Chapter - 1

INTRODUCTION

1.1 Introduction to Music Genre Classification

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. Automatic musical genre classification can potentially automate this process and provide an important component for a complete music information retrieval system for audio signals.

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:**

1. Frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the Mel filter bank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filter bank energies.
5. Take the DCT of the log filter bank energies.
6. 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 sec = 20 ms. Additional, if the overlap is 160 points, then the frame rate is 16000/(320-160) = 100 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…N-1, then the signal after Hamming windowing is s(n)*w(n), where w(n) is the Hamming window defined by:

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

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:

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

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=-MMCm(t+t)t] / [St=-MMt2]

The value of M is usually set to 2. If we add the velocity, the feature dimension is 26. 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, ⋯, 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:

$$y = e\hat{}(b_0 + b_1*x) / (1 + e\hat{}(b_0 + b_1*x))$$

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.

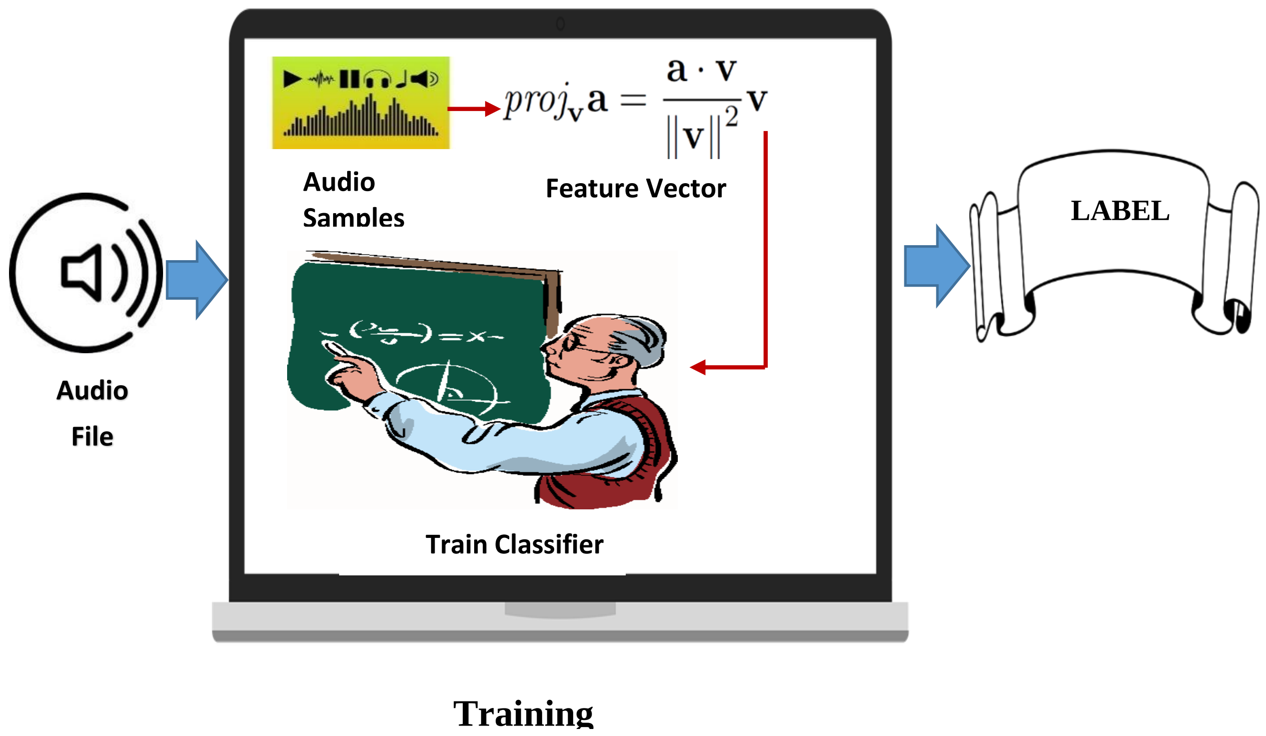
$proj_{\mathbf{v}}\mathbf{a} = \frac{\mathbf{a}\cdot\mathbf{v}}{\|\mathbf{v}\|^2}\mathbf{v}$
Audio Samples
Feature Vector
LABEL
Audio File
Train Classifier
Training

Fig 1.1 Overall Idea

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 Retrevial 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.

