A Low Latency Pipeline with Integrated Music Genre Classifier

Major project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

Aditya Shukla (181383)

Under the Supervision of

Prof (Dr.) Vivek Sehgal

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

Foreword

In order to properly integrate the work in a neat and intuitive fashion project implementation has been divided into three different sections.

i. The first section (Ch 03) describes work on pipelines

ii. The second section (Ch 04) describes the feature extraction and machine learning models based on GTZAN.

iii. The third section (Ch 05, Ch 06) describes the preprocessing mathematics and model architecture of the proposed model.

iv. Finally, the performance of proposed model is compared with models proposed in other research papers (Ch 07).

Declaration

I certify that

1. The work contained in this report is original and has been done by myself in the general supervision of my supervisor.
2. The work has not been submitted anywhere else for any project.
3. I have given due credit to the materials (data, theoretical analysis, results) taken from other sources by citing them in the report and giving their details in the references.

Supervised by:

Prof (Dr.) Vivek Sehgal
Professor and Head,
Department of Computer Science and Engineering,
Jaypee University of Information Technology

Submitted by:

Aditya Shukla
181383
Department of Computer Science and Engineering,
Jaypee University of Information Technology

Certificate

This is to certify that the work contained in this report entitled “A Low Latency Pipeline with Integrated Music Genre Classification” submitted by Aditya Shukla student of B.Tech (Semester VIII) Computer Science and Engineering JUIT, Waknaghat is a bona fide work performed by him under my supervision during the period February 2022 to May 2022 at Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Prof (Dr.) Vivek Sehgal
Professor and Head,
Department of Computer Science and Engineering,
Jaypee University of Information Technology

Acknowledgement

We are highly indebted to Dr Sehgal for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support throughout the duration of the project.

We would like to express my gratitude towards my parents for their kind co-operation and encouragement which helped me in the completion of this project.

My (Aditya Shukla) thanks and appreciation also goes to the research scholars working in the Lab, Department of CSE, IIT (BHU) who have willingly helped me with the compute infrastructure required for this project.

Contents

Chapter 1:

Introduction

Over the past decade significant improvements in accessibility and usability of machine learning technology (hardware and software) have led to their wide adoption of machine learning in various domains. Recently, a campaign spearheaded by Dr Andrew Ng, has highlighted the need the for proper management and delivery of data. A robust data centric system is required for the proper functioning of machine learning.

The project focuses on integrating a deep learning based music genre classification model with a ultra-reliable scalable low latency pipeline. The robust music genre classification system is designed to quickly give information of a given sound clip using only 10 seconds (in this project) or 30 seconds (in most research). This project proposes a variety of different mathematical techniques for preprocessing the input audio signal and converting the audio signal into a Mel-scaled spectrogram. This spectrogram is then used as input for a custom deep learning model which outputs the genre of the model. The model proposed in this report output performs all other models by a margin.

Problem Statement

Over the past few years, the value of data driven and data centric machine learning models has increased significantly. Due to the data intensive nature of machine learning models, it is essential ensure data integrity and security while ensuring low latency. Data pipelines are designed to ensure the integrity, security, low latency and scalability. This project implements a scalable data pipeline as a microservice for transferring audio file from client to the processing server.

Recent years have seen a significant growth in online music services and music databases. These services are dependent heavily on recommender systems for customer retention. Classifying music by genre is one of the most useful techniques used to solve this problem. One of the most reliable methods of music genre classification is based on the study of acoustic characteristic of a given music file. Most of the research in focuses on classification of clean noise free audio signal. The clean data does not properly reflect real life situation. To this end the model used in this project is trained on a specially designed "Noisy GTZAN"[1]**.** This project proposes a novel deep learning model and compares the result with previous research. The problem statement of the is:

"To create a low latency, scalable data pipeline with a music genre classification model"

The Dataset

For audio input model to work properly in the real world, it is essential that the model be robust against noise. Therefore, a special noisy dataset needs to be created.

1. The project uses the GTZAN dataset as a base.
2. The GTZAN dataset is curated by the MARSYAS, an open-source framework for audio processing.
3. The GTZAN dataset is the most widely used dataset for music genre classification.
4. The dataset consists of audio clips of approximately 30 seconds each.
5. Each audio clip represents one of ten different music genres.
6. There are 100 audio clips per genre
7. All the tracks are 22050Hz single channel 16 bit audio files encoded in .wav format

The noise dataset consists of a few different types of noise. These are wind, mechanical, electrical feedback, white noise, factory and machine tools noise. Samples of these types of noise was collected from various open source projects.

The final dataset dubbed ***Noisy GTZAN*** is created by overlaying the every audio clip GTZAN with the noise samples. The resulting dataset consists of 390 to 400 audio clips per genre. It retains the 22050Hz frequency but uses 32 bit encoding instead of 16 bit encoding. The clip size is also reduced to 10 sec in the interest of storage.

Chapter 2:

Literature Survey

Ketan Doshi has written a series of articles on applying various state of the art techniques on audio. He started with what is sound, how it can be represented in numbers and images. He gives brief introduction of spectrograms and Mel Spectrograms and why the latter one is better to represent audio data, what are the different hyperparameters, what are MFCC's and how the data can be augmented. He also discussed various applications of deep learning in audio and human speech field.

Nagesh Singh Chauhan in his article gives a brief introduction to audio data analysis and its application in our day-to-day life. He discussed various features that can be extracted from audio data, relevant to problems the one is trying to solve. He discussed various spectral features (frequency-based **features**), which are obtained by converting the time-based signal into the frequency domain using the Fourier Transform, like fundamental frequency, **spectral** centroid, **spectral** flux, **spectral** density, **spectral** roll-off, etc.

An article on Devopidia discussed various categorization of features in detail. Categorization based on level of abstraction, temporal scope, musical aspect, signal domain.

Martin F. McKinney and Jeroen Breebaart in their research paper titled "Features for Audio and Music Classification" evaluated 4 audio features in classifying 5 audio classes and 7 popular music genres namely

i. low-level signal properties
ii. MFCC
iii. psychoacoustic features and
iv. temporal envelope fluctuations.

They concluded for both audio and music classification temporal behaviour of features is important.

George Tzanetakis and Perry Cook in their research paper titled "Musical Genre Classification of Audio Signals" proposed 3 feature sets for representing rhythmic content, pitch content, and timbral texture (a thirty-dimensional feature vector). They used different machine learning algorithms to evaluate the features.

Michael I. Mandel and Daniel P.W. Ellis in their research paper titled "Song-Level Features and Support Vector Machines for Music Classification" proposed calculating features over the entire length of song rather than short clips. They discussed Support Vector Machines and K-NNN classification with 3 different distance measures for both algorithms.

Lonce Wyse in his research paper titled "Audio spectrogram representations for processing with Convolutional Neural Networks" explored different representation of audio data, focusing particularly on spectrogram for neural networks.

Hareesh Bahuleyan in his research paper titled “Music Genre Classification using Machine Learning Techniques” compared two classes of models in music genre classification. The first one is to use different machine learning algorithms on various hand-crafted features (frequency and time domain). The second is to use convolutional neural network on different spectrograms (treating spectrograms as images).

Table

S.no	Link
Dataset	
1)	http://marsyas.info/downloads/datasets.html
Articles	
2)	https://towardsdatascience.com/audio-deep-learning-made-simple-part-1-state-of-the-art-techniques-da1d3dff2504
3)	https://towardsdatascience.com/audio-deep-learning-made-simple-part-2-why-mel-spectrograms-perform-better-aad889a93505
4)	https://towardsdatascience.com/audio-deep-learning-made-simple-part-3-data-preparation-and-augmentation-24c6e1f6b52
5)	https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5
6)	https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html
7)	https://devopedia.org/audio-feature-extraction
Documentations	
8)	https://librosa.org/doc/latest/index.html
9)	https://pypi.org/project/sounddevice/
Research Papers	
10)	https://jscholarship.library.jhu.edu/handle/1774.2/22
11)	https://ieeexplore.ieee.org/abstract/document/1021072
12)	https://academiccommons.columbia.edu/doi/10.7916/D8QV3WWQ
13)	https://arxiv.org/abs/1706.09559
14)	https://arxiv.org/abs/1804.01149

Chapter 3: The Pipeline

Parts of Pipeline

Containerization

Consider a simple node.js based application. In order to deploy this node.js based application, it need to be pushed on to a server running virtual machine, the virtual machine is required for access control, network and resource management, security etc. The host virtual machine requires its own operating system and a set of binaries and libraries. This significantly increases the size of the application. In order to scale the node.js application, multiple copies of the same virtual machine needs to be created. In addition to this, if the application is platform dependent, then it is likely that some changes will be required to ensure that the application runs properly on the servers. Some of these changes will have to made after deployment owing to limitation of the development platform.

Image of resource comparison

To deploy the same application using a container, first a container is initialized by creating an container image. A container image is created using a file that describes the container. This could be dockerfile for Docker containers or manifest channel in cloud foundry. A container-based deployment offers the following benefits:

1. Containers are easier to deploy and manage using docker daemons or runtime engines (which replace hypervisors).

2. Containers require significantly lower resources and have better security integration.
3. Resources are shared between containers.
4. Containers also allow easier integration of third part tools or new languages to containerized application. The third party application can simply be deployed as a standalone container and connected to other containers using internal network systems e.g. docker network etc.
5. Due to their light weight, system independent nature, it was possible to run containers on local machines, this enabled developers to develop applications within the containers, thereby reducing the time of development.

