E7 PROJECT DS203

Transforming Audio Data: Machine Learning Strategies for Classification and Prediction

Sameer Gupta(23B2509)
Ayush V Awatade(23B2524)
Ruchir Kulkarni(23B2483)
Vidhi Tejas Morzaria(23B2485)

PROBLEM STATEMENT

- We have been given 115 MFCC files extracted from solo songs sung by different singers.
- The files consist of renditions of National Anthem, Marathi lavani and Bhaav Geet songs, Hindi songs by Asha Bhosale and Kishore Kumar and English songs by Michael Jackson.
- The provided MFCC files do not contain the labels belonging to the artists who sung that song.

OBJECTIVES

- Analyzing MFCC files and categorizing them into various clusters, as suggested in Problem Statement.
- Detecting At least 3 Files containing the Indian National Anthem.
- Detecting At least 3 Files each for solo songs by Asha Bhosale, Kishore Kumar and Michael Jackson.

EXECUTIVE OVERVIEW

- The project leverages several audio analysis techniques to handle the unique characteristics of MFCC files
- Data preprocessing, Feature extraction, Model creation using Neural Networks and testing all given 115 files in our model.

EXECUTIVE OVERVIEW

- The process in creating the model, the problems encountered during the same and the results what were expected and what we got.

- Learnings from the project, approaches thought to tackle the accuracy and the coding work inculcated in the project

APPROACH TO THE PROBLEM

Understanding MFCC files

Analyzing various metrics related to the MFCC files by constructing various matrices and quantities for comparison further

Feature Extraction

Audio representations from MFCC will help us to identify features for classification.

Clustering

K-Means clustering to visualize the clusters and identify properties based on that.

Neural Network

Using Dense Layers Network to process and transform input information into output

UNDERSTANDING LIBROSA

- Here we are just getting acquainted with librosa library.
- We have given input a .wav file and wish to analyze its sampling rate and corresponding audio data.
- Then we plot the Audio Data, basically the Amplitude v/s time graph of the song.
- We can clearly see that the initial and final parts of the song are at amplitude=0, which is indeed obivious.

Checking some functions of librosa for data analysis

```python
### Let's read a sample audio using librosa
import librosa
audio_file_path='train/bollywood_STK 1982 - Kitne Bhi Tu(Female).wav'
librosa_audio_data, librosa_sample_rate=librosa.load(audio_file_path,sr = 44100)
librosa_sample_rate
```

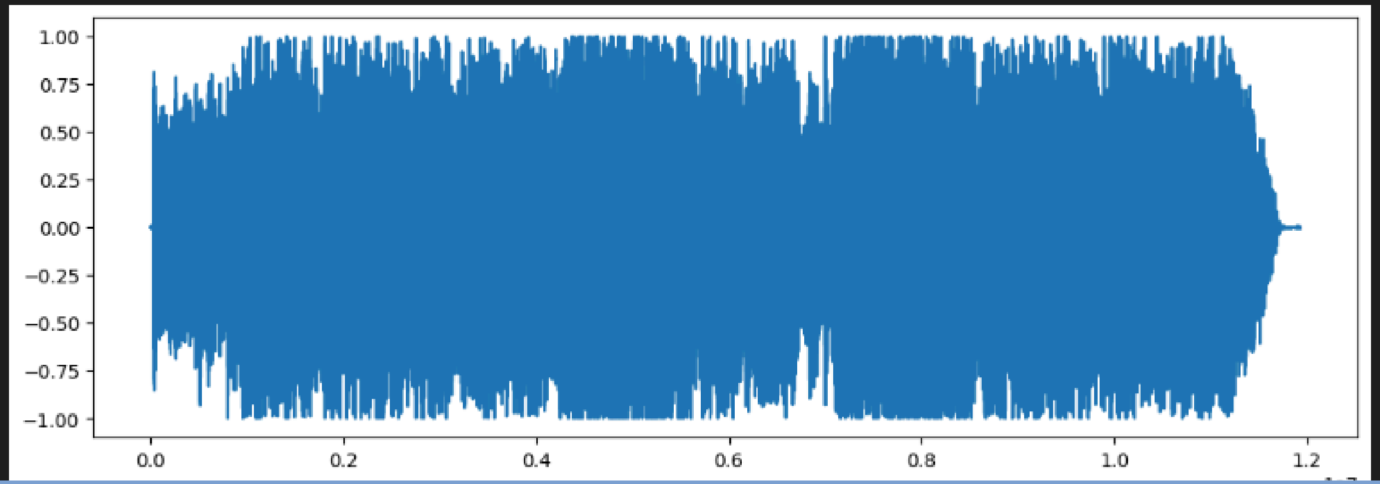
[390] Python

```
44100
```

```python
### Lets plot the librosa audio data
import matplotlib.pyplot as plt
# Original audio with 1 channel
plt.figure(figsize=(12, 4))
plt.plot(librosa_audio_data)
```

