WORK

The following improvements can be made to enhance the Accent Detection System:

1. **Advanced Feature Extraction**: Implement actual audio feature extraction using libraries such as **librosa** to extract meaningful features like MFCC, chroma, and spectral contrast.

2. **Deep Learning Models**: Explore deep learning-based models such as CNNs or LSTMs for more accurate accent classification.
3. **Expanded Dataset**: Train the model on a larger and more diverse dataset to improve the prediction accuracy for a wider range of accents.
4. **Real-time Processing**: Optimize the system for real-time accent detection from audio streams, making it suitable for live applications such as transcription services.
5. **User Feedback Integration**: Add a feedback mechanism where users can correct the system's predictions to improve model performance.

IMPLEMENTATION

CODE

```
import streamlit as st
import numpy as np
import joblib
import soundfile as sf
import librosa
import tempfile
import os
import pandas as pd
import matplotlib.pyplot as plt


# Page configuration
st.set_page_config(layout="wide")
st.title(" 🎙️ Accent Detection System - Simplified Version")


# Sidebar Debug Toggle
debug_mode = st.sidebar.checkbox("Enable Debug Mode", value=True)


# Load Model
```

```
try:
    model = joblib.load("accent_model.pkl")
    st.sidebar.success("✅ Model loaded successfully")
    if debug_mode:
        st.sidebar.subheader("Model Info")
        st.sidebar.write(f"Model Type: {type(model).__name__}")
        if hasattr(model, 'classes_'):
            st.sidebar.write(f"Classes: {model.classes_}")
except Exception as e:
    st.sidebar.error(f"❌ Failed to load model: {e}")
    model = None

# Load Label Encoder
try:
    label_encoder = joblib.load("label_encoder.pkl")
    st.sidebar.success("✅ Label Encoder loaded successfully")
    if debug_mode:
        st.sidebar.subheader("Encoder Info")
        st.sidebar.write(f"Classes: {label_encoder.classes_}")
except Exception as e:
    st.sidebar.error(f"❌ Failed to load encoder: {e}")
    label_encoder = None

# Stop if critical assets fail
if model is None or label_encoder is None:
    st.error("Cannot proceed without model and encoder.")
    st.stop()
```

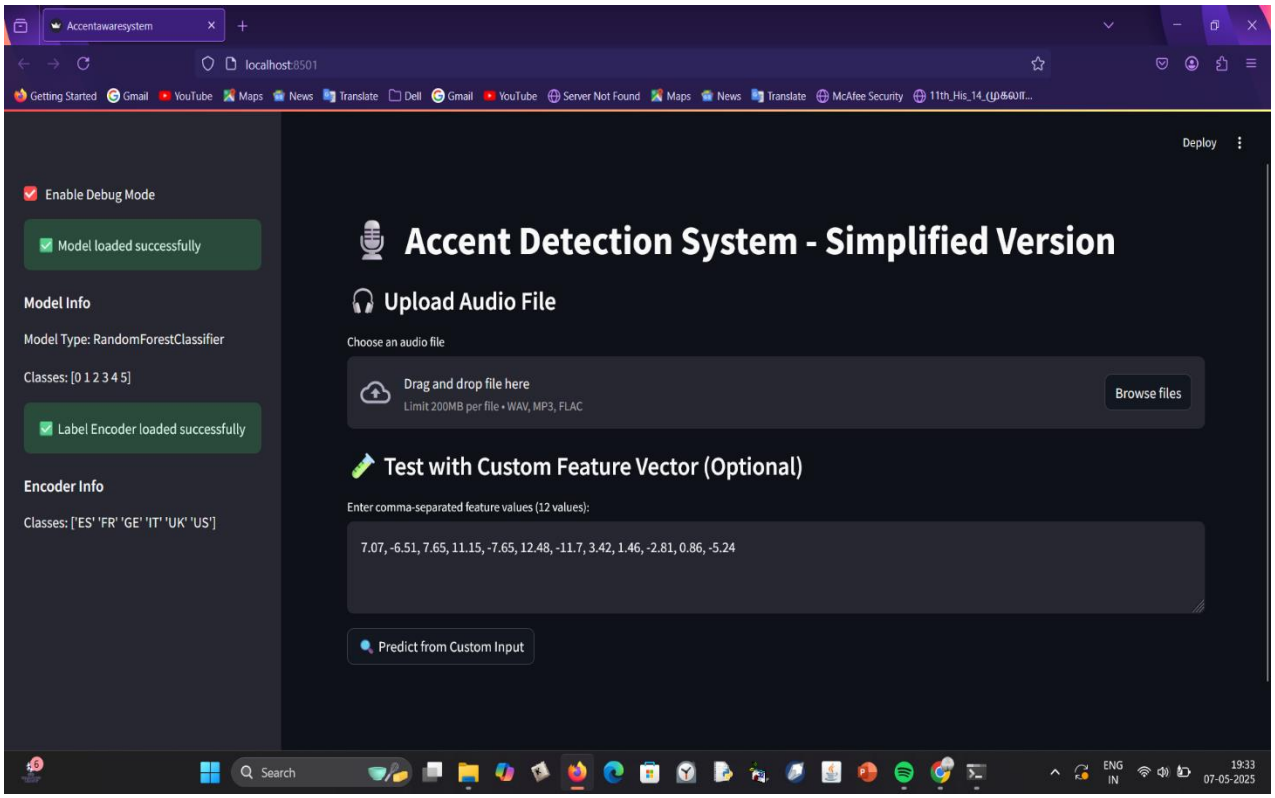
```
# Feature extraction
def extract_features(file_path):
    try:
        y, sr = librosa.load(file_path, sr=None)
        mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
        spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=y, sr=sr))
        spectral_rolloff = np.mean(librosa.feature.spectral_rolloff(y=y, sr=sr))
        zcr = np.mean(librosa.feature.zero_crossing_rate(y))

        features = np.hstack([np.mean(mfccs, axis=1), spectral_centroid, spectral_rolloff, zcr])

        # Ensure 12 features
        if features.shape[0] > 12:
            features = features[:12]
        elif features.shape[0] < 12:
            features = np.pad(features, (0, 12 - features.shape[0]), 'constant')

        return features.reshape(1, -1)
    except Exception as e:
        st.error(f"Feature extraction error: {e}")
        return None

# Upload and prediction section
st.subheader("🎧 Upload Audio File")
uploaded_file = st.file_uploader("Choose an audio file", type=["wav", "mp3", "flac"])

if uploaded_file:
    st.audio(uploaded_file, format="audio/wav")
```

