

# **STUDY OF MUSIC GENRE CLASSIFICATION BASED ON PATTERN RECOGNITION TECHNIQUE**



*Subhrasalil Bhattacharya(17UCS110)*  
*Shibam Datta(17UCS006)*  
*Saurav Shukla(17UCS108)*  
*Paidimukkala Padmaja(17UCS060)*

**COMPUTER SCIENCE & ENGINEERING DEPARTMENT  
NATIONAL INSTITUTE OF TECHNOLOGY,AGARTALA  
INDIA-799046  
APRIL, 2021**

# **STUDY OF MUSIC GENRE CLASSIFICATION BASED ON PATTERN RECOGNITION TECHNIQUE**

*Report submitted to  
National Institute of Technology, Agartala  
for the award of the degree  
of  
Bachelor of Technology*

*by  
Subhrasalil Bhattacharya(17UCS110)  
Shibam Datta(17UCS006)  
Saurav Shukla(17UCS108)  
Paidimukkala Padmaja(17UCS060)*

*Under the Guidance of  
Dr. Dwijen Rudrapal  
Asst. Professor, CSE Department, NIT Agartala, India*



**COMPUTER SCIENCE & ENGINEERING DEPARTMENT  
NATIONAL INSTITUTE OF TECHNOLOGY AGARTALA  
APRIL, 2021**

## **Dedicated To**

To our Project Supervisor Dr. Dwijen Rudrapal, Assistant Professor, CSED, NIT Agartala for sharing his valuable knowledge, encouragement and showing confidence on us all the time. Each of the faculties of the department to contribute in our development as a professional and help us to achieve this goal.

To all those people who have somehow contributed to the creation of this project and who have supported us.

***“Machine learning will automate jobs that most people thought could only be done by people. ”***

**-Dave Waters**

# REPORT APPROVAL FOR B.TECH

This report entitled “*Study of Music Genre Classification based on Pattern Recognition Technique*”, by Subhrasail Bhattacharya(17UCS110), Shibam Datta(17UCS006), Saurav Shukla (17UCS108), Paidimukkala Padmaja(17UCS060), is approved for the award of *Bachelor of Technology* in *Computer Science and Engineering*.



---

Dr. Dwijen Rudrapal

(Project Supervisor)

Assistant Professor

Computer Science and Engineering Department

NIT, Agartala

---

Dr. Mrinal Kanti Debbarma

(Head of the Department)

Associate Professor

Computer Science and Engineering Department

NIT Agartala

Date:27-04-21

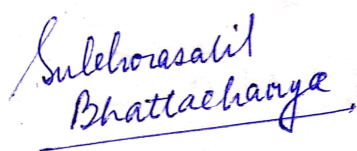
Place:NIT, Agartala

# DECLARATION

We declare that the work presented in this report proposal titled “*Study of Music Genre Classification based on Pattern Recognition Technique*”, submitted to the Computer Science and Engineering Department, National Institute of Technology, Agartala, for the award of the ***Bachelor of Technology*** degree in ***Computer Science and Engineering***, represents our ideas in our own words and where others’ ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

APRIL, 2021

Agartala



---

Subhrasail Bhattacharya  
(17UCS110)




---

Paidimukkala Padmaja  
(17UCS060)



---

Shibam Datta (17UCS006)



---

Saurav Shukla (17UCS108)

# CERTIFICATE

It is certified that the work contained in the report titled “*Study of Music Genre Classification based on Pattern Recognition Technique*”, by Subhrasalil Bhattacharya(17UCS110), Shibam Datta(17UCS006), Saurav Shukla(17UCS108), Paidimukkala Padmaja(17UCS060), has been carried out under my supervision and this work has not been submitted elsewhere for a degree.



---

Dr. Dwijen Rudrapal  
(Project Supervisor)  
Assistant Professor

Computer Science and Engineering Department  
NIT Agartala

---

Dr. Mrinal Kanti Debbarma  
(Head of the Department)  
Associate Professor

Computer Science and Engineering Department  
NIT Agartala

---

## Acknowledgement

---

We would like to take this opportunity to express our deep sense of gratitude to all who helped us directly or indirectly during this thesis work.

Firstly, we would like to thank our supervisor, **Dr. Dwijen Rudrapal**, for being a great mentor and the best adviser we could ever have. Her advise, encouragement and critics are source of innovative ideas, inspiration and causes behind the successful completion of this report. The confidence shown on us by her was the biggest source of inspiration for us. It has been a privilege working with her from last one year.

We are highly obliged to all the faculty members of Computer Science and Engineering Department for their support and encouragement. We also thank **Prof.(Dr.) Harish Kumar Sharma**, Director, NIT Agartala and **Dr. Mrinal Kanti Debbarma**, H.O.D, CSED for providing excellent computing and other facilities without which this work could not achieve its quality goal.

**- Subhrasalil Bhattacharya**

**- Shibam Datta**

**- Saurav Shukla**

**- Paidimukkala Padmaja**



---

## List of Figures

---

1.1	Overview . . . . .	6
2.1	System architecture for overall system . . . . .	13
2.2	System Architecture for Web App . . . . .	15
2.3	Use case diagram for overall system . . . . .	17
2.4	Use case diagram for the classifier . . . . .	17
2.5	Data flow diagram for overall system . . . . .	18
2.6	Flow chart diagram of feature extraction . . . . .	20
2.7	The flow chart diagram for classifier . . . . .	21
2.8	Flow chart diagram for Web App . . . . .	22
2.9	Sequence diagram for overall system of music genre classification . . . . .	24
4.1	Logistic Regression Example . . . . .	54

4.2	Linear Separable Features . . . . .	58
4.3	Front face of Web App . . . . .	62
4.4	Upload the music . . . . .	63
4.5	Finding the genre . . . . .	63
4.6	Finding multiple genres . . . . .	64

---

## List of Tables

---

1	Hardware requirements . . . . .	11
2	Software requirements . . . . .	11

---

## Abstract

---

Music is categorized into subjective categories called genres. With the growth of the internet and multimedia systems applications that deal with the musical databases gained importance and demand for Music Information Retrieval (MIR) applications increased. Musical genres have no strict definitions and boundaries as they arise through a complex interaction between the public, marketing, historical, and cultural factors. This observation has led some researchers to suggest the definition of a new genre classification scheme purely for the purposes of music information retrieval. Genre hierarchies, typically created manually by human experts, are currently one of the ways used to structure music content on the Web. Classifying the music according to their genre is indeed a challenging task. Automatic musical genre classification can potentially automate this process and provide an important component for a complete music information retrieval system for audio signals. Previous research on music genre classification systems centred primarily on the use of timbral characteristics, which restricts the output. In this study, we have used various machine learning algorithms to classify the music based on their genre. In machine learning, we have used the SVM classifier, Logistic Regression and K-Nearest Neighbour (KNN) classifier for the task of genre classification. These algorithms are prevalent in the task of classification. Our work compares the accuracy of these machine learning classification algorithms of which SVM(Kernel=Poly) Classifier has the highest accuracy and has been implemented through a Django Web App to show a running real time example.

This Project aims to test different Machine Learning algorithms in predicting the genre of Tunes and peruse their performance so that data-driven decisions can be taken based on the results. The primary objective of the endeavour is to figure out a suitable algorithm which can be implemented in connection to a fully-fledged system which is capable of making recommendations in real-time with good accuracy resulting to a smart model working smoothly in the back-end.

As a finished product of the aforesaid initiative, it is expected to have successfully deployed a smoothly running portal accepting audio and video tracks, feeding it in the model implemented in the back-end and returning the genre of the particular Music with accuracy.

---

## Contents

---

<b>Acknowledgement</b>	<b>viii</b>
<b>Abstract</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction to Music Genre Classification . . . . .	1
1.2 FEATURES . . . . .	2
1.2.1 Mel-Frequency Cespstral Coefficients (MFCCs) . . . . .	2
1.3 Multiclass Learning Methods: . . . . .	5
1.4 Motivation . . . . .	6
1.5 Problem statement . . . . .	7
1.6 Scope of the Project . . . . .	7
1.7 Objectives . . . . .	7

1.8	Literature Survey . . . . .	8
1.9	Organisation of project . . . . .	9
1.10	Software requirement specification . . . . .	10
1.11	Specific requirements . . . . .	10
1.12	Functional requirements . . . . .	11
1.13	Hardware requirements . . . . .	11
1.14	Software requirements . . . . .	11
1.15	Summary . . . . .	11
<b>2</b>	<b>METHODOLOGY</b>	<b>12</b>
2.1	Flow Diagram . . . . .	12
2.2	Design considerations . . . . .	12
2.3	Architecture . . . . .	13
2.3.1	Overall System Architecture . . . . .	13
2.3.2	Web application architecture . . . . .	15
2.4	Module Specification . . . . .	15
2.4.1	Sound Clip Selection Module . . . . .	16
2.4.2	Feature Extraction Module . . . . .	16
2.4.3	Classification Module . . . . .	16
2.5	Specification using Use Case Diagram . . . . .	16
2.6	Data flow diagram . . . . .	18
2.7	System Design . . . . .	19

---

2.8	Detailed description of music genre classification based on pattern recognition technique . . . . .	19
2.9	Flow chart of system . . . . .	20
2.10	Flow chart of Web App . . . . .	22
2.11	Sequence diagram for overall system . . . . .	23
2.12	Summary . . . . .	24
<b>3</b>	<b>IMPLEMENTATION</b>	<b>25</b>
3.1	Programming Language Selected . . . . .	25
3.1.1	Python . . . . .	25
3.2	Platform Selected . . . . .	26
3.2.1	Linux . . . . .	26
3.3	Version Control . . . . .	26
3.3.1	Git . . . . .	26
3.4	Prototype . . . . .	26
3.4.1	Feature Extraction . . . . .	26
3.4.2	Principal component . . . . .	27
3.4.3	Dimensionality reduction . . . . .	27
3.4.4	Classification . . . . .	28
3.5	Music Genre Classifier App . . . . .	28
3.5.1	Python package <i>mysvm</i> . . . . .	29
3.6	Pseudocode for each Module . . . . .	31

---



3.6.1	Code for classification in KNN . . . . .	31
3.6.2	Code for Logistic Regression . . . . .	32
3.6.3	Logistic regression Cost function . . . . .	34
3.6.4	Logistic regression Gradient Descent . . . . .	34
3.6.5	Code for feature extraction in SVM . . . . .	35
3.6.6	Code for classification in SVM . . . . .	37
3.6.7	Front End . . . . .	44
3.6.8	Back End . . . . .	47
3.7	Summary . . . . .	52
<b>4</b>	<b>MODEL TESTING AND RESULT ANALYSIS</b>	<b>53</b>
4.1	Logistic Regression . . . . .	54
4.2	K-Nearest neighbour . . . . .	55
4.2.1	How does K-NN work? . . . . .	56
4.3	Support Vector Machine . . . . .	58
4.3.1	SVM (Linear Kernel) . . . . .	59
4.3.2	SVM (RBF Kernel) . . . . .	60
4.3.3	SVM (Polynomial Kernel) . . . . .	61
4.4	Snapshots . . . . .	62
4.5	Summary . . . . .	64
<b>5</b>	<b>CONCLUSION AND FUTURE ENHANCEMENTS</b>	<b>65</b>
5.1	Conclusion . . . . .	65

---

5.2 Future Enhancement . . . . .	66
<b>References</b>	<b>67</b>

# CHAPTER 1

---

## Introduction

---

### 1.1 Introduction to Music Genre Classification

Music genre labels are useful to organize songs, albums, and artists into broader groups that share similar musical characteristics. In this work, an approach to learn and combine multi-modal data representations for music genre classification is proposed. Intermediate representations of deep neural networks are learned from audio tracks, text reviews, and cover art images, and further combined for classification. Experiments on single and multi-label genre classification are then carried out, evaluating the effect of the different learned representations and their combinations. Results on both experiments show how the aggregation of learned representations from different modalities improves the accuracy of the classification, suggesting that different modalities embed complementary information. Moreover, a proposed approach for dimensionality reduction of target labels yields major improvements in multi-label classification not only in terms of accuracy, but also in terms of the diversity of the predicted genres, which implies a more fine-grained categorization. Finally, a qualitative analysis of the results sheds some light on the behavior of the different modalities on the classification task.

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. 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.

