Python Mini Project – Speech Emotion Recognition with librosa

BY DATAFLAIR TEAM ·OCTOBER 3, 2019

Python Mini Project

Speech emotion recognition, the best ever python mini project. The best example of it can be seen at call centers. If you ever noticed, call centers employees never talk in the same manner, their way of pitching/talking to the customers changes with customers. Now, this does happen with common people too, but how is this relevant to call centers? Here is your answer, the employees recognize customers' emotions from speech, so they can improve their service and convert more people. In this way, they are using speech emotion recognition. So, let's discuss this project in detail.

Speech emotion recognition is a simple Python mini-project, which you are going to practice with DataFlair. Before, I explain you the terms related to this mini python project, make sure you bookmarked the ***complete list of Python Projects***.

- ***Driver Drowsiness Detection Python Project***
- ***Detecting Fake News Python Project***
- **Speech Emotion Recognition Python Project**
- ***Detecting Parkinson's Disease Python Project***
- **Age Gender Detection Python Project**
- **Breast Cancer Classification Python Project**

What is Speech Emotion Recognition?

Speech Emotion Recognition, abbreviated as SER, is the act of attempting to recognize human emotion and affective states from speech. This is capitalizing on the fact that voice often reflects underlying emotion through tone and pitch. This is also the phenomenon that animals like dogs and horses employ to be able to understand human emotion.

SER is tough because emotions are subjective and annotating audio is challenging.

What is librosa?

librosa is a ***Python library*** for analyzing audio and music. It has a flatter package layout, standardizes interfaces and names, backwards compatibility, modular functions, and readable code. Further, in this Python mini-project, we demonstrate how to install it (and a few other packages) with pip.

What is JupyterLab?

JupyterLab is an open-source, web-based UI for Project Jupyter and it has all basic functionalities of the Jupyter Notebook, like notebooks, terminals, text editors, file browsers, rich outputs, and more. However, it also provides improved support for third party extensions.

To run code in the JupyterLab, you'll first need to run it with the command prompt:

```
1.  C:\Users\DataFlair>jupyter lab
```

This will open for you a new session in your browser. Create a new Console and start typing in your code. JupyterLab can execute multiple lines of code at once; pressing enter will not execute your code, you'll need to press Shift+Enter for the same.

Speech Emotion Recognition – Objective

To build a model to recognize emotion from speech using the librosa and sklearn libraries and the RAVDESS dataset.

Speech Emotion Recognition – About the Python Mini Project

In this Python mini project, we will use the libraries librosa, soundfile, and sklearn (among others) to build a model using an MLPClassifier. This will be able to recognize emotion from sound files. We will load the data, extract features from it, then split the dataset into training and testing sets. Then, we'll initialize an MLPClassifier and train the model. Finally, we'll calculate the accuracy of our model.

The Dataset

For this Python mini project, we'll use the RAVDESS dataset; this is the Ryerson Audio-Visual Database of Emotional Speech and Song dataset, and is free to download. This dataset has 7356 files rated by 247 individuals 10 times on emotional validity, intensity, and genuineness. The entire dataset is 24.8GB from 24 actors, but we've lowered the sample rate on all the files, and you can **download it here**.

Prerequisites

You'll need to install the following libraries with pip:

```
pip install librosa soundfile numpy sklearn pyaudio
```

