MUSIC GENRE CLASSIFCATION AND PREDICTION SYSTEM

J Component Project Report for

CSE4022- NATURAL LANGUAGE PROCESSING

Slot – G2

Bachelor of Technology

in

ECE with Specialization in Internet of Things and Sensors

By

Name	Reg. No.	Phone	Email
Wriddhirup Dutta	*16BIS0145*	*70104470881*	*wriddhirup.dutta2016@vitstudent.ac.in*

Under the guidance of

Prof. SHARMILA BANU K.

School of Computer Science and Engineering

Vellore Institute of Technology, Vellore-632014

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

WINTER 2018-19

CERTIFICATE

This is to certify that the project work entitled "MUSIC GENRE CLASSIFCATION AND PREDICTION SYSTEM" that is being submitted by ***Wriddhirup Dutta*** for **Natural Language Processing** (CSE4022) is a record of bonafide work done under my supervision. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other CAL course.

Place: Vellore

Date : 05. 04 .2018

Signature of Students:

WRIDDHIRUP DUTTA

Signature of Faculty: _____

Sharmila Banu K.

ABSTRACT

This project is primarily aimed to create an automated system to classify and predict music genres. The first step included finding good features that demarcated genre boundaries clearly. A total of five features, namely MFCC vector, Chroma frequencies, spectral roll -off , spectral centroid, zero-crossing rate were used for obtaining feature vectors for the classifiers from the GTZAN genre dataset. The dataset consist of 1000 samples of 10 genres. For detecting the genre of the test audio, an artificial neural network model has been created. The model is further compared with another model made using Support vector machine (SVM). The features from the spectrogram sample is fed as the input and it will show the genre of the song as output. The project is made using python.

INTRODUCTION

Wikipedia states that music genre is a conventional category that identities pieces of music as belonging to a shared tradition or set of conventions." The term genre" is a subject to interpretation and it is often the case that genres may very fuzzy in their definition. Further, genres do not always have sound music theoretic foundations, e.g. - Indian genres are geographically defined, Baroque is classical music genre based on time period. Despite the lack of standard criteria for defining genres, the classification of music based on genres is one of the broadest and most widely used. Genre usually assumes high weight in music recommender systems. Genre classification, until now, had been done manually by appending it to metadata of audio files or including it in album info. This project however aims at content-based classification, focusing on information within the audio rather than extraneously appended information. The traditional machine learning approach for classification is used - find suitable features of data, train classifier on feature data, make redirections. The novel thing that we have tried is the use of ensemble classifier on fundamentally different classifiers to achieve our end goal.

The most influential work on genre classification using machine learning techniques was pioneered by Tzanetakis and Cook. The GTZAN dataset was created by them and is to date considered as a standard for genre classification. Scaringella et al gives a comprehensive survey of both features and classification techniques used in the genre classification. Changsheng Xu et al. have shown how to use support vector machines for this task. Most of the work deals with supervised learning approaches. Riedmiller et al. use unsupervised learning creating a dictionary of features. It gives a detailed account of evaluation of previous work on genre classification.

DATASET

Link: http://opihi.cs.uvic.ca/sound/genres.tar.gz

Getting the dataset might be the most time consuming part of this work. Working with music is a big pain, every file is usually a couple of MBs, there are variety of qualities and parameters of recording (Number of frequencies, Bits per second, etc). However, the biggest pain is copyrighting, there are no legit famous songs dataset, as they would cost money. Tao's paper based on a dataset called GTZAN. This dataset is quite small (100 songs per genre X 10 genres = overall 1,000 songs), and the copyright permission is questionable. This is from my perspective one of the reasons that held him from getting better results. Therefore, I looked up for generating more data to learn from. Eventually I found MSD dataset (Million Song Dataset). It is a freely available collection of audio features and metadata for a million contemporary popular music tracks. Around 280 GB of pure metadata. There is a project on top of MSD called tagtraum which classify MSD songs into genres. The problem now was to get the sound itself, here is where I got a little creative. I found that one of the tags every song have in the dataset is an id from a provider called 7Digital. 7Digital is a SaaS provider for music application, it basically let you stream music for money. I signed up to 7Digital as a developer and after their approval i could access their API. Still any song stream costs money, But I found out that they are enabling to preview random 30 seconds of a song to the user before paying for them. So I started working with the GTZAN dataset as it is free and has less size.

Each audio clips has a length 30 seconds, are 22050Hz Mono 16-bit files. The dataset incorporates samples from variety of sources like CDs, radios, microphone recordings etc. We split the dataset in 0:9 : 0:1 ratio and used k-fold cross validation for reporting the results.

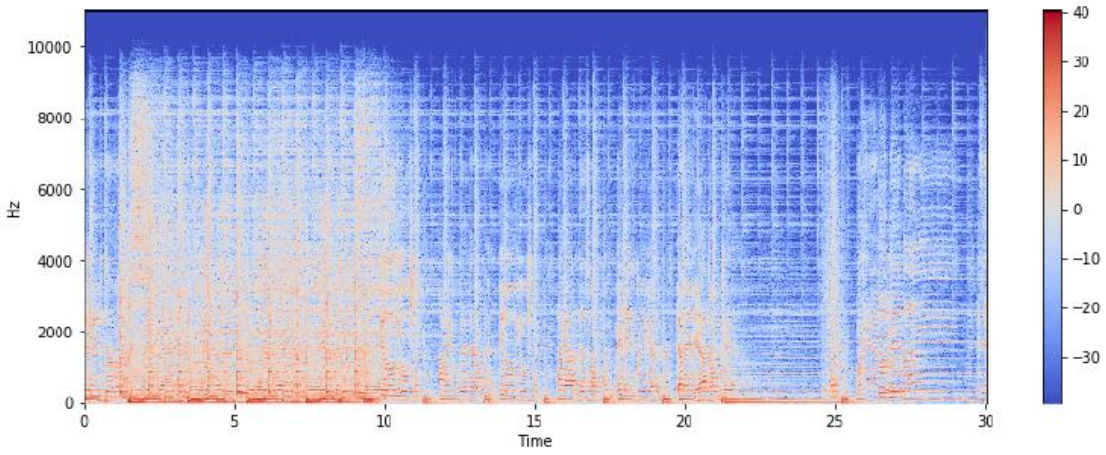
The genres I am classifying are:

1. Blues
2. Classical
3. Country
4. Disco
5. Hip-hop
6. Jazz
7. Metal
8. Pop
9. Reggae
10. Rock

THEORY

Librosa: It is a Python module to analyze audio signals in general but geared more towards music. It includes the nuts and bolts to build a MIR(Music information retrieval) system. It has been very well documented along with a lot of examples and tutorials.

1. **Spectrogram:** A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. Spectrograms are sometimes called sonographs, voiceprints, or voice grams. When the data is represented in a 3D plot, they may be called waterfalls. In 2-dimensional arrays, the first axis is frequency while the second axis is time.