There have been many researches carried out on music genre classification, but none have implemented in a way that the common people without mathematical or programming knowledge can use to find genre of a music. 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

The objectives of music genre classification are as follows

- 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% 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 ± 4.2% on 10 genre set of 1000 music pieces.

There have been no published works that perform largescale genre classification using cross-model methods. Dawen Liang in 2011, proposed a cross-model retrieval framework of model blending which combines features of lyrics and audio. The algorithm scales linearly in the number of songs, and evaluate it on a subset of the MSD (Million Song Dataset) containing 156,289 songs. 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 Organization of Report

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 – High Level Design: 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 – 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 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

SYSTEM REQUIREMENT SPECIFICATION

2.1 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.

2.2 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.

2.3 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.

- Matlab 8.0
- Python v3.4

2.4 Hardware requirements

➢ Processor	:	Pentium –IV and higher
➢ Speed	:	1.1 GHz
➢ RAM	:	4GB
➢ Hard Disk	:	40 GB and above.

2.5 Software requirements

➢ Operating System	:	Linux/Windows
➢ Programming Language	:	Python v3.4
➢ Tool	:	Matlab 8.0

2.6 Summary

Under this chapter we have specified the hardware and the software requirements of the project, and also specified specific requirements.

Chapter-3

HIGH LEVEL DESIGN

3.1 High level Design

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.

3.2Design 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.

3.3 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.

3.3.1 Overall System Architecture

The figure 3.1 shows the system architecture for overall system of music genre classification using pattern recognition technique.

