

BTP PRESENTATION

By -

Tushar Gupta (18085073)

Tushar Singh Diwakar (18085074)

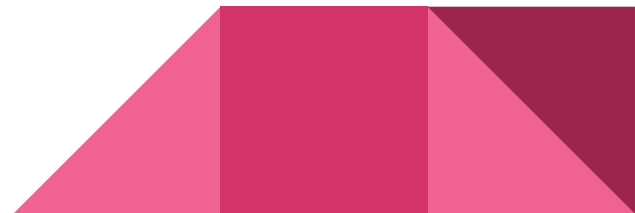
Vipin Sharma(18085080)

Music Genre Recognition using deep learning Practices

- 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,



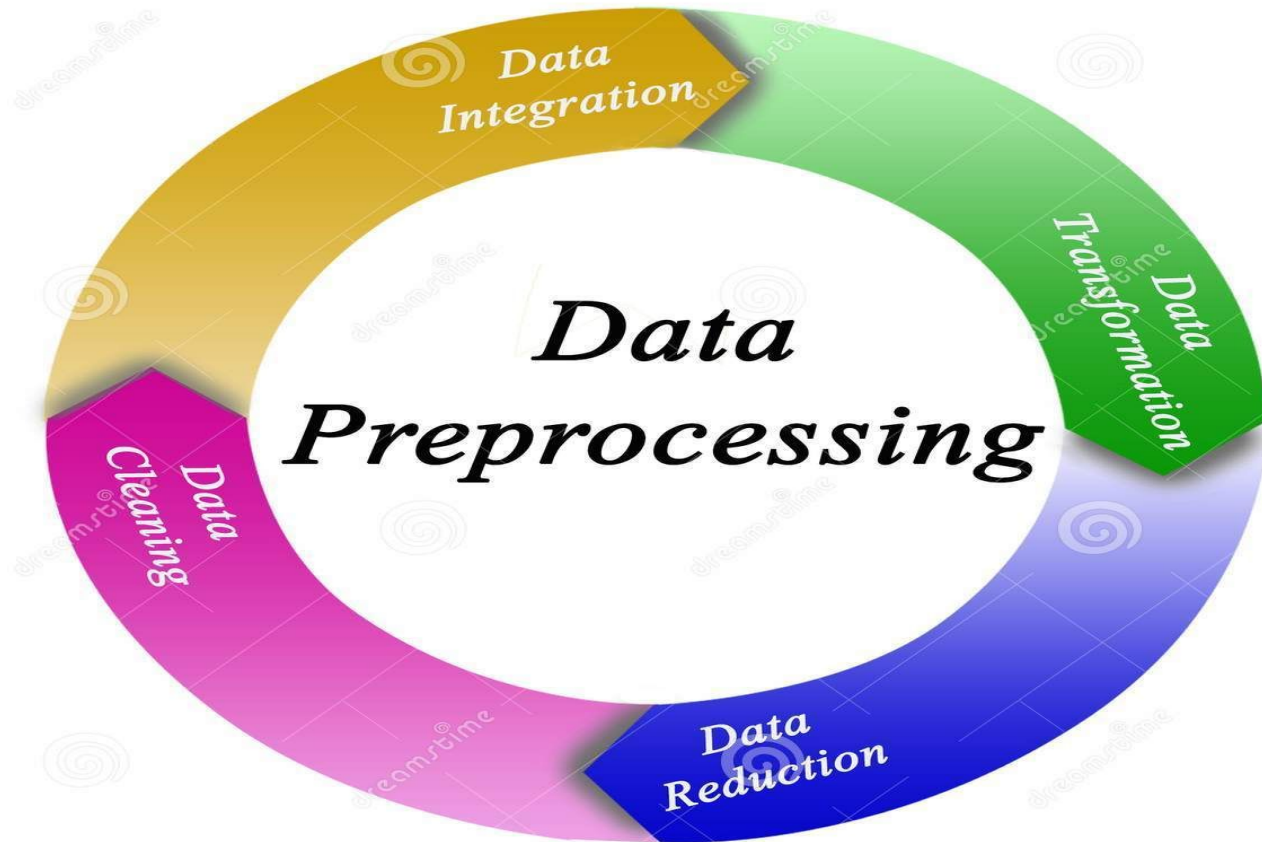
- Genre classification is an important task with many real world applications. As the quantity of music being released on a daily basis continues to sky-rocket, especially on internet platforms such as Soundcloud and Spotify.
- Being able to instantly classify songs in any given playlist or library by 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.



DATASET

We used the famous GTZAN dataset for our project.. 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.





PREPROCESSING THE GIVEN DATA

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.