Apache ZooKeeper

Zookeeper is an open source project which provides multiple features for distributed applications. It is an important system for maintain data across the cluster in a consistent manner.

In a distributed system it is necessary to for every client to have a consistent picture of the following points:

1. which node serves as master node
2. task assigned to different workers
3. list of workers currently available

Therefore it is necessary for a distributed system to have an external system, like Zookeeper, which can manage the clusters in the distributed system.

Zookeeper is an integral part of various different application including Kafka, HBase, HighAvailability Map Reduce etc.

Zookeeper is designed to handle a wide variety of failure modes including master crashes and worker crashes. Zookeeper also has systems in place to detect network trouble, i.e. a situation where part of the network is not visible either due to fault with dynamic DNS or routing error or hardware errors etc.

Zookeeper essentially provides distributed configuration management, has features that facilitate self-election/consensus building, coordination and locks.

Zookeeper Architecture:

The adjacent images shows the architecture of an application running in a cluster using the zookeeper client. Each cluster has a master node, this master node is set by Zookeeper. Every client in the application is linked to the Zookeeper in order to facilitate independent communication with a Zookeeper ensemble. Since all the applications are dependent on ZK for proper functioning, therefore it is advisable to have more than one Zookeeper server.

Figure 1: Architecture of application based on Zookeeper

Apache Kafka

Apache Kafka is a message queue system which is designed to be an asynchronous service to service communication system. It is designed to be used in microserver and serverless environments.

Apache Kafka provides five core APIs which can be used for various different functionalities:

1. Producer API: Publish streams of data to kafka clusters
2. Consumer API: Consume streams of data from Kafka clusters
3. Streams API: Transferring data from input to output stream
4. Connect API: Connecters that manually pull data from predefined sources.
5. Admin API: Admin control of Kafka topics, brokers, clusters etc

This project uses the Producer and Consumer API to create data transfer pipeline.

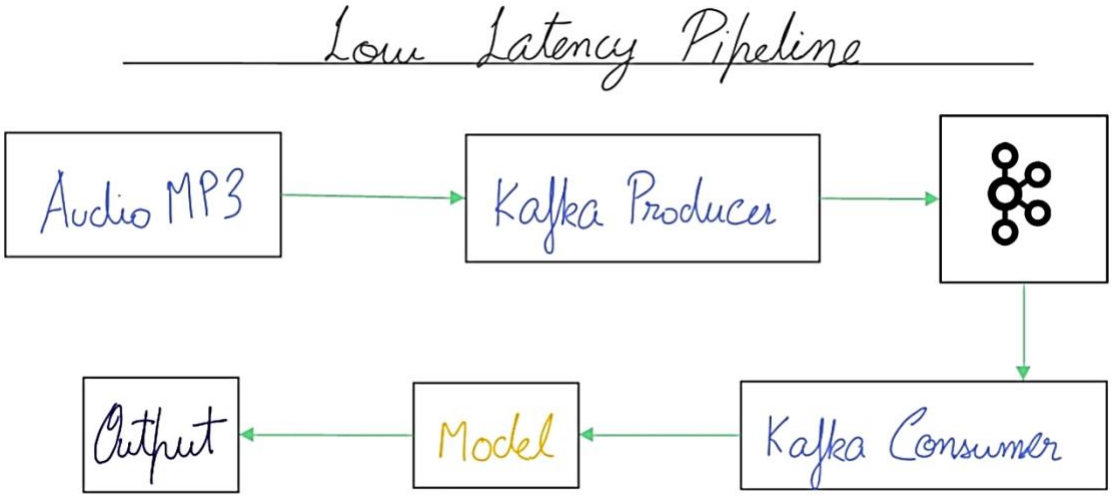
A dedicated data pipeline ensures the following:

1. Data Consistency and Integrity:
    - For a data centric model integrity is paramount for data pipelines.
    - Properly configured pipelines are designed to minimize aberrations.
    - Various techniques such are checksums etc. are used to ensure integrity.
2. Low Latency:
    - Systems deployed in defense, healthcare, fintech and in various other domains often operate in real time.

- For such systems the ability to receive and transmit data at very low latency is paramount.
- Pipelines can provide very low latency while ensuring integrity of data.

3. High Security:
    - The data used for machine learning models is very sensitive.
    - High security features integrated into the pipeline are designed to prevent data leakage and data changes.
4. Standard Integration
    - Pipelines are designed with well testes standard integrations.
    - This reduces the points of failures.
5. Scalability:
    - With increase in load, it is possible to horizontally scale the pipeline

Designing and Implementing the low latency pipeline



The pipeline is composed of 4 essential modules. They are as follows:

1. Kafka Producer:

   The Kafka Producer API is designed to connect nodes with kafka MQ clusters. This project implements the producer api using python programming language. First the input, either mp3 or wav, is read into memory using PyDub library. Next, the file is converted to 256-bit binaries packets and serialized. Finally, the packets thus produced are sent to Kafka cluster.

2. Kafka Brokers:

   The Kafka brokers are a group of servers that run Kafka instances. These servers are managed by zookeeper at the backend and they are stored either at one location or are distributed across several regions. The messages received by Kafka brokers are stored in *Kafka topics*.

These topics are divided into partitions. These partitions are essential for scalability and for backup. During configuration the duration of data persistence (temporary storage) can also be set, by default this value is 3 days. This project deploys Kafka instance(s) and Zookeeper instance(s) as microservices on Docker®™ containers. This modular approach ensures horizontal scalability and independent functionality.

3. Kafka Consumer:

   The Kafka consumer API is designed to consume messages from Kafka brokers. Each consumer instance has to “subscribe” to the topics that it wants to read of Kafka brokers. These messages are then set in order in real-time to the *subscribed* producers. The system maintains every consumer’s last read message’s index number. If the consumer fails it can start from the last read location.

   This project implements Kafka producer using Kafka client for Python. Packets of data are read from the brokers and reserialized. These are concatenated in memory as a binary sequence and then converted to and saved as wav or mp3. They audio file is then processed to reduce noise and finally a mel-spectrogram is sent to the model.

4. Model:

   The model receives a Mel scaled spectrogram as input and outputs the genre of music file with 96% accuracy.

Chapter 4: GTZAN

Feature Extraction of GTZAN

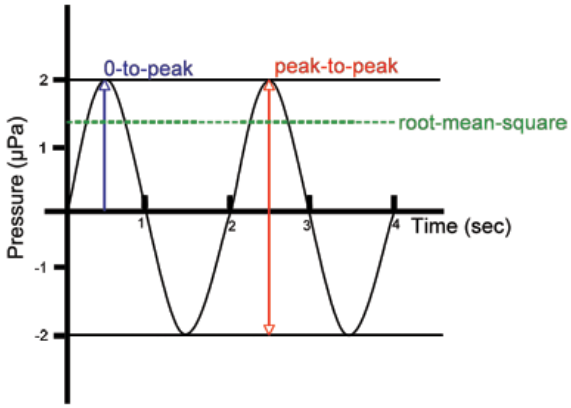
Chromogram

Chromagram or chroma features capture melodic and harmonic characteristics of music. It is used for music whose pitches can be classified into 12 pitch classes. This generates 12 element array for a single audio file, representing energy of each pitch class. We calculated mean and variance of the values to represent the whole array as 2 features.

librosa.feature.chroma_stft() function is used to calculate vector which takes audio file and sample rate as input.

Root Mean Square Energy

This feature indicates the loudness in the audio as energy can be used to represent loudness.



librosa.feature.rms() function is used to calculate Root Mean Square Energy for each frame in audio file. It returns an array from which we calculated mean and variance of values to represent complete vector.

Spectral Centroid

Spectral Centroid used to calculate the frequency band where most of the energy is centred. This gives us the centre of mass of audio data. Mathematically it is the weighted mean of frequency bins:

$$f_c = \frac{\sum_k S(k)f(k)}{\sum_k S(k)}$$

where f(k) is the frequency corresponding to bin k and S(k) is the spectral magnitude of frequency bin k.