Organisations nowadays use music classification, either to be able to place recommendations to their customers or simply as a product. Determining music genres is the first step in that direction. Machine Learning techniques have proved to be quite successful in extracting trends and patterns from the large pool of data. The same principles are applied in Music Analysis also.

We use audio data set which contains 1000 music pieces each of 30 seconds length. There are 100 pieces from each of the following genres: classical (cl), country(co), disco(d), hip-hop(h), jazz(j), rock(ro), blues(b), reggae(re), pop(p), metal(m). Later for our web-app we have chosen six popular genres namely classical, hip-hop, jazz, metal, pop and rock to get more accuracy. We use pattern recognition algorithms with Mel Frequency Cepstral Coefficients (MFCC) [1] as the feature vectors for classification

We have tried different supervised learning algorithms for our classification problem. Input can be in any audio/video format. The final output will be a label from the 6 genres.

## **1.2 FEATURES**

### **1.2.1 Mel-Frequency Cespstral Coefficients (MFCCs)**

The first step in any automatic speech recognition system is to extract features i.e. identify the components of the audio signal that are good for identifying the linguistic content and discarding all the other stuff which carries information like background noise, emotion etc. The main point to understand about speech is that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the phoneme being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

- Steps at a Glance:

Frame the signal into short frames.

For each frame calculate the periodogram estimate of the power spectrum.

Apply the Mel filter bank to the power spectra, sum the energy in each filter.

Take the logarithm of all filter bank energies.

Take the DCT of the log filter bank energies.

Keep DCT coefficients 2-13, discard the rest.

- The Following steps are required to compute MFCC.

1. Pre-emphasis

The goal of pre-emphasis is to compensate the high-frequency part that was suppressed during the sound production mechanism of humans.

2. Frame blocking

The input speech signal is segmented into frames of 20 30 ms with optional overlap of 1/3 1/2 of the frame size. Usually the frame size (in terms of sample points) is equal to power of two in order to facilitate the use of FFT. If this is not the case, we need to do zero padding to the nearest length of power of two. If the sample rate is 16 kHz and the frame size is 320 sample points, then the frame duration is  $320/16000 = 0.02 \text{ sec} = 20 \text{ ms}$ . Additional, if the overlap is 160 points, then the frame rate is  $16000/(320-160) = 100 \text{ frames per second}$ .

3. Hamming windowing

Each frame has to be multiplied with a hamming window in order to keep the continuity of the first and the last points in the frame (to be detailed in the next step). If the signal in a frame is denoted by  $s(n)$ ,  $n = 0 \dots N-1$ , then the signal after Hamming windowing is  $s(n) w(n)$ , where  $w(n)$  is the Hamming window defined by:

$$w(n, a) = (1 - a) - a \cos(2\pi n / (N - 1)) \quad 0 \leq n \leq N - 1 \quad (1)$$

4. Fast Fourier Transform or FFT

Spectral analysis shows that different timbres in speech signals corresponds to different energy distribution over frequencies.

5. Triangular Bandpass Filters

We multiply the magnitude frequency response by a set of 20 triangular bandpass filters to get the log energy of each triangular bandpass filter. The positions of these

filters are equally spaced along the Mel frequency, which is related to the common linear frequency  $f$  by the following equation:

$$mel(f) = 1125 \ln(1 + f/700) \quad (2)$$

Mel-frequency is proportional to the logarithm of the linear frequency, reflecting similar effects in the human's subjective aural perception.

#### 6. Discrete cosine transform or DCT

In this step, we apply DCT on the 20 log energy  $E_k$  obtained from the triangular bandpass filters to have  $L$  mel-scale cepstral coefficients.

Since we have performed FFT, DCT transforms the frequency domain into a time-like domain called quefrency domain. The obtained features are similar to cepstrum, thus it is referred to as the mel-scale cepstral coefficients, or MFCC. MFCC alone can be used as the feature for speech recognition. For better performance, we can add the log energy and perform delta operation, as explained in the next two steps.

#### 7. Log energy

The energy within a frame is also an important feature that can be easily obtained. Hence we usually add the log energy as the 13th feature to MFCC. If necessary, we can add some other features at this step, including pitch, zero cross rate, high-order spectrum momentum, and so on.

#### 8. Delta cepstrum

It is also advantageous to have the time derivatives of (energy+MFCC) as new features, which shows the velocity and acceleration of (energy+MFCC). The equations to compute these features are:

$$Cm(t) = [St = -MM C_m(t+t)/[St = -MMt^2] \quad (3)$$

The value of  $M$  is usually set to 2.

- If we add both the velocity and the acceleration, the feature dimension is 39. Most of the speech recognition systems on PC use these 39-dimensional features for recognition.

## 1.3 Multiclass Learning Methods:

Once the feature set has been extracted, the music genre classification problem is reduced to a multi-class classification problem. The problem can be formally defined as follows: The input to the problem is a set of training samples of the form of  $(x_i, l_i)$ , where  $x_i$  is a data point and  $l_i$  is its label, chosen from a finite set of labels  $c_1, c_2, \dots, c_m$ . In our case the labels are music genres. The goal is to infer a function  $f$  that well approximates the mapping of  $x$ 's to their labels.

- K-Nearest Neighbor (KNN)

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.

- Support vector machines (SVMs)

SVM have shown superb performance in binary classification tasks. Basically, Support Vector Machine aim at searching for a hyperplane that separates the positive data points and the negative data points with maximum margin. To extend SVMs for multi-class classification, we use one-versus-the-rest, pairwise comparison, and multi-class objective functions.

- Logistic Regression

Logistic regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values). Logistic regression uses an equation as the representation, very much like linear regression.

Input values ( $x$ ) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value ( $y$ ). A key difference from linear regression is that the output value being modeled is a binary value (0 or 1) rather than a numeric value. Below is an example logistic regression equation:

$$y = \hat{e}(b_0 + b_1x)/(1 + \hat{e}(b_0 + b_1x)).....(4)$$

Where  $y$  is the predicted output,  $b_0$  is the bias or intercept term and  $b_1$  is the coefficient for the single input value ( $x$ ). Each column in input data has an associated  $b$  coefficient

(a constant real value) that must be learned from training data.

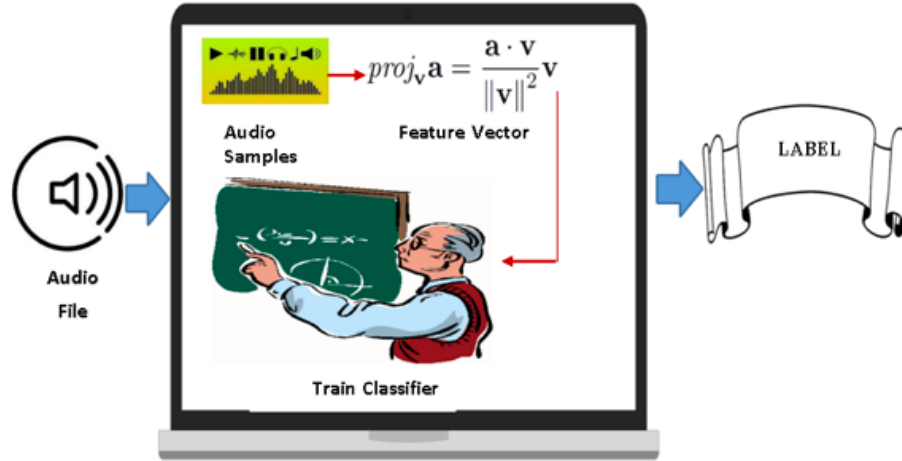


Figure 1.1: Overview

Our attempt is to classify music based on genres, which is a standard problem in Music Information Retrieval (MIR) research. We will be using different supervised learning algorithms along with multiple feature vectors for our classification problem. We have used a Dataset from GTZAN Genre Collection [2] which contains 1000 music pieces each of 30 seconds length. For any given music file, our attempt is to train a classifier which will use the extracted features, and output a label. The overall idea is described in the figure 1.1.

## 1.4 Motivation

Genre classification is a standard problem in Music Information Retrieval research. Most of the music genre classification techniques employ pattern recognition algorithms. The features are extracted from short-time recording segments. Commonly used classifiers are Support Vector Machines (SVMs), Nearest-Neighbor (NN) classifiers, Gaussian Mixture Models, Linear Discriminant Analysis (LDA), etc.

Our attempt is to provide an application that can accept any audio/video file and output a genre label from it. For training the classifier we have used GATZAN Genre Collection dataset from MARSYAS [2].



## **1.5 Problem statement**

Classifying a song to a genre, although an inherently subjective task, comes quite easily to the human ear. Within seconds of hearing a new song one can easily recognize the timbre, distinct instruments, beat, chord progression, lyrics, and genre of the song. For machines on the other hand this is quite a complex and daunting task as the whole “human” experience of listening to a song is transformed into a vector of features about the song. Historically, machines haven’t been able to reliably detect many of these musical characteristics that humans recognize in music.

We are considering a 10-genre classification problem with the following categories: classical (cl), country(co), disco(d), hip-hop(h), jazz(j), rock(ro), blues(b), reggae(re), pop(p), metal(m). Finally, we have chosen six popular genres namely classical, hip-hop, jazz, metal, pop and rock to get more accuracy for our final web-app. We use pattern recognition algorithms with Mel Frequency Cepstral Coefficients (MFCC) as the feature vectors for classification.

## **1.6 Scope of the Project**

- The final Graphical User Interface can be used to identify the genre of any given audio/video file.
- With the help of our classifiers we can label large data set of untagged music.
- The classifiers can be integrated into any application for Music Information Retrieval.

## **1.7 Objectives**

- To extract the features from the given music.
- To classify the input music based on the extracted features and label a class(genre) name to it.

- To quantify the match of any input music to the listed genres.
- To improve upon the accuracy of genre classification.
- To find the multiple genres to which a music belongs to.

## **1.8 Literature Survey**

In an approach proposed by George Tzanetakis for automatic classification of audio signals three feature sets for representing tumbrel texture, rhythmic content and pitch content are proposed. The performance and relative importance of the proposed features is investigated by training statistical pattern recognition classifiers using real-world audio collections. Both whole file and real-time frame-based classification schemes are described. Using the proposed feature sets, classification of 61 percent for ten musical genres is achieved. This result was comparable to results reported for human musical genre classification.

A new feature extraction method for music genre classification were proposed by Tao Li, in 2003. They capture the local and global information of music signals simultaneously by computing histograms on their Daubechies wavelet coefficients. Effectiveness of this new feature and of previously studied features are compared using various machine learning classification algorithms, including Support Vector Machines and Linear Discriminant Analysis.

An approach by Yusuf Yaslan and Zehra Cateltepe in 2006. They examined performances of different classifiers on different audio feature sets to determine the genre of a given music piece. For each classifier, they also evaluated performances of feature sets obtained by dimensionality reduction methods. They used the freely available MARSYAS software to extract the audio features. Finally, they experimented on increasing classification accuracy by combining different classifiers. Using a set of different classifiers, they obtained a test genre classification accuracy of around  $79.6 \pm 4.2$  percent on 10 genre set of 1000 music pieces.

There is no previous work on genre classification measuring the likelihood of different genre-based HMMs (Hidden Markov Models), or bag-of-words lyric features. They also experimented with methods which have not appeared before in genre classification, such as the spectral method for training HMM's, and used Canonical Correlation Analysis (CCA) to combine

features from different domains.

## **1.9 Organisation of project**

The project report is organized as follows

Chapter 1 – Introduction: This chapter gives the brief introduction on music genre classification, features, problem statement and objectives, scope of the project and literature survey.

Chapter 2 – System Requirement Specification: This chapter presents the various software and hardware requirements. It also presents a brief description of the software packages used.

Chapter 3 – Flow Diagram: This chapter presents the system architecture of proposed system and data flow diagram of each module.

Chapter 4 – Detail Design: This chapter briefs about the structural chart diagram and detail functionality and processing of each module.

Chapter 5 – Implementation: This chapter explains about the implementation requirements programming language selection and also coding lines for programming language used in the project.

Chapter 6 – Models and Testing: This chapter deals with the unit testing, experimentations and results for each module.

Chapter 7 – Results and Discussions: This chapter presents snapshots of the Web App.

## 1.10 Software requirement specification

Software Requirement Specification (SRS) basically explains about, a customer or potential client's system requirements and dependencies at a particular point in time prior to any actual design or development work. It involves specific requirements, software and hardware requirements, interfaces.