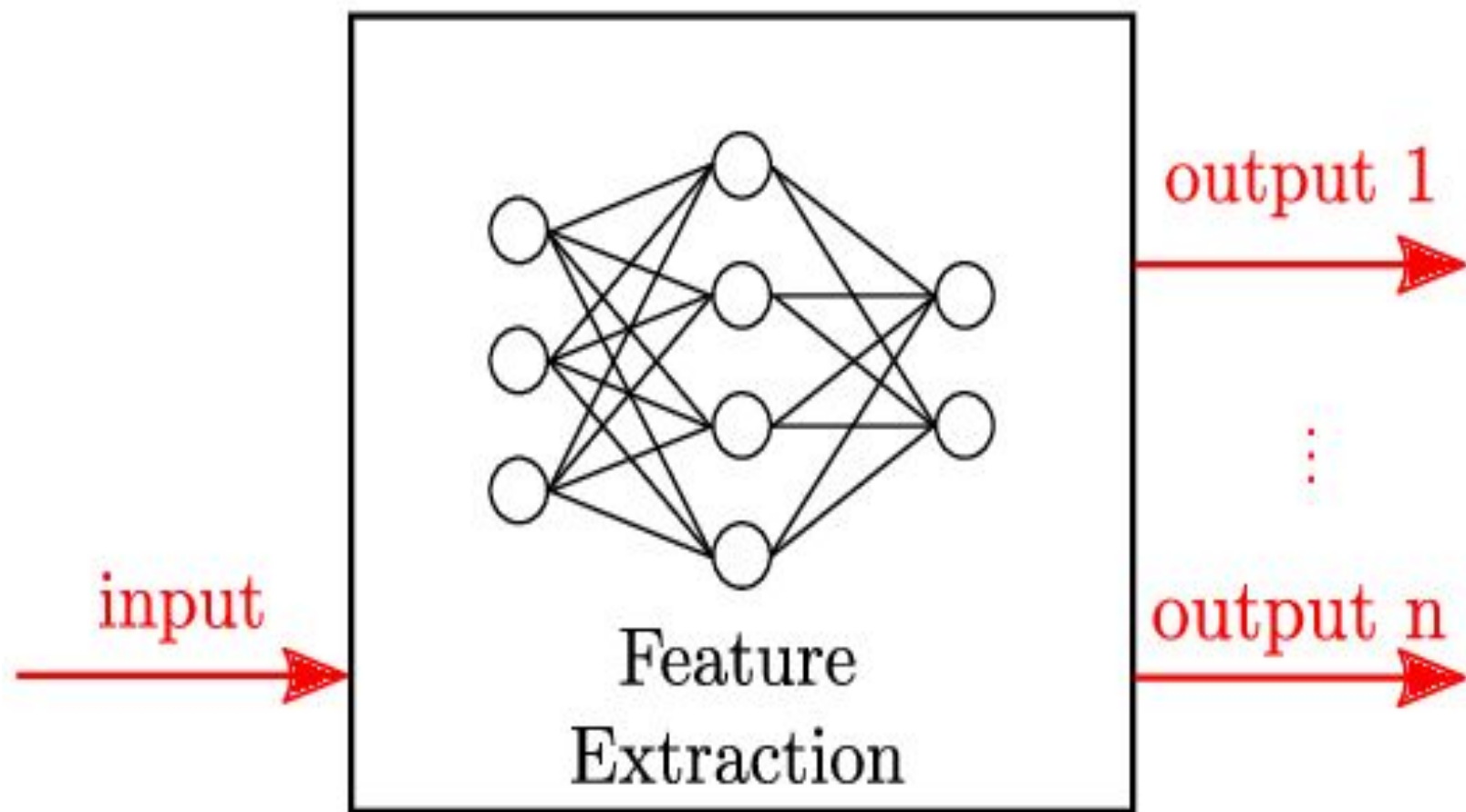


ABOUT SOX UTILITY


Sox is a cross platform command line utility that can be used to convert various formats of computer audio files to the other formats

We used it to convert music files from .au to .wav format. It converted all 1000 files in one go.





Feature extraction is a part of the dimensionality reduction process, in which, an initial set of the raw data is divided and reduced to more manageable groups. So when you want to process it will be easier. The most important characteristic of these large data sets is that they have a large number of variables. These variables require a lot of computing resources to process them. So Feature extraction helps to get the best feature from those big data sets by select and combine variables into features, thus, effectively reducing the amount of data.



EXTRACTED FEATURES

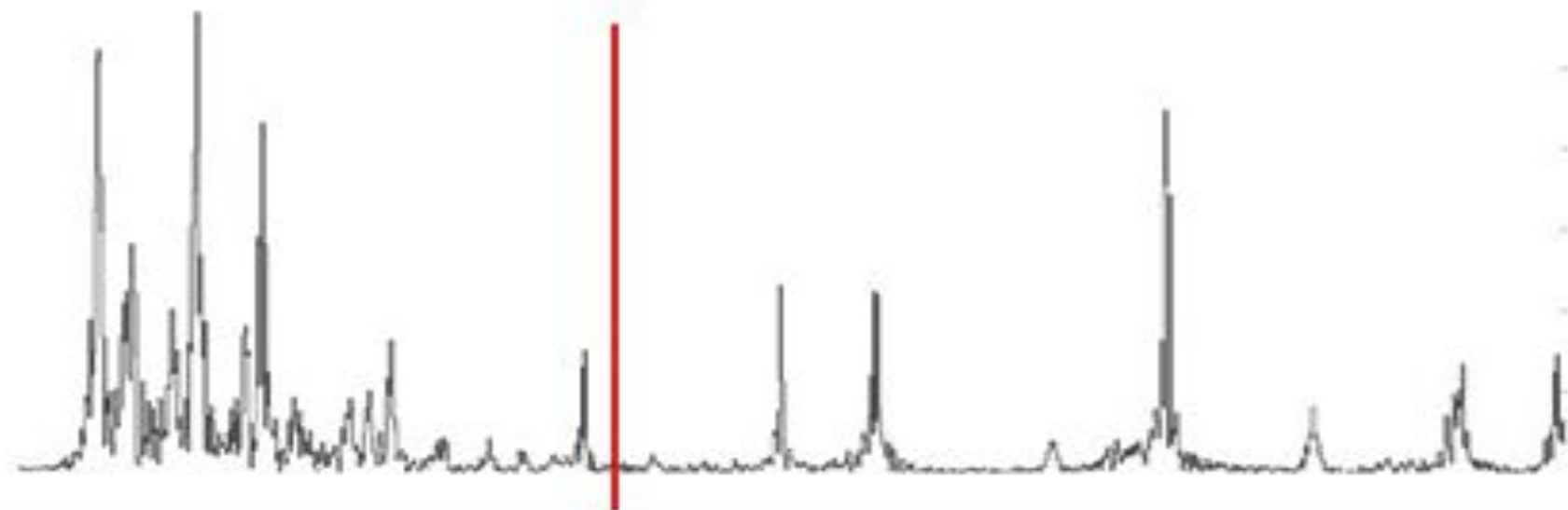
SPECTRAL CENTROID

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 its 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.