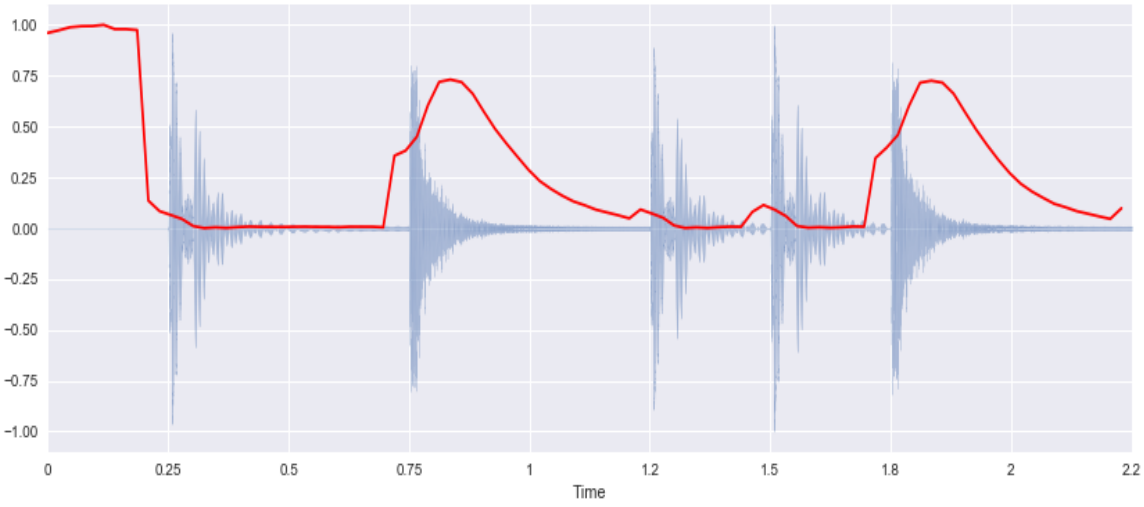


Figure 2: Spectral centroid distribution of an audio sample

librosa.feature.spectral_centroid() function is used to calculate spectral centroid for each frame. It returns an array from which we calculated mean and variance of values to represent complete vector.

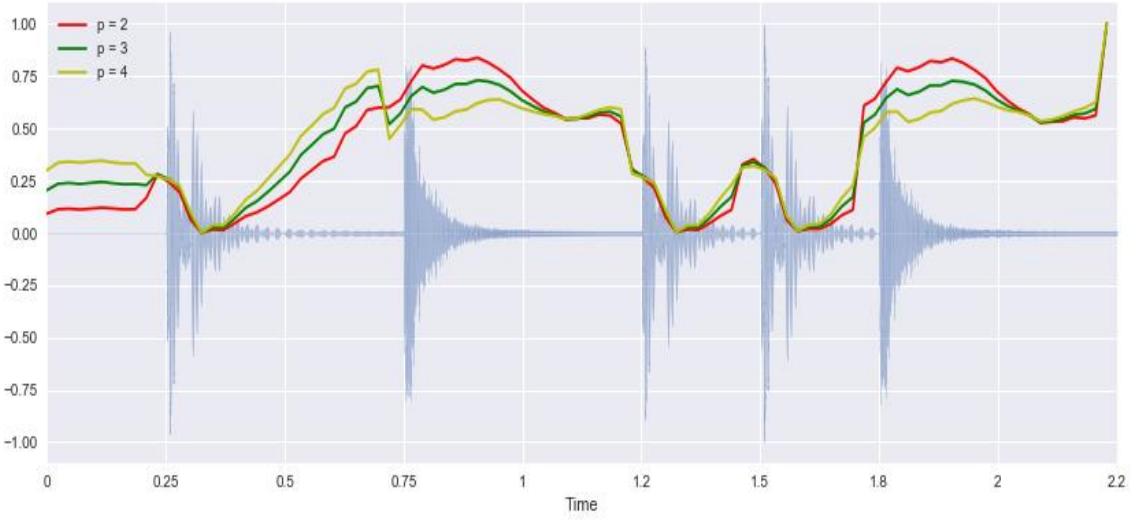
Spectral Bandwidth

Spectral bandwidth is the width of the band of light at one-half the peak maximum. Mathematically, it is the weighted mean of the distances of frequency bands from the Spectral Centroid.

librosa.feature.spectral_bandwidth() is used to computes the order-*p* spectral bandwidth:

$$\left(\sum_k S(k)(f(k) - f_c)^p\right)^{\frac{1}{p}}$$

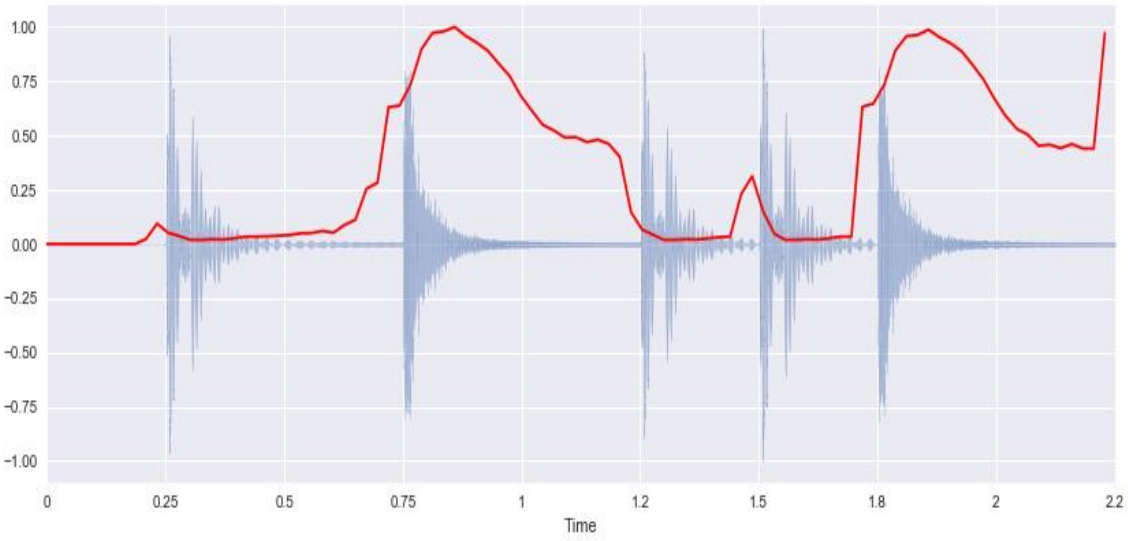
where *f*(*k*) is the frequency of bin *k*, *f(c)* is spectral centroid, *and S*(*k*) is the spectral magnitude of frequency bin *k*. It is like a weighted standard deviation, when *p* = 2. It returns an array from which we calculated mean and variance of values to represent complete vector.



Spectral Rolloff

Spectral Rolloff represents the shape of the signal. It is calculated by the fraction of bins that are below a certain threshold defined by user, ex 85% of total spectrum power.

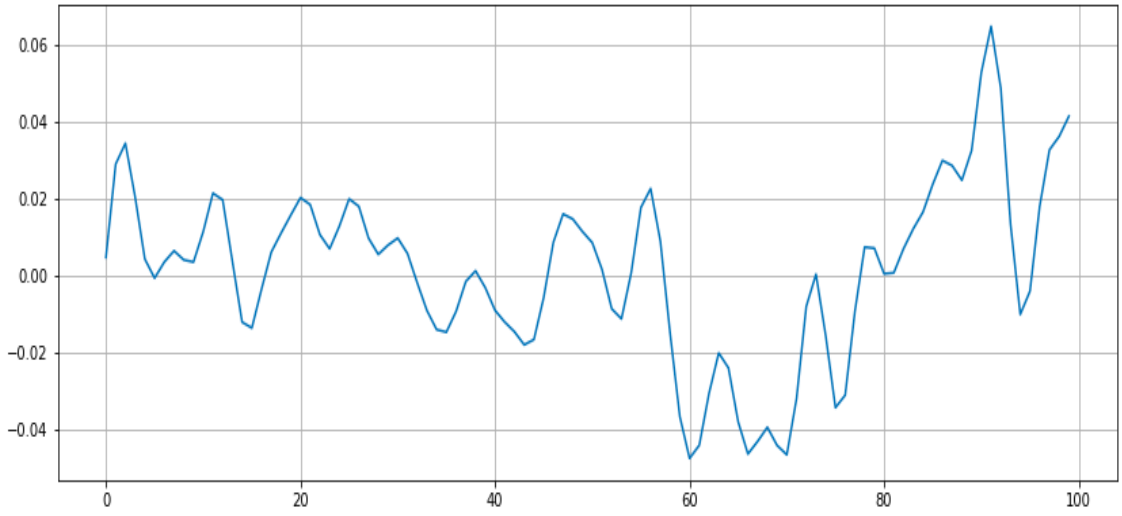
librosa.feature.spectral_rolloff() is used to calculate the rolloff frequency for each frame in an audio file. It takes the audio data and sample rate as input. It returns an array from which we calculated mean and variance of values to represent complete vector.



Zero Crossing Rate

Zero Crossing Rate basically tells us the smoothness of signal. It is calculated by measuring the number of times signal crosses horizontal axis, or the number of times the signal becomes zero. Signal can change its value either from positive to negative or from negative to positive.

librosa.zero_crossings() function is used to calculate the zero crossings rate. The entire audio length is divided into smaller frames and number of zero crossings is calculated in each frame. It returns an array from which we calculated mean and variance of number of zero crossings values to represent complete vector.



There are total of 16 zero crossings in above representation.

Harmony

librosa.effects.harmonic() is used to extract the harmonic content from the audio data. It returns an array from which we calculated mean and variance of values to represent complete vector.

Percussion

librosa.effects.percussive() is used to extract the harmonic content from the audio data. It returns an array from which we calculated mean and variance of values to represent complete vector.

Tempo

Measured in beats per minute (BPM), tempo tells us the pace of the audio file. It is calculated by mean of the tempo across several frame of the audio file.