## 1.11 Specific requirements

The following are the specific requirements for extracting the features, audio manipulation and for the web-app.

- **Django:** Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel.
- **Pydub:** It is a Python library for audio manipulation.
- **NumPy:** NumPy is the fundamental package for scientific computing with Python.
- **python\_speech\_features:** This library provides common speech features for MFCCs and filterbank energies.
- **Scikit-learn:** It is a software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
- **SciPy:** It is an open source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions and FFT and other tasks common in science and engineering.

## 1.12 Functional requirements

The functional requirements for a system describe what the system should do. It describes the system function in detail, its inputs and outputs, exceptions, and so on.

- pythonv3.4

## 1.13 Hardware requirements

Processor	Pentium-IV and higher
Speed	1.1GHz
RAM	4GB
Hard Disk	40GB and above

Table 1: Hardware requirements

## 1.14 Software requirements

Operating System	Linus/Windows
Programming language	pythonv3.4

Table 2: Software requirements

## 1.15 Summary

This chapter gives introduction to the project, brief introduction about music genre classification, the scope of the project, problem statement and objectives of the project, the related paper presented in the literature survey and also finally the organization of the report.

## CHAPTER 2

---

### METHODOLOGY

---

#### **2.1 Flow Diagram**

High level design has a conceptual overview of the system. However, this gives more information for the user to understand the logic. Following are the issues that we see in this part which are the primary components for the design.

#### **2.2 Design considerations**

The key design goals for this system are:

- To design and implement the music genre classification system based on pattern recognition technique.
- Efficient system with probability of getting more accuracy for genres of audio files.

- User friendly scheme with easy to use interface.

## 2.3 Architecture

The System Architecture is a conceptual model that defines the structure, behavior and more views of a system. An architecture description is a formal description and the system architecture.

### 2.3.1 Overall System Architecture

The figure 2.1 shows the system architecture for overall system of music genre classification using pattern recognition technique.

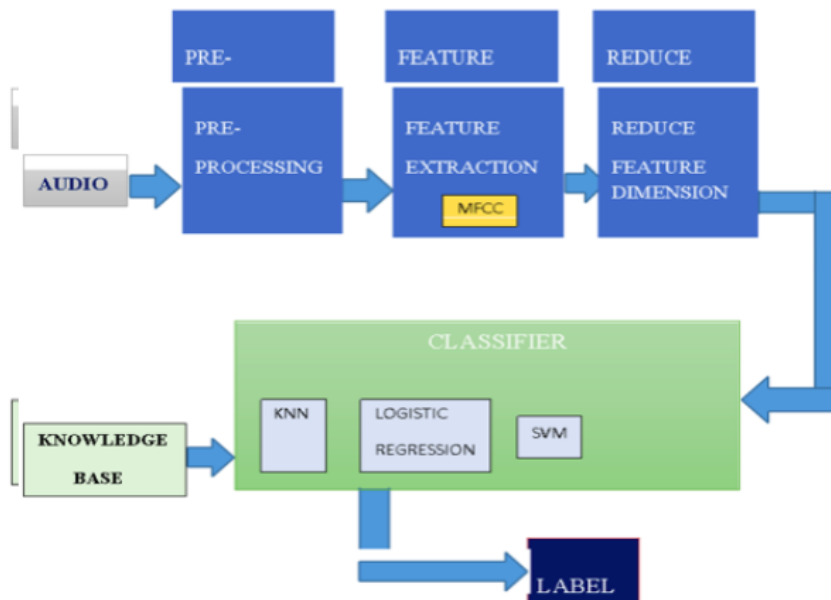


Figure 2.1: System architecture for overall system

## ■ *Dataset*

We have used the GTZAN dataset from the MARYSAS website. It contains 10 music genres; each genre has 1000 audio clips in .au format. The genres are - blues, classical, country, disco, pop, jazz, reggae, rock, metal, hip-hop. Each audio clips have a length 30 seconds, are 22050Hz Mono 16-bit files. The dataset incorporates samples from variety of sources like CDs, radios, microphone recordings etc.

## ■ *Preprocessing*

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

We have done logistic regression, KNN and SVM in python. The final GUI is build using Python Django.

To classify our audio clips, we use feature like Mel-Frequency Cepstral Coefficients. This feature are appended to give a 28 length feature vector. We have used librosa library of python where first we calculated short time Fourier transformation and then applied MFCC considering first 13 coefficients. We took the mean variance of  $13 \times n$  MFCC coefficients, result we get is  $104 \times 1$  matrix or vector of size 104. Then, we used different multi-class classifiers and an ensemble of these to obtain our results as a label.



### 2.3.2 Web application architecture

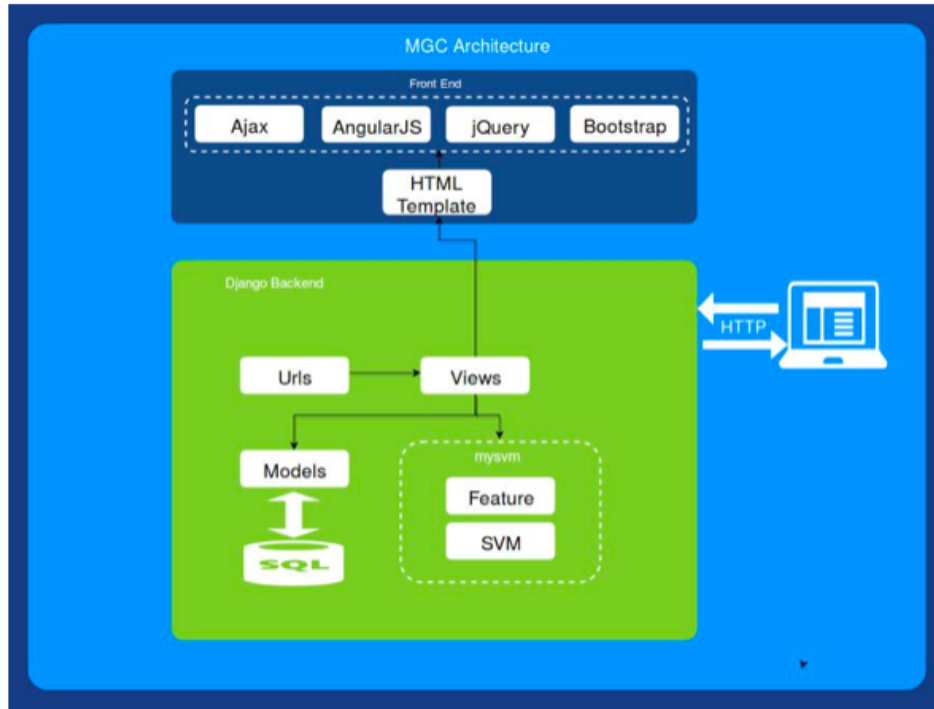


Figure 2.2: System Architecture for Web App

The figure 2.2 shows the Web App architecture explaining the front end and Django backend. The front end consists of Ajax, AngularJS, jQuery and, bootstrap. Ajax is for asynchronous page reload. AngularJS is for front end query. jQuery is a JavaScript library used for event handling and animation. Bootstrap is a responsive front-end framework. The front end sends the HTTP requests to the Django backend where it works with the urls, views and models.

## 2.4 Module Specification

The specification describes the various modules used for the implementation of the project Music Genre Classification Based on Pattern Recognition Technique.

### **2.4.1 Sound Clip Selection Module**

Music information retrieval is a heavy on processor and analyzing the whole waveform of the song can take a good amount of time. So instead of full song, only 30 sec long chunks were taken.

### **2.4.2 Feature Extraction Module**

Feature extraction has been performed in python using librosa library which has primary function of manipulating audio data. The code is executed by various functions on it to extract their features.

### **2.4.3 Classification Module**

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 Neighbor's
- Logical Regression
- SVM
  - Linear Kernel
  - Radial Basis Function (RBF) Kernel
  - Polynomial Kernel

## **2.5 Specification using Use Case Diagram**

A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication associates between the actors and the use cases and generalization among the use case as shown below in figure.

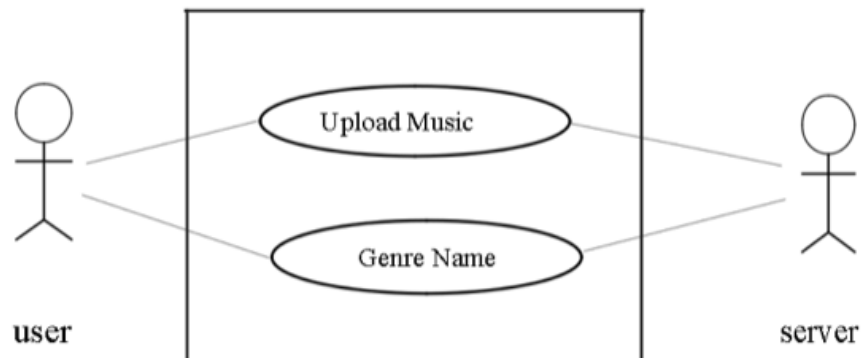


Figure 2.3: Use case diagram for overall system

The figure 2.3 shows the use case diagram for the application. In our use case diagram user uploads the audio file to the server, then the server response with the genre or label to the user.

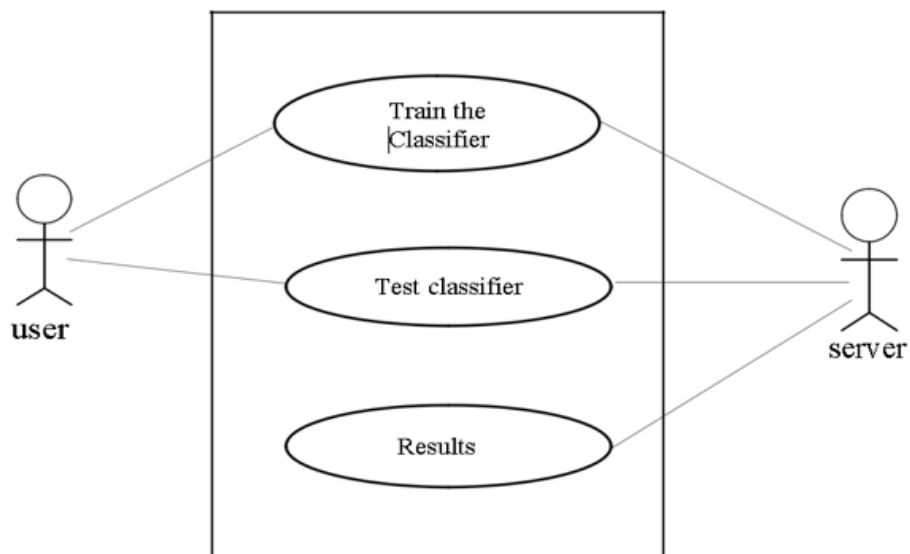


Figure 2.4: Use case diagram for the classifier

The Fig 2.4 shows the use case diagram for the classifier. The above diagram at its simplest is a representation of a user's interaction with the system and depicting the specification of a use case. It deals with training the classifier with the extracted features vectors and combines the

classifiers to improve the accuracy of the genre after that testing the classifier for the result and display it.

## 2.6 Data flow diagram

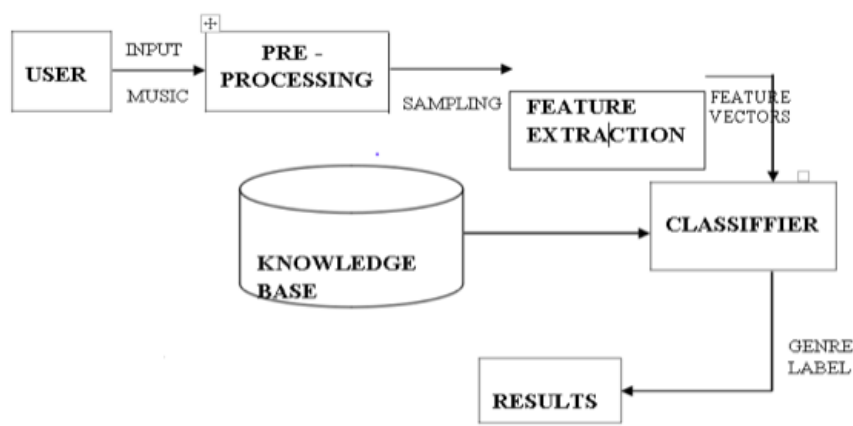


Figure 2.5: Data flow diagram for overall system

The above fig 2.5 shows the data flow diagram for the overall system of music genre classification. We have used the GTZAN dataset from the MARYSAS website. It contains 10 music genres; each genre has 1000 audio clips in .wav format. The genres are - blues, classical, country, disco, pop, jazz, reggae, rock, metal, hip-hop. Each audio clips have a length 30 seconds. Then 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.

