```
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


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


# Emotions to recognize
emotions_to_recognize=['angry','disgust','surprised','calm','neutral','happy','sad','fearful']


#Extracting features from audio files
def feature_extraction(file_name):
    with soundfile.SoundFile(file_name) as sound_file:
        X = sound_file.read(dtype="float32")
        sample_rate=sound_file.samplerate
        result=np.array([])
        # STFT
        stft=np.abs(librosa.stft(X))
        # MFCC
        mfccs=np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
        result=np.hstack((result, mfccs))
        # Chroma_STFT
        chroma=np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
        result=np.hstack((result, chroma))
        # Mel Spectrogram
        mel=np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
        result=np.hstack((result, mel))
        # As part of Phase II, Adding More features
        # Zero Crossing Rate
        zcr = np.mean(librosa.feature.zero_crossing_rate(y=X).T, axis=0)
        result=np.hstack((result, zcr))
        # CQT
        cqt = np.mean(librosa.feature.chroma_cqt(y=X, sr=sample_rate).T, axis=0)
        result = np.hstack((result, cqt))
        # Chroma CENS
        chroma_cens = np.mean(librosa.feature.chroma_cens(y=X, sr=sample_rate).T, axis=0)
        result = np.hstack((result, chroma_cens))
        # Spectral Centroid & Bandwidth
        chroma_spectral_centroid = np.mean(librosa.feature.spectral_centroid(S=stft, sr=sample_rate).T, axis=0)
        chroma_spectral_bandwidth = np.mean(librosa.feature.spectral_bandwidth(S=stft, sr=sample_rate).T, axis=0)
        chroma_spectral_contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T, axis=0)
        chroma_spectral_flatness = np.mean(librosa.feature.spectral_flatness(S=stft).T, axis=0)
        chroma_spectral_rolloff = np.mean(librosa.feature.spectral_rolloff(S=stft, sr=sample_rate).T, axis=0)
        chroma_spectral_poly_features = np.mean(librosa.feature.poly_features(S=stft, order=2).T, axis=0)
        result = np.hstack((result, chroma_spectral_poly_features))
        result = np.hstack((result, chroma_spectral_centroid))
        result = np.hstack((result, chroma_spectral_rolloff))
        result = np.hstack((result, chroma_spectral_flatness))
        result = np.hstack((result, chroma_spectral_bandwidth))
        result = np.hstack((result, chroma_spectral_contrast))
        # Root Mean Square Value
        rms = np.mean(librosa.feature.rms(y=X).T, axis=0)
        result = np.hstack((result, rms))
        ## Tonnetz
        harmonic = np.abs(librosa.effects.harmonic(X))
        tonnetz = np.mean(librosa.feature.tonnetz(y=harmonic, sr=sample_rate).T, axis=0)
        result = np.hstack((result, tonnetz))
        return result


# Data Loading and feature extraction for each sound file
def load_data(test_size=0.2):
    x,y=[],[]
    for file in glob.glob('/content/drive/MyDrive/RAVDESS/**/*.wav'):
        file_name=os.path.basename(file)
        emotion=emotions[file_name.split("-")[2]]
```

```
            if emotion not in emotions_to_recognize:
                continue
            feature=feature_extraction(file)
            x.append(feature)
            y.append(emotion)
        return train_test_split(np.array(x), y, test_size=test_size, random_state=9)

# Dataset splitting for training and testing
x_train,x_test,y_train,y_test= load_data(test_size=0.2)


# training and testing datasets
print((x_train.shape[0], x_test.shape[0]))
```

```
(1152, 288)
```

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

```
Features extracted: 226
```

```
# Model Initialization -> 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)


# Model Training
model.fit(x_train,y_train)
```

```
MLPClassifier(alpha=0.01, batch_size=256, hidden_layer_sizes=(300,),
              learning_rate='adaptive', max_iter=500)
```

```
# Model Prediction
y_pred=model.predict(x_test)


# Model Accuracy
accuracy=accuracy_score(y_true=y_test, y_pred=y_pred)
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 23.26%
```

Therefore adding all features has negative effect (curse of dimensionality), let's assess features individually and decide which features to be combined accordingly