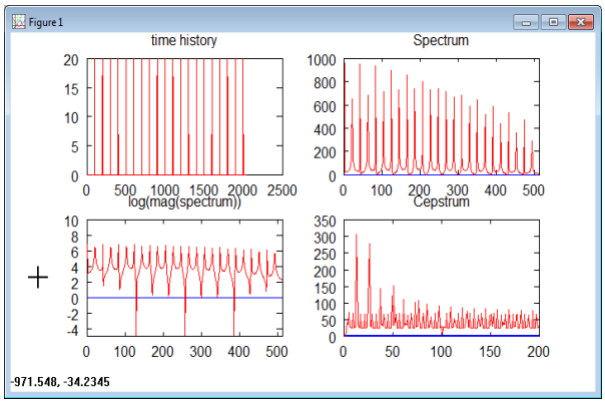
librosa.beat.tempo() is used to calculate the tempo of an audio file.

Mel-frequency cepstral coefficients

One of the most useful type of features, they are widely used to represent the whole audio file. Mathematically, first the audio is broken into multiple overlapping frames, then the fourier transform is applied to the time signal representation of the audio to get the fourier spectrum. Then log is taken of the magnitude of the fourier spectrum, and after that cosine transformation is applied to get the spectrum of the log. This resulting spectrum is neither in time domain nor in the frequency domain since transformation in applied to fourier spectrum itself. This domain is specifically named as quefrency domain and the resultant spectrum is named as cepstrum.

Therefore, Mel-frequency cepstral coefficients are just the coefficients that make up the mel-frequency cepstrum.

librosa.feature.mfcc() is used calculate mfcc's of an audio signal. It returns 20 vectors. For each vector we calculate the mean and variance to generate to features. In this way 40 features are generated to represent the audio signal.



Description of Machine Learning Techniques Used

Naïve Bayes

Naïve Bayes is a supervised classification algorithm based on bayes theorem. It uses concepts of conditional probability assuming each feature to be independent of other features and can be applied in both binary and multiclass classification.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

P(A)= The probability of A occurring P(B)= The probability of B occurring

P(A? B)=The probability of A given B P(B? A)= The probability of B given A

P(A? B))= The probability of both A and B occurring

Without normalization Naïve bayes achieved an accuracy of 43.11%, and after normalization algorithm achieved an accuracy of 51.95% accuracy.

K-Nearest Neighbourhood

KNN is a supervised machine learning technique which can be used for both classification and regression. It does not assume anything about the dataset beforehand (non-parametric). K in KNN is a hyperparameter for which we have to try various values to find the best one. This algorithm predicts the value on the basis nearest neighbours. Different distance metrics can be used

to calculate nearest neighbourhood, most commonly used is eucledian distance:

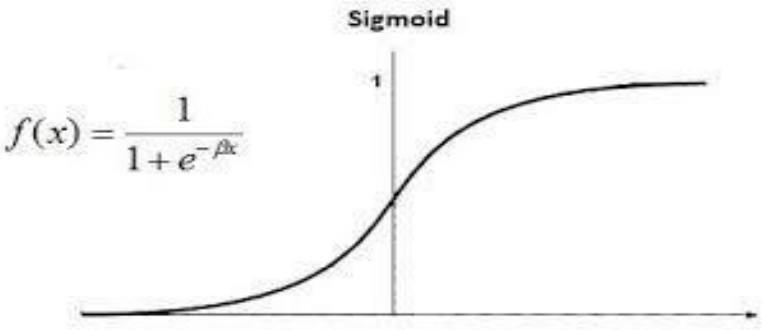
$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Without normalization KNN algorithm achieved an accuracy of 29.36%, and after normalization, algorithm achieved an accuracy of 80.58% accuracy.

Logistic Regression

Logistic regression is a supervised machine learning algorithm which is used to predict the probability of a dependent, categorical target variable given one or more independent input variables. It can be applied to both binomial and multinomial classification problems. It uses sigmoid activation function for modelling.



Cost function used by Logistic regression is:

$$\text{Cost}(h_\Theta(x), Y(\text{actual})) = -\log(h_\Theta(x)) \text{ if } y=1$$

$$-\log(1- h_\Theta(x)) \text{ if } y=0$$

Without normalization Logistic regression algorithm achieved an accuracy of 30.19%, and after normalization, algorithm achieved an accuracy of 69.77% accuracy.

XGB Classifier

XGB classifier is an advance implementation of gradient boosting algorithms using decision trees. It works on the principle of ensemble learning. It uses various regularization techniques to reduce overfitting and parallel computing to increase the speed.

Without normalization XGB Classifier algorithm achieved an accuracy of 90.29%, and after normalization, algorithm achieved an accuracy of 90.22% accuracy.

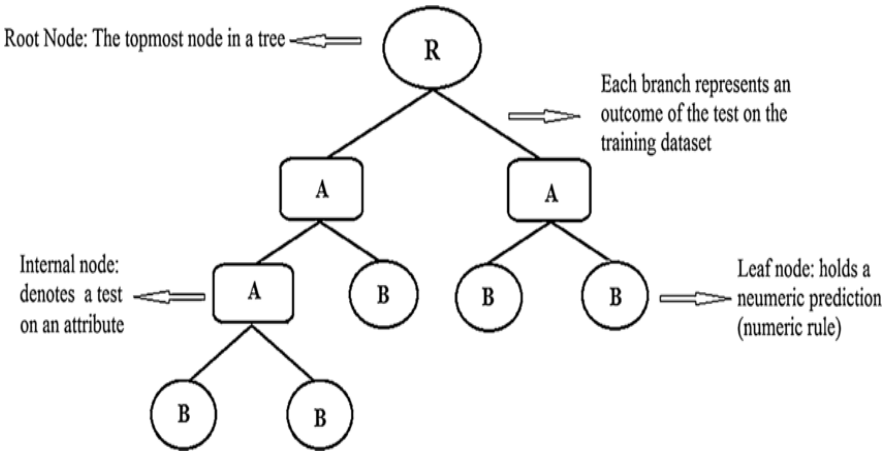
Decision Trees

Decision trees is a supervised machine learning algorithm which uses a tree like structure of decisions for predictive modelling. It also uses the concept of entropy and information gain as the basic building block of the algorithm. For calculating uncertainty, entropy is used as a metric and information is used to determine how to reduce the uncertainty.

Each internal node represents the test on the variable, the branches in a decision tree represents the observation for an item or the result of the test and leaf nodes are used to represent the value of the target variable. Decision trees

can be used for both regression analysis and classification analysis. If the target variable (represented by leaves) can take discrete values, it is a classification model and if the target variable can take continuous values, it is a regression model.

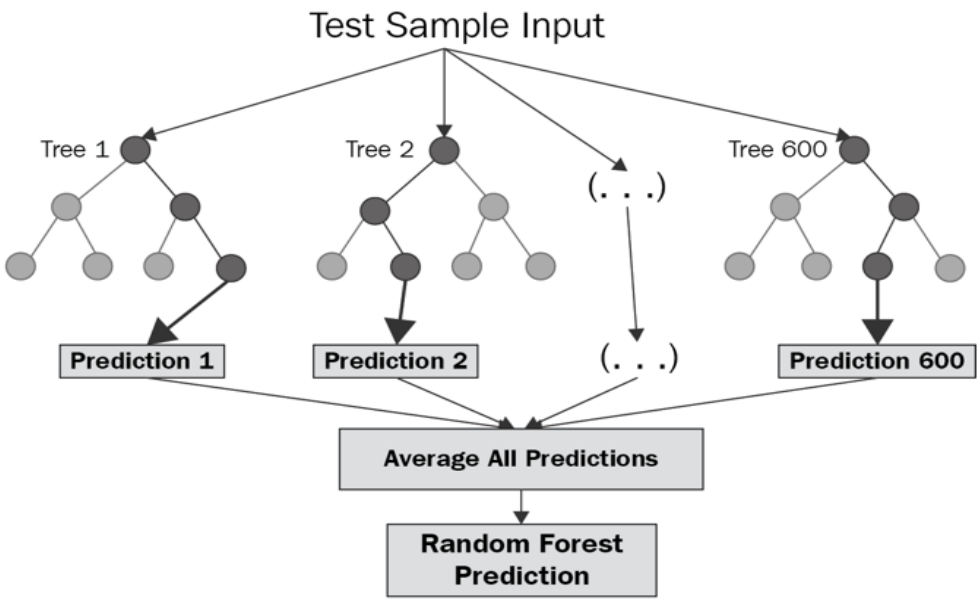
Without normalization decision tree algorithm achieved an accuracy of 64.78%, and after normalization, algorithm achieved an accuracy of 63.99% accuracy.