To classify our audio clips, we chose features like Mel-Frequency Cepstral Coefficients, and also reduce the feature dimensions Then, we used different multi-class classifiers and an ensemble of these to obtain our results as a genre label.

## **2.7 System Design**

After high level design, a designer's focus shifts to low level design, each module's responsibilities should be specified as precisely as possible. Constraints on the use of its interface should be specified, pre and post conditions can be identified. Module wide in variants can be specified internal structures and algorithm can be suggested.

## **2.8 Detailed description of music genre classification based on pattern recognition technique**

This section describes the functionalities and process of all the modules. Different modules that are implemented for music genre classification based on pattern recognition technique are:

We have taken the audio files of the 100 songs per genre of .wav format. We then read the .wav file into python, and extract the MFCC features for each song. We further reduced this matrix representation of each song by taking the mean vector and covariance matrix of the cepstral features and storing them as a cell matrix, effectively modeling the frequency features of each song as a multi-variate Gaussian distribution. Lastly, we applied both supervised and unsupervised machine learning algorithms, using the reduced mean vector and covariance matrix as the features for each song to train on.

## 2.9 Flow chart of system

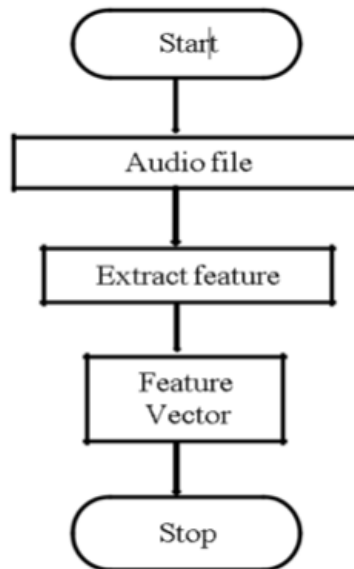


Figure 2.6: Flow chart diagram of feature extraction

The flow chart diagram 2.6 of feature extraction deals with the initial process of extracting features in which, from the given input audio file feature extraction takes place in the form of feature vectors like MFCC, which is to be used in the next module to classify the music by the classifiers.

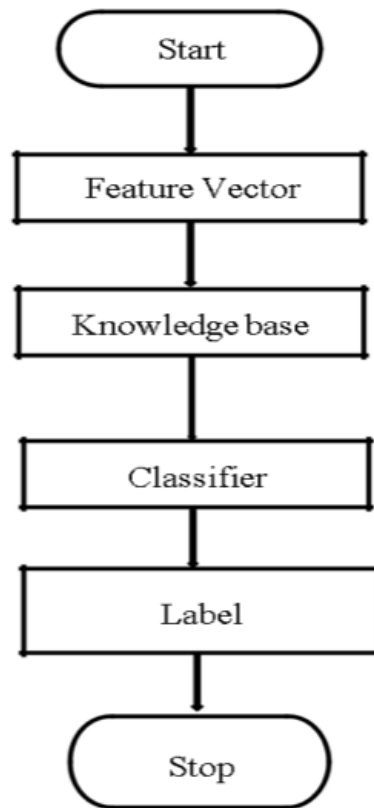


Figure 2.7: The flow chart diagram for classifier

The flow chart diagram of fig 2.7 for classifier deals with the training and testing the different types of classifiers by the feature vectors to produce the output to the user as a label. The different types of classifier that can be used are k-NN, SVM, Logistic regression, etc.

## 2.10 Flow chart of Web App

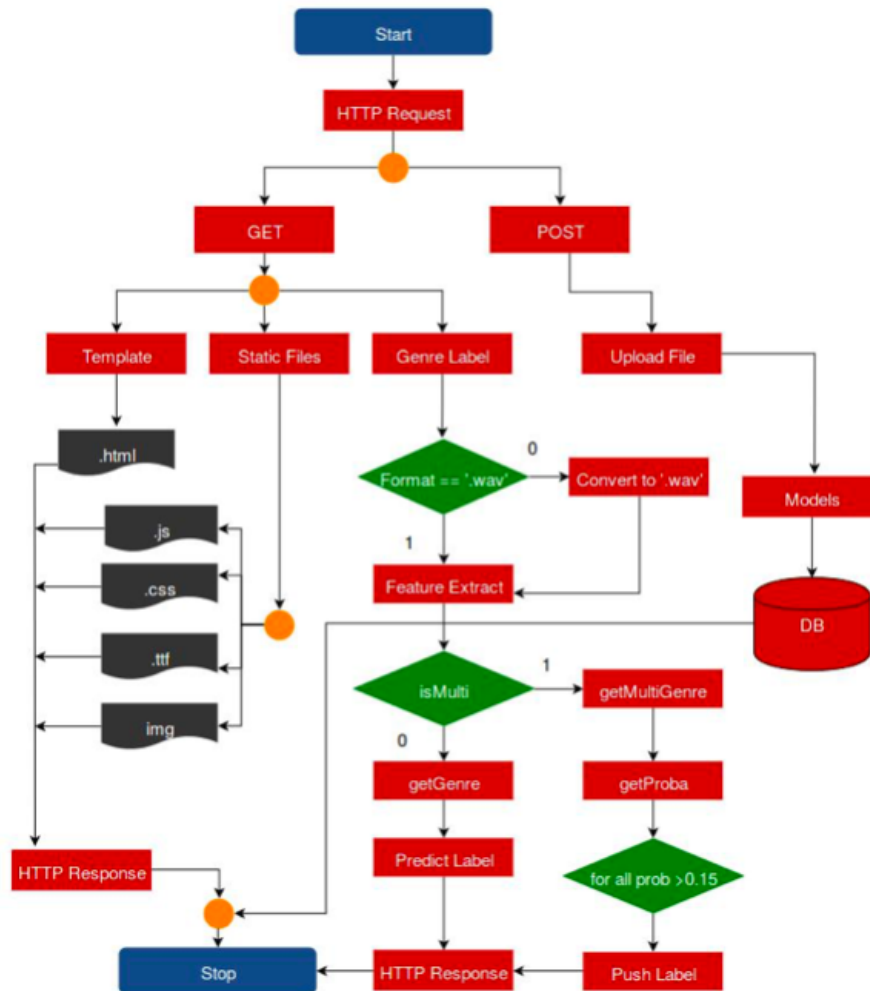


Figure 2.8: Flow chart diagram for Web App

Figure 2.8 illustrates the flow chart of Web App. The browser will send HTTP request as GET or POST from our Web App. GET request can be for loading templates which is an html file or for a static file, which includes front end JavaScript files (js), stylesheets (css), true type fonts (ttf) and image files. User upload the file by a POST request. Our front end java script will convert the given file as BLOB (binary large objects) files of size 1MB before POST. After a successful POST, user can send a GET request for genre label for the uploaded file. We have developed a python package called ‘mysvm’ for extracting features and classifying



music. This package is added to our Web App. On receiving a GET request for genre label, we convert the file to .wav if it is in other format. Then the features are extracted from the audio file using 'mysvm.feature.extract(filename)'. If multi option is selected by the user then 'mysvm.svm.getMultiGenre' function is called. This will get all the probabilities of a genres which the given music belongs to. If the probability is greater than 0.15, we will push the label into the stack of maximum size 3. The labels are sent as JSON data in the response. If single genre label is selected the label which is having highest probability is sent in response.

## 2.11 Sequence diagram for overall system

The sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. A sequence diagram shows object interaction arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between objects needed to carry out the functionality of the scenario.

In our scenario, we have the objects namely user, web app and knowledge base. The lifeline of that an objects begins as follows.

The user uploads the input music and the web app processes the audio and saves it in the knowledge base. Consecutively the web app extracts the feature and classifies it. The genre name is displayed to the user by the web app and additionally the web app scans for the music from other genre interacting with knowledge base. Finally, the knowledge base outputs 10 different music to the user.

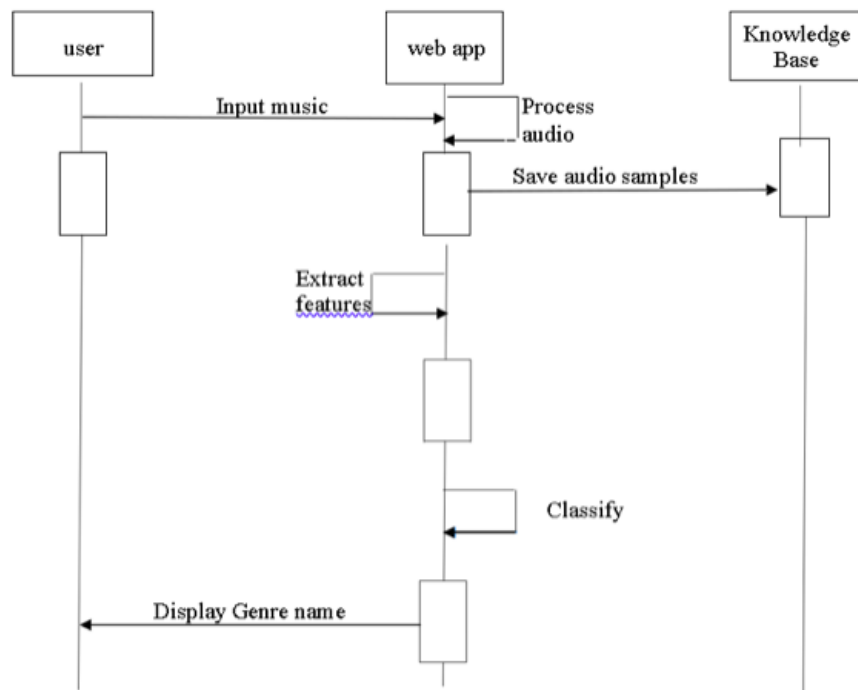


Figure 2.9: Sequence diagram for overall system of music genre classification

## 2.12 Summary

This chapter explains the details design of this project with the flow chart diagram, activity model for each model and sequence diagram. The next chapter clearly explains the implementation part of this project.

## CHAPTER 3

---

### IMPLEMENTATION

---

### 3.1 Programming Language Selected

#### 3.1.1 Python

Python is a high-level, interpreted object-oriented language and is platform independent. There are many machine learning libraries available in Python. Django a framework written in python is well suited for developing web applications. Python has a good community support in field of machine learning and data preprocessing .Libraries like Sklearn and numpy helps in evaluating heavy mathematical calculation and training various complicated Machine Learning and Deep Learning Models .In context of our project along with Sklearn Python also provides Libraries like librosa which provides us with various functions like Mel-frequency Cepstral Coefficients (MFCCs) and Fast Fourier Transformation (FFT) which has been extensively used in our project.

## **3.2 Platform Selected**

### **3.2.1 Linux**

Our prototype is written in python which is platform independent. The Web App is written in Python is also platform independent. Our development environment is Linux.

## **3.3 Version Control**

### **3.3.1 Git**

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. We have chosen git for version control.

## **3.4 Prototype**

We need to find the best classification algorithm that can be used in our Web App to classify music based on genres. python is ideal to implement machine learning algorithms in minimum lines of code. Before making the Web App in python we made a prototype in python.

### **3.4.1 Feature Extraction**

We chose to extract MFCC from the audio files as the feature. For finding MFCC we are using librosa library built on python. The output will be a matrix of  $13 \times n$  dimensional vector. Where  $n$  depends on the total duration of the audio.  $13 \times (100 \times \text{sec})$ .