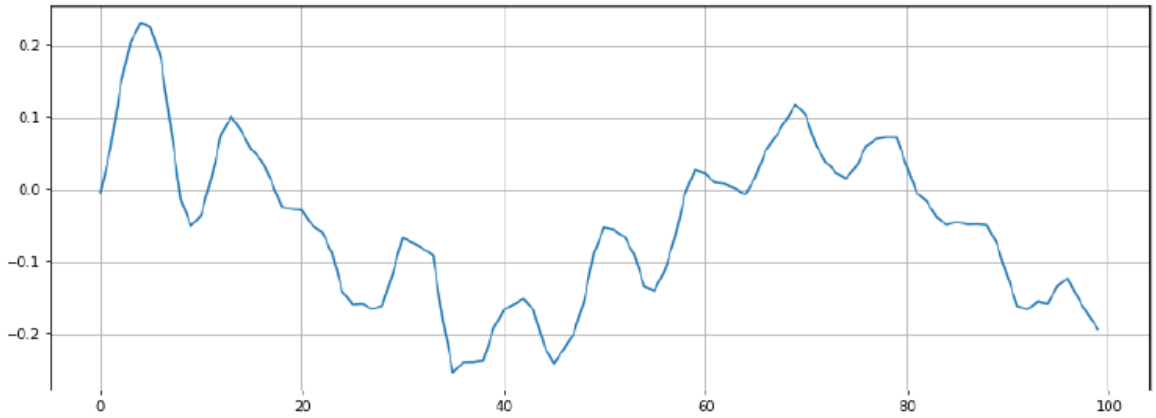
FEATURES EXTRACTED FOR THE PROJECT

1. **Zero Crossing Rate :** The zero crossing rate is the rate of sign-changes along a signal, i.e., the rate at which the signal changes from positive to negative or back. This feature has been used heavily in both speech recognition and music information retrieval. It usually has higher values for highly percussive sounds like those in metal and rock.

ZCR is defined formally as

$$zcr = \frac{1}{T-1}\sum_{t=1}^{T-1} 1_{\mathbb{R}<0}(s_t s_{t-1})$$

, where s is a signal of length T and $1_{R<0}$ is an indicator function.



Here the total ZCR is 8.

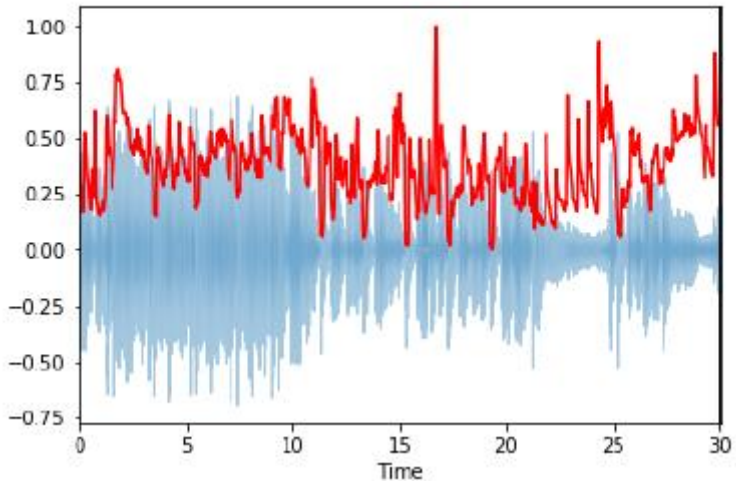
2. **Spectral Centroid:** It indicates where the "centre of mass" for a sound is located and is calculated as the weighted mean of the frequencies present in the sound. Consider two songs, one from a blues genre and the other belonging to metal. Now as compared to the blues genre song, which is the same throughout its length, the metal song has more frequencies towards the end. So spectral centroid for blues song will lie somewhere near the middle of its spectrum while that for a metal song would be towards its end**.**

   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

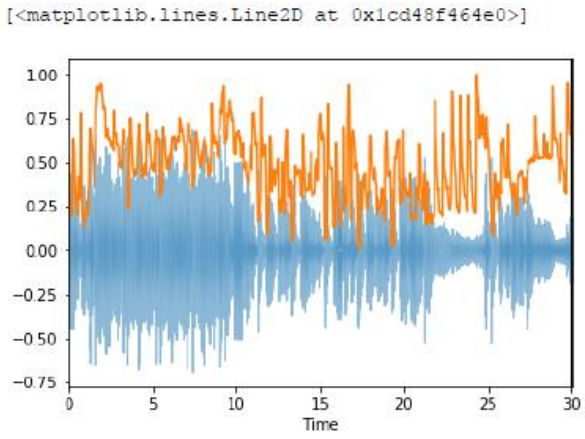
$$\text{Centroid} = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

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

```
[<matplotlib.lines.Line2D at 0x1cd4c11ae10>]
```



3. **Spectral Roll off :** It is a measure of the shape of the signal. It represents the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies.



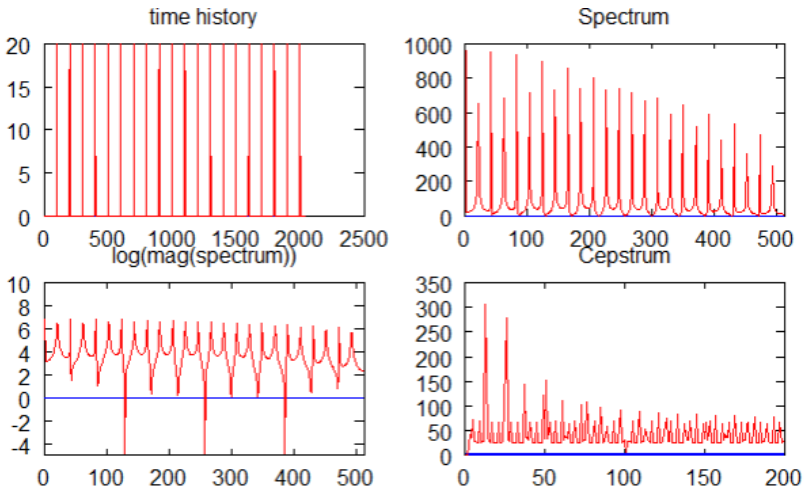
4. **Mel-Frequency Cepstral Coefficients:** The Mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice.

   In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

   They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

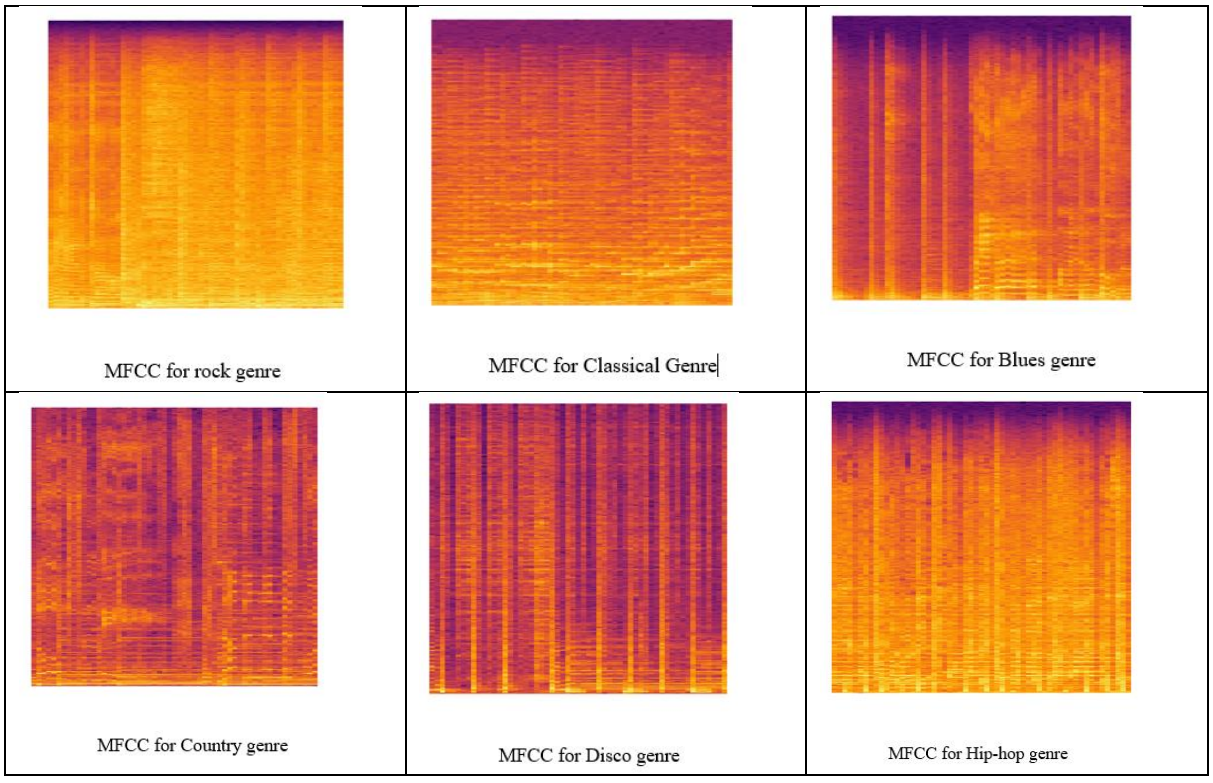
   MFCCs are commonly derived as follows:[2]