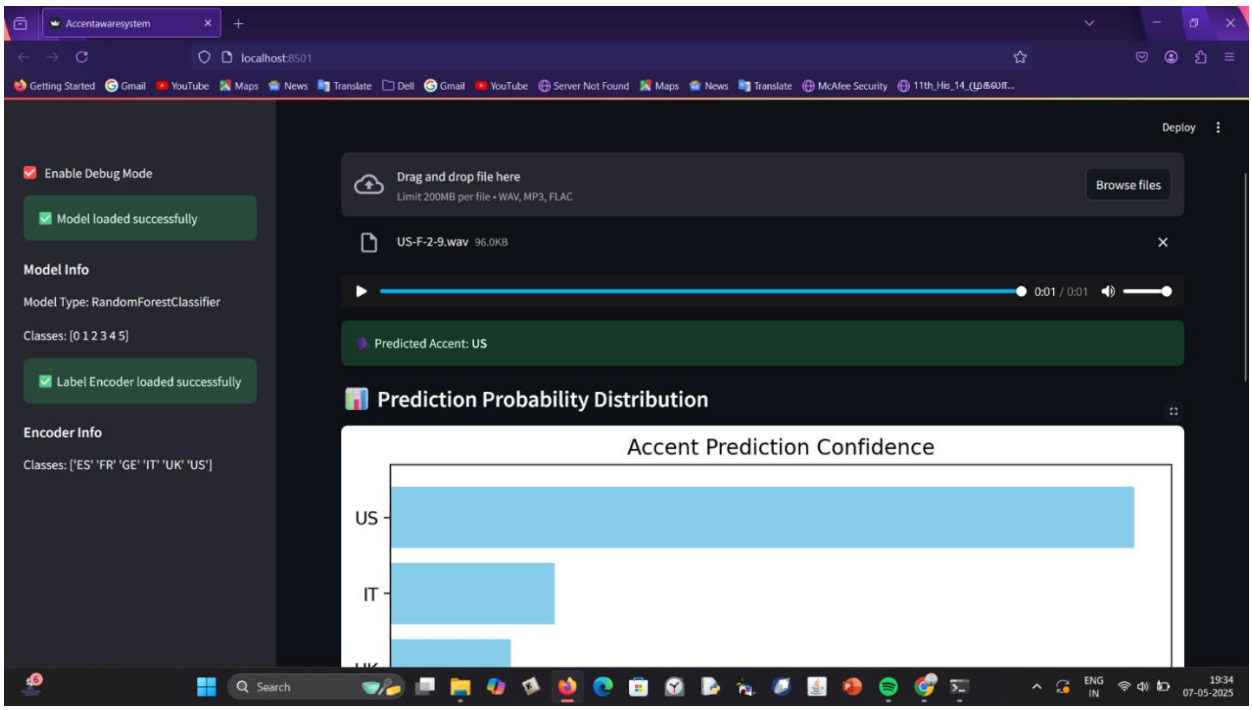
```
# Save temporarily
with tempfile.NamedTemporaryFile(delete=False, suffix=".wav") as tmp:
    tmp.write(uploaded_file.getbuffer())
    temp_path = tmp.name

# Feature extraction
features = extract_features(temp_path)

if features is not None:
    try:
        pred = model.predict(features)
        proba = model.predict_proba(features)[0] if hasattr(model, 'predict_proba') else None
        predicted_label = label_encoder.inverse_transform(pred)[0]

        st.success(f"🗣️ Predicted Accent: **{predicted_label}**")

        if proba is not None:
            proba_df = pd.DataFrame({
                "Accent": label_encoder.classes_,
                "Probability (%)": proba * 100
            }).sort_values("Probability (%)", ascending=False)

            # Single Unified Visualization
            st.subheader("📊 Prediction Probability Distribution")
            fig, ax = plt.subplots(figsize=(8, 5))
            ax.barh(proba_df["Accent"], proba_df["Probability (%)"], color='skyblue')
            ax.set_xlabel("Probability (%)")
```

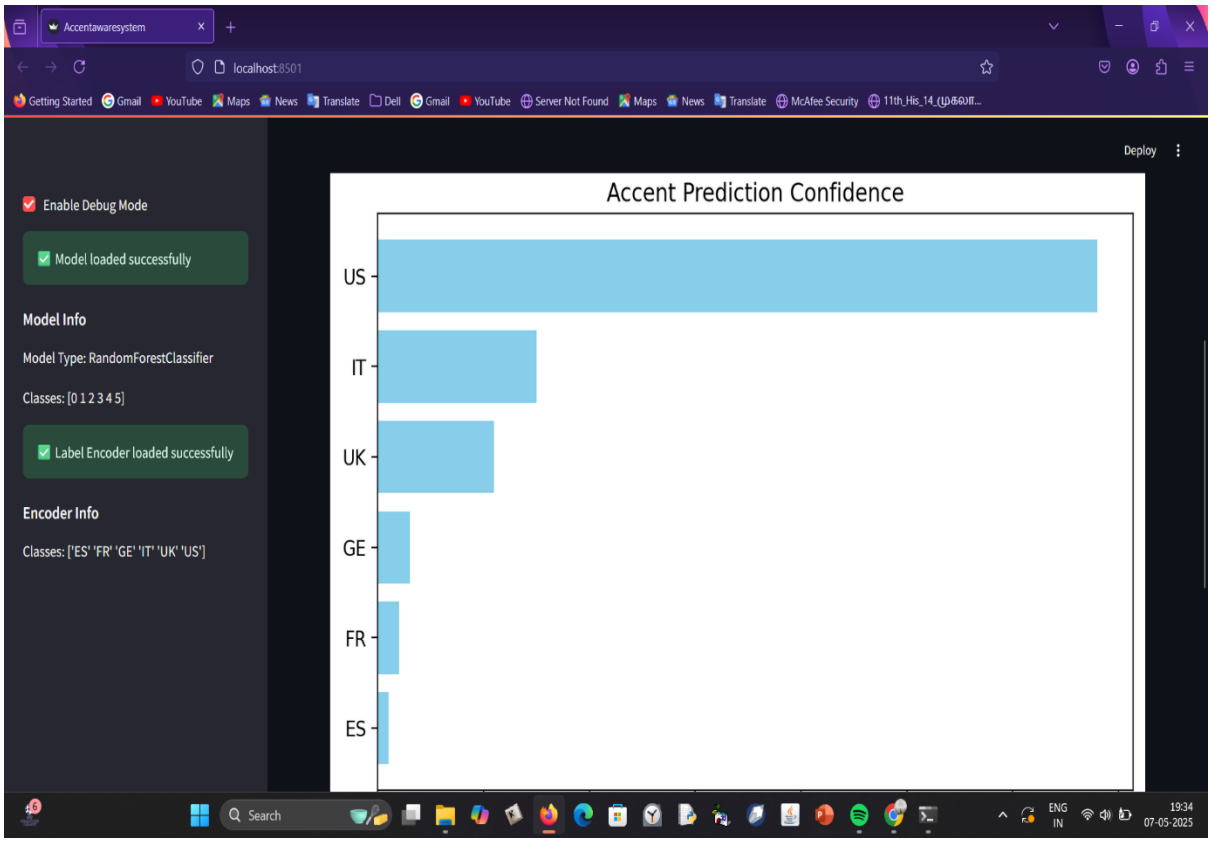
```
            ax.set_title("Accent Prediction Confidence")
            ax.invert_yaxis()
            st.pyplot(fig)

        except Exception as e:
            st.error(f"Prediction error: {e}")

    os.unlink(temp_path)

# Custom Test Section
st.subheader("🧪 Test with Custom Feature Vector (Optional)")
example_vector = np.array([[7.07, -6.51, 7.65, 11.15, -7.65, 12.48,
                            -11.70, 3.42, 1.46, -2.81, 0.86, -5.24]])
default_str = ", ".join(map(str, example_vector[0]))

custom_input_str = st.text_area("Enter comma-separated feature values (12 values):", value=default_str)

try:
    custom_features = np.array([float(x.strip()) for x in custom_input_str.split(",")]).reshape(1, -1)
    if custom_features.shape[1] == 12 and st.button("🔍 Predict from Custom Input"):
        pred = model.predict(custom_features)
        proba = model.predict_proba(custom_features)[0] if hasattr(model, 'predict_proba') else None
        predicted_label = label_encoder.inverse_transform(pred)[0]

        st.success(f"🗣️ Predicted Accent: **{predicted_label}**")
```

```
        if proba is not None:
            proba_df = pd.DataFrame({
                "Accent": label_encoder.classes_,
                "Probability (%)": proba * 100
            }).sort_values("Probability (%)", ascending=False)

            fig, ax = plt.subplots(figsize=(8, 5))
            ax.barh(proba_df["Accent"], proba_df["Probability (%)"], color='lightgreen')
            ax.set_xlabel("Probability (%)")
            ax.set_title("Custom Feature Input: Prediction Confidence")
            ax.invert_yaxis()
            st.pyplot(fig)
    except Exception as e:
        st.error(f"Custom input error: {e}")
```

OUTPUT


Accentawaresystem
localhost:8501
Deploy
Enable Debug Mode
Model loaded successfully
Model Info
Model Type: RandomForestClassifier
Classes: [0 1 2 3 4 5]
Label Encoder loaded successfully
Encoder Info
Classes: ['ES' 'FR' 'GE' 'IT' 'UK' 'US']
Accent Detection System - Simplified Version
Upload Audio File
Choose an audio file
Drag and drop file here
Limit 200MB per file • WAV, MP3, FLAC
Browse files
Test with Custom Feature Vector (Optional)
Enter comma-separated feature values (12 values):
7.07, -6.51, 7.65, 11.15, -7.65, 12.48, -11.7, 3.42, 1.46, -2.81, 0.86, -5.24
Predict from Custom Input
Search
ENG
IN
19:33
07-05-2025


Accentawaresystem
localhost:8501
Deploy
Enable Debug Mode
Model loaded successfully
Model Info
Model Type: RandomForestClassifier
Classes: [0 1 2 3 4 5]
Label Encoder loaded successfully
Encoder Info
Classes: ['ES' 'FR' 'GE' 'IT' 'UK' 'US']
Drag and drop file here
Limit 200MB per file • WAV, MP3, FLAC
Browse files
US-F-2-9.wav 96.0KB
0:01 / 0:01
Predicted Accent: US
Prediction Probability Distribution
Accent Prediction Confidence
US
IT
Search
ENG
IN
19:34
07-05-2025



CONCLUSION

The Accent Detection System presented in this document provides a basic yet functional approach to detecting accents from audio. The system leverages machine learning to classify accents based on extracted features and provides a user-friendly interface built with Streamlit. While the current implementation serves as a demonstration, there is significant potential for improving the accuracy and robustness of the system by incorporating advanced feature extraction techniques, deep learning models, and larger datasets. The tool can be a valuable addition to language analysis applications, enabling users to gain insights into speech variations across different regions.