### 3.4.2 Principal component

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components (or sometimes, principal modes of variation). The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components [7]. After doing a PCA on the data we got 90percent variance and should reduce the feature dimension.

```
[input2, eigvec, eigvalue] = pca (ds. input);
```

```
cumvar = cumsum(eigvalue); //cumulative sum n(n+1)/2 cumvarpercent = cumvar/cumvar(end)*100;
```

### 3.4.3 Dimensionality reduction

Various researchers take statistics such as mean variance IQR, etc., to reduce the feature dimension. Some researchers model it using multivariate regression and some fit it to a Gaussian mixture model. Here we are taking the mean and upper diagonal variance of 13\*n MFCC coefficients. The result is a feature vector of size 104.

```
%Reducing Feature Dimeansion
```

```
mf = mean(mm,2); %row mean
```

```
cf = cov(mm'); % covariance
```

```
for i=0:(size(mm,1)-1)
```

```
= [ff;diag(cf,i)]; %use diagonals  
end
```

```
t(:,end+1) = ff(:,1);
```

### 3.4.4 Classification

- K-nearest neighbors (KNN)

Principle is that the data instance of the same class should be closer in the feature space.

For a given data point  $x$  of unknown class, we can compute the distance between  $x$  and all the data points in the training data and assign the class determined by  $k$  nearest points of  $x$ . Suppose we are given training dataset of  $n$  points

$(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$

For a new data point  $x$  the most likely class is determined by finding the distance from all training data points (Euclidian distance). The output class will be the class which  $k$  nearest neighbors belongs to.  $K$  is a predefined integer ( $k=1, k=2, k=3$ .)

- Logistic Regression

Logistic Regression is one of the widely used classification algorithm. This algorithm is used in medical as well as business fields for analytics and classification.

This model has the hypothesis function  $0 \leq \hat{y}(x) \leq 1$ . Where  $\hat{y}(x) = \text{mod}(1 + e^{-z})$  called as Sigmoid or Logistic Function. For binary class classification  $0, 1$ . The output of this classifier will be a probability of the given input belonging to class 1. If  $\hat{y}(x)$  outputs 0.7 it means that the given input has 70% chance of belonging to class 1. Since we have 10 genre classes  $0, 1 \dots 9$  we used one-vs-all method for classification

## 3.5 Music Genre Classifier App

Our web application is written in Python using Django framework. Every HTTP request is first gone to the url dispatcher `urls.py` which will contain a view present in `views.py`. We can write

rules in `views.py` to handle each HTTP request on that url.

We have written views for HTTP POST and GET requests to upload music and find genre. There need to be models (`models.py`) for every files that we store in the database. The uploaded music is stored in sqlite database. The file is automatically deleted once we find the genre. Python objects can be saved in to the disk by using a framework called Joblib: `joblib.dump` (object, filename). Our trained classifier is saved in to the disk and loaded by `joblib.load`(filename). Since our training dataset is small, our classifier object does not need to be compressed. Our web application uses the package `mysvm` which we developed to extract features and to find the genre label.

#### 3.5.1 Python package *mysvm*

We developed a python package called '`mysvm`' which contains three modules: `features`, `svm`, `acc`. These are used by the web application in feature extraction and finding genre. This package also contains many other functions to do complicated feature extraction and classification.

■ *This module is used to extract MFCC features from a given file. It contains the following functions*

- `extract (file)`: Extract features from a given file. Files in other formats are converted to .wav format. Returns numpy array.
- `extract_all (audio_dir)`: Extracts features from all files in a directory.
- `extract_ratio (train_ratio, test_ratio, audio_dir)` : Extract features from all files in a directory in a ratio. Returns two numpy arrays.
- `geny(n)`: Generates Y values for n classes. Returns numpy array.
- `gen_suby(sub, n)`: Generates Y values for a subset of classes. Returns numpy array.
- `gen_labels( )`: Returns a list of all genre labels.
- `flatten( x)` : Flatterns a numpy array

## ■ svm

This module contains various functions for classification using support vector machines.

- *poly(X,Y)*: Trains a poly kernel SVM by fitting X, Y dataset. Returns a trained poly kernel SVM classifier.
- *fit ( training\_percentage, fold)*: Randomly choose songs from the dataset, and train the classifier. Accepts parameter: train\_percentage, fold; Returns trained classifier.
- *getprob (filename)*: Find the probabilities for a song belongs to each genre. Returns a dictionary mapping genre names to probability and a list of top 3 genres which is having probability of more than 0.15.
- *random\_cross\_validation (train\_percentage, fold)*: Randomly cross validate with training percentage and fold. Accepts parameter: train\_percentage, fold;
- *findsubclass (class\_count)*: Returns all possible ways we can combine the classes. Accepts an integer as class count. Returns numpy array of all possible combination.
- *gen\_sub\_data (class\_l)*: Generate a subset of the dataset for the given list of classes. Returns numpy array.
- *fitsvm (Xall, Yall, class\_l, train\_percentage, fold)*: Fits a poly svm and returns the accuracy. Accepts parameter: train\_percentage; fold; Returns: classifier, Accuracy.
- *best\_combinations (class\_l, train\_percentage, fold)*: Finds all possible combination of classes and the accuracy for the given number of classes Accepts: Training percentage, and number of folds Returns: A List of best combination possible for given the class count.
- *getgenre (filename)*: Accepts a filename and returns a genre label for a given file.
- *getgenreMulti (filename)*: Accepts a filename and returns top three genre labels based on the probability.



## ■ acc

Module for finding the accuracy. *get ( res, test )* : Compares two arrays and returns the accuracy of match.

## 3.6 Pseudocode for each Module

### 3.6.1 Code for classification in KNN

```
fprintf('classificationany key to continue. '); pause;
```

```
fprintf('knn using 10 feature vectors');
```

```
sumr=0;sumk=0;
```

```
for i=1:5
```

```
[md, rloss, kloss] = myKnn(ds2, i);
```

```
sumr = sumr + rloss;
```

```
sumk = sumk + kloss;
```

```
End
```

```
fprintf('\naverage resubstitution loss = %g %%\n',sumr*100/5); fprintf('\naverage cross-validation  
loss loss = %g %%\n',sumk*100/5); fprintf('knn using 156 feature vectors'); sumr=0;sumk=0;
```

```
for i=1:5
```

```
[md, rloss, kloss] = myKnn(ds, i);
```

```
sumr = sumr + rloss;
```

```
sumk = sumk + kloss;
```

```
End
```

Md

```
fprintf('\naverage resubstitution loss = %g %%\n',sumr*100/5);
```

```
fprintf('\naverage cross-validation loss loss = %g %%\n',sumk*100/5);
```

```
fprintf('knn using 2 feature vectors: LDA (linear discriminant analysis)');
```

```
sumr=0;sumk=0;
```

```
for i=1:5
```

```
[md, rloss, kloss] = myKnn(ds, i);
```

```
sumr = sumr + rloss;
```

```
sumk = sumk + kloss;
```

End

```
Md fprintf('resubstitution loss = %g %%\n',sumr*100/5); fprintf('\naverage cross-validation  
loss loss = %g %%\n',sumk*100/5); fprintf('\nprogram finished executing \n\npress any key  
to continue..\n'); pause;
```

```
fprintf('————END————\n')
```

```
toc(scriptStartTime)
```

### 3.6.2 Code for Logistic Regression

```
dim=size(x, 1)
```

```
if isPca
```

```
[input2, eigVec, eigValue]=pca(x');
```

```
cumVar=cumsum(eigValue);
```

```
cumVarPercent=cumVar/cumVar(end)*100;
```

```
figure; plot(cumVarPercent, '.-');

xlabel('No. of eigenvalues');

ylabel('Cumulated variance percentage (%)');

title('Variance percentage vs. no. of eigenvalues');

cumVarTh=95;

index=find(cumVarPercent<cumVarTh);

newDim=index(1);

x2=input2(1:newDim, :);

end

%x2=x2';

lambda = 0.1;

[all_theta] = oneVsAll(x, y, num_labels, lambda); pred = predictOneVsAll(all_theta, x);

fprintf('Set Accuracy: %f\n', mean(double(pred == y)) * 100);

X=x;

Y=y;

if isTest

load('mgcTest.mat'); lambda = 0.1;

pred = predictOneVsAll(all_theta, x);

fprintf('Testing Set Accuracy: %f\n', mean(double(pred == y)) * 100);

end
```

### 3.6.3 Logistic regression Cost function

```
function [J, grad] = lrCostFunction(theta, X, y, lambda)

m = length(y); % number of training examples

J=0;

grad = zeros(size(theta));

htheta = sigmoid(X * theta);

%J = -(1 / m) * (y .* log(htheta) + (1-y) .* log(1 - htheta)) + lambda/(2*m) * sum(theta(2:end)
.â²)

J = 1 / m * sum(-y .* log(htheta) - (1 - y) .* log(1 - htheta)) + lambda / (2 * m) *

sum(theta(2:end) .â²);

theta = [0;theta(2:end)];

grad = (1/m) * (X' * (htheta - y) + lambda * theta);
```

### 3.6.4 Logistic regression Gradient Descent

```
function [all_theta] = oneVsAll(X, y, num_labels, lambda)

m = size(X, 1);

n = size(X, 2);

all_theta = zeros(num_labels, n + 1);

Add ones to the X data matrix X = [ones(m, 1) X];

for c = 1:num_labels

initial_theta = zeros(n + 1, 1);

options = optimset('GradObj', 'on', 'MaxIter', 50); [theta] = ...
```

```
fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)), ...  
initial_theta, options); all_theta(c,:) = theta;  
  
end  
  
end
```

### 3.6.5 Code for feature extraction in SVM

```
a = glob.glob('wav/*/*.wav') //get the relative path of the files  
  
Testing = False  
  
Training = True  
  
a.sort()  
  
all_mfcc = np.array([]) //initialize the array  
  
count = 0; //training count = 30; //test  
  
loop_count = -1  
  
flag = True  
  
for i in a: // extract mfcc features for the audio files  
  
    if Training: // for training select only 90 songs from each loop_count += 1  
  
        if loop_count % 100 == 0:  
  
            count = 0  
  
            if count == 70:  
  
                continue //selects only 90 songs in every 100 songs count += 1  
  
            if Testing: // for testing select last 10 songs from each genre loop_count += 1
```

```
if (loop_count + 30) % 100 == 0 and loop_count:

    count = 0

    print('-'*10)

    if count == 30:

        continue

    count += 1

    (rate, data) = scipy.io.wavfile.read(i)

    mfcc_feat = mfcc(data,rate)

    redusing mfcc dimension to 104

    = np.transpose(mfcc_feat) mf = np.mean(mm,axis=1)

    cf = np.cov(mm) ff=mf

    for i in range(mm.shape[0]): // ff is a vector of size 104 ff = np.append(ff,np.diag(cf,i))

    if flag: // re initializing to size 104

        all_mfcc = ff;

    print('*'*20)

    flag = False

    else:

        all_mfcc = np.vstack([all_mfcc,ff])

    print("looping——",loop_count)

    print("all_mfcc.shape:",all_mfcc.shape)

    y=[np.ones(70),np.ones(70)*2,np.ones(70)*3,np.ones(70)*4,np.ones(70)*5,
```

```
np.ones(70)*6,np.ones(70)*7,np.ones(70)*8,np.ones(70)*9,np.ones(70)*10]

yt=[np.ones(30),np.ones(30)*2,np.ones(30)*3,np.ones(30)*4,np.ones(30)*5,

np.ones(30)*6,np.ones(30)*7,np.ones(30)*8,np.ones(30)*9,np.ones(30)*10]

y = flatten(y)

yt = flatten(yt)
```

### 3.6.6 Code for classification in SVM

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

```
resource_package = __name__ // load pre saved variables resource_path = '/' .join(('','data/Xall.npy'))

Xall_path = pkg_resources.resource_filename(resource_package, resource_path)

Xall = np.load(Xall_path)

Yall = feature.geny(100)

resource_path = '/' .join(('','data/classifier_10class.pkl'))

clf_path = pkg_resources.resource_filename(resource_package, resource_path)

myclf = joblib.load(clf_path)

def poly(X,Y):

clf = svm.SVC(kernel='poly',C=1) //Polynomial kernel

clf.fit(X,Y)

return clf

def random_cross_validation(train_percentage,fold): resTrain =0 //creates a matrix of size 10x100x104
```

```
resTest = 0

score = 0

scores = 0

for folds in range(fold):

    flag = True // init

    flag_train = True

    start = 0

    train_matrix = np.array([])

    test_matrix = np.array([])

    Xindex = []

    Tindex = []

    for class_counter in range(10):

        stack = list(range(start, start+100)) //create an index of size 100 for song_counter in range(
        int(train_percentage) ):

        index = random.choice(stack) //randomly choose numbers from index

        stack.remove(index) //remove the choosen number from index

        random_song = Xall[index] //select songs from that index for training

        Xindex.append(index)

    if flag:

        train_matrix = random_song

        flag = False

    else:
```