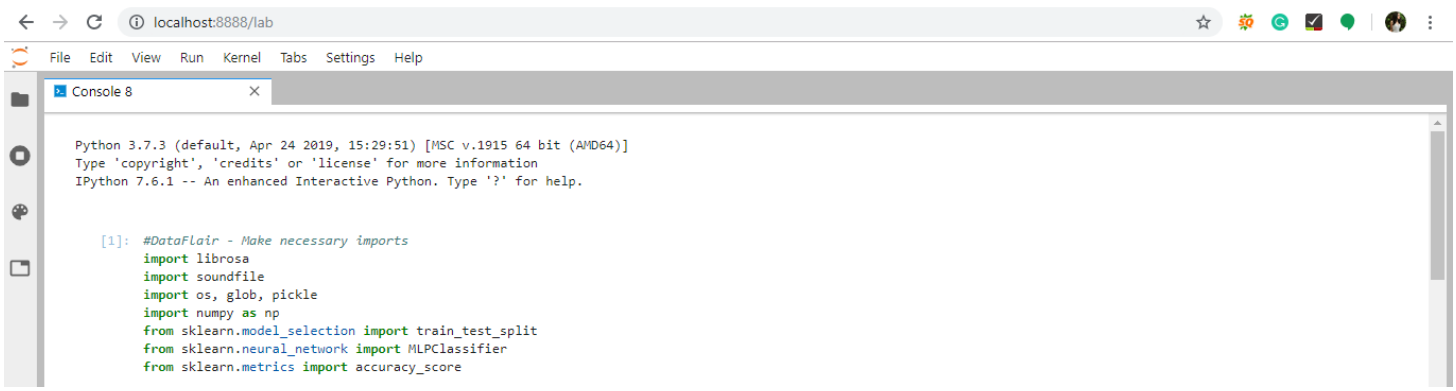
If you run into issues installing librosa with pip, you can try it with conda.

Steps for speech emotion recognition python projects

1. Make the necessary imports:

```
import librosa
import soundfile
import os, glob, pickle
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

Screenshot:



2. Define a function extract_feature to extract the mfcc, chroma, and mel features from a sound file. This function takes 4 parameters- the file name and three Boolean parameters for the three features:

- **mfcc:**Mel Frequency Cepstral Coefficient, represents the short-term power spectrum of a sound
- **chroma:**Pertains to the 12 different pitch classes
- **mel:**Mel Spectrogram Frequency

Learn more about Python Sets and Booleans

Open the sound file with soundfile.SoundFile using with-as so it's automatically closed once we're done. Read from it and call it X. Also, get the sample rate. If chroma is True, get the Short-Time Fourier Transform of X.

Let result be an empty numpy array. Now, for each feature of the three, if it exists, make a call to the corresponding function from librosa.feature (eg- librosa.feature.mfcc for mfcc), and get the mean value. Call the function hstack() from numpy with result and the feature value, and store this in result. hstack() stacks arrays in sequence horizontally (in a columnar fashion). Then, return the result.

```
#DataFlair - Extract features (mfcc, chroma, mel) from a sound file
def extract_feature(file_name, mfcc, chroma, mel):
    with soundfile.SoundFile(file_name) as sound_file:
        X = sound_file.read(dtype="float32")
        sample_rate=sound_file.samplerate
        if chroma:
            stft=np.abs(librosa.stft(X))
        result=np.array([])
        if mfcc:
            mfccs=np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,
axis=0)
            result=np.hstack((result, mfccs))
        if chroma:
            chroma=np.mean(librosa.feature.chroma_stft(S=stft,
sr=sample_rate).T,axis=0)
            result=np.hstack((result, chroma))
if mel:
            mel=np.mean(librosa.feature.melspectrogram(X,
sr=sample_rate).T,axis=0)
            result=np.hstack((result, mel))
return result
```

Screenshot:

```
[2]: #DataFlair - Extract features (mfcc, chroma, mel) from a sound file
     def extract_feature(file_name, mfcc, chroma, mel):
         with soundfile.SoundFile(file_name) as sound_file:
             X = sound_file.read(dtype="float32")
             sample_rate=sound_file.samplerate
             if chroma:
                 stft=np.abs(librosa.stft(X))
             result=np.array([])
             if mfcc:
                 mfccs=np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
                 result=np.hstack((result, mfccs))
             if chroma:
                 chroma=np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
                 result=np.hstack((result, chroma))
             if mel:
                 mel=np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
                 result=np.hstack((result, mel))
         return result
```

3. Now, let's define a ***dictionary***to hold numbers and the emotions available in the RAVDESS dataset, and a list to hold those we want- calm, happy, fearful, disgust.