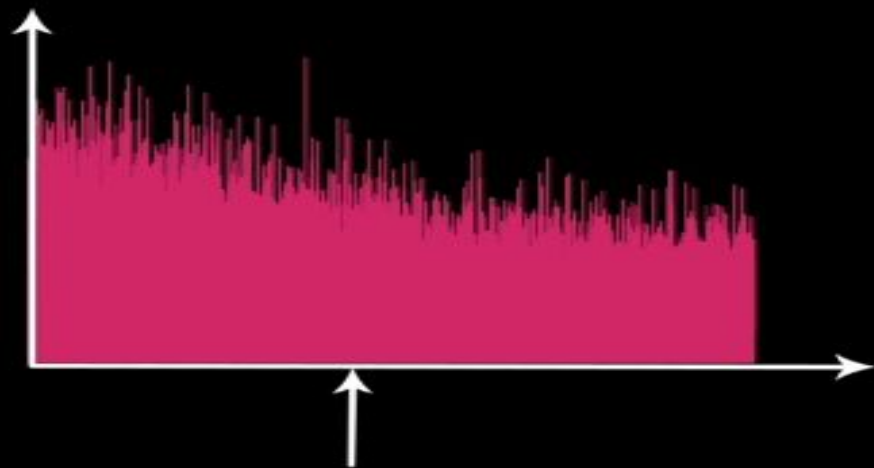


Spectral Centroid

where is the '*center of mass*' of the spectrum



Brighter sound
Higher centroid



Warmer sound
Lower centroid

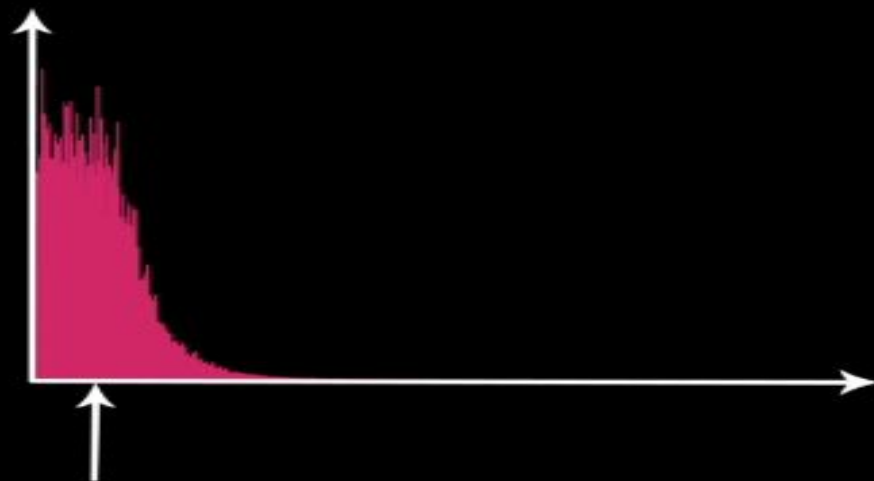


Figure 2.3: Blues genre Spectrogram

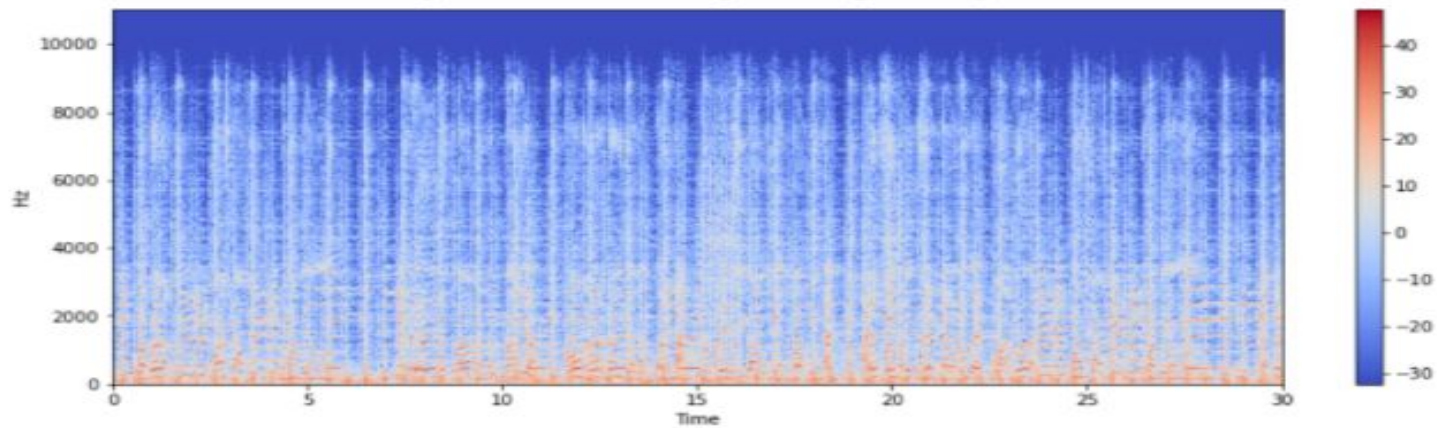
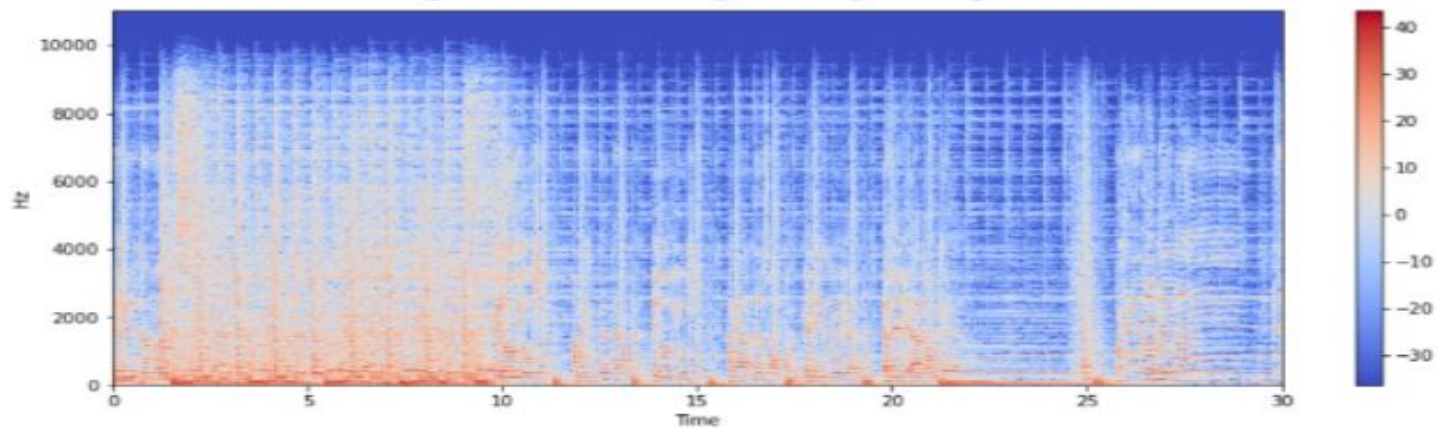


Figure 2.4: Rock genre Spectrogram



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.



