Music Genre Recognition Using Deep Learning

A project report submitted in partial fulfillment of the requirements
for the award of the degree of

B.Tech in

Electrical Engineering

By

Tushar Gupta(18085073)

Tushar Singh Diwakar (18085074)

Vipin Sharma (18085080)

Under the supervision of

Ms. Sobhita Mehar

Assistant Professor

Department of Electrical Engineering Indian Institute of Technology,

BHU

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, BHU
VARANASI 221005, INDIA

CERTIFICATE

This is to certify that the project titled **Music Genre Recognition Using Deep Learning** is a bonafide record of the work done by

Tushar Gupta (18085073)

Tushar Singh Diwakar (18085074)

Vipin Sharma (18085080)

under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electrical Engineering** of the **Department of electrical engineering Indian Institute of Technology, BHU Varanasi 221005, India**, during the year 2020-2021.

Their work is genuine and has not been submitted for the award of any other degree to the best of my knowledge.

DATE: 24.04.2021

Ms. Sobhita Mehar

Assistant Professor

Electrical Engineering

Indian Institute of Technology,

BHU Varanasi

DECLARATION

This is to certify that the work which is being hereby presented by us in this project titled "**Music Genre Recognition Using Deep Learning**" in partial fulfilment of the award of the Bachelor of Technology submitted at the Department of Electrical Engineering at Indian Institute of Technology , BHU ,Varanasi, is a genuine account of our work carried out during the period from January to April 2021 under the guidance of Ms. Shobhita Mehar, Assistant Professor, the Department of Electrical Engineering at Indian Institute of Technology , BHU ,Varanasi.

The matter embodied in the project report to the best of our knowledge has not been submitted for the award of any other degree elsewhere.

DATE:24/04/2021

Tushar Gupta	**Tushar Singh Diwakar**	**Vipin Sharma**
(18085073)	(18085074)	(18085080)

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

1.1 The Problem statement

Genre classification is an important task with many real world applications. As the quantity of music being released on a daily basis continues to sky-rocket, especially on internet platforms such as Soundcloud and Spotify – a 2016 number suggests that tens of thousands of songs were released every month on Spotify – the need for accurate meta-data required for database management and search/storage purposes climbs in proportion. Being able to instantly classify songs in any given playlist or library by genre is an important functionality for any music streaming/purchasing service, and the capacity for statistical analysis that correct and complete labeling of music and audio provides is essentially limitless.

Wikipedia states that "music genre is a conventional category that identifies 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 a 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, till 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 predictions.

1.2 Dataset

We will be using the famous GTZAN dataset for our case study. This dataset was used for the well-known paper in genre classification " Musical genre classification of audio signals " by G. Tzanetakis and P. Cook in IEEE Transactions on Audio and Speech Processing 2002. The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres namely, blues, classical, country, disco, hiphop, jazz, reggae, rock, metal and pop. Each genre consists of 100 sound clips.

We have used the GTZAN dataset from the MARYSAS website. Each audio clip 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 datset in 0.9 : 0.1 ratio and used 5-fold cross validation for reporting the results.

1.3 Related Work

The most influential work on genre classification using machine learning techniques was pioneered by Tzanetakis and Cook [5]. The GTZAN dataset was created by them and is to date considered as a standard for genre classification. Scaringella et al.[2] gives a comprehensive survey of both features and classification techniques used in the genre classification. Changsheng Xu et al.[7] have shown how to use support vector machines for this task. Most of the work deals with supervised learning approaches. Riedmiller et al.[6] use unsupervised learning creating a dictionary of features. [4] gives a detailed account of evaluation of previous work on genre classification.

Chapter 2

Data Analysis and Preprocessing

2.1 Preprocessing

Before training the classification model, we have to transform raw data from audio samples into more meaningful representations. The audio clips need to be converted from .au format to .wav format to make it compatible with python's wave module for reading audio files. We used the open source SoX module for the conversion.

Using Open Source SOX utility using GIT BASH CMD.

Code:

```
>>for i in *.au
>>do
>>sox "$i" "waves/$(basename -s .au "$i").wav"
>>done
```

The preprocessing part involved converting the audio from .au format to .wav format to make it compatible to python's wave module for reading audio files.

2.2 Data Analysis

2.2.1 Wave Plots

Audio time series as a numpy array with a default sampling rate(sr) was computed using the librosa library in python. (Please refer the notebook file)

Wave Plots for the files were extracted and analysed. We plotted the audio array using librosa.display.waveplot.

Figure 2.1: Blues genre Waveplot

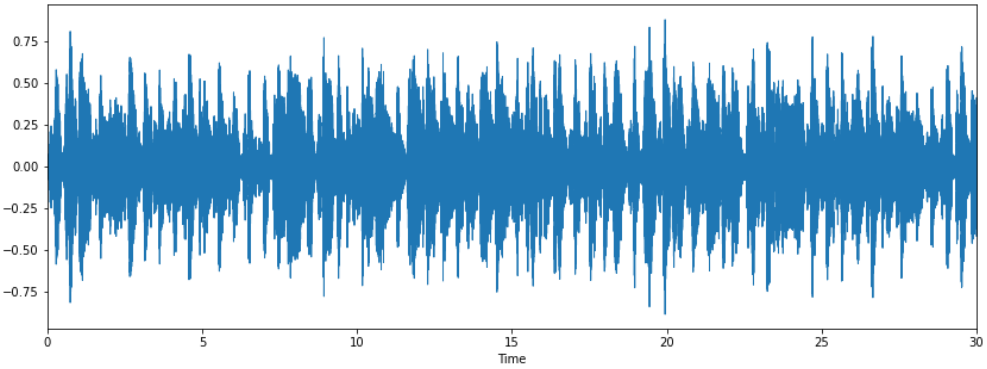
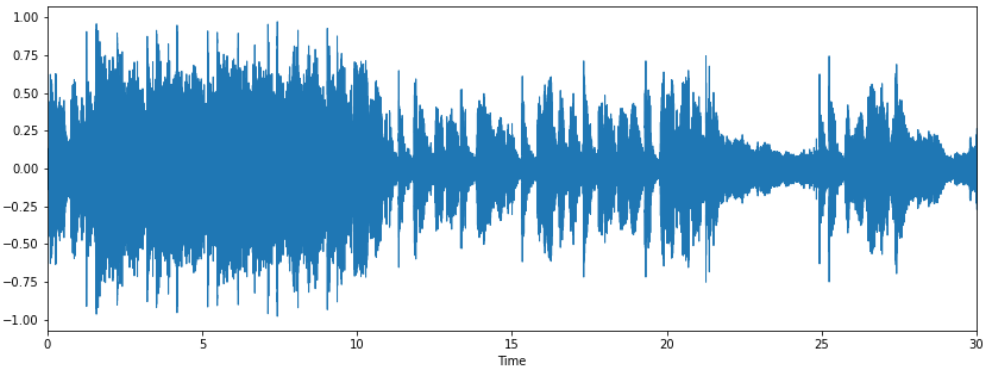


Figure 2.2: Rock genre Waveplot



2.2.2 Spectrograms

We then extracted the spectrograms for each genre. 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 voicegrams. 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.

Code:

```
X = librosa.stft(x)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
```

2.2.3 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

Figure 2.3: Blues genre Spectrogram

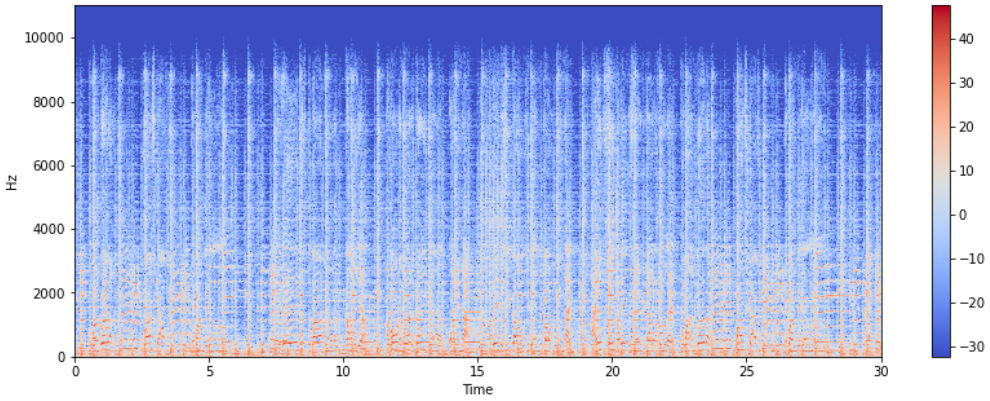
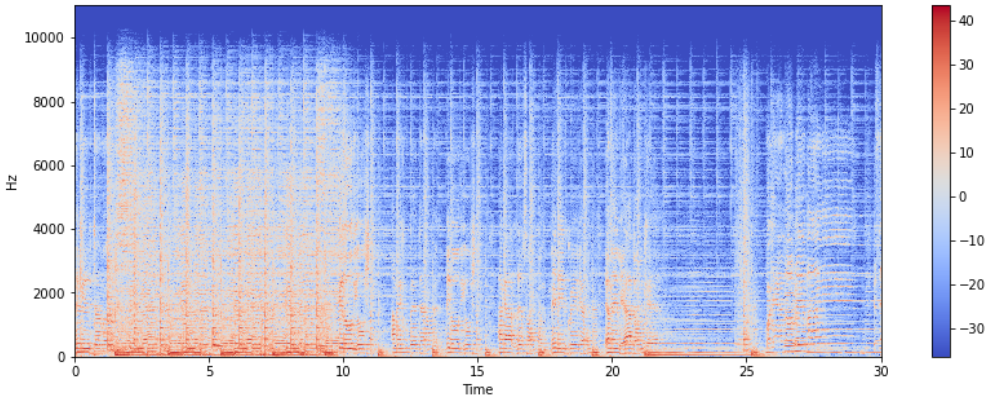


Figure 2.4: Rock genre Spectrogram



in both speech recognition and music information retrieval. It usually has higher values for highly percussive sounds like those in metal and rock.

Code:

```
x, sr = librosa.load('genres/rock/rock.00000.wav')
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
n0 = 9000
n1 = 9100
plt.figure(figsize=(14, 5))
plt.plot(x[n0:n1])
plt.grid()
```

So it can be seen zero crossing rate of blues genre here is 16 and that of rock is 8.

2.2.4 Spectral Centroids

Then we used Spectral Centroids .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

Figure 2.5: Blues genre Zero Crossing

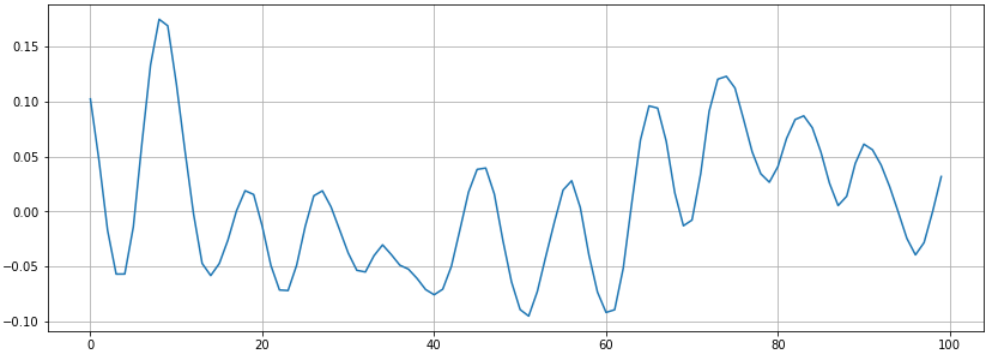
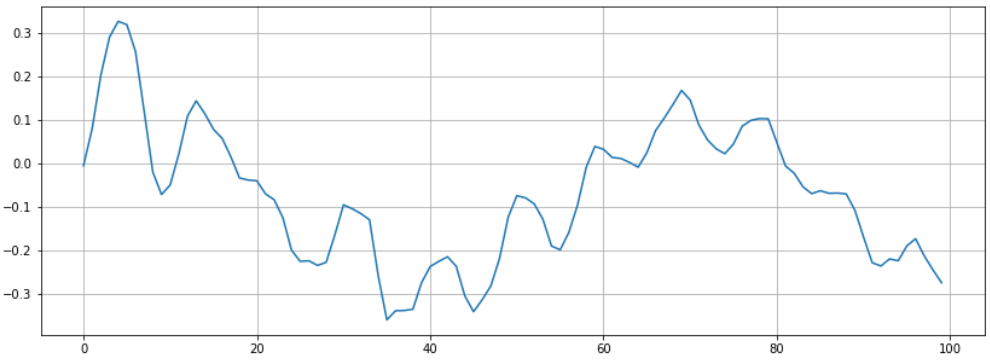


Figure 2.6: Rock genre Zero Crossing



compared to the blues genre song which is the same throughout its length, the metal song has more frequencies towards the end. So the 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.

```
librosa.feature.spectral_centroid
```

computes the spectral centroid for each frame in a signal:

Figure 2.7: Rock genre Spectral Centroid

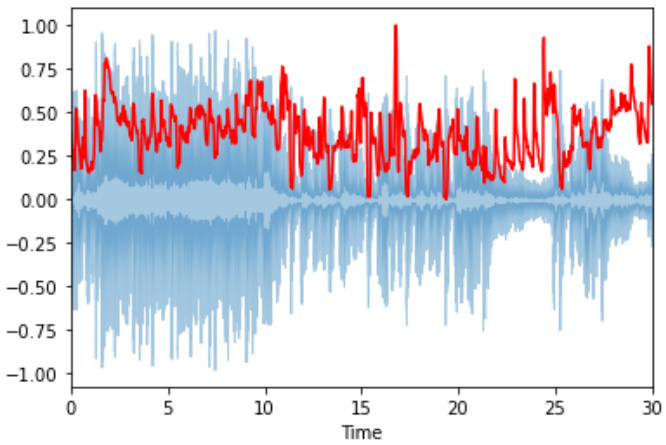
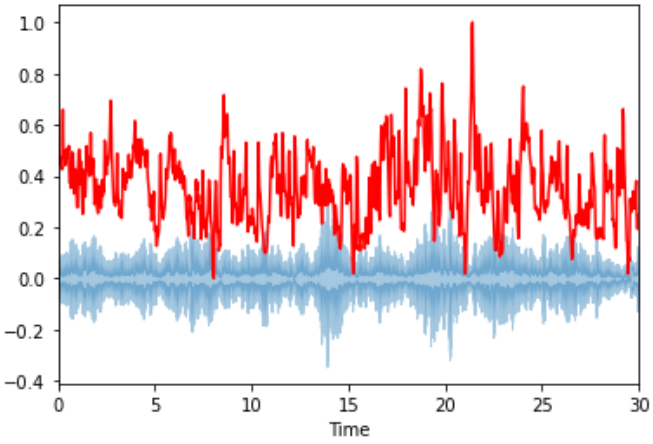


Figure 2.8: Classical genre Spectral Centroid



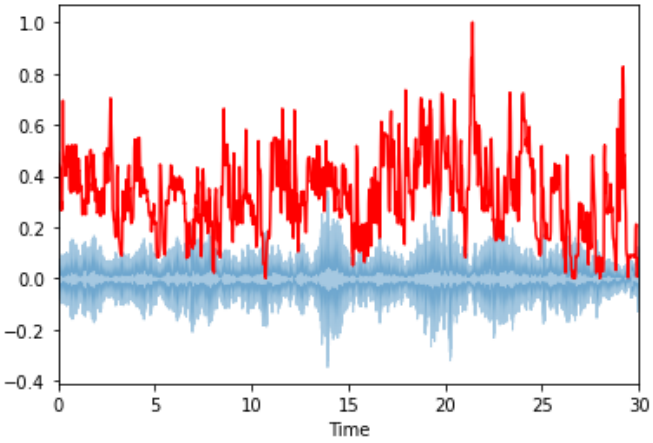
2.2.5 Spectral Rolloff

Spectral Rollof 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

```
librosa.feature.spectral_rolloff
```

computes the rolloff frequency for each frame in a signal:

Figure 2.9: Classical genre Spectral Rolloff



2.2.6 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.

Figure 2.10: Rock genre Spectral Rolloff

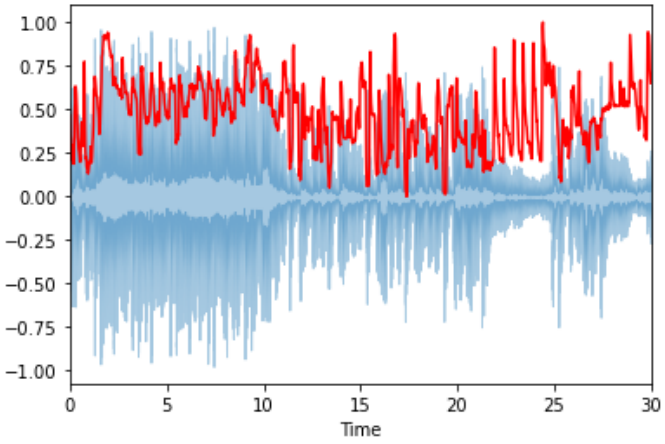
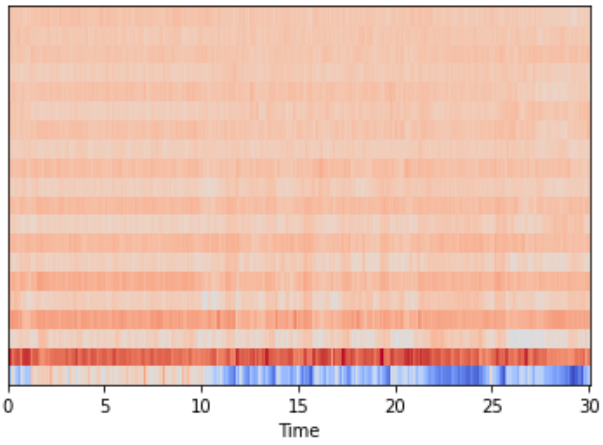


Figure 2.11: Rock genre MFCC



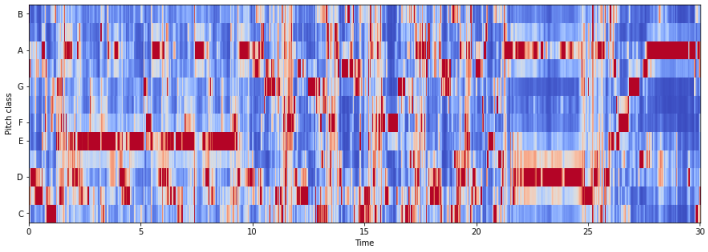
2.2.7 Chroma Frequencies

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.

```
librosa.feature.chroma_stft
```

is used for computation

Figure 2.12: Rock Genre Chroma Frequencies



Chapter 3

Feature Extraction

3.1 Spectral Centroid

Spectral Centroid It describes where the ”centre of mass” for sound is. It essentially is the weighted mean of the frequencies present in the sound. Consider two songs, one from blues and one from metal. A blues song is generally consistent throughout it length while a metal song usually has more frequencies accumulated towards the end part. So spectral centroid for blues song will lie somewhere near the middle of its spectrum while that for a metal song would usually be towards its end.

$$Centroid = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)} \tag{3.1}$$

Figure 3.1: Formula to calculate the Spectral Centroid
x(n) is the weighted frequency value of bin number n
f(n) is the center frequency of that bin

3.2 Zero Crossing Rate

Zero Crossing Rate It represents the number of times the waveform crosses 0. It usually has higher values for highly percussive sounds like those in metal and rock.

$$z_{cr} = \frac{1}{T-1} \sum_{t=1}^{T-1} \| \{s_t s_{t-1} < 0\} \tag{3.2}$$

Figure 3.2: Formula to calculate the Zero Crossing Rate
st is the signal of length t
IIX is the indicator function (=1 if X true, else =0

3.3 Mel-Frequency Cepstral Coefficients

MFCC represents a set of short term power spectrum characteristics of the sound and have been used in the state-ofthe-art recognition and sound categorisation techniques. It models the characterics of human voice. MFCC values mimic human hearing, and they are commonly used in speech recognition applications as well as music genre detection. These MFCC values will be fed directly into the neural network. This features is a large part of the final feature vector (13 coefficients). The method to implement this feature is below :

- Dividing the signal into several short frames. The aim of this step is to keep an audio signal constant.
- For each frame, we calculated the periodogram estimate of the power spectrum. This is to know frequencies present in the short frames.
- Pushing the power spectra into the mel filterbank and collecting the energy in each filter to sum it. We will then know the number of energy existing in the various frequency regions.

$$M(f) = 1125 \ln(1 + \frac{f}{700}) \tag{3.3}$$

Figure 3.3: Formula to work with Mel Scale

- Calculating the logarithm of the filterbank energies in the previous It enables humans to have our features closer to what humans can hear.
- Calculating the Discrete Cosine Transform (DCT) of the result. It decorrelates the filterbank energies with each others.
- Keep first 13 DCT coefficients. We remove the higher DCT coefficients which can introduce errors by representing changing in the filterbank energies.

3.4 Chroma Frequencies

Chroma frequency vector discretizes the spectrum into chromatic keys, and represents the presence of each key. We take the histogram of present notes on a 12-note scale as a 12 length feature vector. The chroma frequency have a music theory interpretation. The histogram over the 12-note scale actually is sufficient to describe the chord played in that window. It provides a robust way to describe a similarity measure between music pieces.

3.5 Spectral Roll-of

It is a measure of the shape of the signal. It represents the frequency at which high frequencies decline to 0. To obtain it, we have to calculate the fraction of bins in the power spectrum where 85at lower frequencies.

```
file = open('data.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'genres/{g}'):
        songname = 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)
        to_append = f'{filename} {np.mean(chroma_stft)} {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('data.csv', 'a', newline='')
        with file:
            writer = csv.writer(file)
            writer.writerow(to_append.split())
```

Chapter 4

Classification

4.1 Ensembling Approach

Ensemble methods combine several trees base algorithms to construct better predictive performance than a single tree base algorithm. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model. When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are noise, variance, and bias. Ensemble helps to reduce these factors (except noise, which is irreducible error).

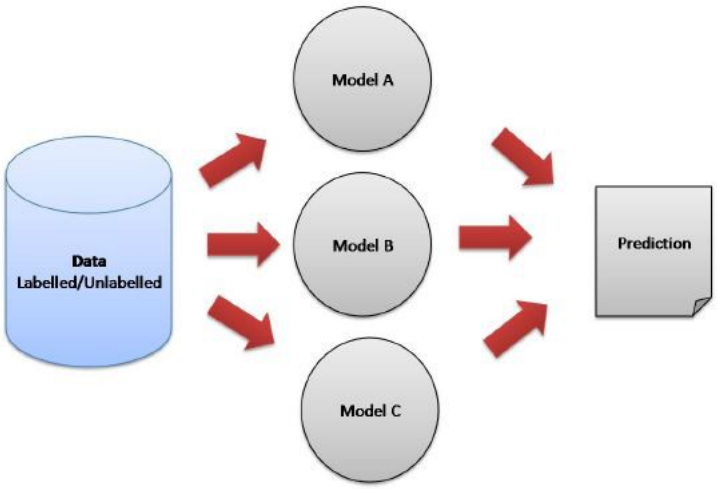


Figure 4.1: Ensemble Learning

The fundamental principle of the ensemble model is that a group of weak learners come together to form a strong learner, which increases the accuracy of the model.

When we try to predict the target variable by any machine learning technique, the main causes of the difference between the actual and predicted values are noise, variance and bias. The set reduces these factors (except noise, which is an irreducible error).

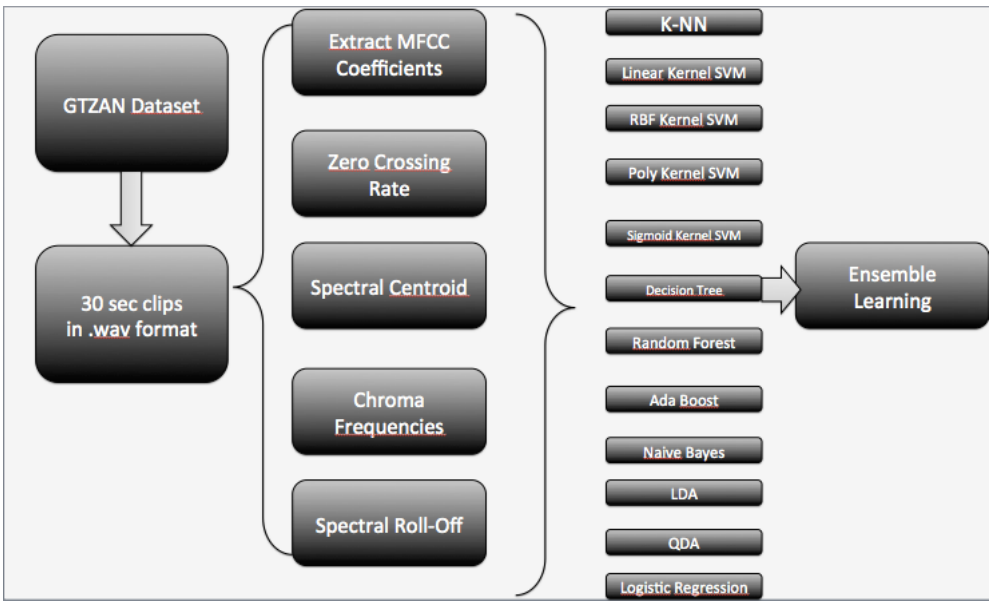


Figure 4.2: Model

Once the feature vectors are obtained, we train different classifiers on the training set of feature vectors. Following are the different classifiers that were used – K Nearest Neighbours

– Linear Kernel SVM

– Radial Basis Function (RBF) Kernel SVM

– Polynomial Kernel SVM

– Sigmoid Kernel SVM

– Decision Tree

– Random Forest

– Ada Boost

– Naives Bayes

– Logic Regression

The parameters used for various classifiers were obtained by manual tuning. It was

observed that any single classifier did not classify all the genres well. For example in the SVM with polynomial kernel worked well for most genres except blues and rock (See fig 5). This could have been due to the fact that many other genres are derived from blues.

Code:

```
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
    , BaggingClassifier, ExtraTreesClassifier
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor,
     ExtraTreesRegressor
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler


clf6 = AdaBoostClassifier(n_estimators=50,learning_rate=1)
rfClf = RandomForestClassifier(n_estimators=500, random_state=0) #
    500 trees.
svmClf = SVC(probability=True, random_state=0,kernel='rbf') # force a
     probability calculation
logClf = LogisticRegression(random_state=0)
clf7 = DecisionTreeClassifier()
clf3 = GaussianNB()
clf5 = KNeighborsClassifier(n_neighbors=3)

clf = VotingClassifier(estimators = [('rf',rfClf), ('svm',svmClf), ('
    log', logClf),('Dec',clf7),('NB',clf3),('KNN',clf5),('ada',clf6)],
     voting='hard')
```

This ensemble classifier used the prediction of each classifier and run a majority voting heuristic to obtain the optimal class label for given test input. The weights used to average the classifiers were proportional to the accuracy of the classifiers. The ensemble seemed to give a better overall precision but the accuracy dropped by a small amount.

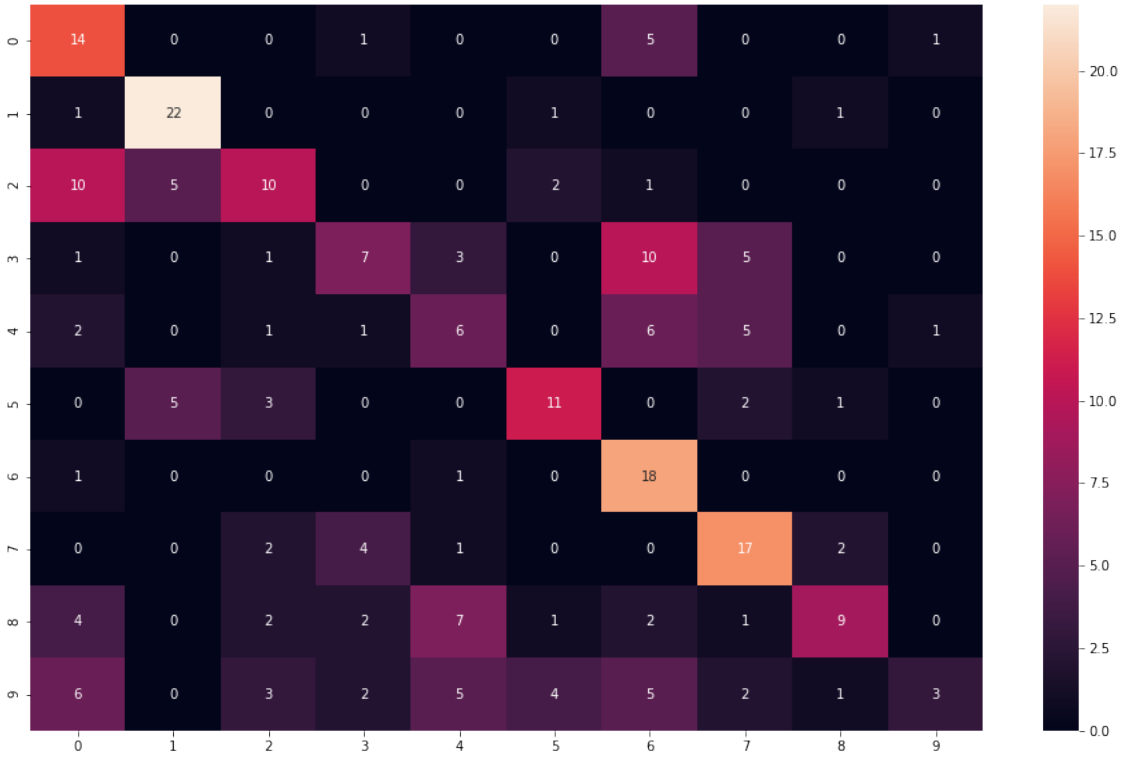


Figure 4.3: Confusion matrix For Ensemble Classification

Algorithm	Classification Accuracy
Logistic Regression	0.40
Random Forest	0.61
Naive Bayes	0.44
KNeighborsClassifier	0.33
AdaBoost Classifier	0.26
Decision Classifier	0.46
SVM RBF	0.30
SVM POLY	0.31
Ensemble	0.50

Table 4.1: Classification Accuracies

4.2 Convolutional Neural Networks based Approach

(CNN) are specially designed neural networks for processing data that has a grid-like topology Convolutional neural networks have shown to be effective in a wide range of applications, including computer vision, speech recognition, and natural language processing. Long short term memory (LSTM) networks are also often use Convolutional neural networksd to collect long-term dependencies in sequential time series data.

In this work, we apply variants of CNN models for musical genre prediction. We use 1-D convolution in our models. Here, the extracted features have dimensions of (646, k) , and our convolutional filters have dimension of (3, k). The 1D convolution operation is performed by sliding the filters over the 646 windowed time-steps. The operation is denoted as 1-D convolution because the convolutional filters and the features have same length and hence the sliding of the filters are performed only over the width (time dimension) of the features. We've made use of the above-mentioned features.

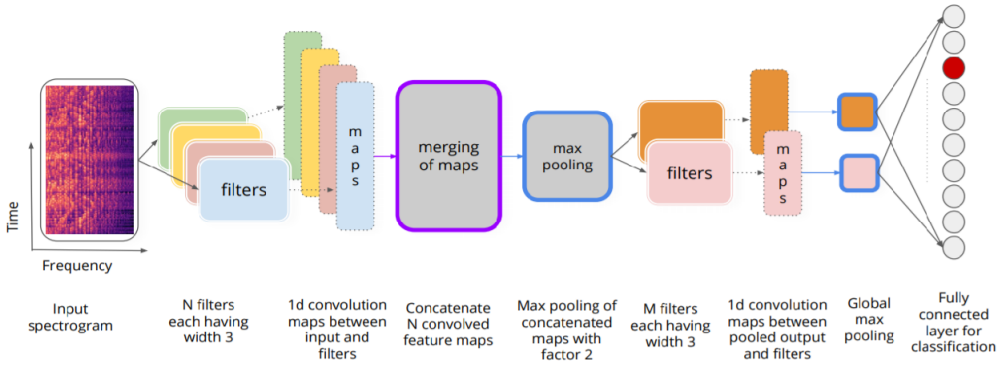


Figure 4.4: CNN Max Pooling Model

The first layer has 1024 neurons and the output layer has 10 neurons .It will be 10 because we have 10 binary numbers in that encoding. In total there are 11 layers. Total parameters are 724554.

The activation used in the code, softmax, tells you to take the output of the 10 and normalize them so that they add up to 1. That way, they end up being probabilities. Next, we compile the model, choose an optimizer such as Adam, and define the loss function.

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 1024)              26624
_________________________________________________________________
dropout_8 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_27 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_28 (Dense)             (None, 256)               131328
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_29 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_30 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_31 (Dense)             (None, 10)                650
=================================================================
Total params: 724,554
Trainable params: 724,554
Non-trainable params: 0
```

Figure 4.5: Layers of the Model

Our main aim is to not just generate a single genre per song, but generate a continuous genre prediction through the song. This stems from the nature of songs: although they are categorized to some parent genre, it contains elements of a lot of different genres. We believe a continuous prediction of genre for the song will do justice to the recognition

After each convolutional layer, the model is passed through ReLU activation and max pooling. We have used Convolution in 1-D and not 2-D as we are interested in finding out patterns with respect to time. After getting the features, a time distributed layer has been used, which gives out probabilities for the 10 genres we intend to recognize with every timestep. Since our genres are one-hot encoded, we use a categorical cross-entropy loss metric along with Adam optimizer with an initial learning rate of 0.001.

We apply CNN models on all the extracted features separately to predict the genre of the song. We have used the ReLU activation function. The first layer will do the weighted sum of its inputs, its weights, and bias term, and then run the relu activation function. relu states that anything less than 0 will turn out to be a 0, anything higher than 0 will just be the value itself.

The test loss is 4.2284 and the best test accuracy is 0.6569.

```
Epoch 486/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0099 - accuracy: 0.9987 - val_loss: 3.3265 - val_accuracy: 0.6869
Epoch 487/500
6/6 [==============================] - 0s 21ms/step - loss: 0.0342 - accuracy: 0.9939 - val_loss: 3.0429 - val_accuracy: 0.7020
Epoch 488/500
6/6 [==============================] - 0s 19ms/step - loss: 0.0202 - accuracy: 0.9942 - val_loss: 3.3219 - val_accuracy: 0.6717
Epoch 489/500
6/6 [==============================] - 0s 19ms/step - loss: 0.0369 - accuracy: 0.9916 - val_loss: 3.3407 - val_accuracy: 0.6818
Epoch 490/500
6/6 [==============================] - 0s 23ms/step - loss: 0.0504 - accuracy: 0.9925 - val_loss: 3.2777 - val_accuracy: 0.6768
Epoch 491/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0087 - accuracy: 0.9958 - val_loss: 3.1600 - val_accuracy: 0.6919
Epoch 492/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0091 - accuracy: 0.9975 - val_loss: 3.1672 - val_accuracy: 0.6919
Epoch 493/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 3.3975 - val_accuracy: 0.6818
Epoch 494/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0235 - accuracy: 0.9956 - val_loss: 3.3946 - val_accuracy: 0.6818
Epoch 495/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0157 - accuracy: 0.9919 - val_loss: 3.1676 - val_accuracy: 0.7121
Epoch 496/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0063 - accuracy: 0.9994 - val_loss: 3.2933 - val_accuracy: 0.6970
Epoch 497/500
6/6 [==============================] - 0s 20ms/step - loss: 0.0169 - accuracy: 0.9952 - val_loss: 3.4386 - val_accuracy: 0.6919
Epoch 498/500
6/6 [==============================] - 0s 18ms/step - loss: 0.0101 - accuracy: 0.9964 - val_loss: 3.3702 - val_accuracy: 0.6869
Epoch 499/500
6/6 [==============================] - 0s 18ms/step - loss: 0.0349 - accuracy: 0.9917 - val_loss: 3.3231 - val_accuracy: 0.6768
Epoch 500/500
6/6 [==============================] - 0s 18ms/step - loss: 0.0229 - accuracy: 0.9916 - val_loss: 3.8219 - val_accuracy: 0.6768
```

Figure 4.6: The value of loss and accuracy at some of the Epochs

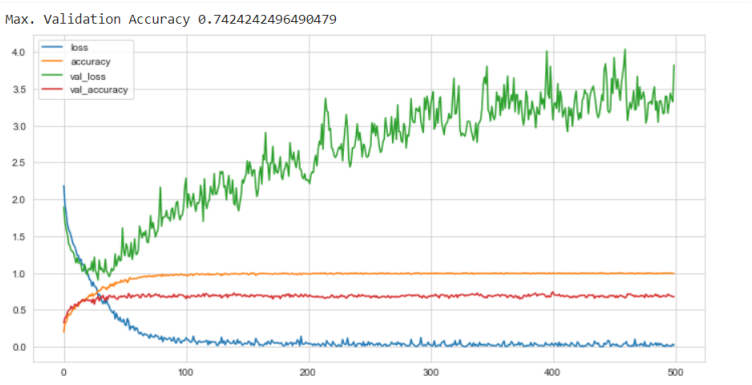


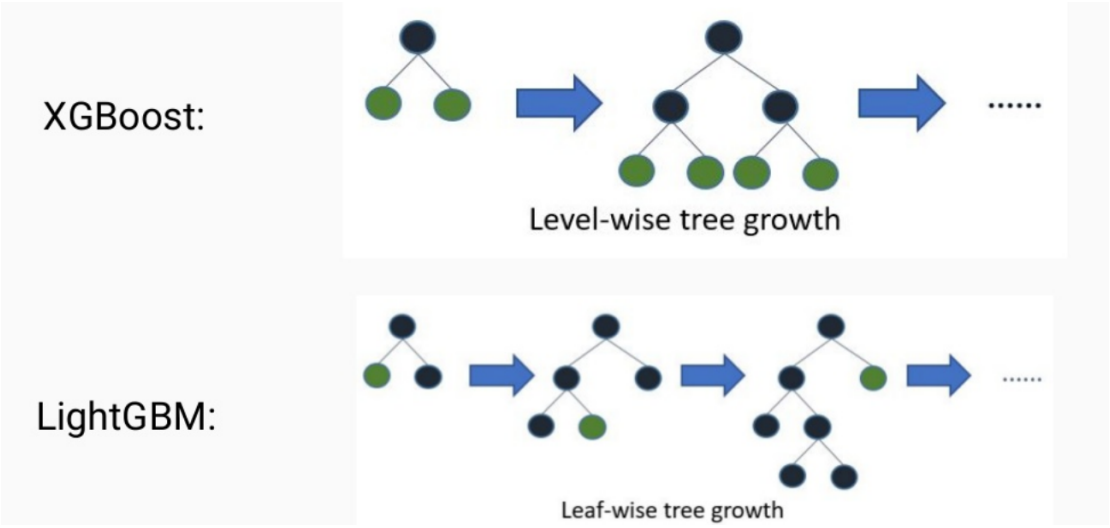
Figure 4.7: Final Graph

4.3 Light Gradient Boosting Machine based Approach

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel, distributed, and GPU learning.
- Capable of handling large-scale data.

The LightGBM framework supports different algorithms including GBT, GBDT, GBRT, GBM, MART and RF. LightGBM has many of XGBoost's advantages, including sparse optimization, parallel training, multiple loss functions, regularization, bagging, and early stopping. A major difference between the two lies in the construction of trees. LightGBM does not grow a tree level-wise — row by row — as most other implementations do. Instead it grows trees leaf-wise. It chooses the leaf it believes will yield the largest decrease in loss. Besides, LightGBM does not use the widely-used sorted-based decision tree learning algorithm, which searches the best split point on sorted feature values, as XGBoost or other implementations do. Instead, LightGBM implements a highly optimized histogram-based decision tree learning algorithm, which yields great advantages on both efficiency and memory consumption. The LightGBM algorithm utilizes two novel techniques called Gradient-Based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) which allow the algorithm to run faster while maintaining a high level of accuracy.

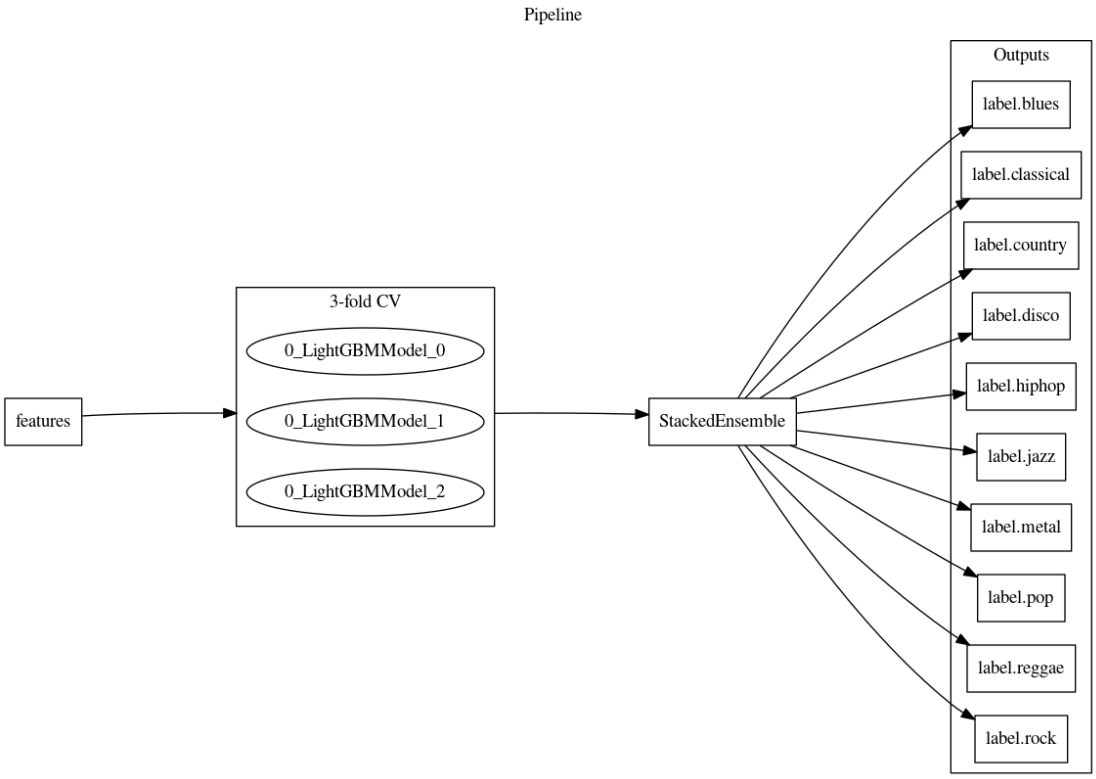


Most of the parameters used in the work are standard, however some of them (ex. Learning rate) were obtained after manual tuning.

A 3-fold CV stacked ensemble of LightGBM models is used which takes in manually selected 39 features and outputs a label vector.

The features used here were obtained by splitting the original 30 seconds clip of the

data set into 10, 3 seconds clips, and then the aforementioned procedure was applied to get the input vectors.



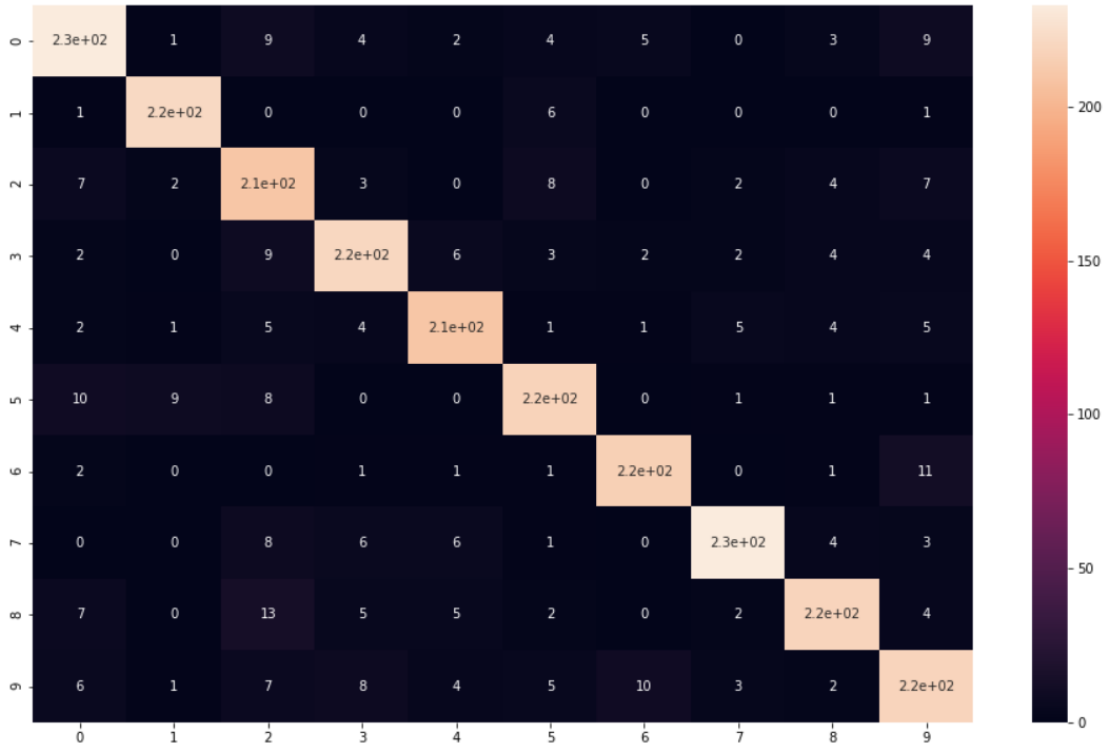
The parameters used for each of these LightGBM model were,

```
[ ]  d_train = lgb.Dataset(x_train, label = y_train)

     params = {}
     params['learning_rate'] = 0.03
     params['objective_type'] = 'gbdt'
     params['objective'] = 'multiclass'
     params['metric'] = 0.03
     params['metric'] = 'multi_logloss'
     params['sub_feature'] = 0.5
     params['num_leaves'] = 8
     params['min_data'] = 50
     params['max_depth'] = 3
     params['subsample'] = 0.9
     params['max_bin'] = 128
     params['num_iterations'] = 1200
     params['n_jobs'] = 2
     params['random_state'] = 10937361
     params['num_classes'] = 10

     clf = lgb.train(params, d_train, 1200)
```

Labels were mapped to integers for better predictions using the following scheme,

```
[ ]  data.label = data.label.map({
        'blues': 0,
        'classical': 1,
        'country': 2,
        'disco': 3,
        'hiphop': 4,
        'jazz': 5,
        'metal': 6,
        'pop': 7,
        'reggae': 8,
        'rock': 9
    })
```

The results obtained after training has been summarised in the following confusion matrix.



This gives us an overall accuracy of nearly 89% .

Chapter 5

Conclusion

Out of the tried three machine learning and deep learning approaches we got best accuracy using lgbm model (0.89).

Approach	Classification Accuracy
Ensemble	0.50
CNN	0.65
LGBM	0.89

Table 5.1: Classification Accuracies of all used algorithms

Here we proposed a novel approach for music genre recognition. Firstly we tried Ensembling approach using an ensemble of various machine learning algorithms like logistic regression , Random Forest, Decision Trees etc. This classifier used the prediction of each classifier and run a majority voting heuristic to obtain the optimal class label for given test input. The weights used to average the classifiers were proportional to the accuracy of the classifiers. This seemed to give a better overall precision but the accuracy dropped by a small amount. Secondly, a CNN network is trained on extracted features, applying the model on all the features separately to predict the genre of the song. We have used the ReLU activation function. The first layer does weighted sum of its inputs, its weights, and bias term, and then run the relu activation function. It states that anything less than 0 will turn out to be a 0, anything higher than 0 will just be the value itself. Finally, we came up with more advanced and efficient method using light gradient boosting machines which significantly improved our accuracy from 0.65 to 0.89. Our proposed model outperforms the current state- of-the-art systems and achieves a near perfect score for musical genre recognition in the GTZAN dataset.

Bibliography

[1] Hareesh Bahuleyan *Music Genre Classification using Machine Learning Techniques* University of Waterloo, ON, Canada hpallika@uwaterloo.ca

[2] Deepanway Ghosal, Maheshkumar H. Kolekar *Music Genre Recognition using Deep Neural Networks and Transfer Learning*. Indian Institute of Technology Patna, India deepanwayedu@gmail.com, mkolekar@gmail.com

[3] G. Tzanetakis and P. Cook,. *"Musical genre classification of au- dio signals," IEEE Transactions on speech and audio processing,*. vol. 10, no. 5, pp. 293–302, 2002.

[4] Omar Diab, Anthony Manero, and Reid Watson. *Musical Genre Tag Classification With Curated and Crowdsourced Datasets.*. Stanford University, Computer Science, 1 edition, 2012.

[5] Jan W ulfing and Martin Riedmiller. *Unsupervised learning of local features for music classification.*. In ISMIR, pages 139–144, 2012.