---



```
train_matrix = np.vstack([train_matrix, random_song])

start += 100

//select the remaning songs from the stack for testing for test_counter in range(100 - train_percentage):

Tindex.append(stack[test_counter])

if flag_train:

    test_matrix = Xall[stack[test_counter]]

    flag_train = False

else:

    test_matrix = np.vstack([test_matrix, Xall[stack[test_counter]]] Y = feature.geny(train_percentage)

    y = feature.geny(100 - train_percentage)

    clf = svm.SVC(kernel='poly',C=1) //training accuracy clf.fit(train_matrix, Y)

    res = clf.predict(train_matrix)

    resTrain += acc.get(res,Y)

    res = clf.predict(test_matrix)

    resTest += acc.get(res,y)

    print("Training accuracy with %d fold %f: " % (int(fold), resTrain / int(fold)))

    print("Testing accuracy with %d fold %f: " % (int(fold), resTest / int(fold)))

def findsubclass(class_count):

    //Finds the combination of classes NCR

    class_1 = list (range (10))

    flag = True
```

```
labels = np. array ([])

for i in itertools. combinations (class_l, class_count):

    if flag:

        labels = i

        flag = False

    else:

        labels = np. vstack ([labels, i])

    return labels

def gen_sub_data(class_l):

    all_x = np. array ([])

    flag = True;

    for class_index in class_l:

        if class_index != 0:

            class_index *= 100

            if flag:

                all_x = Xall [ class_index: class_index + 100]

                flag = False

            else:

                all_x = np. vstack ([all_x, Xall [ class_index: class_index + 100]])

    return all_x

def fitsvm (Xall, Yall, class_l, train_percentage, fold):
```

```
//creates a matrix of size 10x100x104

resTrain =0

resTest = 0

score = 0

scores = 0

for folds in range(fold):

    //init

    flag = True

    flag train = True

    start = 0

    train matrix = np. array ([])

    test_matrix = np. array ([])

    Xindex = []

    Tindex = []

    for class counter in range(class_1):

        stack = list(range(start, start+100)) //create an index of size 100 for song counter in range(
        int(train percentage) ):

        index = random. Choice(stack) //randomly choose numbers from index

        stack. Remove(index) //remove the choosen number from index

        random song = Xall[index] //select songs from that index for training

        Xindex.append(index)

    if flag:
```

```
train_matrix = random_song

flag = False

else:

train_matrix = np.vstack([train_matrix, random_song])

start += 100

select the remaning songs from the stack for testing for test_counter in range(100 - train_percentage):

Tindex.append(stack[test_counter])

if flag_train:

test_matrix = Xall[stack[test_counter]]

flag_train = False

else:

test_matrix = np.vstack([test_matrix, Xall[stack[test_counter]]])

Y = feature.gen_suby(class_l, train_percentage)

y = feature.gen_suby(class_l, 100 - train_percentage)

//training accuracy

clf = svm.SVC(kernel='poly',C=1)

clf.fit(train_matrix, Y)

//train case

res = clf.predict(train_matrix)

//print(acc.get(res,Y))

resTrain += acc.get(res,Y)
```

```
res = clf.predict(test_matrix)

resTest += acc.get(res,y)

return clf , resTest / int(fold)

def best_combinations(class_l, train_percentage, fold):

class_comb = findsubclass(class_l)

avg = []

count = 0

X = gen_sub_data(class_comb[0])

Y = feature.gen_suby(class_l,100)

for class_count in range(class_comb.shape[0]):

all_x = gen_sub_data( class_comb[ class_count ] )

all_y = feature.gen_suby(class_l,100)

clf , score = fitsvm(all_x, all_y, class_l, train_percentage, fold)

avg.append(score)

print(score)

print(class_count)

count += 1

if count == 10:

break

maxAvg = max(avg)

maxIndex = [i for i, j in enumerate(avg) if j <= (maxAvg - 2)] print("Maximum accuracy:",maxAvg)
print("Best combinations:")
```

```
for i in maxIndex:

    print(class_comb[i])

return avg

def getgenre(filename):

    music_feature = feature.extract(os.path.abspath(os.path.dirname(__name__) + '/django-jquery-
file-upload/' + filename))

    clf = myclf

    return clf.predict(music_feature)
```

### 3.6.7 Front End

We have used Ajax, AngularJS, and jQuery for front end JavaScript and Bootstrap for responsive design. We have also used Font Awesome iconic fonts and CSS framework.

For finding genre labels

Resuest is sent as Http POST

\$svm is for finding single label

\$multisvm is for finding multiple labels

File: app.js

```
.controller('FileDestroyController', [ '$scope', '$http',
```

```
function ($scope, $http) var file = $scope.file,
```

```
state;
```

```
if (file.url)
```

```
file.$state = function () return state;
```

```
;

file.$svm = function () state = 'pending'; return $http(

url: '/upload/svm/' ,

method: 'POST',

data: 'file': file.url, 'delete' : file.deleteId,

xsrferHeaderName: 'X-CSRFToken',

xsrferCookieName: 'csrftoken',

headers:

'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8' ,

).then(

function (response)

state = 'resolved';

$scope.resp = response.data;

,

function () state = 'rejected';

);

;

file.$multisvm = function ()

state = 'pending';

return $http(

url: '/upload/multisvm/' ,
```

```
method: 'POST',

data: 'file': file.url, 'delete' : file.deleteId,

xsrHeaderName: 'X-CSRFToken',

xsrCookieName: 'csrftoken',

headers:

'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8' ,

).then(

function (response)

state = 'resolved';

$scope.resp = response.data;

// $scope.resp = JSON.parse(response.data);

,

function ()

state = 'rejected';

);

;

]);

// Submitting file and toggle options // File: jquery.fileupload-angular.js

$scope.submit = function ()

this.applyOnQueue('$submit');

;
```



```
$scope.find = function()

var x = document.getElementById('ismulti').checked; if(x)

this.applyOnQueue('$svm');

else

this.applyOnQueue('$multisvm');

$scope.findGenre = function ()

this.applyOnQueue('$svm');

;

$scope.findMultiGenre = function ()

this.applyOnQueue('$multisvm');
```

### 3.6.8 Back End

We have used Django-jQuery-File-Upload [8] which is an open source project ported from jQuery-File-Upload [9] by Sebastian Tschan.

Url Dispatcher

File: urls.py

encoding: utf-8

```
from django.conf.urls import url
```

```
from fileupload.views import (
```

```
MusicDeleteView,MultiSvm,
```

```
info,MusicCreateView,
```

```
)

from . import views urlpatterns = [

url(r'^help/$', info.as_view(), name='info'),

url(r'^musicUpload/$', MusicCreateView.as_view(), name='upload-music'),

url(r'^$', MultiSvm.as_view(), name='upload'),

url(r'^delete/(?P<pk>\d+)$', MusicDeleteView.as_view(), name='upload-delete'),

url(r'^svm/$', views.music_genre, name='music_genre'),

url(r'^multisvm/$', views.multi_music_genre, name='multi_music_genre'),

]

//Views

//file: views.py

encoding: utf-8 import json

from django.http import HttpResponseRedirect

from django.views.generic import CreateView, DeleteView, ListView, DetailView from .models
import Picture, Music

from .response import JSONResponse, response_mimetype from .serialize import serialize

import json import random

from mysvm import feature from mysvm import svm

class MusicCreateView(CreateView): model = Music fields = "__all__"

def form_valid(self, form): self.object = form.save() files = [serialize(self.object)] data = 'files':
files

response = JSONResponse(data, mimetype=response_mimetype(self.request)) response['Content-
Disposition'] = 'inline; filename=files.json'
```

```
def form_invalid(self, form):

data = json.dumps(form.errors)

return HttpResponse(content=data, status=400, content_type='application/json')

class info(MusicCreateView):

template_name_suffix = '_svm_info'

class MultiSvm(MusicCreateView):

template_name_suffix = '_svm_multi'

class MusicDeleteView(DeleteView):

model = Music

def delete(self, request, *args, **kwargs):

self.object = self.get_object()

self.object.delete()

response = JsonResponse(True, mimetype=response_mimetype(request))

response['Content-Disposition'] = 'inline; filename=files.json'

return response

def music_genre(request):

model = Music

if request.method == 'POST':

context = feature.getlabels()

context = ['Classical', 'Hipop', 'Jazz', 'Metal', 'Pop', 'Rock']

return (request, "love")
```

```
try:

JSONdata = json.loads(str(request.body, encoding='utf-8'))

except:

JSONdata = 'ERROR'

get index of the genre

genre = svm.getgenre(JSONdata['file'])

delete file after finding genre

id = JSONdata['delete']

instance = model.objects.get(id=id)

instance.delete()

return HttpResponse(context[int(genre[0]) - 1 ])

if request.method == 'GET':

return HttpResponse('nothing here')

def multi_music_genre(request):

model = Music

if request.method == 'POST':

try:

JSONdata = json.loads(str(request.body, encoding='utf-8'))

except:

JSONdata = 'ERROR'

print(JSONdata['file'])
```

```
get index of the genre

dd, genre = svm.getgenreMulti(JSONdata['file'])

print(dd)

dt = json.dumps(dd)

delete file after finding genre

id = JSONdata['delete']

instance = model.objects.get(id=id)

instance.delete()

return HttpResponse(', '.join(genre))

return HttpResponse(dt)

if request.method == 'GET':

return HttpResponse('nothing here')
```

#### Models

File: models.py

encoding: utf-8

```
from django.db import models class Music(models.Model):
```

```
"""
```

Model to upload music file

```
"""
```

```
file = models.FileField(upload_to="audio")
```

```
slug = models.SlugField(max_length=50, blank=True)
```

```
def __str__(self):  
  
    return self.file.name  
  
    @models.permalink  
  
    def get_absolute_url(self):  
  
        return ('upload-new', )  
  
    def save(self, *args, **kwargs):  
  
        self.slug = self.file.name  
  
        super(Music, self).save(*args, **kwargs)  
  
    def delete(self, *args, **kwargs):  
  
        """delete – Remove to leave file."""  
  
        self.file.delete(False)  
  
        super(Music, self).delete(*args, **kwargs)
```

## **3.7 Summary**

This chapter discusses implementation for implementing the brief description about the programming language selection, platform selected and finally the codes for each process.

## CHAPTER 4

---

### MODEL TESTING AND RESULT ANALYSIS

---

This section of the report shows the testing of various machine learning models which we used for training over the preprocessed dataset .We have tried training over 8 genre classes and predicting the accuracy of each model .