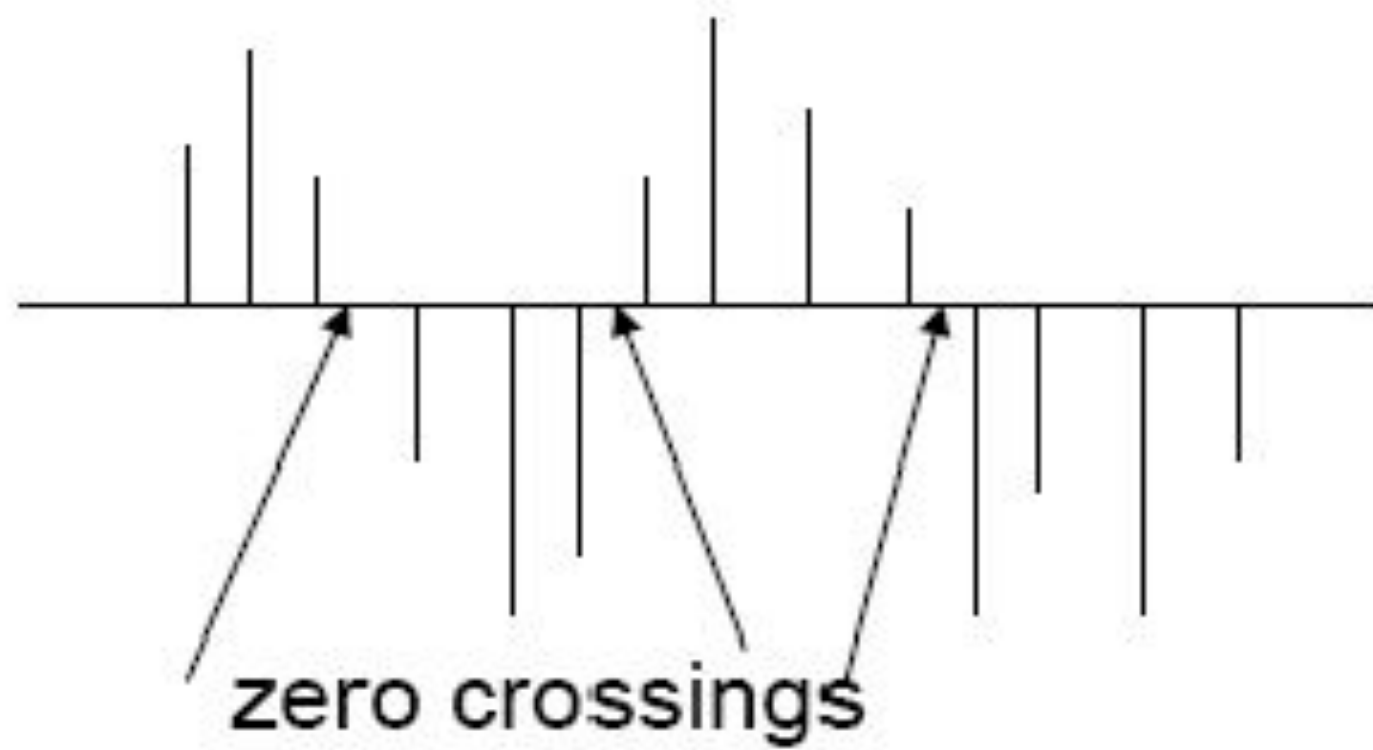


Figure 2.5: Blues genre Zero Crossing

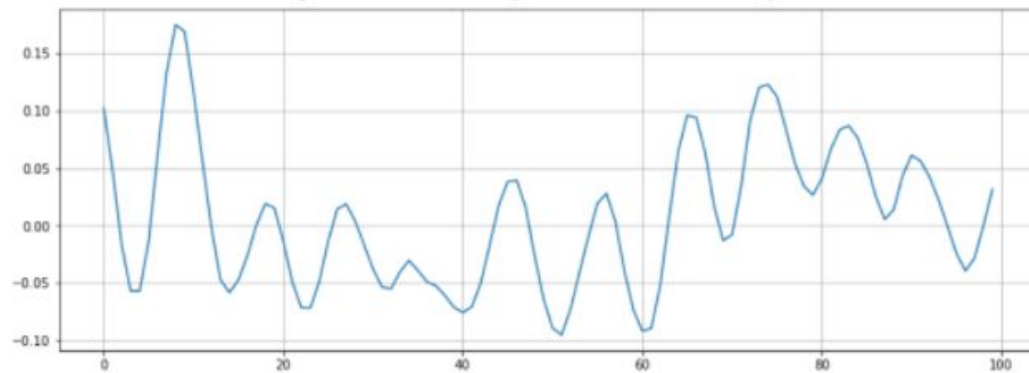
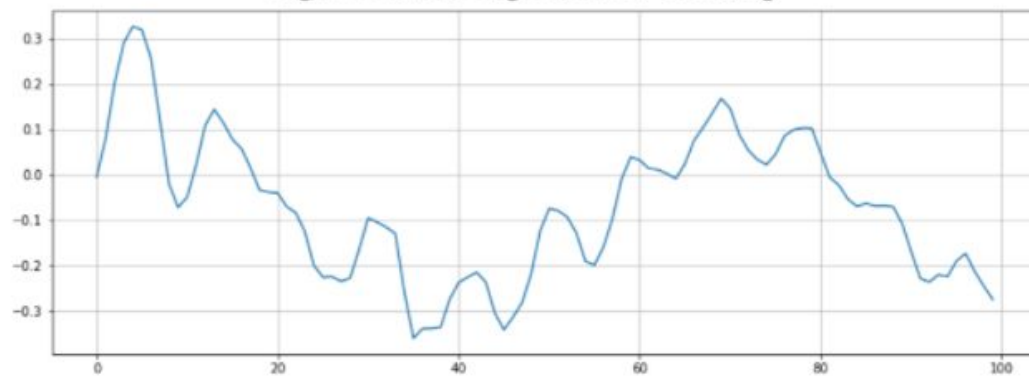


Figure 2.6: Rock genre Zero Crossing



Spectral Rolloff

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 percent. It represents the frequency at which high frequencies decline to 0.