Random Forest

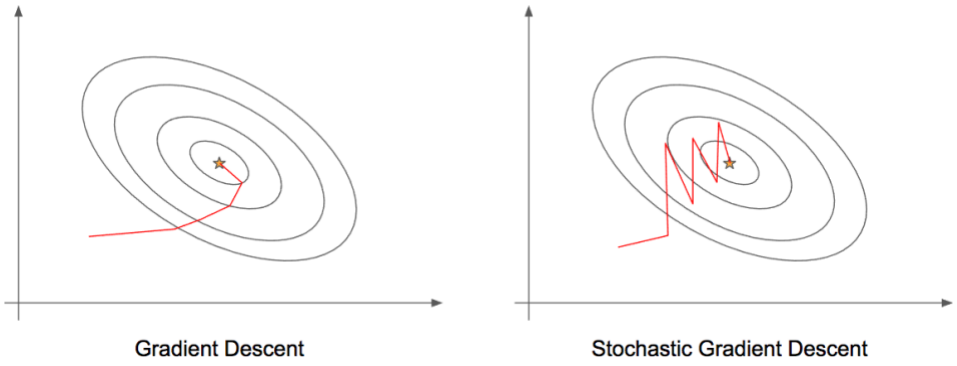
Random forest is a supervised machine learning algorithm based on ensemble learning. It uses many decision trees to solve various complex problems of classification as well as regression. The training process in this algorithm is based on bootstrap aggregating. The training data is divided and supplied to decision in trees in random order. In classification, majority voting system is followed to get final output and in regression we take the mean of the output of each decision to tree to get the final output.

Without normalization random forest algorithm achieved an accuracy of 81.34%, and after normalization, algorithm achieved an accuracy of 81.41% accuracy. There is no noticeable difference in accuracy after normalization.



Stochastic Gradient Descent Classifier

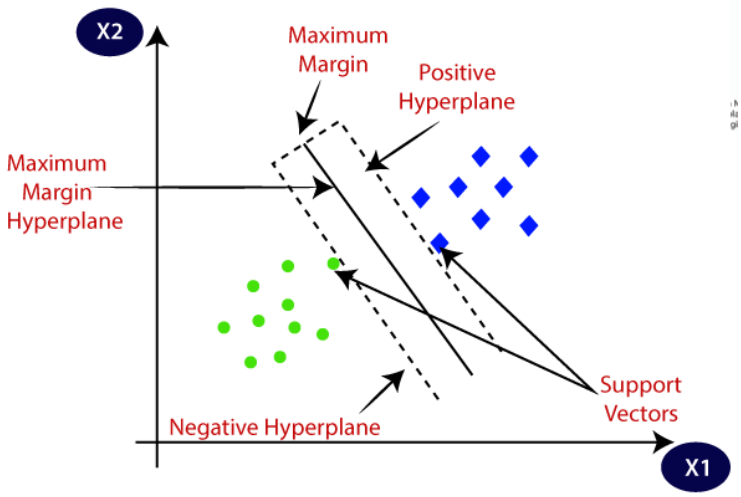
SGDClassifier uses Support vector Machines as a default classifier with stochastic gradient descent as an optimization technique. In normal gradient process, whole data is selected for training purpose in each iteration. With Stochastic gradient descent technique, a single data point from the whole data is selected randomly for training purpose in each iteration. This reduces computation time for very large dataset at cost of noisier path to reach the minima as compare to normal gradient descent process.



Without normalization SGDClassifier algorithm achieved an accuracy of 16.85%, and after normalization, algorithm achieved an accuracy of 65.53% accuracy.

Support Vector Machines

Support vector machines is a supervised machine learning algorithm mostly used for classification problems bur can also be used in regression problems. The target of the SVM algorithm is to find the best decision boundary or line between different classes in a n dimensional space, also known as hyperplane. The closest data points are known as support vectors, they act as the boundary of the hyperplane or support for the hyperplane. The hyperplane is selected such that the margin between the data points or support vectors between 2 classes should be maximum. The dimension of the hyperplane depends on the number of the features in the dataset.



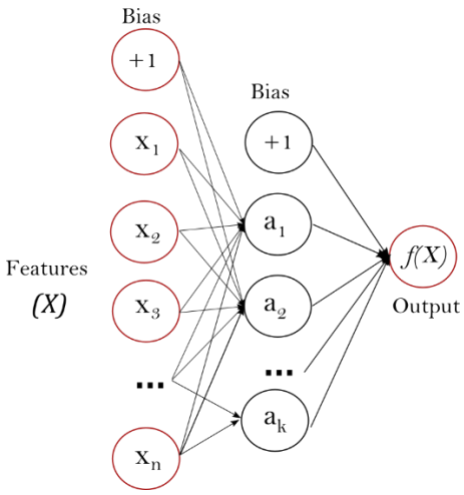
The loss function to maximize the margin by SVM algorithm is known as hinge loss function. The first term in below loss function is regularized parameter and other one is loss function.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Without normalization SVM algorithm achieved an accuracy of 28.99%, and after normalization, algorithm achieved an accuracy of 75.44% accuracy.

Multilayer Perceptron Classifier

MLPClassifier is a basic neural network (comes under supervised machine learning technique) which aims at learning a function which maps the input variables to the output variables. It can be used for both classification as well as regression problems. In logistic regression input directly maps to output using layer, but in case of MLPClassifier there can many hidden layers. This classifier can take a multiple number of parameters like, size of the hidden layer, activation function used, number of training iterations, optimizers, etc.



Without normalization MLPClassifier algorithm achieved an accuracy of 10.04%, and after normalization, algorithm achieved an accuracy of 67.73% accuracy.

Results and Performance:

The following graphs compare the train accuracies of various models

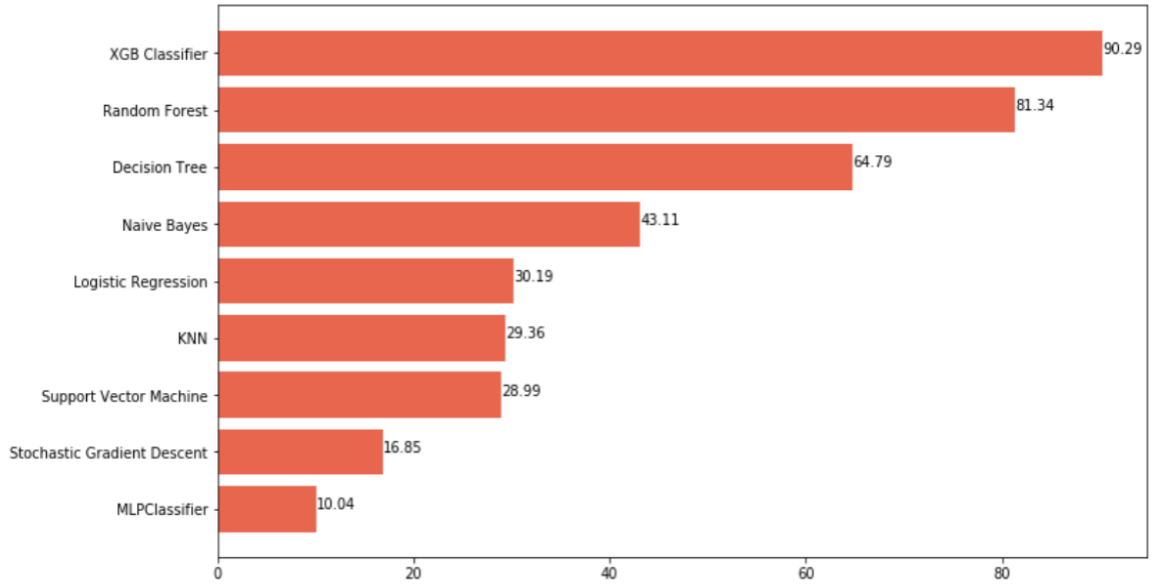


Figure 3:Accuracies of different machine learning techniques without normalizing data

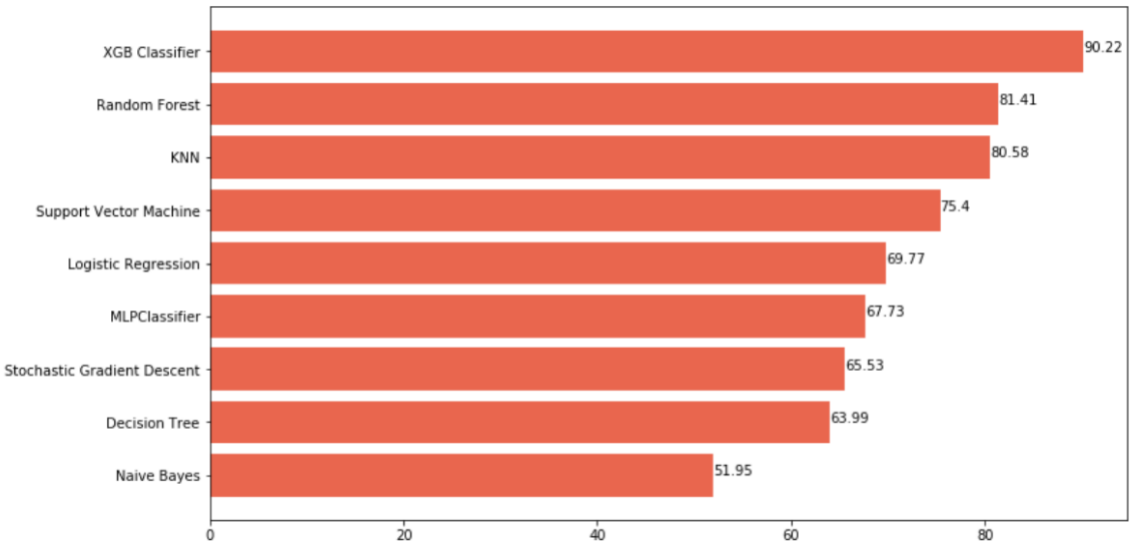


Figure 4: Accuracies of different machine learning techniques after normalizing data

XGBoost classifier achieved the highest accuracy of 90.224%.