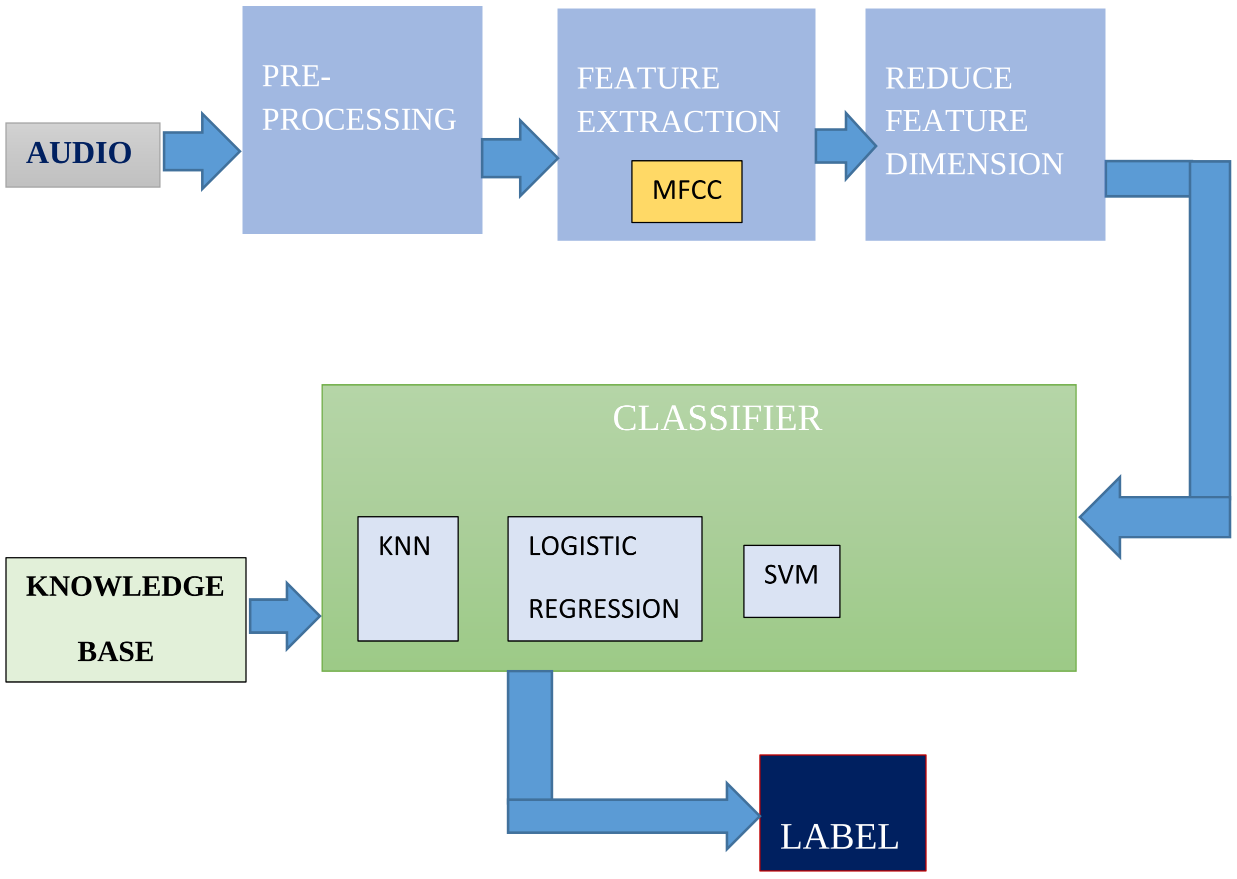

Fig 3.1. System Architecture for Overall System

3.3.1.1 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.

3.3.1.2 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.

3.3.1.3 Workflow

We have done logistic regression and KNN in Matlab 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 also used HTK MFCC MATLAB toolkit for finding MFCC. We took the mean variance of 13*n MFCC coefficients, result we get is 104*1 matrix or vector of size104. Then, we used different multi-class classifiers and an ensemble of these to obtain our results as a label.

3.3.2 Web application architecture

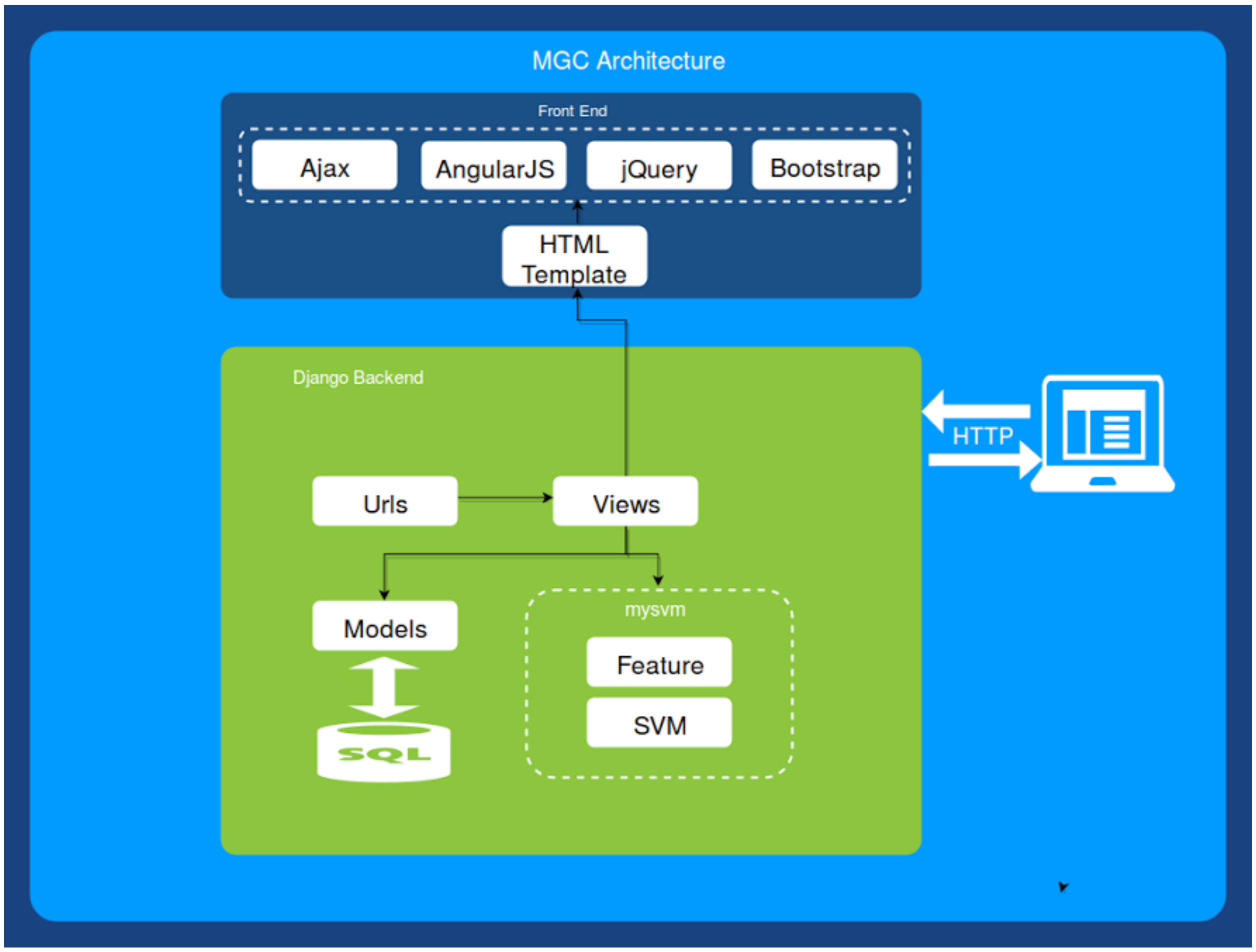

Fig 3.2 System Architecture for Web App

The figure 3.2 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*.

3.4 Module Specification

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

3.4.1 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.

3.4.2 Feature Extraction Module

The sound clips were then processed by the MATLAB code to extract features. The code is executed by various functions on it to extract their features.

3.4.3 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
  - Sigmoid Kernel

3.5 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.

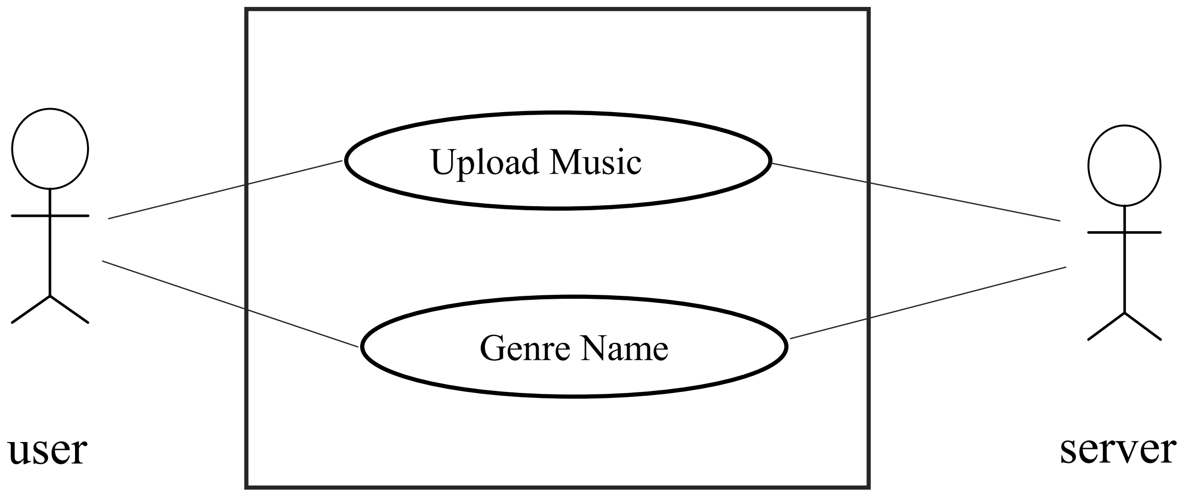

Fig 3.2. Use case diagram for overall system

The figure 3.2 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.

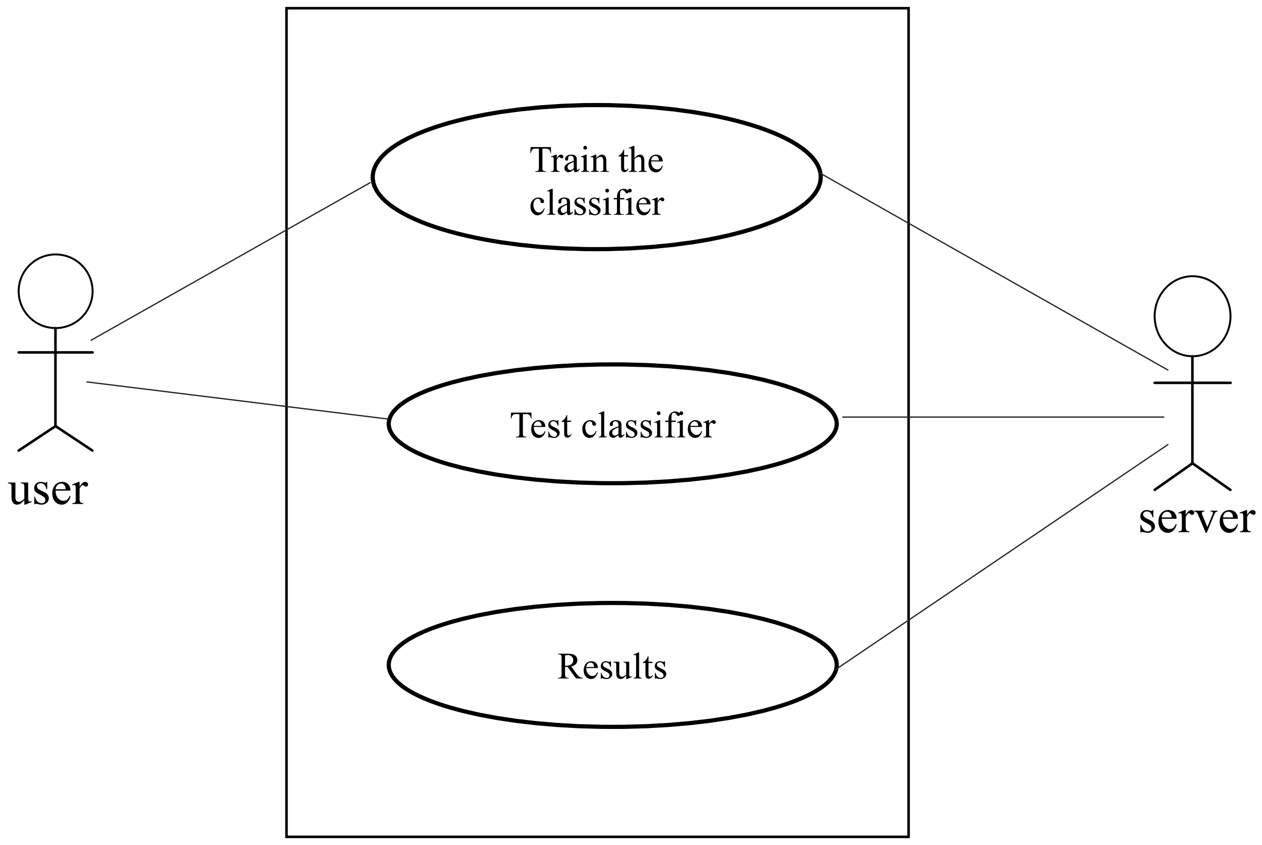

Fig 3.3. Use case diagram for classifier

The Fig 3.3 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.