Figure 2.9: Classical genre Spectral Rolloff

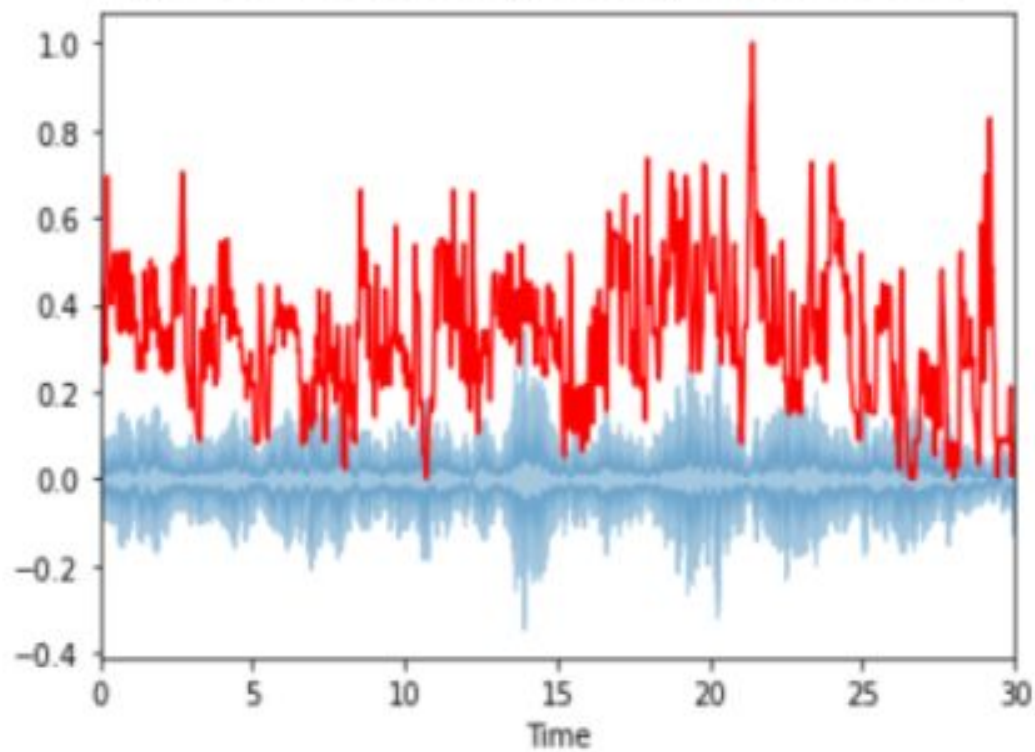
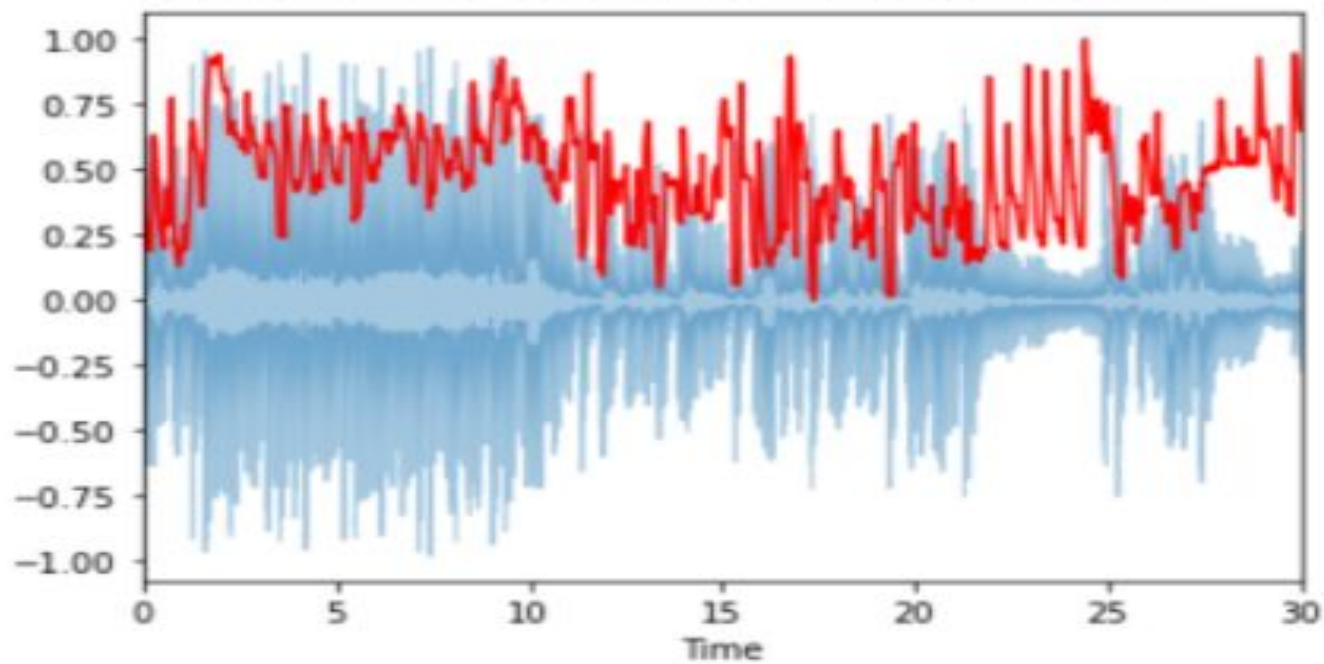