```
#DataFlair - Emotions in the RAVDESS dataset
emotions={
  '01':'neutral',
  '02':'calm',
  '03':'happy',
  '04':'sad',
  '05':'angry',
  '06':'fearful',
  '07':'disgust',
  '08':'surprised'
}

#DataFlair - Emotions to observe
observed_emotions=['calm', 'happy', 'fearful', 'disgust']
```

Screenshot:

4. Now, let's load the data with a function load_data() – this takes in the relative size of the test set as parameter. x and y are empty lists; we'll use the glob() function from the glob module to get all the pathnames for the sound files in our dataset. The pattern we use for

this is: “D:\\DataFlair\\ravdess data\\Actor_**.wav”. This is because our dataset looks like this:

Screenshot:

So, for each such path, get the basename of the file, the emotion by splitting the name around ‘-’ and extracting the third value:

Screenshot:

Using our emotions dictionary, this number is turned into an emotion, and our function checks whether this emotion is in our list of observed_emotions; if not, it continues to the next file. It makes a call to extract_feature and stores what is returned in ‘feature’. Then, it appends the feature to x and the emotion to y. So, the list x holds the features and y holds the emotions. We call the function train_test_split with these, the test size, and a random state value, and return that.

```
#DataFlair - Load the data and extract features for each sound file
def load_data(test_size=0.2):
    x,y=[],[]
    for file in glob.glob("D:\\DataFlair\\ravdess data\\Actor_*\\*.wav"):
        file_name=os.path.basename(file)
        emotion=emotions[file_name.split("-")[2]]
        if emotion not in observed_emotions:
            continue
        feature=extract_feature(file, mfcc=True, chroma=True, mel=True)
        x.append(feature)
        y.append(emotion)
    return train_test_split(np.array(x), y, test_size=test_size, random_state=9)
```

Screenshot:

```
[4]: #DataFlair - Load the data and extract features for each sound file
     def load_data(test_size=0.2):
         x,y=[],[]
         for file in glob.glob("D:\\DataFlair\\ravdess data\\Actor_*\\*.wav"):
             file_name=os.path.basename(file)
             emotion=emotions[file_name.split("-")[2]]
             if emotion not in observed_emotions:
                 continue
             feature=extract_feature(file, mfcc=True, chroma=True, mel=True)
             x.append(feature)
             y.append(emotion)
         return train_test_split(np.array(x), y, test_size=test_size, random_state=9)
```

5. Time to split the dataset into training and testing sets! Let's keep the test set 25% of everything and use the load_data function for this.

```
#DataFlair - Split the dataset
x_train,x_test,y_train,y_test=load_data(test_size=0.25)
```

Screenshot:

6. Observe the shape of the training and testing datasets:

```
#DataFlair - Get the shape of the training and testing datasets
print((x_train.shape[0], x_test.shape[0]))
```

Screenshot:

7. And get the number of features extracted.

```
#DataFlair - Get the number of features extracted
print(f'Features extracted: {x_train.shape[1]}')
```

Output Screenshot:

8. Now, let's initialize an MLPClassifier. This is a Multi-layer Perceptron Classifier; it optimizes the log-loss function using LBFGS or stochastic gradient descent. Unlike SVM or ***Naive Bayes***, the MLPClassifier has an internal neural network for the purpose of classification. This is a feedforward ANN model.

```
#DataFlair - Initialize the Multi Layer Perceptron Classifier
model=MLPClassifier(alpha=0.01, batch_size=256, epsilon=1e-08, hidden_layer_sizes=
(300,), learning_rate='adaptive', max_iter=500)
```

Screenshot:

```
[8]: #DataFlair - Initialize the Multi Layer Perceptron Classifier
     model=MLPClassifier(alpha=0.01, batch_size=256, epsilon=1e-08, hidden_layer_sizes=(300,), learning_rate='adaptive', max_iter=500)
```

9. Fit/train the model.