## 4.1 Logistic Regression

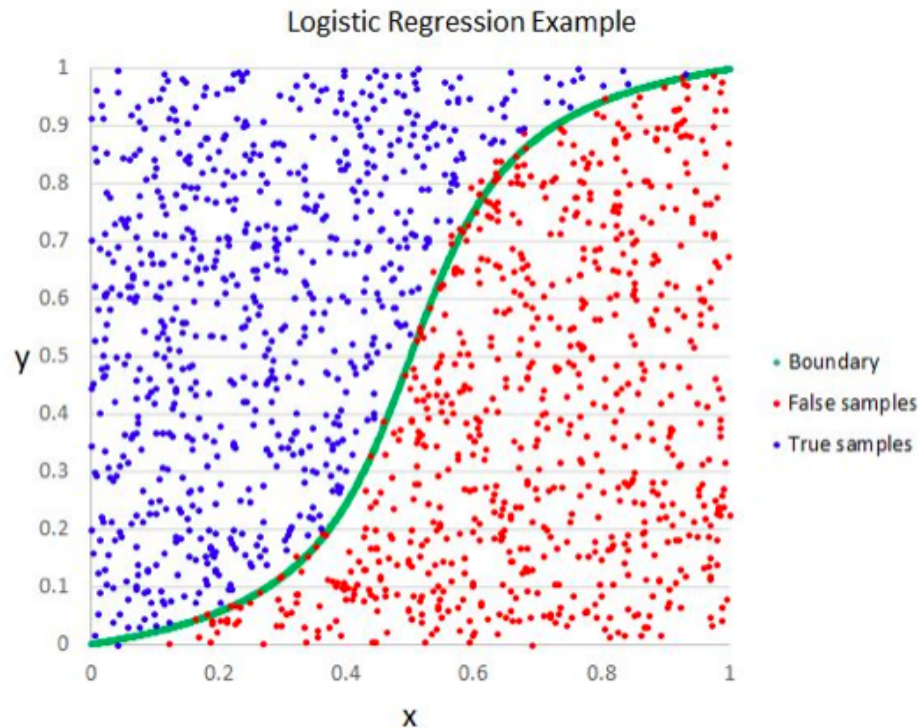


Figure 4.1: Logistic Regression Example

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes. The following section shows the python code for training the logistic regression model and evaluation of the Confusion Matrix and Classification Report over the 8 genre classes : -

1. The model is trained over the training data and then tested over the test data. And we were able to achieve accuracy of 33.5 %.

```
In [45]: cls = LogisticRegression()
cls.fit(train_x , train_y)
Y = cls.predict(test_x)
models.append('logistic Regression')
accuracy.append(accuracy_score(test_y , Y)*100)
print('Accuracy : ', accuracy_score(test_y , Y)*100 )

Accuracy : 33.5
```



## 2. Evaluated the confusion Matrix

```
In [47]: print('CONFUSION MATRIX')
         confusion_matrix(test_y, Y)

CONFUSION MATRIX
out[47]: array([[ 7,  2,  5,  1,  4,  1,  4,  4],
               [ 1, 14,  1,  0,  0,  1,  0,  0],
               [ 6,  1,  7,  4,  0,  2,  0,  3],
               [ 4,  3,  2,  6,  3,  4,  6,  2],
               [ 0,  0,  1,  1, 13,  0,  0,  2],
               [ 1, 10,  4,  0,  0, 10,  1,  2],
               [ 5,  3,  2,  7,  4,  2,  4,  3],
               [ 3,  0,  3,  5,  6,  0,  4,  6]])
```

## 3. Finally we evaluated the classification Report

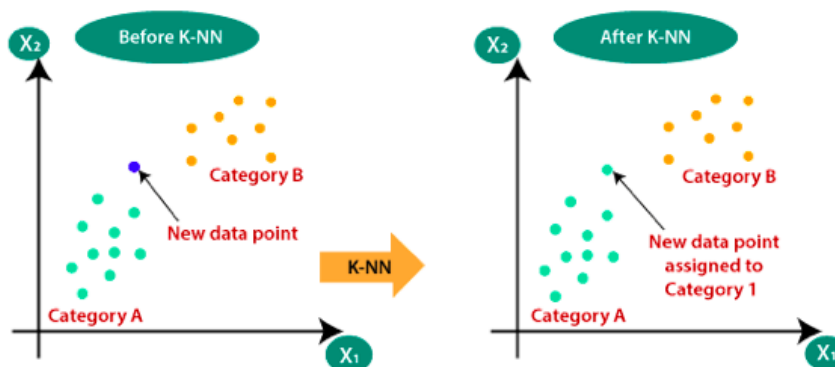
```
In [48]: print("CLASSIFICATION REPORT")
         print(classification_report(test_y, Y))

CLASSIFICATION REPORT
precision    recall  f1-score   support

     1      0.26      0.25      0.25        28
     2      0.42      0.82      0.56        17
     3      0.28      0.30      0.29        23
     4      0.25      0.20      0.22        30
     5      0.43      0.76      0.55        17
     6      0.50      0.36      0.42        28
     7      0.21      0.13      0.16        30
     8      0.27      0.22      0.24        27

 avg / total      0.32      0.34      0.31       200
```

## 4.2 K-Nearest neighbour



K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the

distance between the test data and all the training points. Then select the K number of points which is closet to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

### 4.2.1 How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready.

The following section shows the python code for training the K Nearest Neighbour model and evaluation of the Confusion Matrix and Classification Report over the 8 genre classes : -

1. The model is trained over the training data and then tested over the test data. And we were able to achieve accuracy of 42.5 %.

```
In [157]: cls = sklearn.neighbors.KNeighborsClassifier()
          cls.fit(train_x , train_y)
          Y = cls.predict(test_x)
          models.append('k-neighbors')
          accuracy.append(accuracy_score(test_y , Y)*100)
          print('Accuracy : ',accuracy_score(test_y , Y)*100 )

Accuracy : 42.5
```

2. Evaluated the confusion Matrix

```
In [59]: print("CONFUSION MATRIX")
         confusion_matrix(test_y, y)

CONFUSION MATRIX
Out[59]: array([[ 2, 16,  3,  1,  3,  0,  1,  2],
                [ 0, 16,  1,  0,  0,  0,  0,  0],
                [ 0, 12,  6,  1,  0,  0,  1,  3],
                [ 1, 12,  3,  4,  3,  0,  2,  5],
                [ 1, 10,  0,  0,  4,  0,  0,  2],
                [ 0, 20,  1,  0,  0,  7,  0,  0],
                [ 1, 14,  5,  1,  2,  1,  5,  1],
                [ 2, 16,  0,  3,  1,  0,  1,  4]])
```

3. Finally we evaluated the classification Report

```
In [60]: print("CLASSIFICATION REPORT")
         print(classification_report(test_y, y))

CLASSIFICATION REPORT
precision    recall  f1-score   support

     1      0.29      0.07      0.11      28
     2      0.14      0.94      0.24      17
     3      0.32      0.26      0.29      23
     4      0.40      0.13      0.20      30
     5      0.31      0.24      0.27      17
     6      0.88      0.25      0.39      28
     7      0.50      0.17      0.25      30
     8      0.24      0.15      0.18      27

 avg / total      0.40      0.24      0.24      200
```

### 4.3 Support Vector Machine

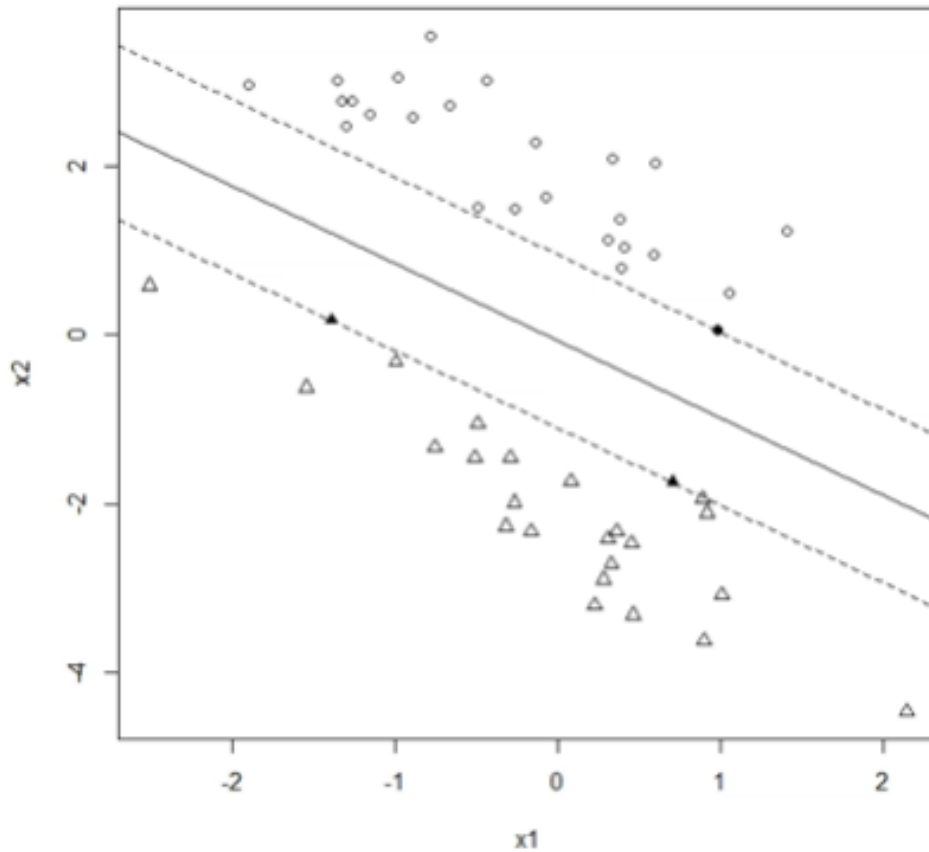


Figure 4.2: Linear Separable Features

Consider a support vector machine (SVM) for a classification task. Given a set of pairs of feature data-point vectors  $x$  and classifier labels  $y = -1, 1$ , the task of the SVM algorithm is to learn to group features  $x$  by classifiers. After training on a known data set the SVM machine is intended to correctly predict the class  $y$  of an previously unseen feature vector  $x$ . Applications in quantitative finance of support vector machines include for example predictive tasks, where  $x$  consists of features derived from a historical stock indicator time series and  $y$  is a sell or buy signal. Another example could be that  $x$  consist of counts of key-words within a text such as an news announcements and  $y$  categorizes it again according on its impact to market movements. Outside of finance a text based SVM could be used to filter e-mail to be forwarded to either the inbox or the spam folder.

### 4.3.1 SVM (Linear Kernel)

The SVM algorithm finds the largest possible linear margin that separates these two regions. The marginal separators rest on the outpost points that are right on the front line of their respective regions.

These points, marked as two bold triangles and one bold circle in the picture below, are named the ‘support vectors’ as they are supporting the separation boundary lines. In fact the SVM learning task fully consists of determining these support vector points and the margin distance that separates the regions. After training all other non-support points are not used for prediction.

The following section shows the python code for training the K Nearest Neighbour model and evaluation of the Confusion Matrix and Classification Report over the 8 genre classes : -

1. The model is trained over the training data and then tested over the test data. And we were able to achieve accuracy of 42.5 %

```
In [121]: cls = sklearn.svm.SVC(C = 5000, gamma = 0.0005, kernel = 'linear')
          cls.fit(train_x, train_y)
          Y = cls.predict(test_x)
          models.append('svm')
          accuracy.append(accuracy_score(test_y, Y)*100)
          print('Accuracy : ', accuracy_score(test_y, Y)*100)

Accuracy : 45.5
```

2. Evaluate the confusion matrix

```
In [124]: print('CONFUSION MATRIX')
          confusion_matrix(test_y, Y)

CONFUSION MATRIX
Out[124]: array([[11, 1, 3, 4, 3, 1, 2, 3],
                [1, 15, 1, 0, 0, 0, 0, 0],
                [2, 3, 9, 4, 1, 0, 2, 2],
                [2, 2, 0, 14, 2, 1, 4, 5],
                [1, 0, 0, 4, 9, 1, 1, 1],
                [0, 8, 3, 2, 0, 12, 3, 0],
                [2, 1, 2, 4, 2, 0, 14, 5],
                [3, 2, 4, 3, 2, 0, 6, 7]])
```

3. Evaluate the classification report

```
In [125]: print("CLASSIFICATION REPORT")
          print(classification_report(test_y, Y))
```

	precision	recall	f1-score	support
1	0.50	0.39	0.44	28
2	0.47	0.88	0.61	17
3	0.41	0.39	0.40	23
4	0.40	0.47	0.43	30
5	0.47	0.53	0.50	17
6	0.80	0.43	0.56	28
7	0.44	0.47	0.45	30
8	0.30	0.26	0.28	27
avg / total	0.48	0.46	0.45	200

### 4.3.2 SVM (RBF Kernel)

1. The model was trained with accuracy of 49.5 %

```
In [38]: cls = sklearn.svm.SVC(C = 5000, gamma = 0.0005, kernel = 'rbf')
          cls.fit(train_x, train_y)
          Y = cls.predict(test_x)
          models.append('svm')
          accuracy.append(accuracy_score(test_y, Y)*100)
          print('Accuracy : ', accuracy_score(test_y, Y)*100)

Accuracy : 49.5
```

2. Confusion Matrix.

```
In [42]: print('CONFUSION MATRIX')
          confusion_matrix(test_y, Y)
```

CONFUSION MATRIX

```
Out[42]: array([[ 9, 1, 2, 5, 3, 1, 3, 4],
                [ 0, 13, 2, 0, 0, 1, 1, 0],
                [ 2, 0, 9, 4, 0, 4, 2, 2],
                [ 0, 1, 0, 12, 5, 1, 6, 5],
                [ 1, 0, 0, 1, 11, 0, 1, 3],
                [ 1, 2, 4, 1, 0, 15, 0, 0],
                [ 5, 0, 2, 2, 2, 1, 17, 1],
                [ 2, 0, 3, 3, 4, 0, 2, 13]])
```

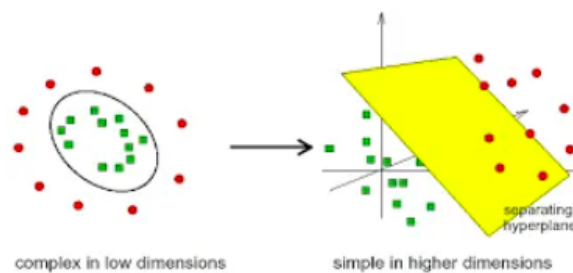
3. Classification Report

```
In [43]: print("CLASSIFICATION REPORT")
          print(classification_report(test_y, Y))
```

	precision	recall	f1-score	support
1	0.45	0.32	0.38	28
2	0.59	0.76	0.67	17
3	0.41	0.39	0.40	23
4	0.43	0.40	0.41	30
5	0.44	0.65	0.52	17
6	0.65	0.54	0.59	28
7	0.53	0.52	0.55	30
8	0.46	0.48	0.47	27
avg / total	0.50	0.49	0.49	200

### 4.3.3 SVM (Polynomial Kernel)

When it is not possible to classify or divide the genres using any type of linear classification model (Low Accuracy) .We use Poly kernel Classification .This classifier where if  $n$  is the dimension of the input then what it does it increase the dimension of the input and then with the help of a hyperplane it draws the boundary plane between the different intended output genres .



