Speech Emotion Recognition

What is Speech Emotion Recognition?

The technique of attempting to understand human emotion and affective states from speech is known as Speech Emotion Recognition (SER). This takes use of the fact that voice tone and pitch frequently reflect underlying emotion. This is the same phenomena that animals like dogs and horses use to comprehend human emotion. Because emotions are subjective and annotating audio is difficult, SER is difficult.

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.

Speech Emotion Recognition – Objective

Using the librosa and sklearn libraries, as well as the RAVDESS dataset, create a model that distinguish emotion from voice.

Download dataset from here

1. Make the necessary imports:

In [1]:
```
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
import warnings
warnings.filterwarnings("ignore")
```

1. 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

In [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
```

1. 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.

In [3]:
```
#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']
```

This number is converted to an emotion using our emotions dictionary, and our function checks whether this emotion is in our list of observed emotions; if not, it moves on to the next file. It calls extract feature and saves the result in the 'feature' variable. The feature is then appended to x, while the feeling is appended to y. As a result, the list x contains the features and the list y has the feelings. With these, the test size, and a random state value, we use train test split and return it.

In [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("speech-emotion-recognition-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)
```

1. 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.

In [5]:
```
#DataFlair - Split the dataset
x_train,x_test,y_train,y_test=load_data(test_size=0.25)
```

6.Let's now create an MLPClassifier. This is a Multi-layer Perceptron Classifier that uses LBFGS or stochastic gradient descent to improve the log-loss function. The MLPClassifier, unlike SVM or Naive Bayes, has an inbuilt neural network for classification. This is an ANN model that is fed forward.

In [6]:
```
#DataFlair - Initialize the Multi Layer Perceptron Classifier
model=MLPClassifier(alpha=0.01, batch_size=256, epsilon=1e-08, hidden_layer_sizes=(300,), learning_rate='adapti
```

1. Fit/train the model.

In [7]:
```
#DataFlair - Train the model
model.fit(x_train,y_train)
```

Out[7]:
```
MLPClassifier(alpha=0.01, batch_size=256, hidden_layer_sizes=(300,),
              learning_rate='adaptive', max_iter=500)
```

1. Let's predict the values for the test set. This gives us y_pred

In [8]:
```
#DataFlair - Predict for the test set
y_pred=model.predict(x_test)
```

1. To calculate the accuracy of our model, we'll call up the accuracy_score() function we imported from sklearn. Finally, we'll round the accuracy to 2 decimal places and print it out.

In [9]:
```
#DataFlair - Calculate the accuracy of our model
accuracy=accuracy_score(y_true=y_test, y_pred=y_pred)

#DataFlair - Print the accuracy
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 73.96%
```

Summary

We learned to discern emotions from speech in this project. We utilised an MLPClassifier for this, as well as the soundfile and librosa libraries to read and extract features from the sound file. As you can see, the model had a **73.96%** percent accuracy rate. That will suffice for now.

In [ ]:
```

```