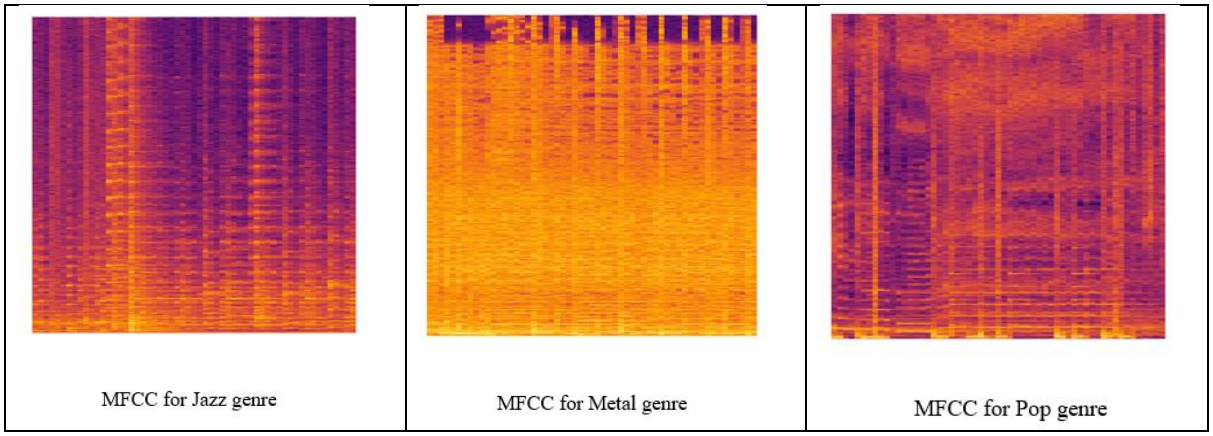
   1. Take the Fourier transform of (a windowed excerpt of) a signal.
   2. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
   3. Take the logs of the powers at each of the mel frequencies.
   4. Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
   5. The MFCCs are the amplitudes of the resulting spectrum.



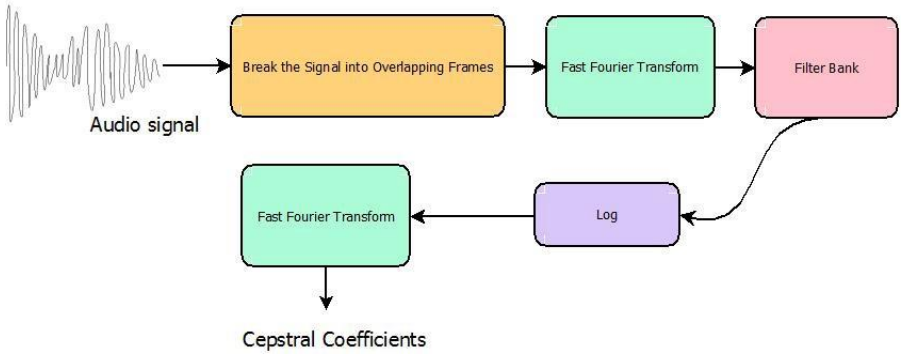
$$\mathrm{Mel}(f) = 2595 \log\left(1 + \frac{f}{700}\right)$$

f is the frequency obtained after Fourier transform of the signal from time domain.





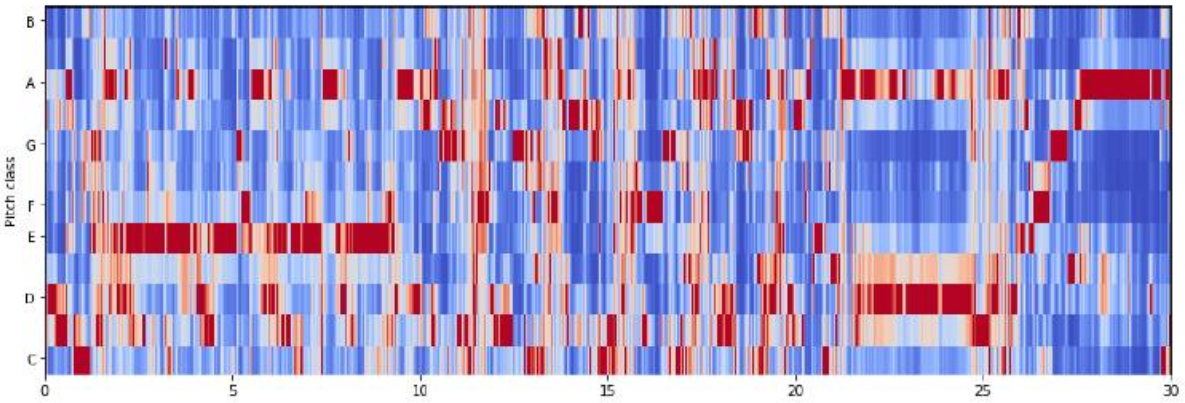
MFCC for Jazz genre | MFCC for Metal genre | MFCC for Pop genre



Here, Filter Bank refers to the Mel filters (converting to mel scale) and Cepstral Coefficients are nothing but MFCCs.

5. **Chroma Frequencies:** Chroma features are an interesting and powerful representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (or chroma) of the musical octave.

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cd4c55e978>
```

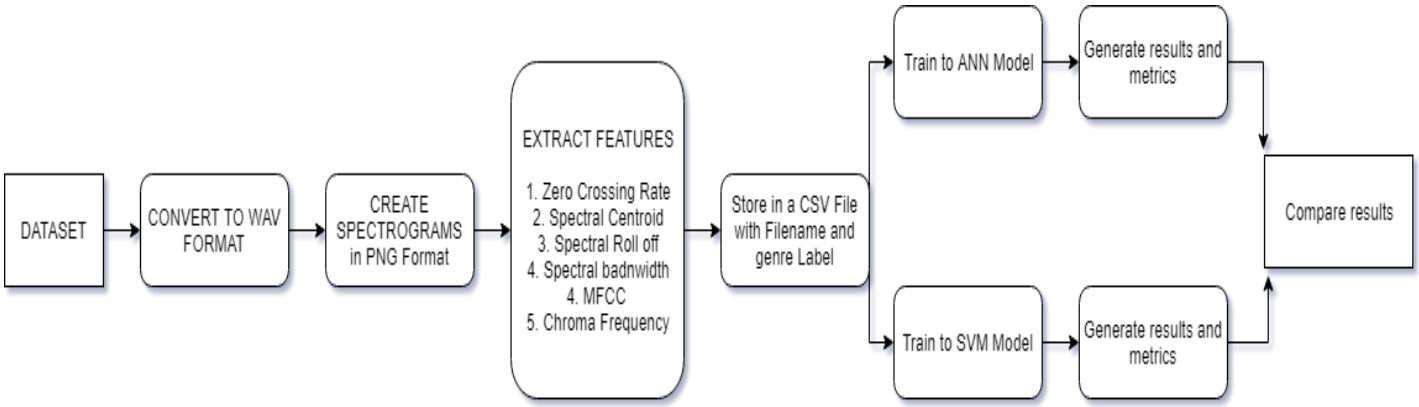


Since, in music, notes exactly one octave apart are perceived as particularly similar, knowing the distribution of chroma even without the absolute frequency (i.e. the

original octave) can give useful musical information about the audio -- and may even reveal perceived musical similarity that is not apparent in the original spectra.

Identifying pitches that differ by an octave, chroma features show a high degree of robustness to variations in timbre and closely correlate to the musical aspect of harmony. This is the reason why chroma features are a well-established tool for processing and analyzing music data. For example, basically every chord recognition procedure relies on some kind of chroma representation. Also, chroma features have become the de facto standard for tasks such as music alignment and synchronization as well as audio structure analysis. Finally, chroma features have turned out to be a powerful mid-level feature representation in content-based audio retrieval such as cover song identification or audio matching.

FLOW DIAGRAM of the PROJECT:



PROCESS:

STEP 1. PRE PROCESSING

I. Data format Conversion

Initially the dataset will be in zip format. The unzipped folder will contain 10 folders of 100 sings each in .au format. The songs cant be read in python directly.

So it needs to be converted to wav/mp3/flac/m4a format. I chose .WAV format for the project as processing for wav is faster and easier. Since there were some .dll file probem while using SOX or FFPEG4 extension in my Laptop, I used online converter to convert .au files to.wav file.

It is done from https://www.docspal.com/convert/au-to-wav

It works faster as 5 files can be converted at a time in a zip folder.

II. Conversion Of Wav Format To Spectrograms

For feature extraction, the spectrograms are needed to be generated. It is one in python

```
plt.figure(figsize=(10,10))
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'.split()
for g in genres:
    pathlib.Path(f'F:/img_data/{g}').mkdir(parents=True, exist_ok=True)
    for filename in os.listdir(f'F:/genres/{g}'):
        songname = f'F:/genres/{g}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=5)
        plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='default', mode='default', scale='dB');
        plt.axis('off');
        plt.savefig(f'F:/img_data/{g}/{filename[:-3].replace(".", "")}.png')
        plt.clf()
```

Here 1000 images are formed in PNG format and placed in 10 folders as made before for the wav file generations. The images are of shape 720x720, almost sizing 100kB each.

Here PNG format is chosen as it has zero or very less compression ratio in comparison to JPEG or JPG format. So the features will be stored properly in the images.

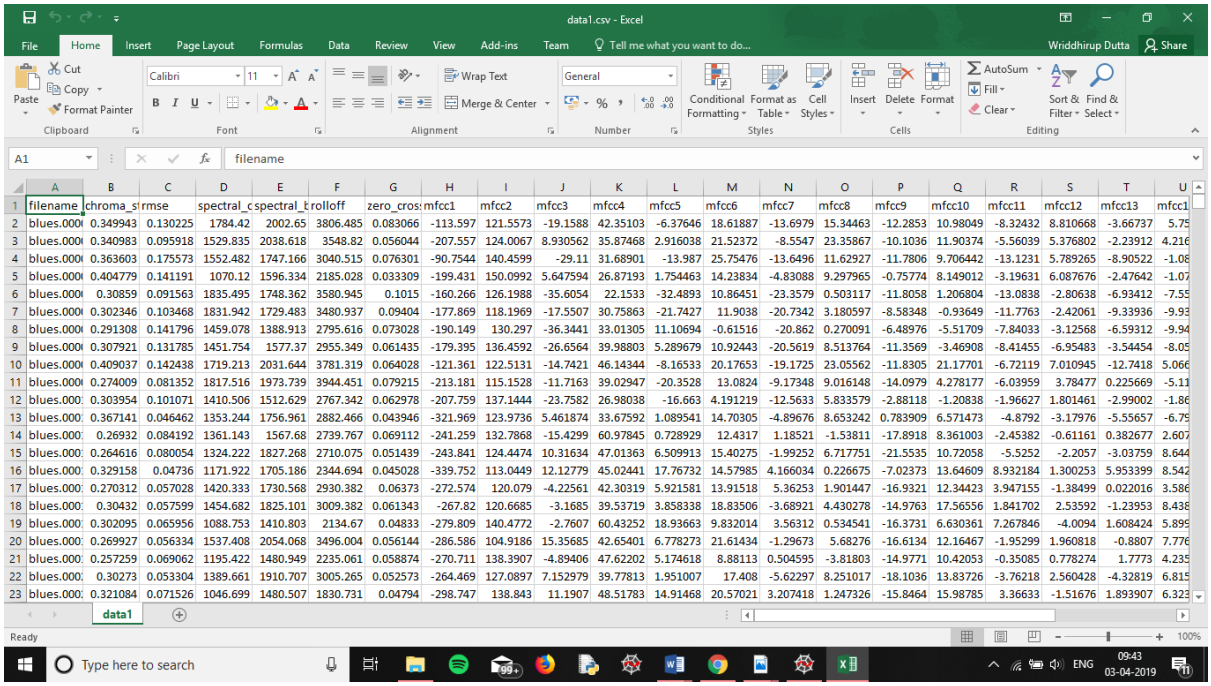
III. *Feature extraction in CSV format*

The Features like Mel-frequency cepstral coefficients (MFCC)(20 in number), Spectral Centroid, Zero Crossing Rate, Chroma Frequencies, Spectral Roll-off are taken for the project.

Here 20 features are chosen for MFCC because after 20, the calculations differ and the system becomes complex and the time complexity increases. So, it is suggested to use till 20 as it is more than enough to extract features.

```
header = 'filename chroma_stft rmse spectral_centroid spectral_bandwidth rolloff zero_crossing_rate'
for i in range(1, 21):
    header += f' mfcc{i}'
header += ' label'
header = header.split()

file = open('F:/data1.csv', 'w', newline='')
with file:
    writer = csv.writer(file)
    writer.writerow(header)
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'.split()
for g in genres:
    for filename in os.listdir(f'F:/genres/{g}'):
        songname = f'F:/genres/{g}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=30)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        rmse = librosa.feature.rmse(y=y)
        to_append = f'{filename} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(spec_cent)} {np.mean(spec_bw)} {np.mean(rolloff)} {np.mean(zcr)}'
        for e in mfcc:
            to_append += f' {np.mean(e)}'
        to_append += f' {g}'
        file = open('F:/data1.csv', 'a', newline='')
        with file:
            writer = csv.writer(file)
            writer.writerow(to_append.split())
```

The features are stored in a data1.csv file. It includes 1001 rows and 28 columns.



filename	chroma_s	rmse	spectral_c	spectral_b	rolloff	zero_cros	mfcc1	mfcc2	mfcc3	mfcc4	mfcc5	mfcc6	mfcc7	mfcc8	mfcc9	mfcc10	mfcc11	mfcc12	mfcc13	mfcc1
blues.000	0.349943	0.130225	1784.42	2002.65	3806.485	0.083066	-113.597	121.5573	-19.1588	42.35103	-6.37646	18.61887	-13.6979	15.34463	-12.2853	10.98049	-8.32432	8.810668	-3.66737	5.75
blues.000	0.340983	0.095918	1529.835	2038.618	3548.82	0.056044	-207.557	124.0067	8.930562	35.87468	2.916038	21.52372	-8.5547	23.35867	-10.1036	11.90374	-5.56039	5.376802	-2.23912	4.216
blues.000	0.363603	0.175573	1552.482	1747.166	3040.515	0.076301	-90.7544	140.4599	-29.11	31.68901	-13.987	25.75476	-13.6496	11.62927	-11.7806	9.706442	-13.1231	5.789265	-8.90522	-1.08
blues.000	0.404779	0.141191	1070.12	1596.334	2185.028	0.033309	-199.431	150.0992	5.647594	26.87193	1.754463	14.23834	-4.83088	9.297965	-0.75774	8.149012	-3.19631	6.087676	-2.47642	-1.07
blues.000	0.30859	0.091563	1835.495	1748.362	3580.945	0.1015	-160.266	126.1988	-35.6054	22.1533	-32.4893	10.86451	-23.3579	0.503117	-11.8058	1.206804	-13.0838	-2.80638	-6.93412	-7.55
blues.000	0.302346	0.103468	1831.942	1729.483	3480.937	0.09404	-177.869	118.1969	-17.5507	30.75863	-21.7427	11.9038	-20.7342	3.180597	-8.58348	-0.93649	-11.7763	-2.42061	-9.33936	-9.93
blues.000	0.291308	0.141796	1459.078	1388.913	2795.616	0.073028	-190.149	130.297	-36.3441	33.01305	11.10694	-0.61516	-20.862	0.270091	-6.48976	-5.51709	-7.84033	-3.12568	-6.59312	-9.94
blues.000	0.307921	0.131785	1451.754	1577.37	2955.349	0.061435	-179.395	136.4592	-26.6564	39.98803	5.289679	10.92443	-20.5619	8.513764	-11.3569	-3.46908	-8.41455	-6.95483	-3.54454	-8.05
blues.000	0.409037	0.142438	1719.213	2031.644	3781.319	0.064028	-121.361	122.5131	-14.7421	46.14344	-8.16533	20.17653	-19.1725	23.05562	-11.8305	21.17701	-6.72119	7.010945	-12.7418	5.066
blues.000	0.274009	0.081352	1817.516	1973.739	3944.451	0.079215	-213.181	115.1528	-11.7163	39.02947	-20.3528	13.0824	-9.17348	9.016148	-14.0979	4.278177	-6.03959	3.78477	0.225669	-5.11
blues.000	0.303954	0.101071	1410.506	1512.629	2767.342	0.062978	-207.759	137.1444	-23.7582	26.98038	-16.663	4.191219	-12.5633	5.833579	-2.88118	-1.20838	-1.96627	1.801461	-2.99002	-1.86
blues.000	0.367141	0.046462	1353.244	1756.961	2882.466	0.043946	-321.969	123.9736	5.461874	33.67592	1.089541	14.70305	-4.89676	8.653242	0.783909	6.571473	-4.8792	-3.17976	-5.55657	-6.75
blues.000	0.26932	0.084192	1361.143	1567.68	2739.767	0.069112	-241.259	132.7868	-15.4299	60.97845	0.728929	12.4317	1.18521	-1.53811	-17.8918	8.361003	-2.45382	-0.61161	0.382677	2.607
blues.000	0.264616	0.080054	1324.222	1827.268	2710.075	0.051439	-243.841	124.4474	10.31634	47.01363	6.509913	15.40275	-1.99252	6.717751	-21.5535	10.72058	-5.5252	-2.2057	-3.03759	8.644
blues.000	0.329158	0.04736	1171.922	1705.186	2344.694	0.045028	-339.752	113.0449	12.12779	45.02441	17.76732	14.57985	4.166034	0.226675	-7.02373	13.64609	8.932184	1.300253	5.953399	8.542
blues.000	0.270312	0.057028	1420.333	1730.568	2930.382	0.06373	-272.574	120.079	-4.22561	42.30319	5.921581	13.91518	5.36253	1.901447	-16.9321	12.34423	3.947155	-1.38499	0.022016	3.586
blues.000	0.30432	0.057599	1454.682	1825.101	3009.382	0.061343	-267.82	120.6685	-3.1685	39.53719	3.858338	18.83506	-3.68921	4.430278	-14.9763	17.56556	1.841702	2.53592	-1.23953	8.438
blues.000	0.302095	0.065956	1088.753	1410.803	2134.67	0.04833	-279.809	140.4772	-2.7607	60.43252	18.93663	9.832014	3.56312	0.534541	-16.3731	6.630361	7.267846	-4.0094	1.608424	5.899
blues.000	0.269927	0.056334	1537.408	2054.068	3496.004	0.056144	-286.586	104.9186	15.35685	42.65401	6.778273	21.61434	-1.29673	5.68276	-16.6134	12.16467	-1.95299	1.960818	-0.8807	7.776
blues.000	0.257259	0.069062	1195.422	1480.949	2235.061	0.058874	-270.711	138.3907	-4.89406	47.62202	5.174618	8.88113	0.504595	-3.81803	-14.9771	10.42053	-0.35085	0.778274	1.7773	4.235
blues.000	0.30273	0.053304	1389.661	1910.707	3005.265	0.052573	-264.469	127.0897	7.152979	39.77813	1.951007	17.408	-5.62297	8.251017	-18.1036	13.83726	-3.76218	2.560428	-4.32819	6.815
blues.000	0.321084	0.071526	1046.699	1480.507	1830.731	0.04794	-298.747	138.843	11.1907	48.51783	14.91468	20.57021	3.207418	1.247326	-15.8464	15.98785	3.36633	-1.51676	1.893907	6.323

IV. *Using the dataset and formatting unnecessary attributes.*

1) Dropping the ‘filename’ column as it will be of no use for training a model.
2) Select the column that has the genre list
3) Label encode the genre list as it will be helpful while using in the model
4) Selecting all the features that will be needed for training the model as X.
5) Scaling X using standard scaler for better accuracy of the model.

```
# Dropping unneccesary columns
data = data.drop(['filename'],axis=1)

#Encoding the Labels
genre_list = data.iloc[:, -1]
#genre_list
encoder = LabelEncoder()
y = encoder.fit_transform(genre_list)

#Scaling the Feature columns
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data.iloc[:, :-1], dtype = float))

```

STEP 2. CREATING THE ANN MODEL

- Libraries used for creating this model are

```
#MADE BY WRIDDHIRUP DUTTA
#AS A ROJECT FOR NATURAL LANGUAGE PROCESSING(CSE4022)

#IMPORT LIBRARIES

import librosa
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#get_ipython().run_line_magic('matplotlib', 'inline')
import os
from PIL import Image
import pathlib
import csv
import itertools

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import confusion_matrix

import keras
from keras import models
from keras import layers
from keras.layers import Dropout

from sklearn import preprocessing
from sklearn.metrics import roc_auc_score

import warnings
warnings.filterwarnings('ignore')
```

- Here Keras Sequential Model is used. The model has input for 10, has 3 hidden dense layers with 3 dropout layers of dropout probability of 20%.
- The first layers has 256 neurons, the 2nd layer has 128 neurons, the 3rd layer has 64 neurons. The activation function used in each of the 3 layers is Rectified linear unit (ReLu). It is given by y = x line
- The output layer uses a softmax activation function as multiclass classification is used here. It will give a numpy array of 10 size.
- Optimizer used in the model is Adam Optimizer.
- The loss function used here is sparse categorical entropy, as the output will be integers (0,1,2,3,4,5,6,7,8,9).
- The metric used is Accuracy.
- The model is compiled for 20 epochs. After 20 epochs the accuracy and loss values are saturating.
- Batch size used here is 128.
- The validation dataset is Test dataset, which includes 300 samples.
- The above features are fed into the model

In [183]:
```
#Dividing data into training and Testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

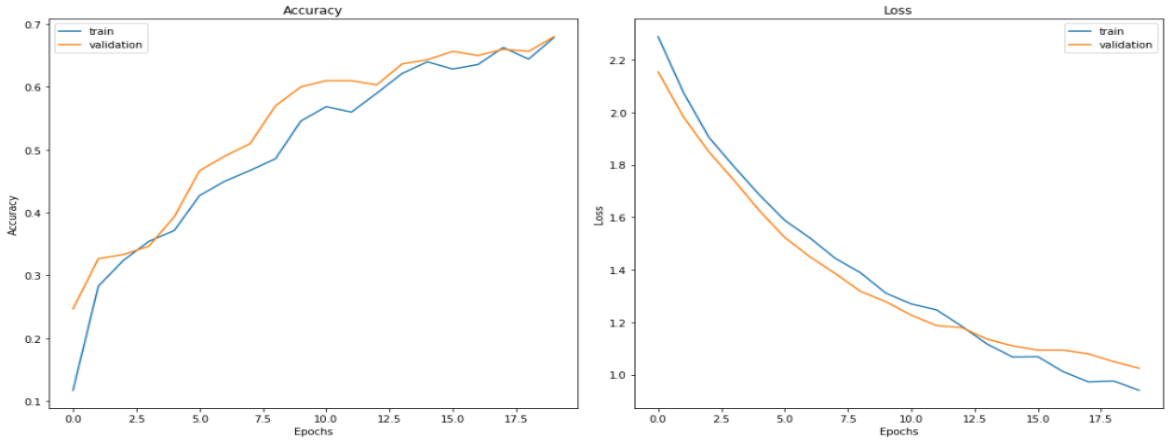
In [184]:
```
#Building our Neural network model

model = models.Sequential()
#model.add(Dropout(0.2, input_shape=(X_train.shape[1],)))
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(layers.Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
#model.add(layers.Dense(32, activation = 'relu'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [185]:
```
history = model.fit(X_train,
                    y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(X_test, y_test)
                    )
```

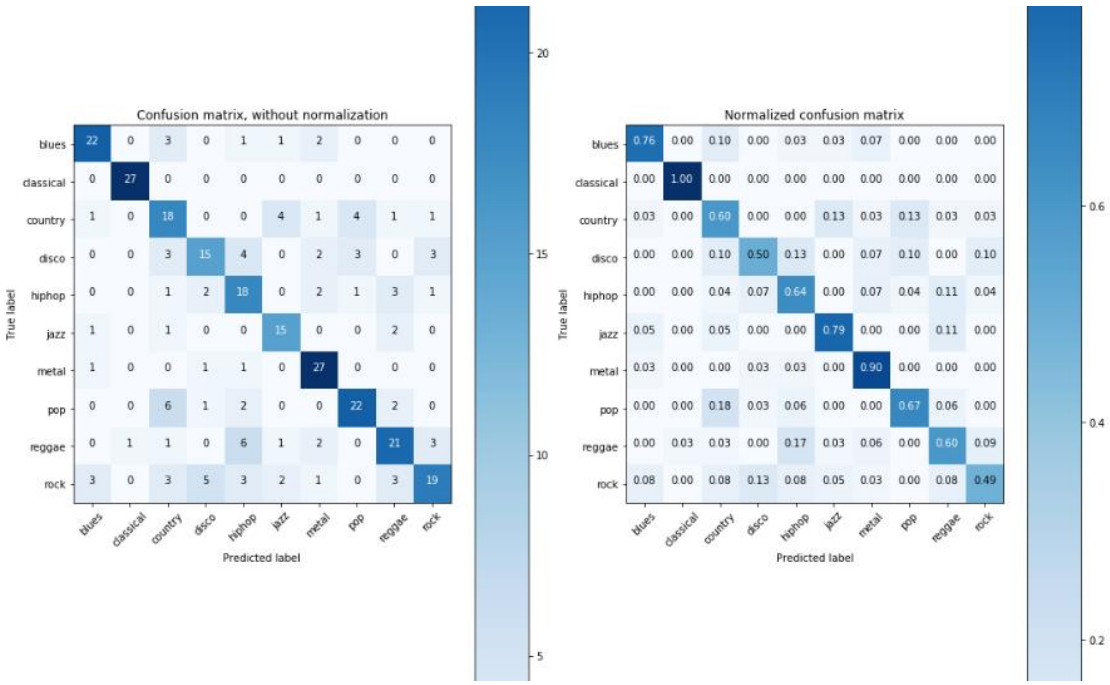
```
Train on 700 samples, validate on 300 samples
Epoch 1/20
700/700 [==============================] - 1s 2ms/step - loss: 2.2886 - acc: 0.1171 - val_loss: 2.1545 - val_acc: 0.2467
Epoch 2/20
700/700 [==============================] - 0s 77us/step - loss: 2.0758 - acc: 0.2829 - val_loss: 1.9827 - val_acc: 0.3267
Epoch 3/20
700/700 [==============================] - 0s 77us/step - loss: 1.9056 - acc: 0.3243 - val_loss: 1.8502 - val_acc: 0.3333
Epoch 4/20
700/700 [==============================] - 0s 83us/step - loss: 1.7928 - acc: 0.3543 - val_loss: 1.7406 - val_acc: 0.3467
```

STEP 3. RESULTS FOR ANN MODEL

- Without Dropout, Accuracy was 75 -80% for cross validation set, which shows overfitting of the model.
- To reduce overfitting, the dropout layers are used.
- Accuracy obtained is 68% (max. 71%)
- Validation dataset loss is 1.024 (min. is 0.985214)
- ROC AUC Score is 0.829443
- Accuracy and Loss curve for 20 epochs



- Confusion matrix without and with normalisation. The values will be under 30 for the CM without normalisation.



- Finding the *TRUE POSITIVE, FALSE POSITIVE, FALSE NEGATIVE, TRUE NEGATIVE* from the confusion matrix

```
values = np.array([TP, FP, FN, TN])
print('\tTP', '  FP', '  FN', '  TN')
values.T
```

```
        TP   FP   FN   TN

array([[ 22,   6,   7, 265],
       [ 27,   1,   0, 272],
       [ 18,  18,  12, 252],
       [ 15,   9,  15, 261],
       [ 18,  17,  10, 255],
       [ 15,   8,   4, 273],
       [ 27,  10,   3, 260],
       [ 22,   8,  11, 259],
       [ 21,  11,  14, 254],
       [ 19,   8,  20, 253]], dtype=int64)
```

- Finding the precision, recall, sensitivity, specificity from TP, FN, FP, TN

$$\text{Specificity} = \frac{\text{TN}}{\text{TN}+\text{FP}} \times 100\%$$

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \times 100\%$$

$$\text{Prevalence} = \frac{\text{TP}+\text{FN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \times 100\%$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \times 100\%$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

The **support** is the number of samples of the true response that lie in that class.

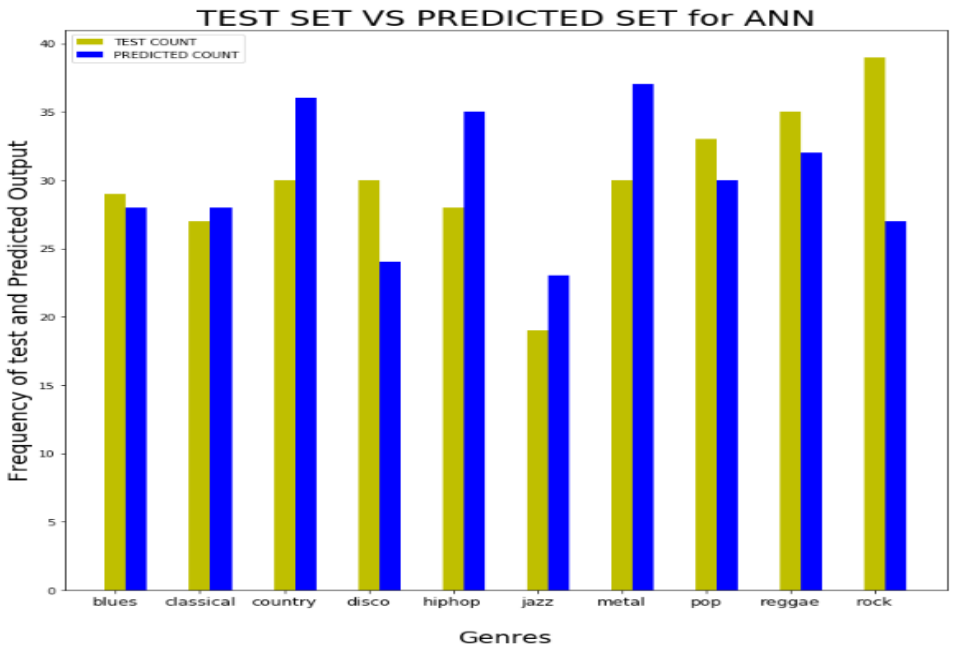
```
print('\navg / total')
print(np.mean(precision).round(2),'\t\t',np.mean(recall).round(2),'\t\t',np.mean(sensitivity)
ty).round(2),'\t\t', np.mean(fscore).round(2),'\t\t', np.mean(support).round(2))
```

precision	recall	sensitivity	specificity	f1-score	support
0.79	0.76	0.76	0.98	0.77	29
0.96	1.0	1.0	1.0	0.98	27
0.5	0.6	0.6	0.93	0.55	30
0.62	0.5	0.5	0.97	0.56	30
0.51	0.64	0.64	0.94	0.57	28
0.65	0.79	0.79	0.97	0.71	19
0.73	0.9	0.9	0.96	0.81	30
0.73	0.67	0.67	0.97	0.7	33
0.66	0.6	0.6	0.96	0.63	35
0.7	0.49	0.49	0.97	0.58	39
avg / total					
0.69	0.69	0.69	0.96	0.68	30.0

- Test set vs Predicted set Plot to show how much it is accurate