Finally lets see the python output for SVM polynomial kernel model and the confusion matrix

1. The model was trained with accuracy of 53.5 %

```
In [130]: cls = sklearn.svm.SVC(C = 5000, gamma = 0.0005, kernel = 'poly')
cls.fit(train_x , train_y)
Y = cls.predict(test_x)
models.append('svm_poly')
accuracy.append(accuracy_score(test_y , Y)*100)
print('Accuracy : ',accuracy_score(test_y , Y)*100)

accuracy : 53.5
```

2. Confusion Matrix

```
In [133]: print('CONFUSION MATRIX')
confusion_matrix(test_y, Y)

CONFUSION MATRIX

Out[133]: array([[ 9,  0,  3,  4,  1,  4,  2,  5],
 [ 0, 14,  2,  0,  0,  1,  0,  0],
 [ 3,  0, 10,  2,  0,  4,  1,  3],
 [ 2,  0,  0, 19,  0,  0,  6,  3],
 [ 3,  0,  0,  0, 13,  0,  1,  0],
 [ 1,  9,  4,  1,  0, 13,  0,  0],
 [ 2,  1,  0,  4,  0,  1, 20,  2],
 [ 5,  1,  1,  6,  3,  1,  1,  9]])
```

### 3. Classification Report

```
In [134]: print("CLASSIFICATION REPORT")
          print(classification_report(test_y, Y))
```

	precision	recall	f1-score	support
1	0.36	0.32	0.34	28
2	0.56	0.82	0.67	17
3	0.50	0.43	0.47	23
4	0.53	0.63	0.58	30
5	0.76	0.76	0.76	17
6	0.54	0.46	0.50	28
7	0.65	0.67	0.66	30
8	0.41	0.33	0.37	27
avg / total	0.53	0.54	0.53	280

Out of all the models ,SVM Poly kernel gives us the maximum accuracy of 53%

## 4.4 Snapshots

The results explanation related to the snapshots of the various modules to the entire process of execution. The different modules can explain the working of procedure.

The user has to select the option depending upon the requirements. The choice based option contains (1) Upload Music. (2) Find Genre. (3) find multiple genre.

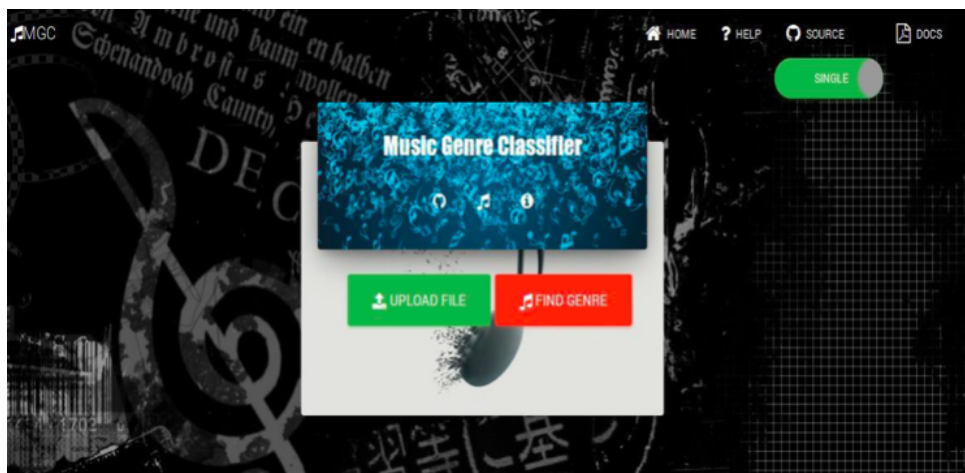


Figure 4.3: Front face of Web App



Figure 4.3 shows the front face of the Web App. The front face consist of mainly 6 buttons. Those are (1) upload file, (2) find genre, (3) docs, (4) help, (5) toggle between single and multiple and (6) home.

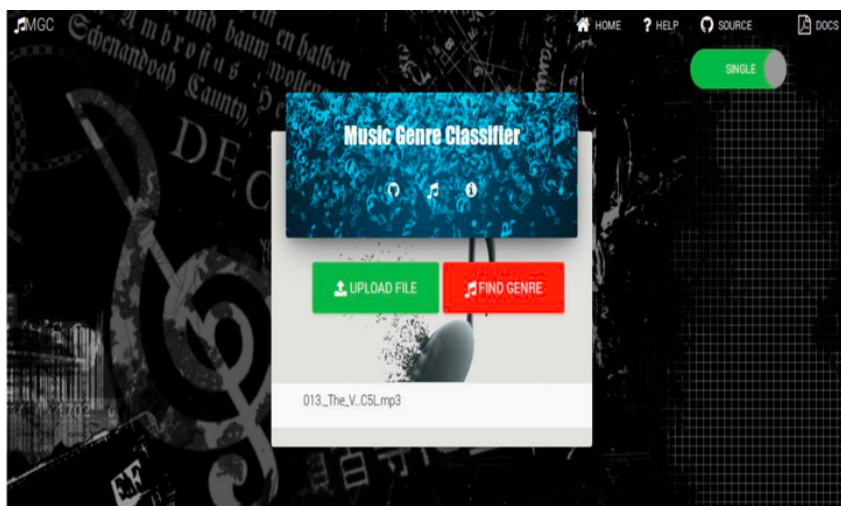


Figure 4.4: Upload the music

Figure 4.4 illustrates the uploading of the music. As the user clicks the ‘upload file’ button the Web App shows the file explorer to select the file.

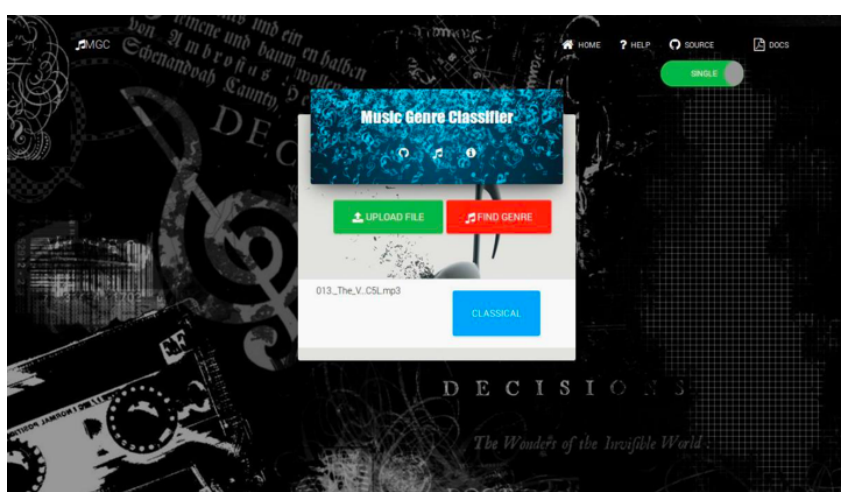


Figure 4.5: Finding the genre

Figure 4.5 illustrates the file being classified according to genre. As the user clicks the ‘find genre’ button the Web App calls `getgenre(JSONdata[‘file’])` function. On receiving the file, the function will use the trained classifier to predict a genre.

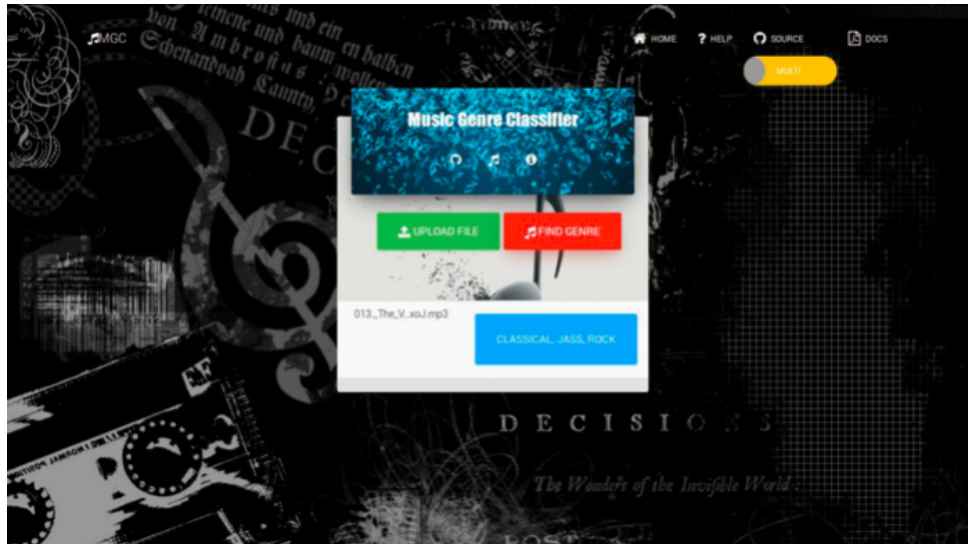


Figure 4.6: Finding multiple genres

The snapshot 4.6 illustrates the result of finding multiple genres

## 4.5 Summary

This chapter discusses about the user interface of the project which has been built upon html/css through which we can upload any audio file in any format and that gets converted to .wav format before processing.

---

# CONCLUSION AND FUTURE ENHANCEMENTS

---

## 5.1 Conclusion

This project on music genre classification was done by using various machine learning algorithms. Our aim was to get maximum accuracy. We have found out from our research that we can get maximum accuracy of 53% by using poly kernel SVM for 8 genre classes. We have also tried to find the best combination of genre classes which will result in maximum accuracy. If we choose 6 genre classes we were able to get an accuracy of 87% for the combination [classical, hip-hop, jazz, metal, pop and rock].

For some songs we can say that this has feature of multiple genres. So we have also tried to get multiple label outputs based on the probability. 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. This could have been due to the fact that many other genres are derived from blues.

## **5.2 Future Enhancement**

We can deploy this as a Web App on the cloud. We can also provide api for developers. Another thing we can try to improve the accuracy is trying a different dataset or dynamically increase the dataset as the user input new music.

---

## References

---

- [1] <http://marsyas.info/downloads/datasets.html>
- [2] <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>
- [3] Archit Rathore and Margaux Dorido, “Music Genre Classification”, Indian Institute of Technology, Kanpur Department of Computer Science and Engineering, 2015.
- [4] Yusuf Yaslan and Zehra Cataltepe, “Audio Music Genre Classification Using Different Classifiers and Feature Selection Methods”, Istanbul Technical University, in 2006.
- [5] Tao Li, Mitsunori Ogihara and Qi Li, “A Comparative Study on Content-Based Music Genre Classification”, University of Rochester, in 2003.
- [6] Omar Diab, Anthony Manero, and Reid Watson. Musical Genre Tag Classification with Curated and Crowdsourced Datasets. Stanford University, Computer Science, 1 edition, 2012.
- [7] [https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis)