Adham Khalifa
COM 307 - Fall 2022

Speech Sentiment Analysis

1. Introduction

This project aimed to create a machine-learning model to analyze voice and speech to recognize emotion. It can be used in many applications, including media understanding, medical fields, etc.

My strategy was to make the model familiar with the audio features by extracting them from an audio file and converting these features to numeric values for training. The primary dataset used is RAVDESS, along with TESS, EMO-DB, and other custom files. These datasets had files for nine emotions: "neutral," "calm," "happy," "sad," "angry", "fear," "disgust", "ps" (pleasant surprise), and "boredom". I focused my output on only the main 5: “calm”, “happy,” “sad”, “angry,” and “fearful”.

2. Dataset

Generally, all the datasets had .wav files with labels and annotations in the file’s name. For example, in RAVDESS, a file could be like this: *03-01-06-01-02-01-12.wav*, with these annotations and corresponding values:

Audio-only (03)
Speech (01)
Fearful (06)
Normal intensity (01)
Statement "dogs" (02)
1st Repetition (01)
12th Actor (12)

The total number of training samples is: 3901 files, and for testing: 1301 files.

Using the Python split() function, I extracted these labels from the file name.

3. Strategy

a. Feature Extraction

This was the most critical part of the application since it allows the model to see numeric values representing the emotion in an audio file. It was done by changing the speech waveform to a form of a parametric representation at a relatively lesser data rate. Using librosa, I extracted these features:

1. MFCC
2. Chromagram (Chroma)
3. MEL Spectogram Frequency (mel)
4. Contrast
5. Tonnetz (tonnetz)

These 5 main features generate 180 sub-features for each audio file.

All these operations were performed in utils.py

Adham Khalifa
COM 307 - Fall 2022

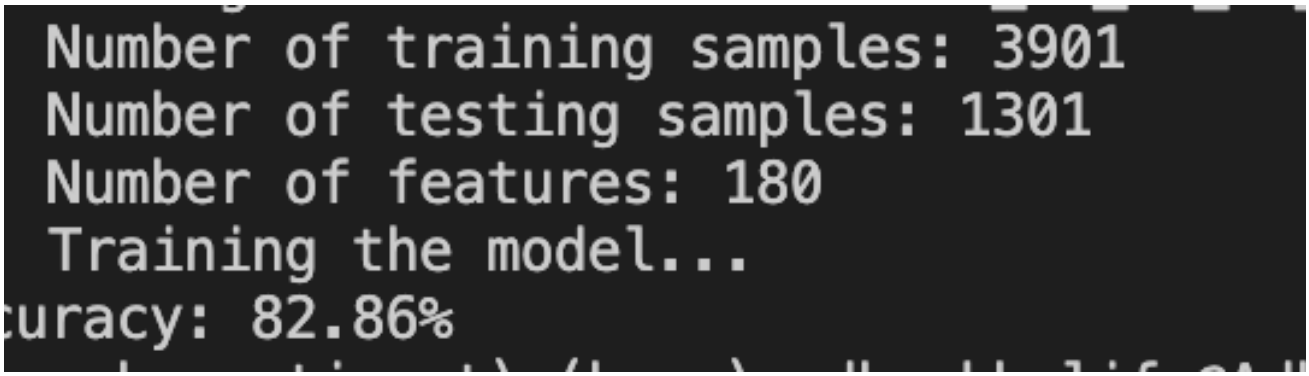
b. Training and evaluation

Using SKlearn, a neural network and a Multi-Layer Perceptron classifier were made to train the model with these parameters:

```
model_params = {
    'alpha': 0.01,
    'batch_size': 256,
    'epsilon': 1e-08,
    'hidden_layer_sizes': (300,),
    'learning_rate': 'adaptive',
    'max_iter': 500,
}
```

The Kaggle community recommended these parameters for that NLP problem. Moreover, that was the maximum iteration my machine was able to achieve. This batch size was enough to finish training in a reasonable time.

After loading the training and testing data using the utils.py helper function load_data, training started and saved the training log in results/log.txt along with the "speech_sentiment.model" model in the same folder. Here's the log for one of the training rounds:



And another round with training and validation was:
Training accuracy: 0.87
Validation accuracy: 0.86

The model accumulated many features mapped to different emotions using the training set that it can now compare any given audio files to that it learned. I think that the most important factor

out of the primary 5 features is the contrast because it allows the computer to identify high contrasts in waves that usually happens in emotions with a loud voice like anger.

This is included in train.py

c. Testing and demo

After achieving high accuracy, it was time for a test. Using PyAudio and Pickle, I implemented a microphone input feature to test the model in real-time. With some helper functions, is_silent, normalize, trim, and add_silence, I was trying to enhance the quality of the given microphone record to be all called in record(). When it's called, a microphone access dialog shows up to grant access, and then the user sees "Please speak after 1 sec," and it can automatically detect silence to close the microphone. Then it saves the recorded audio to a "test.wav" file. Then it calls the model to predict the emotion in that file. Here's the result of this demo when I was saying "Hello everyone," but with different emotions:

```
Please talk - wait 1 sec and start speaking
result: happy
(base) adhamkhalifa@Adhams-MacBook-Pro Final_project_AdhamKhalifa % python test.py
Please talk - wait 1 sec and start speaking
result: angry
(base) adhamkhalifa@Adhams-MacBook-Pro Final_project_AdhamKhalifa % python test.py
Please talk - wait 1 sec and start speaking
result: sad
(base) adhamkhalifa@Adhams-MacBook-Pro Final_project_AdhamKhalifa % python test.py
Please talk - wait 1 sec and start speaking
result: happy
(base) adhamkhalifa@Adhams-MacBook-Pro Final_project_AdhamKhalifa %
```

This is included in test.py

d. Drawbacks

One thing that I noticed when I was testing is that background noise could be a significant distraction. All the audio files were noise-free, and the lack of an augmented dataset made it harder to test out with real-life emotions (like when someone's talking in a crowd). I think a good next step would be either augmenting the dataset, or cancelling noise in detected voice, while I prefer the first to make the model more generic.

e. Applications

I'm looking forward to working on more applications for this project. This can be used to understand the scene of musical performance, for example, to tell if the song is sad, happy, etc.