[3] Python

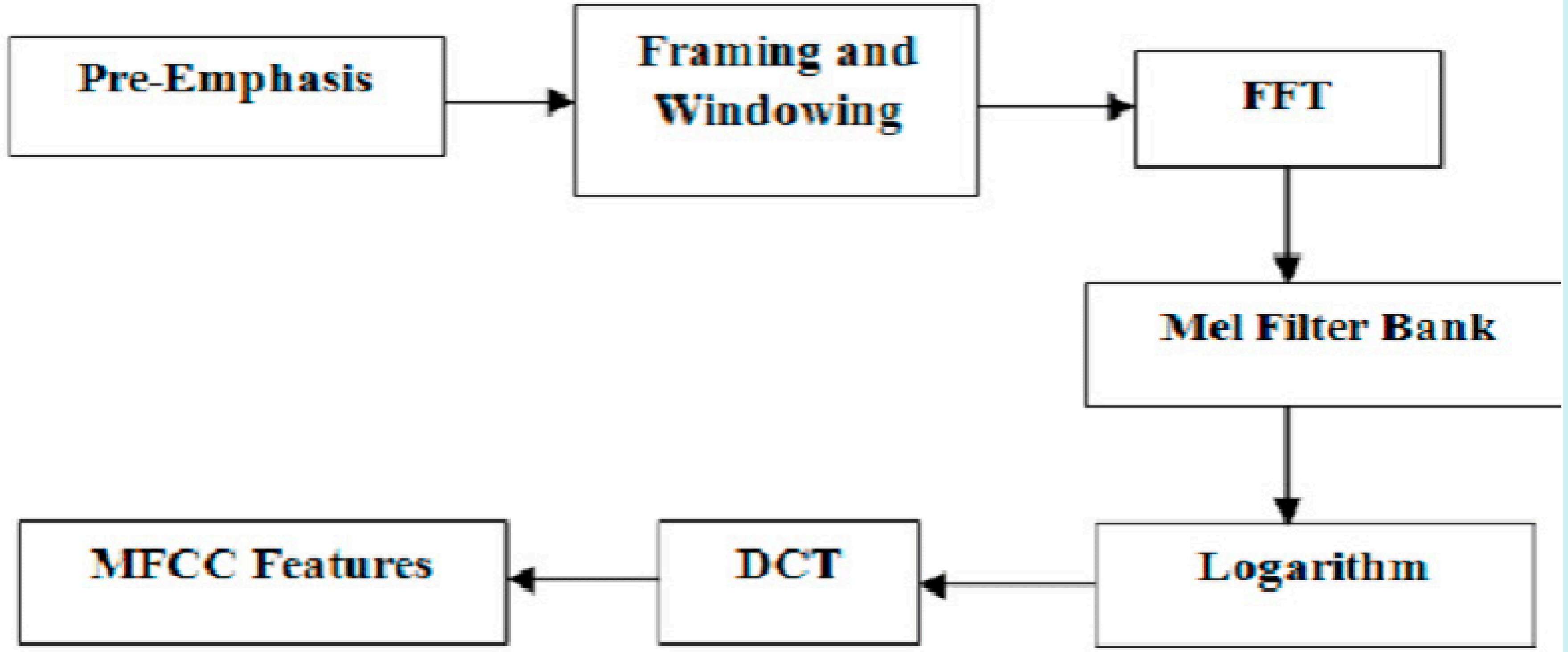
```
[<matplotlib.lines.Line2D at 0x7ecf1a4a80a0>]
```



