End-to-End Speech Emotion Recognition Project using ANN

Introduction

Ever wondered if a computer could understand emotions just by listening? That’s the challenge we took on with this project—teaching machines to "feel" the tone of human speech.

Speech Emotion Recognition (SER) is about training machines to detect and classify emotions like happiness, anger, sadness, and more through audio signals.

And with the growing use cases of this technology—from mental health applications to entertainment and customer service—SER is quickly becoming a game changer.

In this documentation, we’ll walk you through the project, from understanding the problem and data, to training the models, and finally, making predictions.

Problem Statement

Emotions are core to human interaction. While we can sense each other's emotional states through speech and tone, making computers do the same isn’t as easy.

In this project, the goal is to develop a model capable of accurately recognizing emotions from speech audio clips.

This involves using deep learning techniques to capture the hidden essence in audio signals and categorizing them into different emotions, such as happy, sad, angry, and neutral.

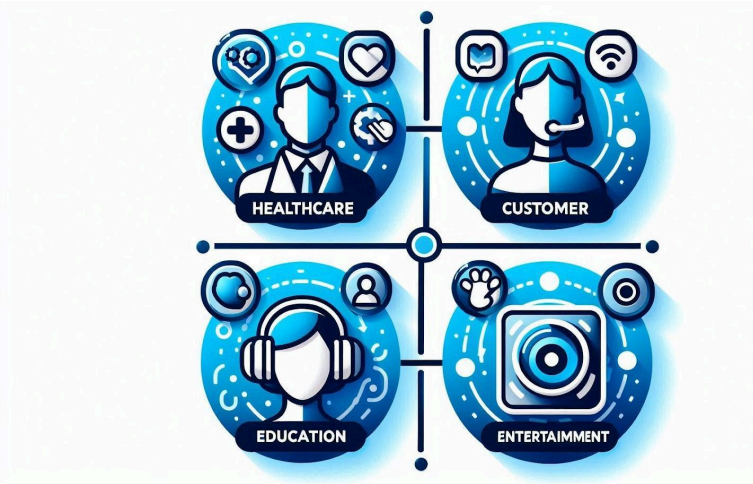
By utilizing the well-structured Ryerson Audio-Visual Database of Emotional Speech and Song **(RAVDESS)** dataset, we will develop a model capable of recognizing emotions from audio files using Keras and TensorFlow.

Additionally, we will implement a Multi-Layer Perceptron (MLP) model utilizing the scikit-learn library.

Business Use Case

Speech Emotion Recognition has vast potential across industries, including:

- **Healthcare**: Enhancing telemedicine experiences by detecting patient emotions and mental health.
- **Customer Service**: Automatically identifying the sentiment in customer support calls to escalate issues or handle angry customers.
- **Education**: Improving virtual learning experiences by recognizing and responding to students' emotions.
- **Entertainment**: Developing more interactive voice assistants or emotional responses in virtual reality and gaming.



Ultimately, implementing SER technology can transform how businesses engage with their customers and clients, creating a more responsive and emotionally aware ecosystem.

Project Objectives

The aim of this project is simple yet exciting -

- To build an Artificial Neural Network (ANN) that can correctly classify speech audio files into various emotions such as happiness, sadness, anger, and neutrality from speech signals, using deep learning libraries like **Keras**, **TensorFlow**, and **scikit-learn**.
- To implement audio processing techniques to extract relevant features, such as Mel-Frequency Cepstral Coefficients (MFCCs), pitch, and spectral features.

Data Overview

We used the **RAVDESS** dataset—the Ryerson Audio-Visual Database of Emotional Speech and Song—to train our models. This dataset includes:

- 7356 audio files.
- 24 actors (12 male, 12 female) delivering two matched statements.
- Emotions: Neutral, calm, happy, sad, angry, fearful, disgust, and surprised.
- File formats: High-quality .wav audio files.

For this project, we only utilized the audio files!

Filename Format Example: *03-01-06-02-02-02-24.wav*

Tech Stack

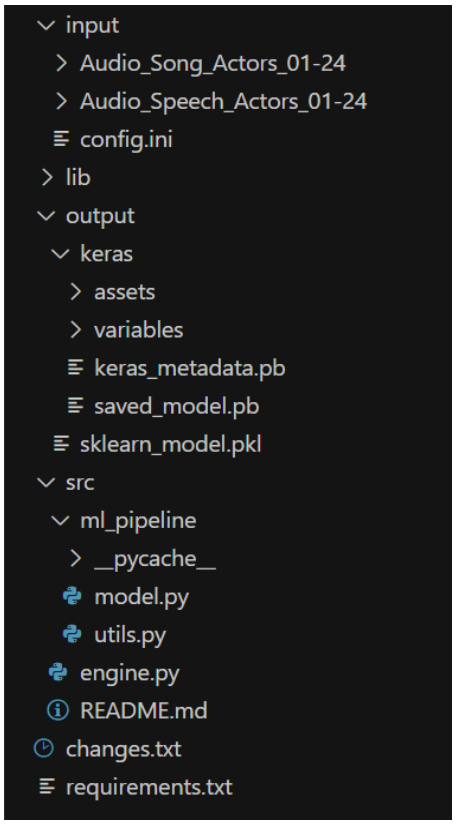
This project is powered by some amazing tools:

- **Python**: The main language used to code everything
- **Keras** & **TensorFlow**: Used for building and training our deep learning models.
- **librosa**: A great library for audio feature extraction.
- **soundfile**: Helps in reading and writing sound files.
- **scikit-learn**: For implementing a Multi-Layer Perceptron (MLP) as an alternative model.
- **pandas**, **numpy :** Used for Data handling
- **matplotlib**: To perform visualization on the data, to make sense of the extracted features.
- **pickle**: For saving and loading our models.

Workflow & Solution Approach

This section covers the end-to-end approach for building the Speech Emotion Recognition model, broken down into key stages:

Here is a glimpse of the folder structure to be followed -



1. Audio Data Preparation

The project starts by loading the **RAVDESS dataset**. We used two folders:

- `Audio_Speech`: Contains speech clips from actors.
- `Audio_Song`: Contains song clips.

The dataset is organized by emotion, actor, and intensity, which helped us structure the data for analysis. And the data is stored in the *input* folder.

2. Feature Extraction

We use the **librosa** library to extract crucial features from the audio files. These include:

- **Mel-Frequency Cepstral Coefficients (MFCC)**: Captures the power spectrum of the audio signal, giving us a compressed version of how the human ear perceives sound.
- **Chroma**: Provides insight into the harmonic content.
- **Spectral Contrast**: Captures the difference between peaks and valleys in the sound spectrum.

Each audio file was transformed into a 193-feature vector that we used to train the model. The script for this is located in the `utils.py` file.

3. Data Preprocessing

Preprocessing is a crucial step in any ML pipeline. In this project, we focused on transforming raw audio features into a format that our models can easily process.

Here's what we did:

1. **Standardizing Audio Features**:
    - After extracting features like **MFCC**, **Chroma**, and **Spectral Contrast** from each audio file, the feature values varied significantly.
    - To ensure the model can efficiently learn patterns, we standardized the features by scaling them.
    - This step transforms the data to have a mean of 0 and a standard deviation of 1, which improves the convergence speed of our neural network during training.

2. **Splitting the Data**:
    - The dataset was split into **training** and **testing** sets, ensuring the model trains on one portion of the data and evaluates its performance on unseen data.
    - We used a typical 70-30 split, where 70% of the data is used for training and 30% for testing.

3. **Handling Class Imbalance**:
    - The dataset had imbalances (some emotions had more samples than others), we monitored class distributions and applied **stratified sampling** during the train-test split to ensure that each emotion is well-represented in both sets.

This preprocessing ensures that our model is trained on clean, well-structured data, maximizing its chances of correctly predicting emotions from speech.

4. Model Building

Building the model is where the magic happens, and for this project, we experimented with two powerful machine learning frameworks: **Keras** (for deep learning) and **Scikit-learn** (for traditional machine learning models).

Here's how we approached model building and saved the trained model for future use:

i). Artificial Neural Network (ANN):

- Using **Keras** and **TensorFlow**, we created a neural network with multiple hidden layers to classify emotions. The architecture is designed to capture patterns in the audio features and predict emotions effectively.
- The architecture was fine-tuned as defined in the `config.ini` file: *hidden_layer_shape=[300, 50, 20, 10]*
- **Activation Functions**: **ReLU** is used in hidden layers for non-linearity, and softmax is used in the output layer for classification.
- The model was trained using **categorical cross-entropy** as the loss function and the **Adam optimizer** for adaptive learning.
- The trained model is saved in the `output` folder, as specified in the `config.ini` file: `model_save_path=../output`
- The model is saved using Keras's `model.save()` function, making it easy to load later .
- The model is loaded using `keras.models.load_model()` for inference or further training.

ii). Multi-Layer Perceptron (MLP):

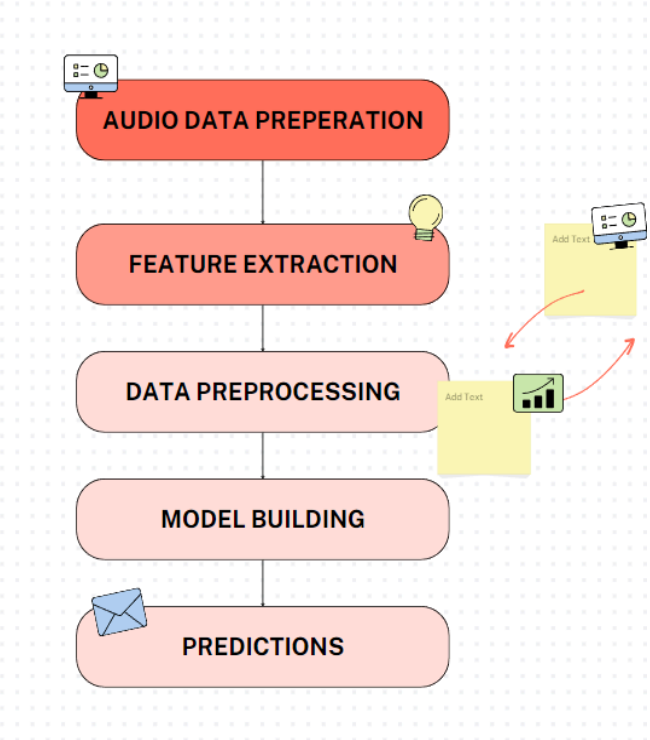
- As a backup, we implemented a traditional MLP using **scikit-learn** to compare performance against the deep learning model.
- This model is not as deep as the Keras-based neural network but provides a good baseline for performance.
- Like the deep learning model, the MLP was trained on the same preprocessed data and evaluated similarly.
- The trained MLP model is saved as a `.pkl` file in the `output` folder for easy reuse: `model_save_path=../output`
- In **scikit-learn**, the model is saved using the `pickle` library, ensuring that it can be loaded later for inference or further training:

```
model_save_path += '/sklearn_model.pkl'

model = train_sklearn(data_x, data_y)

with open(model_save_path, 'wb') as f:

    pickle.dump(model, f)
```



5. *Model.py*: The Core of Model Building

The `model.py` script is the central piece of the model-building process in this project. It handles the entire flow, from loading the preprocessed data to building, training, and saving the models.

Here's a breakdown of what it does:

- **Model Selection**: Depending on the configuration, the script can build and train either an **ANN** using Keras or a **MLP** using scikit-learn. It allows for flexibility in experimenting with both traditional and deep learning approaches.
- **Training**: The script reads the configuration parameters from *`config.ini`* (e.g., batch size, number of epochs, hidden layer sizes) and uses these to train the models on the preprocessed audio features.
- **Evaluation**: After training, the script evaluates model performance using metrics like accuracy and loss. These results help in comparing the performance between deep learning and traditional models.
- **Saving the Model**: Finally, both the trained Keras and scikit-learn models are saved in the `output` folder for future use.

  The saving mechanism is defined in the script:

  - Keras models are saved using the *`model.save()`* function.
  - Scikit-learn model is saved using *`pickle.dump()`*.

By centralizing the entire model-building workflow, *`model.py`* ensures that all components (data, config, and models) work together seamlessly.

6. Making Predictions

With the trained model, we can predict the emotion from any new audio file.This is the exciting part where we put the model to use.

The framework flag (*`--framework=keras`* or `--framework=sklearn`) lets us choose which model to use for inference, giving flexibility to test both deep learning and traditional machine learning models. The output will display the predicted emotion for the given audio file.

Execution Steps

To run this project on your local machine, follow these steps:

1. **Create a Virtual Environment**:
   - For Windows : ``` *python -m venv venv*

     venv\Scripts\activate ```
   - For macOS/Linux: ``` *python3 -m venv myenv*

     source myenv/bin/activate ```

2. **Install Dependencies:**

   With the virtual environment activated, install the necessary libraries by running:

   ``` *pip install -r requirements.txt* ```

3. **Extract Features:**

   Extract features from the RAVDESS dataset audio files. This step will preprocess the audio data and prepare it for model training:

   ```
   cd src

   python ml_pipeline/utils.py
   ```

4. **Train the Model:**

   After extracting features, train the model (Keras-based ANN and Scikit-learn MLP) by running the following command:

   ```
   python ml_pipeline/model.py
   ```

   The trained model will be saved to the `output` folder.
   ```

5. **Make Predictions:**

   Once the model is trained, you can use it to predict the emotion from any new audio file. Run the following command to make predictions:

```
python engine.py --framework=keras --infer--infer-file-path="path_to_audio_file.wav"
```

```
python engine.py --framework=sklearn --infer--infer-file-path="path_to_audio_file.wav"
```

   You can switch between frameworks (*keras* or *sklearn*) by adjusting the `--framework` flag.

Conclusion

Speech Emotion Recognition has huge potential in creating emotionally aware systems that can interact more naturally with humans. From detecting emotions in customer service to enhancing mental health tools, SER can make technology more intuitive and empathetic. This project showcases the power of deep learning in decoding the subtle nuances of human speech, and we hope it inspires more developments in this exciting field!