Figure 2.10: Rock genre Spectral Rolloff



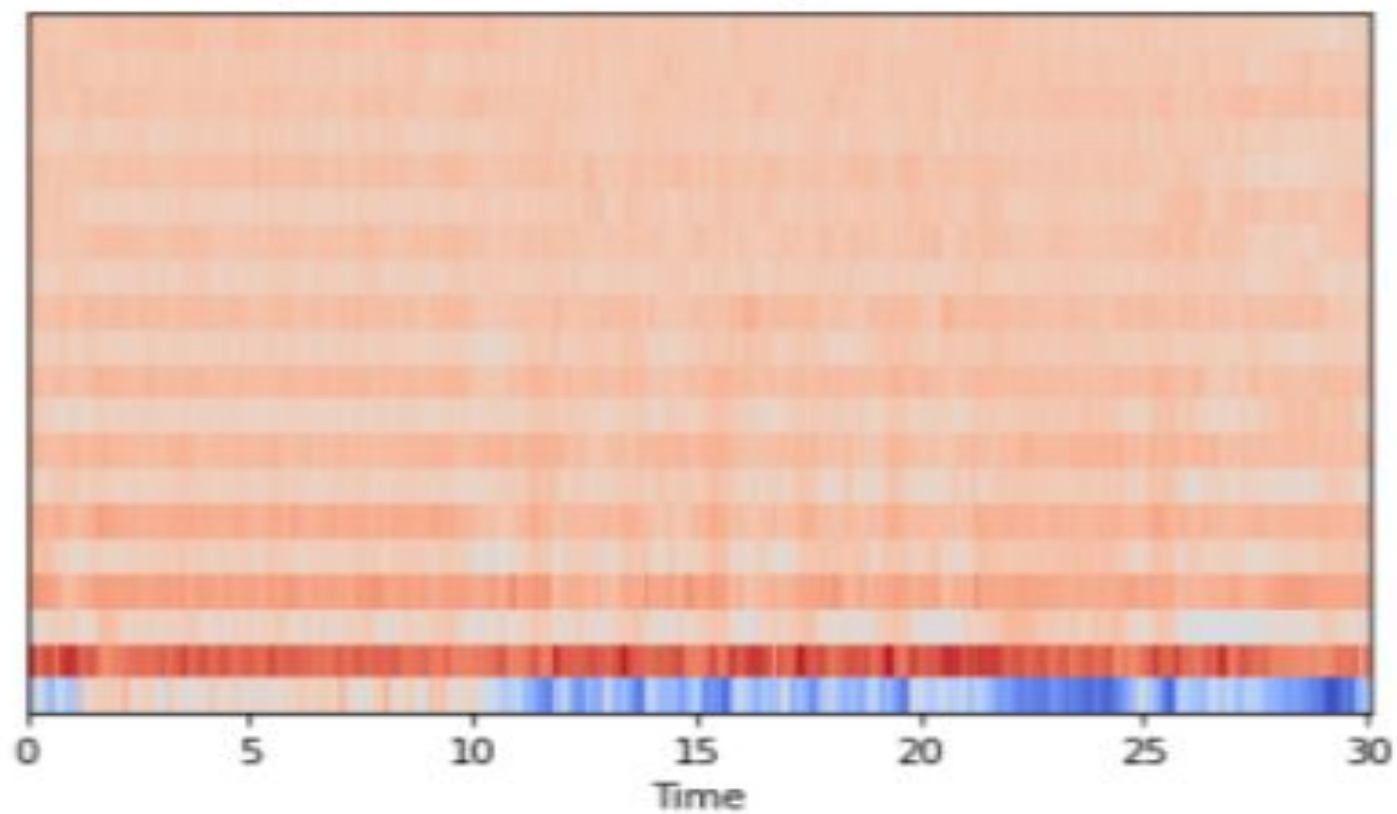
Mel-Frequency Cepstral Coefficients

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. MFCC values mimic human hearing, and they are commonly used in speech recognition applications as well as music genre detection.



Figure 2.11: Rock genre MFCC

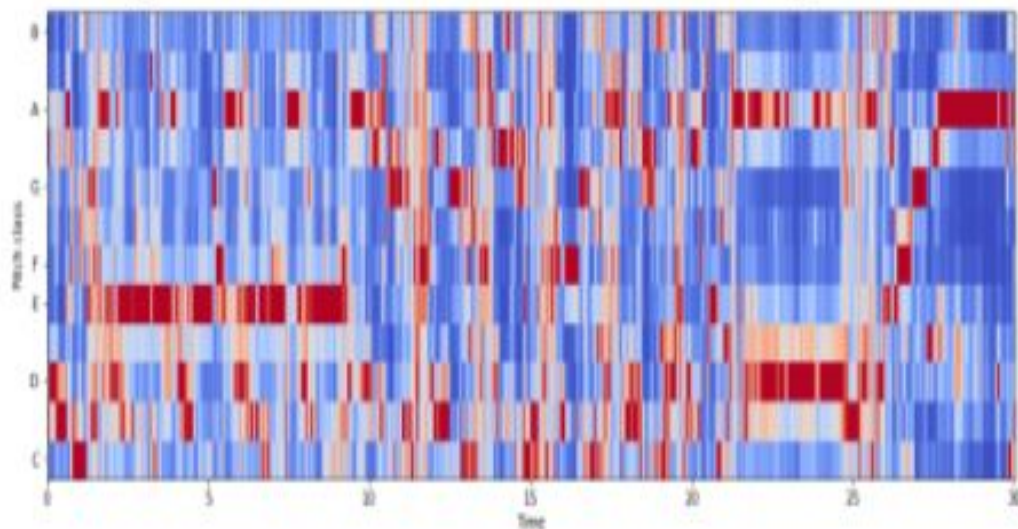


Chroma Frequencies

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. 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.



Figure 2.12: Rock Genre Chroma Frequencies

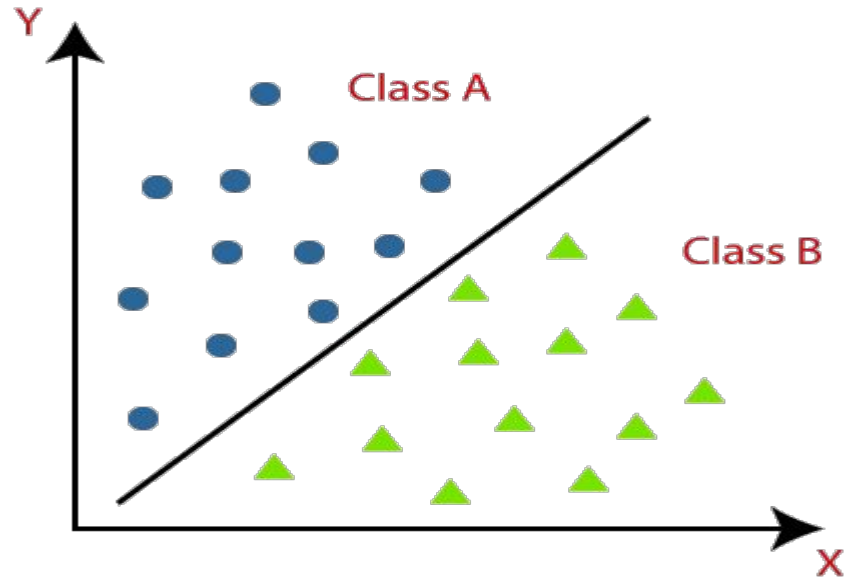




Librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

We used this python package to extract all the features from the dataset.

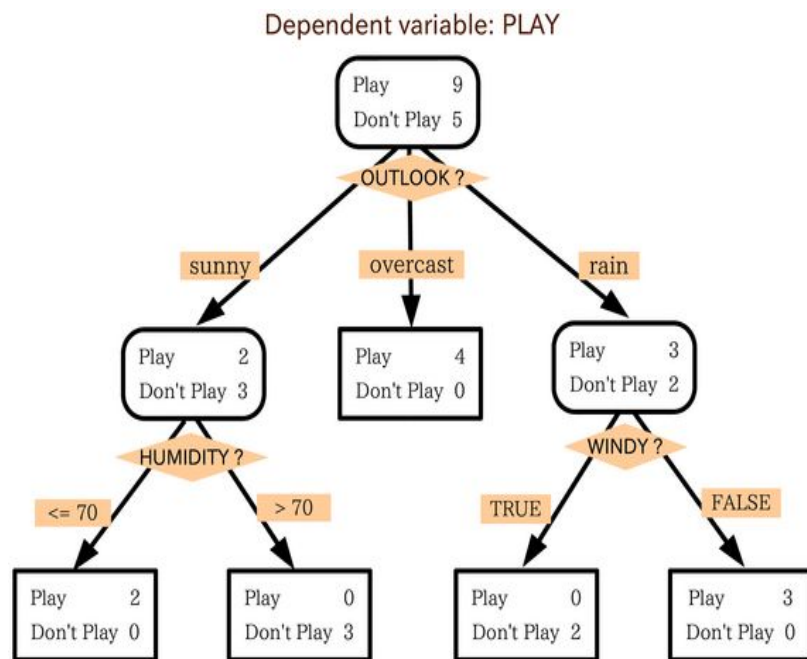
CLASSIFICATION



ENSEMBLING APPROACH

Ensemble methods combine several tree 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).