UNDERSTANDING MFCC

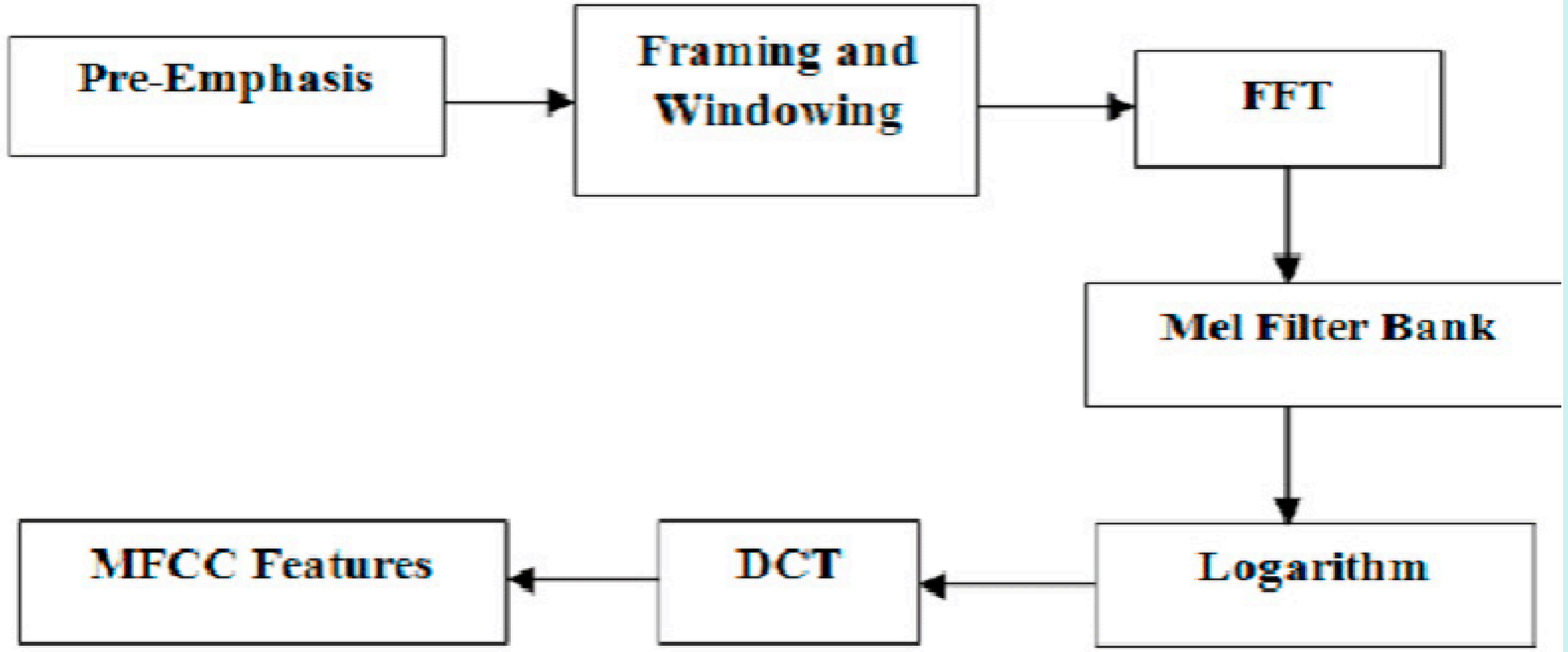
- **Framing and Windowing:** The audio signal is divided into small frames (typically 20-40ms) and a window function with a standard stepsize of 10ms is applied to each frame to reduce spectral leakage

- **Fast Fourier Transform (FFT):** Converts each frame from the time domain to the frequency domain.



UNDERSTANDING MFCC

- **Mel Filter Bank:** The spectrum is passed through a set of filters spaced according to the Mel scale, mimicking human auditory perception
- **Logarithmic Compression:** Converts the filter bank energies into a logarithmic scale to match human loudness perception
- **Discrete Cosine Transform (DCT):** Used to decorrelate and compress Mel spectrum features into a set of cepstral coefficients



FEATURE EXTRACTION

- The MFCC summarizes the frequency distribution across the window size, so it is possible to analyze both frequency and time characteristics of the sound.

- These MFCC files have features in form of coefficients of various quantities and span a 2D space of 20 X 23304

```python
mfccs = librosa.feature.mfcc(y=librosa_audio_data, sr=librosa_sample_rate, n_mfcc=20)
print(mfccs.shape)
```

```
(20, 23304)
```

```python
mfccs
```

[5] Python

```
array([[-505.37363, -505.37363, -505.37363, ..., -505.37363, -505.37363,
        -505.37363],
       [   0.     ,    0.     ,    0.     , ...,    0.     ,    0.     ,
           0.     ],
       [   0.     ,    0.     ,    0.     , ...,    0.     ,    0.     ,
           0.     ],
       ...,
       [   0.     ,    0.     ,    0.     , ...,    0.     ,    0.     ,
           0.     ],
       [   0.     ,    0.     ,    0.     , ...,    0.     ,    0.     ,
           0.     ],
       [   0.     ,    0.     ,    0.     , ...,    0.     ,    0.     ,
           0.     ]], dtype=float32)
```

FEATURE EXTRACTION

- The .wav file is converted into MFCC and store it into a Merged csv file named metadata.

- We create a function named features_extractor which takes .wav file as input and returns the mfcc_features after scaling them.

```python
#### Extracting MFCC's For every audio file
import pandas as pd
import os
import librosa

audio_dataset_path='train/'
metadata=pd.read_csv('train/Merged csv - song_files_with_class_exact.csv')
# metadata
```

```python
def features_extractor(file):
    audio, sample_rate = librosa.load(file_name,sr = 44100)
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=20)
    # print(mfccs_features)
    mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

    return mfccs_scaled_features
```

Python

STORING FEATURES

- We iterate through all the files and store their extracted features in an array named extracted_features

- We also checked how would this work using a input song and extracted its features.

```
import numpy as np
### Now we iterate through every audio file and extract features
### using Mel-Frequency Cepstral Coefficients
extracted_features=[]
for index_num,row in (metadata.iterrows()):
    file_name = os.path.join('train/',str(row["slice_file_name"]))
    # print(file_name)
    final_class_labels=row["class"]
    data=features_extractor(file_name)
    extracted_features.append([data,final_class_labels])
```

```
# file_name = os.path.join('train/Baj Gayi Ghanti.wav')
# audio, sample_rate = librosa.load(file_name,sr = 44100)
```

Python

STORING FEATURES

- One of our teammates' had already collected the data for national anthem in the form of .csv file instead of .wav.

- So for this reason we slightly modified our code to extract the features of national anthem from the .csv file and append it in same array extracted_features

```python
# extracted_features=[]
# filename = 'train/old_MJS-Aao Kanhai Mere Dham.wav'
# data=features_extractor(filename)
# data
# # extracted_features.append([data,final_class_labels])
# # extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','class'])
# # extracted_features_df.tail()

# extracted_features=[]
metadata=pd.read_csv('anthem/anthem-data.csv')
for index_num,row in (metadata.iterrows()):
    file_name = os.path.join('anthem/',str(row["slice_file_name"]))
    # print(file_name)
    final_class_labels=row["class"]
    # data=features_extractor(file_name)
    mfcc_features = pd.read_csv(file_name)
    # mfcc_features = mfcc_features.drop(index=0).reset_index(drop=True)
    num_frames, num_coeffs = mfcc_features.shape
    # Convert the DataFrame to a NumPy array
    mfcc_array = mfcc_features.values
    # Reshape the array to its original dimensions
    mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
    # print(mfcc_matrix)
    mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
    extracted_features.append([mfccs_scaled_features,final_class_labels])
```

TEST FEATURES OTHER THAN MFCC

1) Zero Crossing Rate:

- The Number of times the soundwave cross zero.

2) Rollof Frquency:

- The frequency above or below which a filter begins to filter out the harmonics of the waveform.

3)Spectral contrast:

- Spectral contrast considers the spectral peak, the spectral valley, and their difference in each frequency sub band.

TEST FEATURES OTHER THAN MFCC

4) Chroma_stft:

- Compute a chromagram from a waveform or power spectrogram.

Spectral Centroid:

It is calculated as the [[weighted mean]] of the frequencies present in the signal, determined using a [[Fourier transform]], with their magnitudes as the weights:

$$\text{Centroid} = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

where "x(n)" represents the weighted frequency value, or magnitude, of [[Histogram|bin]] number "n", and "f(n)" represents the center frequency of that bin.

As the test files provided on Moodle are MFCC files there is no need to account for these features rather only Mel frequency Cepstral Coefficients is enough.

CREATING A DATAFRAME

```python
### converting extracted_features to Pandas dataframe
extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','class']
extracted_features_df.tail()
```

[28] Python

...

	feature	class
69	[-256.01001272733987, 139.39327876617466, -7.1...	anthem
70	[-226.047499140475, 153.0845083156622, -5.7326...	anthem
71	[-138.5192946992875, 168.25097320949138, -66.3...	anthem
72	[-234.53966056481298, 126.21139462388994, -17....	anthem
73	[-355.473327295414, 174.76897189419591, -14.28...	anthem

- Before moving on to the clustering of data we convert the array extracted_features into pandas Dataframe for clustering and encoding.
- To re-iterate, our dataframe consists of features from songs of Asha Bhosale, Kishore Kumar, Michael Jackson and the National Anthem.

CLUSTERING

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Convert the 'feature' column to a list of lists
feature_list = extracted_features_df['feature'].tolist()

# Determining the number of elements in each feature list
num_features = len(feature_list[0])
column_names = [f"feature_{i}" for i in range(num_features)]

# Create a new DataFrame with the split features
new_df = pd.DataFrame(feature_list, columns=column_names)

# Concatenate the new DataFrame with the 'class' column
final_df = pd.concat([new_df, extracted_features_df['class']], axis=1)

# Prepare data
X = final_df.drop(['class'], axis=1)
Y = final_df['class']
```

- Here we are preparing the Data for clustering by altering the Dataframe into a new dataframe.
- We determine the total elements in each feature list and store them with the names of their respective class by concatenating with the class column.

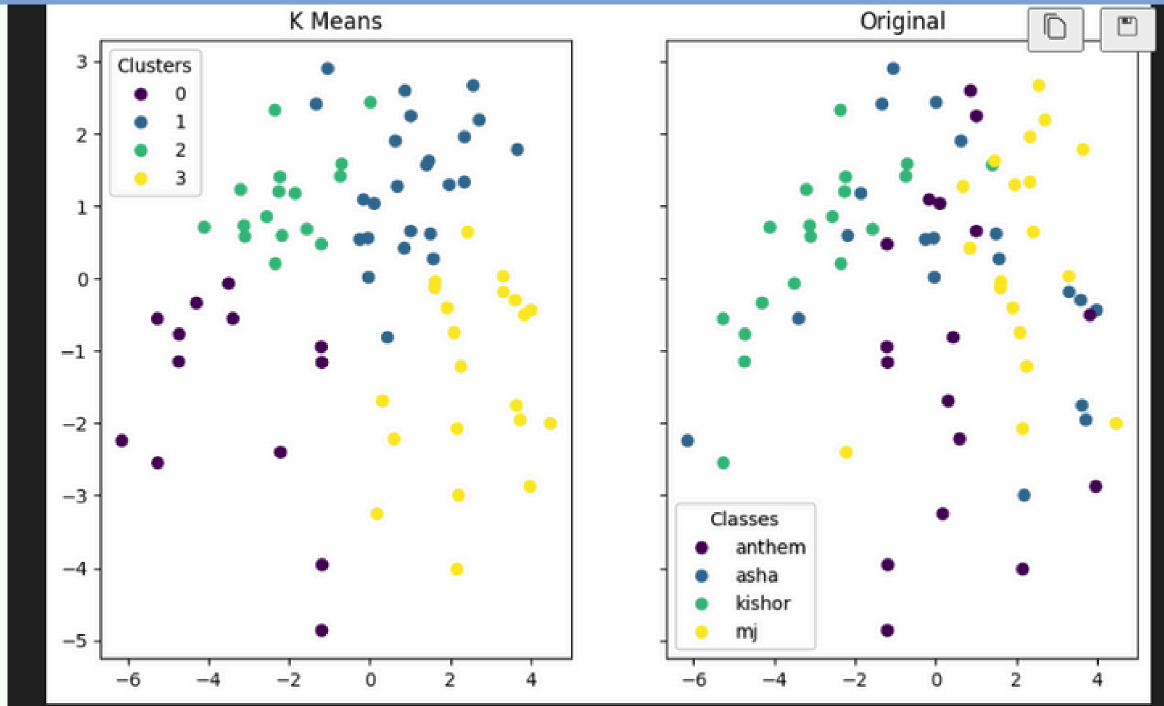
CLUSTERING

```python
# Standardize features
scale = StandardScaler()
X_scaled = pd.DataFrame(scale.fit_transform(X), columns=X.columns)

# Encode labels
encoder = LabelEncoder()
Y_enc = encoder.fit_transform(Y)
from sklearn.decomposition import PCA

pca = PCA(n_components=3,whiten=False);
pca.fit(X_scaled)
xPCA = pca.transform(X_scaled)
# Perform KMeans clustering
kmeans = KMeans(n_clusters=4)
kmeans.fit(xPCA)

# Plot with legends
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(10, 6))
ax1.set_title('K Means')
scatter1 = ax1.scatter(xPCA[:, 0], xPCA[:, 1], c=kmeans.labels_, cmap='viridis')
legend1 = ax1.legend(*scatter1.legend_elements(), title="Clusters")
ax1.add_artist(legend1)

ax2.set_title("Original")
scatter2 = ax2.scatter(xPCA[:, 0], xPCA[:, 1], c=Y_enc, cmap='viridis')
# Map class names to encoded values for legend labels
legend_labels = {i: label for i, label in enumerate(encoder.classes_)}
legend2 = ax2.legend(handles=scatter2.legend_elements()[0], labels=[legend_labels[i] for i in range(len(legend_labels))], title="Classes")
ax2.add_artist(legend2)

plt.show()
```

- We then Standardize or scale the features and store them in X-scaled dataframe.
- The Target variable y is encoded for the classes as 0, 1, 2, 3 and stored in Y_enc.
- Further to reduce the Dimensionality of our data and to make the clustering more efficient we apply PCA considering 3 most important feature DImensions.

- Then we perform K-means clustering on our Dataframe for n=4 clusters.

CLUSTERING




```
Y_enc
```

Python

```
array([2, 1, 1, 2, 2, 2, 1, 2, 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 1, 2, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0])
```

```
### Label Encoding
# y=np.array(pd.get_dummies(y))
genre_nums = {}
cnt = 0
for genre in np.unique(y):
    genre_nums[genre] = cnt
    cnt += 1

# Use vectorized operations for efficiency
y = np.vectorize(lambda x: genre_nums[x])(y)
```

[33]

```
genre_nums
```

[34]

```
{np.str_('anthem'): 0,
 np.str_('asha'): 1,
 np.str_('kishor'): 2,
 np.str_('mj'): 3}
```

- Due to encoding of the Target variable 0 is mapped to anthem, 1 to Asha, 2 to Kishor and 3 to Michael Jackson.
- The Y_enc frame contains the encoded data of our training dataset and maps the songs to respective classes.
- The clustering helps us to analyze the pitch difference and some outliers of various songs according to their Euclidean distances from each other and cluster centroids.

TRAINING DATASET FOR MODEL

- Using the train_test_split from sklearn we make our test and train datasets from the dataframe processed till now. (20% test data)
- We then check the dimensions of X_train, X_test, Y_train, Y_test and move on towards our model creation.

```python
[39] X_train.shape
... (59, 20)

[40] X_test.shape
... (15, 20)

[41] y_train.shape
... (59, 4)

[42] y_test.shape
... (15, 4)
```

```python
[35]
y = np.array(extracted_features_df['class'].tolist())
y=np.array(pd.get_dummies(y))
# y.shape
```
Python

```python
[36]
### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```
Python

```python
[37]
X_train
```
Python

```
array([[-177.75219727,   97.46325684,  -23.72146606, ...,    0.49535388,
          -1.08798146,    1.6080426 ],
       [-138.5192947 ,  168.25097321,  -66.30995151, ...,   -0.57857085,
          -3.55409103,    4.23771722],
       [-127.4241333 ,  121.16482544,  -37.57203293, ...,   -9.34692764,
           8.75722313,   -1.25331104],
       ...,
       [ -90.48006439,  103.31533813,  -74.73695374, ...,   -2.86094785,
           5.38835049,   -2.31573343],
       [-108.60476685,  125.31771851,  -44.90581512, ...,   -3.75181627,
           8.47409153,   -1.95529771],
       [-183.94369507,  185.28294373,  -77.26333618, ...,   -5.48226309,
          -1.31454599,   -6.8033843 ]])
```

MODEL CREATION

- We first Import Tensorflow module as we are using Neural network to train our model.
- As we know that neural Network uses the layering of perceptrons to train the model, so we need an activation function for each layer of our model.

```
import tensorflow as tf
print(tf.__version__)
```

```
2.18.0
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten
from tensorflow.keras.optimizers import Adam
from sklearn import metrics
```

MODEL CREATION

- We use Dense type of layering for building our sequential model.
- The first 3 layers are activated by the ReLU function which is Recitified Linear Unit.
- The Final layer is activated by the softmax function.

```
model=Sequential()
###first layer
model.add(Dense(100,input_shape=(20,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###second layer
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###third layer
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.5))

###final layer
model.add(Dense(num_labels))
model.add(Activation('softmax'))
```

[48]

MODEL CREATION

- We on getting the model.summary() see that there is no Non-trainable parameter in our dataset.
- On analyzing the summary we now go on to train our model with epochs=200 and find out the accuracy and loss of our model.

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 100)	2,100
activation (Activation)	(None, 100)	0
dropout (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 200)	20,200
activation_1 (Activation)	(None, 200)	0
dropout_1 (Dropout)	(None, 200)	0
dense_2 (Dense)	(None, 100)	20,100
activation_2 (Activation)	(None, 100)	0
dropout_2 (Dropout)	(None, 100)	0
dense_3 (Dense)	(None, 4)	404
activation_3 (Activation)	(None, 4)	0

Total params: 42,804 (167.20 KB)

Trainable params: 42,804 (167.20 KB)

Non-trainable params: 0 (0.00 B)

TRAINING MODEL

- The Training of the model does not a lot of time as our Dataset is not too large.
- This is because finding solo songs of the artists given without any other pitch interruption was too difficult and we were able to find around 74 songs in all.

```
## Training my model
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 200
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/audio_classification.keras',
                               verbose=1, save_best_only=True)
# checkpointer = ModelCheckpoint(filepath='saved_models/audio_classification.hdf5',
#                                verbose=1, save_best_only=True)
start = datetime.now()

model.fit(X_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(X_test, y_test), callbacks=[checkpointer], verbose=1)


duration = datetime.now() - start
print("Training completed in time: ", duration)
```

```
Epoch 1/200
1/2 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.7500 - loss: 0.7245
Epoch 1: val_loss improved from inf to 1.35584, saving model to saved_models/audio_classification.keras
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step - accuracy: 0.6794 - loss: 0.7422 - val_accuracy: 0.4000 - val_loss: 1.3558
Epoch 2/200
1/2 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.6562 - loss: 0.8302
Epoch 2: val_loss improved from 1.35584 to 1.35327, saving model to saved_models/audio_classification.keras
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step - accuracy: 0.6142 - loss: 0.8542 - val_accuracy: 0.4000 - val_loss: 1.3533
```

TESTING GIVEN FILES

- The test accuracy of our model on the 20% split test dataset turns out to be 53.33%.

```
test_accuracy=model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])
```

[64]

```
0.5333333611488342
```

- Now we will test the model on the test data provided in the 115 MFCC on Moodle.

PREDICTED RESULTS

```
[211]: filename="test/02-MFCC.csv"
       df = pd.read_csv(filename,header = None)
       num_frames, num_coeffs = df.shape

       # Convert the DataFrame to a NumPy array
       mfcc_array = df.values

       # Reshape the array to its original dimensions
       mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
       # print(mfcc_matrix)
       mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
       prediction_feature=mfccs_scaled_features.reshape(1,-1)
       prediction = model.predict(prediction_feature)
       predicted_class = np.argmax(prediction, axis=1)

       1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step

[213]: print(prediction)
       [[1.07302465e-01 8.69345814e-02 9.11501786e-07 8.05762053e-01]]

[214]: print(predicted_class)
       [3]
```

MICHAEL JACKSON (MFCC-2,3,86)

```
[215]: filename="test/03-MFCC.csv"
       df = pd.read_csv(filename,header = None)
       num_frames, num_coeffs = df.shape

       # Convert the DataFrame to a NumPy array
       mfcc_array = df.values

       # Reshape the array to its original dimensions
       mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
       # print(mfcc_matrix)
       mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
       prediction_feature=mfccs_scaled_features.reshape(1,-1)
       prediction = model.predict(prediction_feature)
       predicted_class = np.argmax(prediction, axis=1)

       1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[216]: print(prediction)
       [[1.0738335e-01 2.2003841e-01 3.8831195e-04 6.7218989e-01]]

[217]: print(predicted_class)
       [3]
```

```
[218]: filename="test/86-MFCC.csv"
       df = pd.read_csv(filename,header = None)
       num_frames, num_coeffs = df.shape

       # Convert the DataFrame to a NumPy array
       mfcc_array = df.values

       # Reshape the array to its original dimensions
       mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
       # print(mfcc_matrix)
       mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
       prediction_feature=mfccs_scaled_features.reshape(1,-1)
       prediction = model.predict(prediction_feature)
       predicted_class = np.argmax(prediction, axis=1)

       1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[219]: print(prediction)
       [[0.32603878 0.13281558 0.00183972 0.53930587]]

[220]: print(predicted_class)
       [3]
```

PREDICTED RESULTS

KISHORE KUMAR (MFCC-09,10,29)

```
[221]: filename="test/09-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[222]: print(prediction)
[[2.8210401e-05 3.0785409e-01 6.9211769e-01 1.1220884e-07]]

[223]: print(predicted_class)
[2]
```

```
[224]: filename="test/29-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[225]: print(prediction)
[[4.2926902e-05 1.8038455e-01 8.1957251e-01 1.4500684e-08]]

[226]: print(predicted_class)
[2]
```

```
[227]: filename="test/10-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[228]: print(prediction)
[[1.9859825e-01 5.4420050e-02 7.4694371e-01 3.8045688e-05]]

[229]: print(predicted_class)
[2]
```