```
#DEFINE Y_TEST AND Y_PRED AS A STACKED LIST SO THAT IT CAN BE USED FOR
#DISPLAYING THE TOTAL COUNTS BEFORE AND AFTER PREDICTION
y_plot = [y_test, y_pred]
x_plot = np.arange(10)

y_test_x, y_test_y = np.unique(np.argmax(y_plot[0],axis = 1), return_counts = True)
y_pred_x, y_pred_y = np.unique(np.argmax(y_plot[1],axis = 1), return_counts = True)
```

```
plt.figure(figsize=(12,12))
plt.bar(x_plot +0.00 , y_test_y, color = 'y', width = 0.25)
plt.bar(x_plot + 0.25 , y_pred_y, color = 'b', width = 0.25)
plt.xlabel('\nGenres',fontsize=20)
plt.ylabel('Frequency of test and Predicted Output',fontsize=20)
classes = genres
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=0,fontsize=13)
#plt.yticks(tick_marks, classes)
plt.title('TEST SET VS PREDICTED SET for ANN',fontsize=25)
plt.legend(('TEST COUNT', 'PREDICTED COUNT'))
```



- Prediction result for the model

```
In [228]: #TAKE THE 50TH SONG FROM THE PREDICTED VALUE AND FIND THE GENRE OF THE SONG
          prediction = np.argmax(y_pred[50])
          genres[prediction]

Out[228]: 'reggae'
```

STEP 4. CREATING THE SVM MODEL

- Libraries used for the model are

```
#MADE BY WRIDDHIRUP DUTTA
#AS A ROJECT FOR NATURAL LANGUAGE PROCESSING(CSE4022)

#IMPORT LIBRARIES

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC
import itertools
```

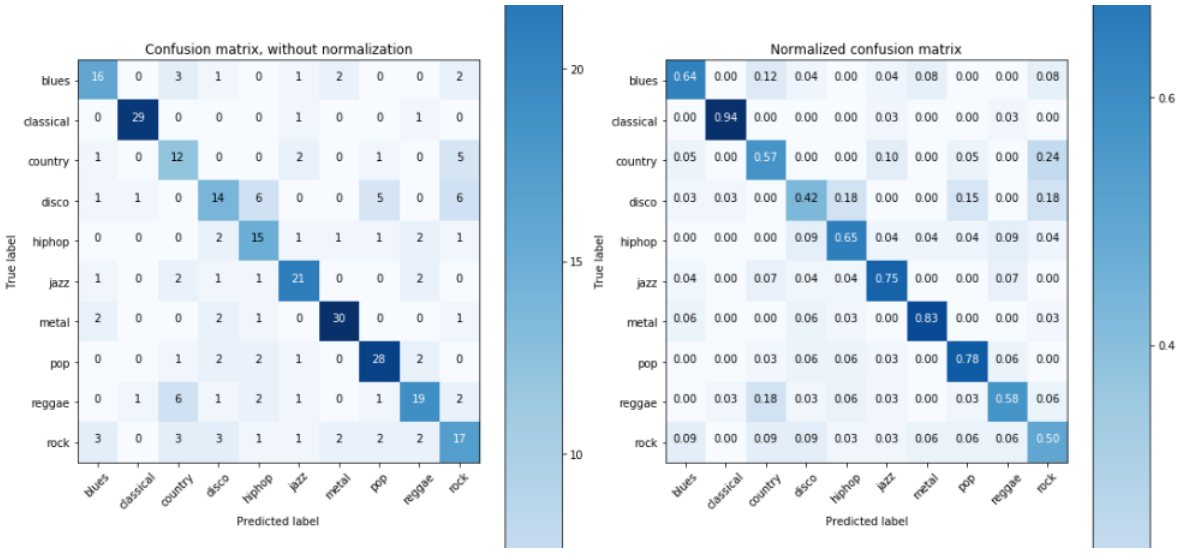
- SVM Classifier is used with RBF kernel which is widely used for multi class classification as the system is non linear. The gamma factor is set to auto, otherwise for manual values the accuracies are getting less.
- The above features are fed into the model

```
In [30]: #DEFINING THE SVM CLASSIFER WITH RBF KERNEL WITH GAMMA IN AUTO.
         svmclassifier = SVC(kernel = 'rbf', random_state = 42, gamma = 'auto')
         svmclassifier.fit(X_train, y_train)

Out[30]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=42, shrinking=True,
           tol=0.001, verbose=False)
```

STEP 5. RESULTS FOR SVM MODEL

- Accuracy obtained is 67% (max. 70.8%)
- ROC AUC Score is 0. 81470926
- Confusion matrix without and with normalisation. The values will be under 30 for the CM without normalisation.

array([[**16**, 0, 3, 1, 0, 1, 2, 0, 0, 2],
[0, **29**, 0, 0, 0, 1, 0, 0, 1, 0],
[1, 0, **12**, 0, 0, 2, 0, 1, 0, 5],
[1, 1, 0, **14**, 6, 0, 0, 5, 0, 6],
[0, 0, 0, 2, **15**, 1, 1, 1, 2, 1],
[1, 0, 2, 1, 1, **21**, 0, 0, 2, 0],
[2, 0, 0, 2, 1, 0, **30**, 0, 0, 1],
[0, 0, 1, 2, 2, 1, 0, **28**, 2, 0],
[0, 1, 6, 1, 2, 1, 0, 1, **19**, 2],
[3, 0, 3, 3, 1, 1, 2, 2, 2, **17**]], dtype=int64)



- Finding the *TRUE POSITIVE, FALSE POSITIVE, FALSE NEGATIVE, TRUE NEGATIVE* from the confusion matrix

```
                TP    FP    FN    TN

array([[ 16,    8,    9, 267],
       [ 29,    2,    2, 267],
       [ 12,   15,    9, 264],
       [ 14,   12,   19, 255],
       [ 15,   13,    8, 264],
       [ 21,    8,    7, 264],
       [ 30,    5,    6, 259],
       [ 28,   10,    8, 254],
       [ 19,    9,   14, 258],
       [ 17,   17,   17, 249]], dtype=int64)
```

- Finding the precision, recall, sensitivity, specificity from TP, FN, FP, TN

```
print(np.mean(precision).round(2),'\t\t',np.mean(recall).round(2),'\t\t',np.mean(sensitivity)
ty).round(2),'\t\t', np.mean(fscore).round(2),'\t\t', np.mean(support).round(2))
```

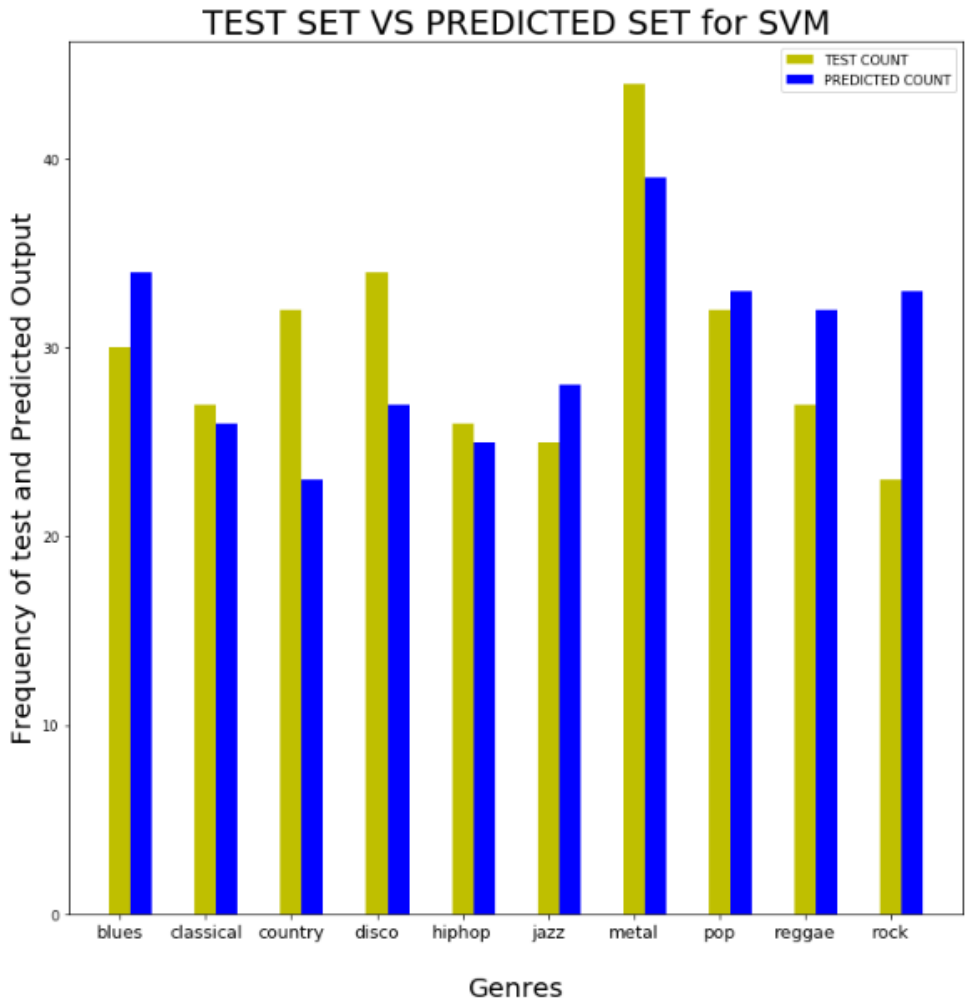
```
precision       recall     sensitivity    specificity         f1-score       support
0.67             0.64            0.64            0.97             0.65            25
0.94             0.94            0.94            0.99             0.94            31
0.44             0.57            0.57            0.95             0.5             21
0.54             0.42            0.42            0.96             0.47            33
0.54             0.65            0.65            0.95             0.59            23
0.72             0.75            0.75            0.97             0.74            28
0.86             0.83            0.83            0.98             0.85            36
0.74             0.78            0.78            0.96             0.76            36
0.68             0.58            0.58            0.97             0.62            33
0.5              0.5             0.5             0.94             0.5             34

avg / total
0.66             0.67            0.67            0.96             0.66            30.0
```

- From the cross validation score used for further analysis of a machine learning model, the accuracy **mean** and **standard deviation** for **10 iterations** obtained are **0.647205, 0.063989** respectively.
- Test set vs Predicted set Plot to show how much it is accurate

```
#DEFINE Y_TEST AND Y_PRED AS A STACKED LIST SO THAT IT CAN BE USED FOR
#DISPLAYING THE TOTAL COUNTS BEFORE AND AFTER PREDICTION
y_plot = [y_test, y_pred]
x_plot = np.arange(10)
# for i, j in zip(np.argmax(y_plot[0],axis = 1),np.argmax(y_plot[1],axis = 1)):
#     print(i,j)
```

```
y_test_x, y_test_y = np.unique(np.argmax(y_plot[0],axis = 1), return_counts = True)
y_pred_x, y_pred_y = np.unique(np.argmax(y_plot[1],axis = 1), return_counts = True)
```

```
plt.figure(figsize=(12,12))
plt.bar(x_plot +0.00 , y_test_y, color = 'y', width = 0.25)
plt.bar(x_plot + 0.25 , y_pred_y, color = 'b', width = 0.25)
plt.xlabel('\nGenres',fontsize=20)
plt.ylabel('Frequency of test and Predicted Output',fontsize=20)
classes = genres
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=0,fontsize=13)
#plt.yticks(tick_marks, classes)
plt.title('TEST SET VS PREDICTED SET for SVM',fontsize=25)
plt.legend(('TEST COUNT', 'PREDICTED COUNT'))
```



- Prediction result:

```
In [49]: #TAKE THE 50TH SONG FROM THE PREDICTED VALUE AND FIND THE GENRE OF THE SONG
         prediction = np.argmax(y_pred[50])
         genres[prediction]

Out[49]: 'classical'
```

RESULT:

1. The accuracy is more in case of ANN Classifier than SVM classifier (68 % vs 67 %).
2. The ROC AUC Score is more for ANN model than SVM Model.
3. Overall the classifiers work same and the accuracies are almost same
4. The ANN Classifier takes more time to train the model, but it produces better output.
5. The loss decreases more some cases as the data is shuffled.
6. The accuracy and efficiency of both the models can be increased with more dataset
7. The loss in case of both the cases can be increased by using regularization parameters separately.

FUTURE WORK:

1. The validation set accuracy can be increased with more amount of dataset
2. CNN model can be used in this case, but the accuracy will be low as all the images have same kind of features, hence the model wont be able to differentiate much.
3. The dataset can also be increased by 50% overlap for every 5s of the song, hence the data will increase by a factor of 1000.
4. Recommendation system can be made by content based or collaborative filtering model in such a way that the system will listen to a song and predict the genre and recommend top 5 songs like the given song.

CONCLUSION:

The music genre classification and prediction system has been successfully completed and the results are compared for better analysis. The prediction is almost accurate for ANN model whereas the SVM model needs more training set.

REFERENCES:

1. https://medium.com/@matanlachmish/music-genre-classification-470aaac9833d
2. https://github.com/mlachmish/MusicGenreClassification
3. Omar Diab, Anthony Manero, and Reid Watson. Musical Genre Tag Classification With Curated and Crowdsourced Datasets. Stanford University, Computer Science,1 edition, 2012.
4. https://thesai.org/Downloads/Volume8No8/Paper_44-Automatic_Music_Genres_Classification.pdf
5. https://ieeexplore.ieee.org/document/7806258/
6. Music Genre Classification by Archit Rathore, Margaux Dorido
7. stackoverflow.com
8. towardsdatascience.com
9. https://librosa.github.io/librosa/
10. https://librosa.github.io/librosa/feature.html
11. https://www.youtube.com/watch?v=KzevshgDv8g
12. https://www.youtube.com/watch?v=dxSfkRLQbUs