Confusion Matrix of XGBoost:

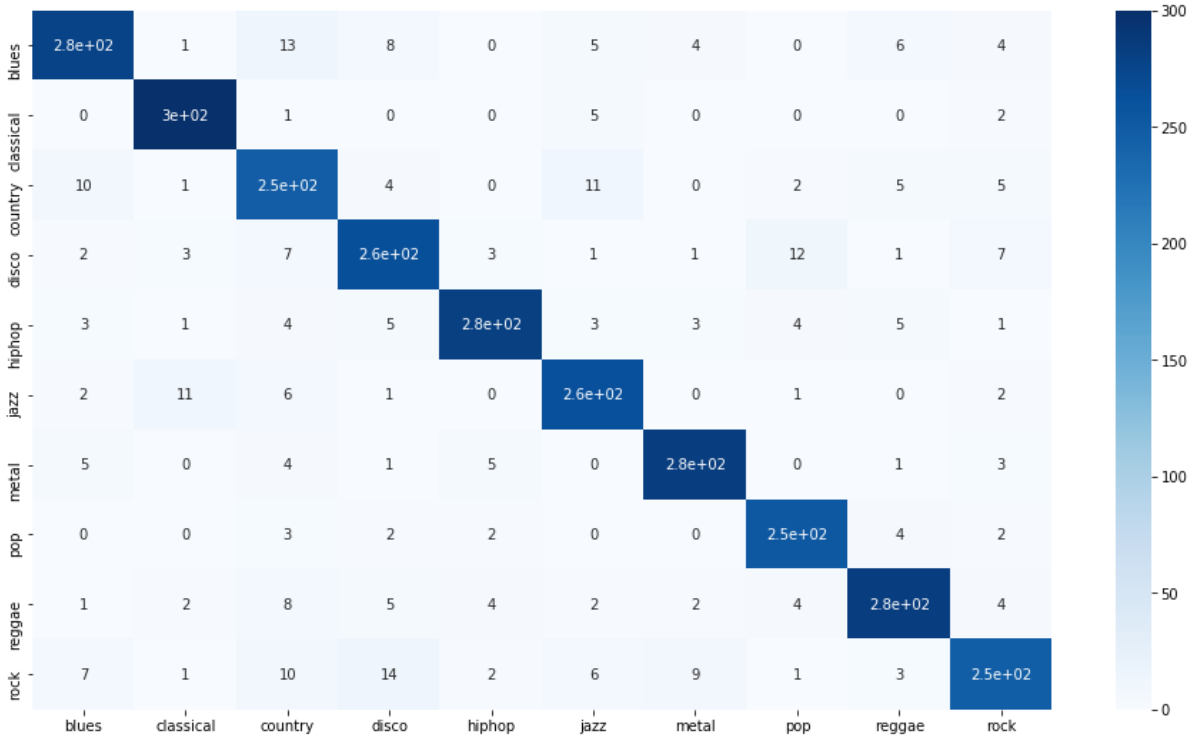


Figure 5: Confusion matrix of the XGBoost model

Chapter 5: GTZAN with Noise

Mathematical Preprocessing Techniques used for Noisy GTZAN

Discrete Fourier Transform

The discrete Fourier transform is used to convert data vector to its sine and cosine components i.e. it is used to decompose a function which depends on space or time into functions depending on spatial or temporal frequency.

Given a vector of data:

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

For each data point f_k (above), we can compute a vector fourier coeff represented by

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_n \end{bmatrix}$$

The formula for the k[th] vector frequency:

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j e^{-\frac{i2\pi jk}{n}}$$

The formula to convert Fourier transform coeff back to data:

$$f_k = \sum_{j=0}^{n-1} \hat{f}_j e^{\frac{i2\pi jk}{n}}$$

Observe that DFT can be represented as the product of a specific data point to an integral power of ω_n; where ω_n is given by:

$$e^{-\frac{2\pi i}{n}}$$

Based on this observation DFT on a large vector dataset can be represented as a matrix multiplication system:

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

Since the matrix consists of complex numbers, therefore the resulting Fourier coefficients are complex valued. The magnitude of the kth coefficients gives the magnitude of the kth frequency sine and cosine waves and the phase values represent the phase between cosine and sine.

Fast Fourier Transform

It is one of the most widely used algorithm in use today. It is used for image compression, digital compression, high performance scientific computing etc.

The FFT algorithm is the go to algorithm when trying to compute DFT.

DFT calculation is an order $O(n^2)$ operation. This makes it a very computationally expensive computation especially when used in image processing or audio processing.

The fast Fourier transform is a computationally efficient implementation of DFT that scales to very large dataset. It is an order $nlog(n)$ algorithm.

This difference in complexity becomes significant for a ten second audio clip sampled at 44 kHz. In this case $n = 4.4 * 10^5$ then DFT will require 10^{11} multiplications while FFT requires only 10^6 multiplications.

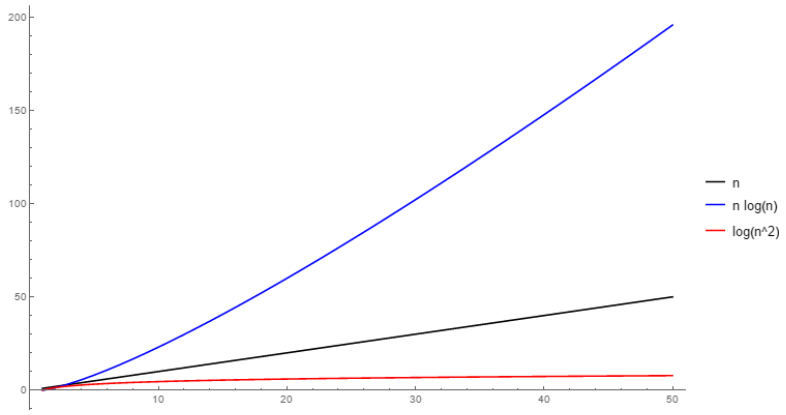


Figure 6: Growth curve of different complexities

FFTs are used for computing derivatives, denoising data, data analysis, image and audio compression etc.

The FFT uses the a special technique which works for values of n that are positive integral power of two. If n is a power of 2 then the FFT can be written as:

$$\hat{f} = F_n f = \begin{bmatrix} I_{\frac{n}{2}} & -D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{\frac{n}{2}} & 0 \\ 0 & F_{\frac{n}{2}} \end{bmatrix} \begin{bmatrix} f_{even} \\ f_{odd} \end{bmatrix}$$

$$where\ D_{\frac{n}{2}} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \omega^{\frac{n}{2}} \end{bmatrix}$$

The aforementioned formula can be recursively repeated until a simple 2 by 2 matrix is formed. The fast Fourier transform is based on this fundamental observation of symmetry.

Even if n is not an whole power of 2, the matrix can be modified by simply adding zero padding.

Short Term Fourier Transform:

The Short-Term Fourier Transform is implemented using the fast Fourier transform. It was developed to adapt DFT to a specific window. In order to adapt DFT to specific windows, STFT uses convolution windows.

Algorithm for STFT:

i. Given a signal, we select a window of time. This window is then assumed to be a complete signal with attenuated edges.

ii. Fourier transform of the windowed signal is calculated using FFT

iii. The power spectrum of FFT output is calculated.

iv. The power spectrum is inverted and colored such that the end result is a bar with 2 features:

    a. frequency represented by height

    b. Power represented by color

A graph comprised of STFTs calculated by moving the window over the entire em-wave gives a spectrogram. The STFT is used to analyse how the frequency content of a non-stationary signal changes over time.

In order to compensate for the attenuation at the edges, a nonzero overlap length can be used in the window function. The magnitude squared of the stft yields the spectrogram representation of the power spectral density of the function.

Spectrogram Representation of audio signals

A spectrogram displays the strength of a signal over time at a waveforms various frequencies. Spectrogram can either be two dimensional with color representing the third dimension or it can be 3 dimensional with color representing the fourth dimension.

A spectrogram is generated by dividing the audio signal into equal segments. Fast Fourier transform is applied on each segment. The spectrum is created by combining the processed output of each segment.

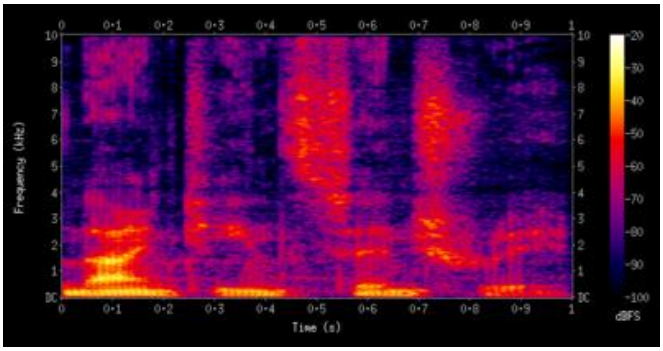


Figure 7: Spectrogram of words "nineteenth century"

The size of the spectrogram depends on the number of STFT outputs which depends on the size of the frame. Therefore, it is possible to define a spectrogram covering 10 hours with only 10 FFT frames. The drawback of skipping too many frames is that there are too many gaps between FFT analysis. This could lead to potential loss of useful information.

Spectrograms are used in a wide variety of applications, these include:

1. study of phonetics and speech synthesis
2. spectrogram used with RNN for speech recognition
3. USGS use spectrogram for seismic study
4. Used for development of RF and microwave systems

Mel-scaled Spectrogram

The mel-scaled spectrogram is the most widely used spectrogram type for machine learning research in the audio domain.

Consider two samples, each containing two notes.

1. Sample 1: C2(65 Hz) – C4(262 HZ)
2. Sample 2: G6(1568 Hz) – A6(1760 HZ)

Mathematically speaking the audio notes in both samples are 202 Hz apart. However when we listen to sample 1, there is a significant difference in the between the notes whereas in sample 2 the notes sound similar.

This experiment demonstrates that the human audio pitch perception is non-linear i.e. humans perceive frequency logarithmically.

The Mel-scaled is a log scale in which equal distances on the scale have same "perpetual" distance. It is based on the following empirically determined formula:

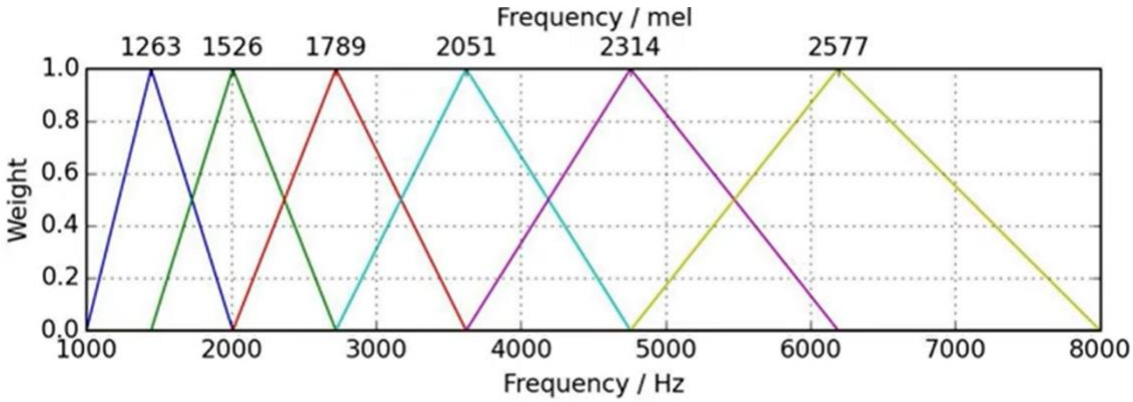
$$m = 2595 \cdot \log\left(1 + \frac{f}{500}\right)$$

$$f = 700\left(10^{\frac{m}{2595}} - 1\right)$$

In order to generate convert frequency to Mel-scale, the first step is to select the number of Mel bands. On the basis of these bands a Mel filter bank is generated. This Mel filter bank is then applied to spectrogram.

The number of Mel bands varies from problem to problem and therefore has to be empirically determined.

Fig.:



Graph of Mel filter banks

Algorithm for generating Mel spectrogram:

i. Divide audio into segments

ii. Extract STFT

iii. Convert amplitude to DBs

iv. Convert frequency to Mel scale

v. Combine to generate Mel spectrogram

Chapter 6: Transformers

Transformers were proposed in the 2017 paper Attention is All You Need by a team from Google Brain lead by A. Vaswani. At the time, the authors intended to use it exclusively for natural language processing.

A recent paper ***An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*** builds on the success of transformers in NLP domain and proposes their use in place of Convolutional Neural Networks.

The paper suggests that transformer can capture sequential and contextual data which is not possible when using a convolution network. Although there are works that show that a combination of CNNs and RNNs (simple, GRU, LSTM) **may sometimes** be able to capture this information, RNN create a performance bottleneck due to their limited parallelizability.

Since the model relies on spectrograms for classification, it was theorised that the addition of sequential and contextual data would improve model performance by making it more stable and making the train test performance similar.

Transformer Architecture

Input Embedding:

- It is a lookup table containing learned vector representation of each word.
- Each input word maps to a vector with continuous values to represent that word.

Positional Encoding:

- This is used to encode data about the relative position of every part of the sequence.
- For every odd time step a vector is created using the following function

$$\cos\left[\frac{Pos}{10000^{2i/d_{model}}}\right]$$

- For every even time step a vector based on the following function is generated

$$\sin\left[\frac{Pos}{10000^{2i/d_{model}}}\right]$$

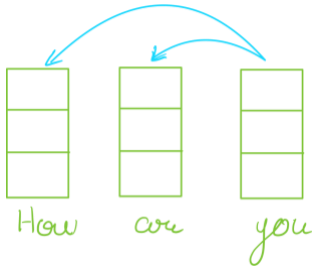
Multi-Head Attention:

In the encoder sequence, multi-head attention applies a specific attention mechanism called self-attention.

Self-attention learns the relevance/dependence of an input sequence to other sequences.

Consider the following input sequence:

Based on the adjacent sequence the model might learn that this pattern resembles a question and 'are' is paired with 'you'.



Self-Attention:

In order to achieve self-attention, the input is feed to three distinct fully connected layers that generate query-key-value vectors.

This concept is derived from retrieval systems and can be understood with the following example.

Consider a search request:

Query: The search itself is the query

Key: Each query is mapped against a set of keys.

Value: These keys will be associated to (result) candidates which are termed values

Implementation of Query, Key and Values:

1. The query and key vectors undergo a dot product matrix multiplication to create a score matrix.
2. The score matrix determines how much focus should a word be but on other words.
3. Each word will have a score that corresponds to other words in the time step.

the higher the score the more the focus

In multi-head attention the query, keys and values are split and go through the self-attention process. Each self-attention is called a head and the output of each head is concatenated into a vector.

The concatenated vector is fed to a linear layer to generate outfit. In theory each head should learn something different thereby giving the encoder model more representation tower.

Residual Connection, layer normalization and feedforward:

- The outbid of multi head attention is added to the input to form a residual connection [for same the reason].
- This goes through a layer normalization before being fed to a Point Wise Feed Forward network for further process.
- The PWFF network are a couple of linear layers with ReLU in between.
- The output of PWFF is again added to the input and normalized.

Function of every step:

- The residual connections help the network train by allowing gradients to flow through the networks directly.
- The layer normalizations are used to stabling the network which results in substantial training time reduction.
- The PWFF layer are used to further from the attention output to get a richer representation.

Developing a SOTA model capable of music genre classification in noisy environment:

Music genre classification is a challenging task due to it variety. Various methods of classification have been proposed over the years. Recent research has focused on two different approaches:

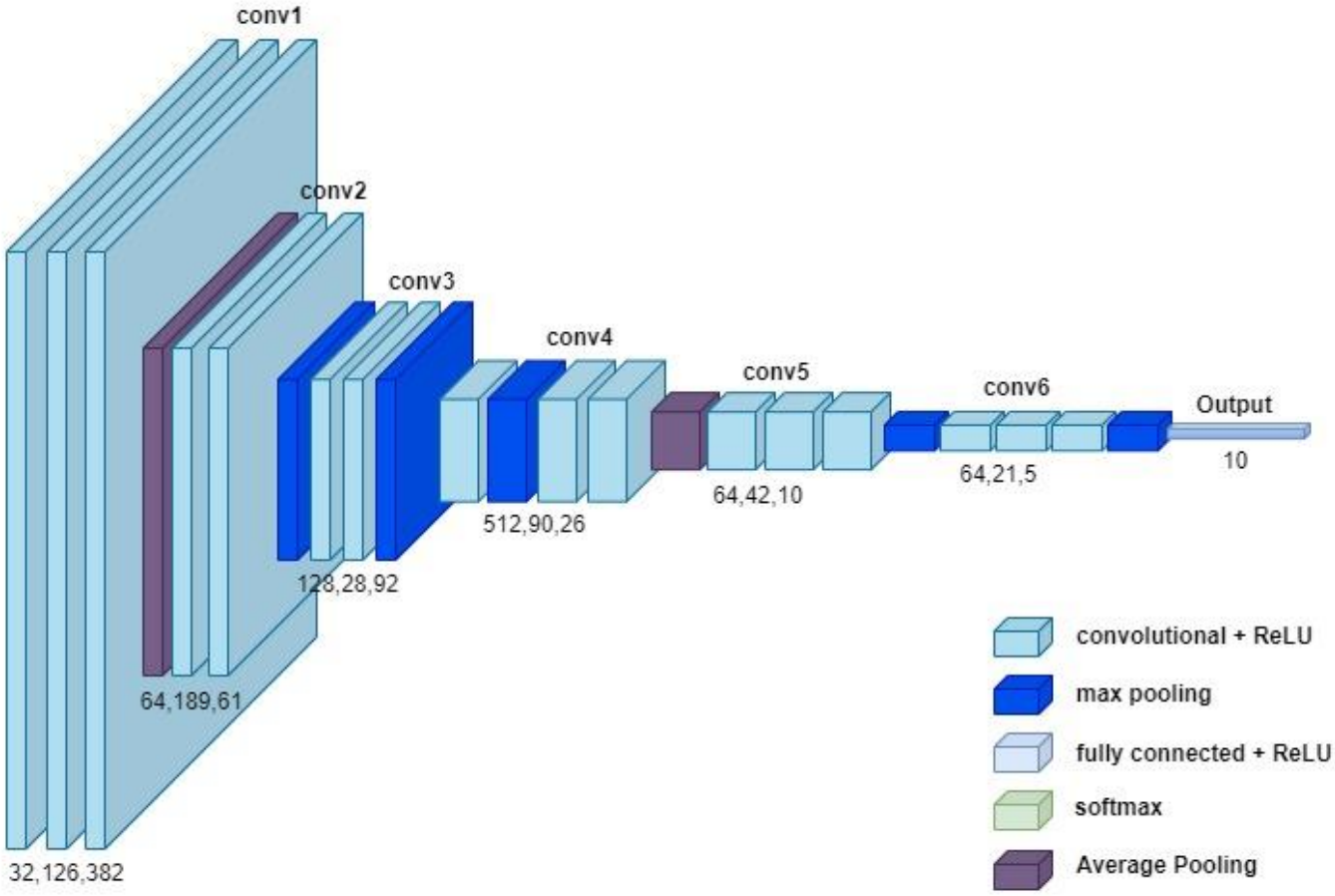
1. The use of feature engineering and feature extraction to represent the major features in a tabular format and using classical machine learning tools on it.
2. The use of deep learning either on the audio data directly or on a simple spectrogram generated by the audio data.