PREDICTED RESULTS

```
[230]: filename="test/49-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[231]: print(prediction)
[[0.01210119 0.9248261  0.00418948 0.05888318]]

[232]: print(predicted_class)
[1]
```

ASHA BHOSALE (MFCC-39,11,49)

```
[233]: filename="test/11-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[234]: print(prediction)
[[0.03823081 0.84771407 0.03733438 0.07672071]]

[235]: print(predicted_class)
[1]
```

```
[236]: filename="test/39-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[237]: print(prediction)
[[0.01471743 0.8827947  0.00140638 0.10108155]]

[238]: print(predicted_class)
[1]
```

PREDICTED RESULTS

NATIONAL ANTHEM (MFCC-90,16,107)

```
[242]: filename="test/90-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[243]: print(prediction)
[[5.0513059e-01 8.2814381e-02 4.1561705e-04 4.1163939e-01]]

[244]: print(predicted_class)
[0]
```

```
[249]: filename="test/16-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step

[250]: print(prediction)
[[0.64284164 0.20657974 0.04286269 0.107716  ]]

[251]: print(predicted_class)
[0]
```

```
[252]: filename="test/107-MFCC.csv"
df = pd.read_csv(filename,header = None)
num_frames, num_coeffs = df.shape

# Convert the DataFrame to a NumPy array
mfcc_array = df.values

# Reshape the array to its original dimensions
mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
# print(mfcc_matrix)
mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
prediction_feature=mfccs_scaled_features.reshape(1,-1)
prediction = model.predict(prediction_feature)
predicted_class = np.argmax(prediction, axis=1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

[253]: print(prediction)
[[0.9131814  0.04083217 0.01155213 0.03443427]]

[254]: print(predicted_class)
[0]
```

HURDLES ENCOUNTERED

```
[255]: filename="test/15-MFCC.csv"
       df = pd.read_csv(filename,header = None)
       num_frames, num_coeffs = df.shape

       # Convert the DataFrame to a NumPy array
       mfcc_array = df.values

       # Reshape the array to its original dimensions
       mfcc_matrix = mfcc_array.reshape(num_frames, num_coeffs)
       # print(mfcc_matrix)
       mfccs_scaled_features = np.mean(mfcc_matrix.T,axis=0)
       prediction_feature=mfccs_scaled_features.reshape(1,-1)
       prediction = model.predict(prediction_feature)
       predicted_class = np.argmax(prediction, axis=1)

       1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step

[256]: print(prediction)

       [[0.0784916  0.44603604 0.00213619 0.4733362 ]]
```

- As you can see here the prediction probabilities for Michael jackson as well as Asha are quite close we cant say that the song is of Michael although it is of Michael.
- This is because the pitch and timbre interference of both singers is constructive and requires large dataset for distinguish them at every aspect of a song.

CREATIVE THINKING

- One Approach thought was to take 3/4/5 sec interval of every song where Asha Bhosale has sung to increase the data set and then do the analysis.
- Another approach is to take a song where asha with sung with some other singer and cut out those where some other singer is present and then do the processing.
- We realised this late and also this is time consuming task to cut out voice clips and train, so we have not implemented it in our project.

RESULTS

1)Our Model is able to organize the 115 MFCC files into groups such as:

- Michael Jackson
- Asha Bhosale
- Kishor Kumar
- National Anthem

2)Atleast 3 files containing National Anthem have been identified.

3)Atleast 3 files of songs sung by Asha, Kishor, Michael have been identified.

So we have solved technically 3 problems in the problem statement

LEARNINGS FROM THE PROJECT

1. Getting introduced to features and analysis of working with song classification.

2. Learning the theory behind calculating the MFCC coefficients as well learning a new library librosa which contains other song metrics such as roll off, zero crossing rate etc.

3. As told combining the codes written by us, online resources and LLM is not easy, but introduces to the world of debugging and error solving which in general increased our analytical skills.

4.Working in a cooperative team helped all of us to explore each others' suggestions to build out a best model collectively.

APPENDIX

LINK TO DRIVE LINK WHERE ALL SOURCE AND FILES AND DATA IS STORED - HTTPS://DRIVE.GOOGLE.COM/DRIVE/FOLDERS/1EKQ5KGZXHDDZGAD9UMJXVSAUS323P0LB?USP=SHARING

THANK YOU