A Decision Tree determines the predictive value based on series of questions and conditions. For instance, this simple Decision Tree determining on whether an individual should play outside or not. The tree takes several weather factors into account, and given each factor either makes a decision or asks another question. In this example, every time it is overcast, we will play outside.

Voting Ensembles

A voting ensemble (or a “*majority voting ensemble*”) is an ensemble machine learning model that combines the predictions from multiple other models.

It is a technique that may be used to improve model performance, ideally achieving better performance than any single model used in the ensemble.



There are two approaches to the majority vote prediction for classification; they are hard voting and soft voting.

Hard voting involves summing the predictions for each class label and predicting the class label with the most votes. Soft voting involves summing the predicted probabilities (or probability-like scores) for each class label and predicting the class label with the largest probability.

- **Hard Voting.** Predict the class with the largest sum of votes from models
- **Soft Voting.** Predict the class with the largest summed probability from models.



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

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.



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



Convolutional **N**eural **N**etworks based Approach

CNNs

- CNNs are regularized versions of multilayer perceptrons
- specially designed neural networks for processing data that has a grid-like topology.
- effective in a wide range of applications, including computer vision, speech recognition, and natural language processing



The Approach

- We use 1-D convolution in our models.
- Called 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.
- the extracted features have dimensions of $(646, k)$, and our convolutional filters have dimension of $(3, k)$.



- 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
- Total 11 layers are there. Total parameters are 724554.
- We normalize the features so that they add up to 1, ending being probabilities.
- choose an optimizer such as Adam, and define the loss function
- aim is to not just generate a single genre per song, but generate a continuous genre prediction through the song
- although they are categorized to some parent genre, it contains elements of a lot of different genres



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		

Layers of the Model

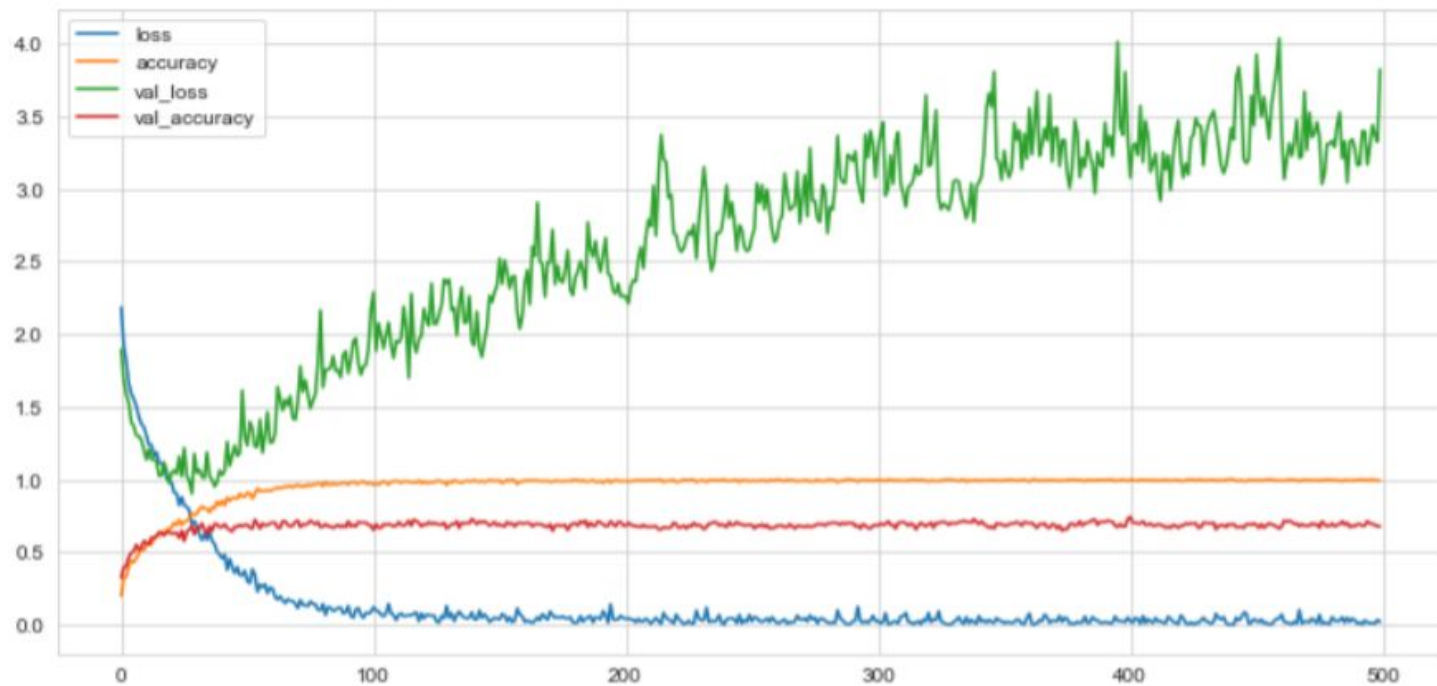
- After each convolutional layer, the model is passed through ReLU activation
- we use a categorical cross-entropy loss metric along with Adam optimizer with an initial learning rate of 0.001.
- The first layer will do the weighted sum of its inputs, its weights, and bias term, and then run the relu 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.



The value of loss and accuracy at some of the Epochs

```
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
```

Max. Validation Accuracy 0.7424242496490479



Final Graph

LightGBM based Approach

LightGBM

- LightGBM is a fast, distributed, high-performance gradient boosting framework based on decision trees algorithm.
- Developed by Microsoft and first version released in 2017.

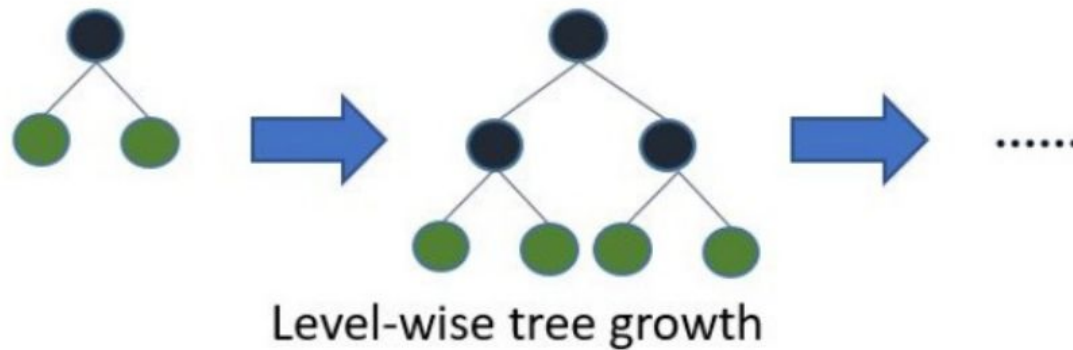


Why LightGBM?

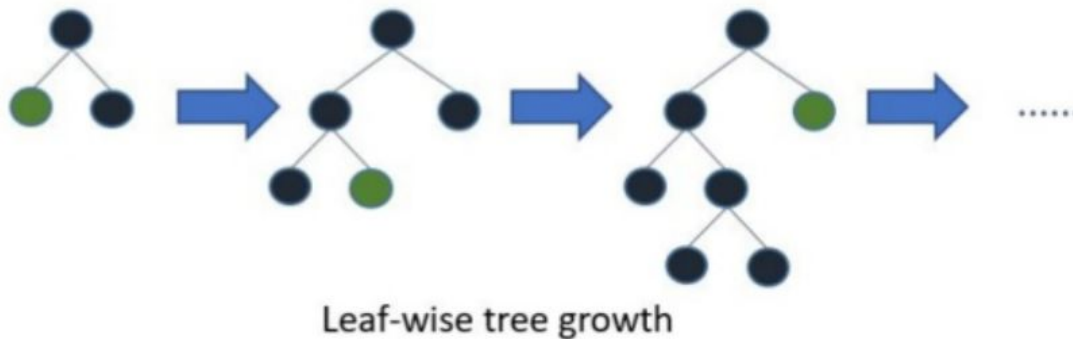
- LightGBM grows tree vertically while other algorithms grow trees horizontally meaning that LightGBM grows tree leaf-wise while others grow level-wise.
- It will choose the leaf with max delta loss to grow. When growing the same leaf, leaf-wise algorithm reduces loss much rapidly than a level-wise algorithm.



XGBoost:



LightGBM:



Tree-growth scheme

Features of LightGBM

- **Speed**
 - LightGBM is prefixed as 'Light' because of its high speed. LightGBM can handle the large size of data and takes lower memory to run.
- **Accuracy**
 - LightGBM focuses on accuracy of results.
- **Distributed / Parallel Computing**
 - LightGBM also supports GPU Learning.



Proposal

A LightGBM based system, that systematically classifies, unlabeled music files into predetermined bins of different genres.



Approach

- The 30 second music clips in the data, were further split into 10 mini clips, each 3 seconds long.
- These clips were then processed to extract necessary features, using the aforementioned procedure.
- A 3-fold cross validation method was used for training-testing.
- Most of the parameters used are standard, however, some (ex. Learning rate) were arrived after manual testing.



```
[ ] 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)
```

The parameters used in the model.

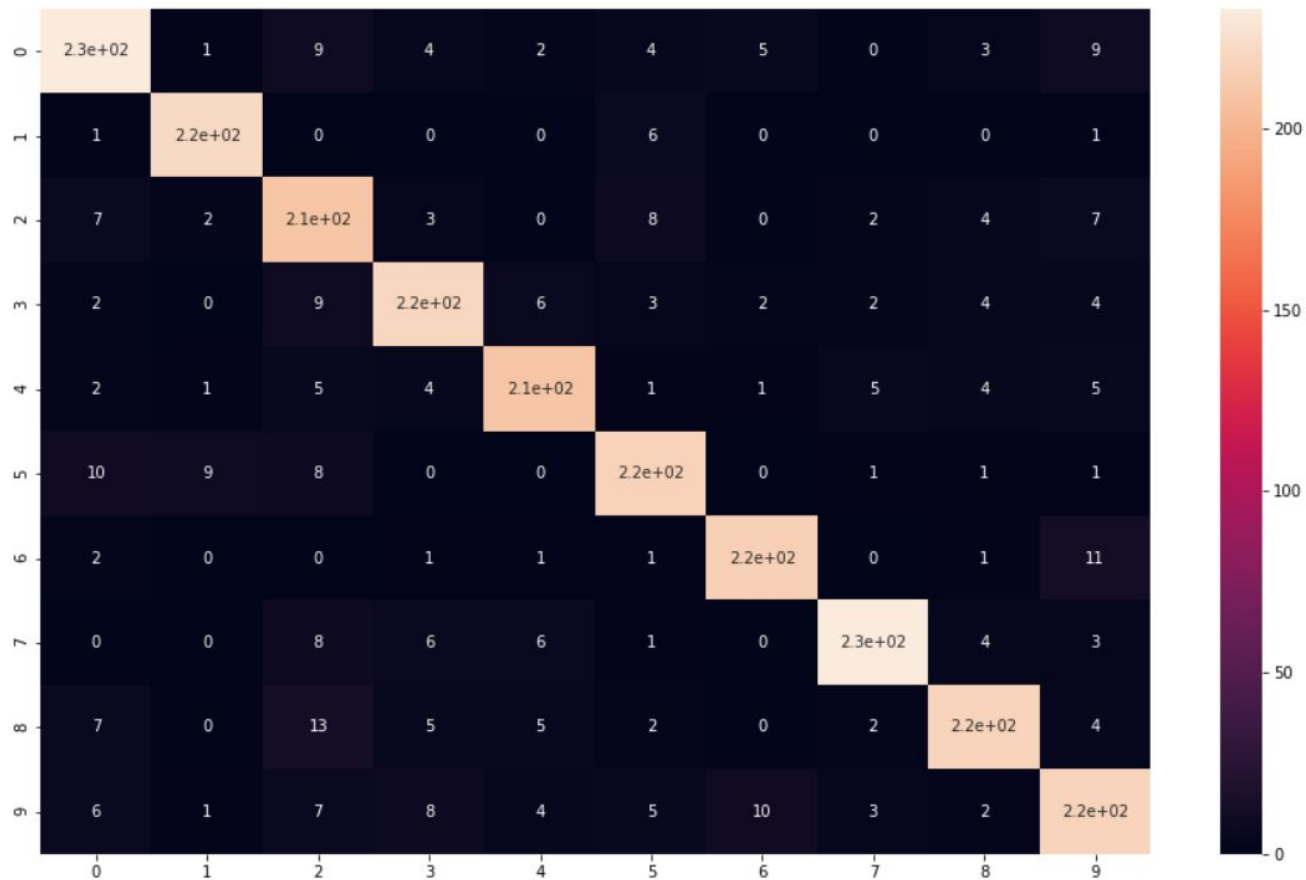

```
[ ] data.label = data.label.map({  
    'blues': 0,  
    'classical': 1,  
    'country': 2,  
    'disco': 3,  
    'hiphop': 4,  
    'jazz': 5,  
    'metal': 6,  
    'pop': 7,  
    'reggae': 8,  
    'rock': 9  
})
```

Output labels mapped to integers.

Observations

The model produces consistent and accurate results, giving an overall accuracy of 89% in testing.





Final test results presented in
form of confusion matrix.



Thank You