This project proposes the use of Mel-scaled spectrogram with a new application specific neural network designed for the purpose of music genre classification.

The audio data is first preprocessed using highly compute and memory efficient mathematical algorithm, next a Mel-scaled spectrogram is generated and stored as image. This image is then sent on to the model for analysis.

The model development process is empirical and therefore, in the interest of time, only the model selected on the basis of accuracy, bias and compute efficiency is described in the project report.

Proposed SOTA Model Architecture:



Model Details for Noisy GTZAN

Adam Optimization:

1. Adaptive Moment Estimation (Adam) is a method for computation of parameter wise adaptive learning rates.
2. Adam optimizer uses a combination of exponentially decaying average of the past gradients and past squared gradient.
3. The formula for exponentially decaying average of the past gradients

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

4. The formula for exponentially decaying average of past squared gradients

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

5. In the formulae mentioned above m_t and v_t represent the first mean and the uncentered variance.
6. Research has demonstrated that m_t and v_t have bias towards zero if they are initialized as a zero vector. This is specially significant when the decay rates are small i.e. in the initial phase.
7. The researcher proposed bias correction by computing the first and second moment estimates:

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \; ; \; \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

8. Adam is one of the most useful and versatile optimization algorithms to date.

Categorical Cross-entropy:

The loss function plays an essential role in machine learning. It acts as an objective function for the optimization algorithm used for determining optimal weights and biases for a model.

The most commonly used loss function for multi-class classification is categorical cross-entropy. The categorical cross-entropy comes from the field of information theory and is based on the principle of entropy and distance between probability distributions.

The cross entropy loss function is given by the following function:

$$Loss_{CE} = -\sum_{i=1}^{\#(classes)} P_i \log(q_i)$$

For the cross entropy function to work properly, the sum of output (q_i) should sum up to exactly 1. This can be ensured by using a SoftMax layer after the value.

Demonstration of cross entropy:

Let $\vec{a}$ represent output of a model and let $\vec{z}$ represent the ground truth labels, then:

$$\underset{\vec{a}}{\begin{bmatrix}0.2\\1.5\\0.1\\0.4\end{bmatrix}} \rightarrow \underset{Softmaxed}{\begin{bmatrix}0.147\\0.540\\0.133\\0.180\end{bmatrix}} \quad and \quad \underset{\vec{z}}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}$$

Then the cross entropy is calculated as:

$$L = -0 \times \log(0.147) - 0 \times \log(0.540) - 0 \times \log(0.133) - 0 \times \log(0.180) \approx 2.9$$

The difference between standard categorical cross entropy and sparse categorical cross entropy is the format in which the ground truth labels are passed to the function.

In case of categorical cross entropy, the ground truth labels are one hot encoded. For example in case of three class classification the expected format would be $[1,0,0], [0,1,0]\ and\ [0,0,1]$.

In case of sparse categorical cross entropy, the ground truth labels are integers. For example in case of three class classification the expected format would be $[1], [2]\ and\ [3]$.

In the interest of memory, this project uses *sparse* categorical cross-entropy.

Proposed Model at a Glance:

Total params: 1,404,202

Trainable params: 1,404,202

Optimizer: Adam

Loss: Sparse Categorical cross-entropy

Model Training accuracy: 99.91%

Model Test accuracy (700 unseen balanced examples)**:** 96.66 %

Chapter 7:

Comparison with pre-existing research:

The model outperforms all pre-exiting works (with and without noise) to the best of author's knowledge. This improvement in performance can be attributed to the use of mathematical techniques used for preprocessing and the development of a custom model.

The following is a table comparing the results of the model performance with previously known results:

Studies (Result on a variety of different datasets)	**Validation accuracy, %**
Tzanetakis and Cook	**79.5**
Li and Tzanetakis	**74.0**
Holzapfel and Stylianou	**63.5**
Shin et al.	**84.5**
Elbir and Aydin	**66.0**
Proposed **Transformer** models (train 95%)	**94.0**
Model Proposed in this project on noisy data	**95.6**

Table 1: Model Performance Comparison Table

Future extension of the work:

The current project is two be used a proof of concept. The pipeline can have tighter integration with the model. For this, a lot of development on the servers need to be done. While the performance of the model is already significantly better than all the existing systems, it can be optimized for ASICs using quantization and other such techniques. In addition to this the complexity can be increase by increasing the amount of information given by the model and adding a built-in recommender system. Finally, there is scope for an end-to-end model which can perform the mathematical pre-processing in a more efficient manner.

Bibliography:

1. Tzanetakis, G., Cook, P.: ‘Musical genre classification of audio signal’, IEEE Trans. Speech Audio Process., 2002, 10, (3), pp. 293–302 (https://doi/org/10.1109/TSA.2002.800560)
2. Mandel, Michael I., and Daniel PW Ellis. "Song-level features and support vector machines for music classification." (2005): 594-599.
3. Wyse, Lonce. "Audio spectrogram representations for processing with convolutional neural networks." arXiv preprint arXiv:1706.09559 (2017).
4. Bahuleyan, Hareesh. "Music genre classification using machine learning techniques." arXiv preprint arXiv:1804.01149 (2018).
5. Convolutional Neural Network Benchmarks: https://github.com/jcjohnson/cnn-benchmarks
6. Balci, B., Saadati, D., & Shiferaw, D. (2017). Handwritten Text Recognition using Deep Learning. http://cs231n.stanford.edu/reports/2017/pdfs/810.pdf
7. Carbune, V., Gonnet, P., Deselaers, T., Rowley, H. A., Daryin, A., Calvo, M., … Gervais, P. (2020, January 24). *Fast Multi-language LSTM-based Online Handwriting Recognition*. arXiv.org. https://arxiv.org/abs/1902.10525.
8. Harris, C., Millman, S., Gommers, P., Cournapeau, E., Taylor, J., Berg, N., Kern, R., Picus, S., Kerkwijk, M., Haldane, J., Wiebe, P., Gérard-Marchant, K., Reddy, T., Weckesser, H., & Gohlke, T. (2020). Array programming with NumPy. *Nature, 585, 357–362.*

9. John D. Hunter. **Matplotlib: A 2D Graphics Environment**, Computing in Science & Engineering, **9**, 90-95 (2007), DOI:10.1109/MCSE.2007.55 (publisher link)
10. Wes McKinney. **Data Structures for Statistical Computing in Python**, Proceedings of the 9th Python in Science Conference, 51-56 (2010)
11. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch.
12. Bahuleyan, Hareesh. "Music genre classification using machine learning techniques." arXiv preprint arXiv:1804.01149 (2018).
13. Ignatius Moses Setiadi D.R., Satriya Rahardwika D., Rachmawanto E.H., Atika Sari C., Irawan C., Kusumaningrum D.P., Nuri, Trusthi S.L. **Comparison of SVM, KNN, and NB Classifier for Genre Music Classification based on Metadata**
14. Li T, Ogihara M, Li Q. A Comparative study on content-based music genre classification. In: Proceedings of the 26th annual international ACM SI-GIR conference on research and development in information retrieval. Toronto: ACM Press; 2003. p. 282–9.
15. Antonio Jose Homsi Goulart, Rodrigo Capobianco Guido, Carlos Dias Maciel, **Exploring different approaches for music genre classification, Egyptian Informatics Journal** (https://doi.org/10.1016/j.eij.2012.03.001)