3.6Data Flow Diagram

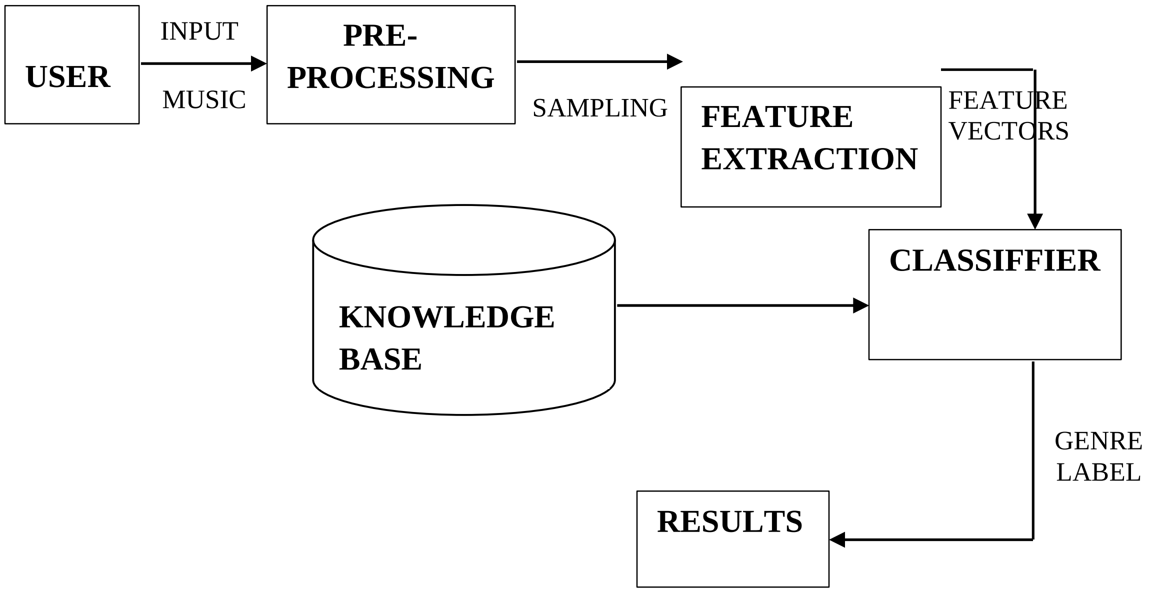

Fig 3.4. Data Flow diagram for overall system.

The above fig 3.4 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 .au 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.

3.6 Summary

This chapter discussed the high level design, design consideration, system architecture, module specification with the data flow diagram for each module of the Music genre classification based on pattern recognition system.

Chapter- 4

DETAILED DESIGN

4.1 Introduction

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.

4.2 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 Matlab, 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.

4.3 Flow Chart of system

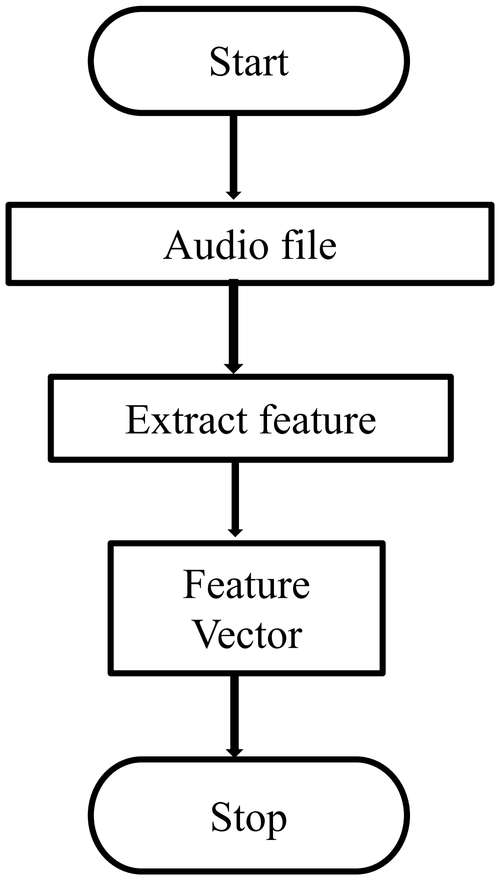

Fig 4.1 Flow chart diagram for feature extraction

The flow chart diagram 4.1 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.

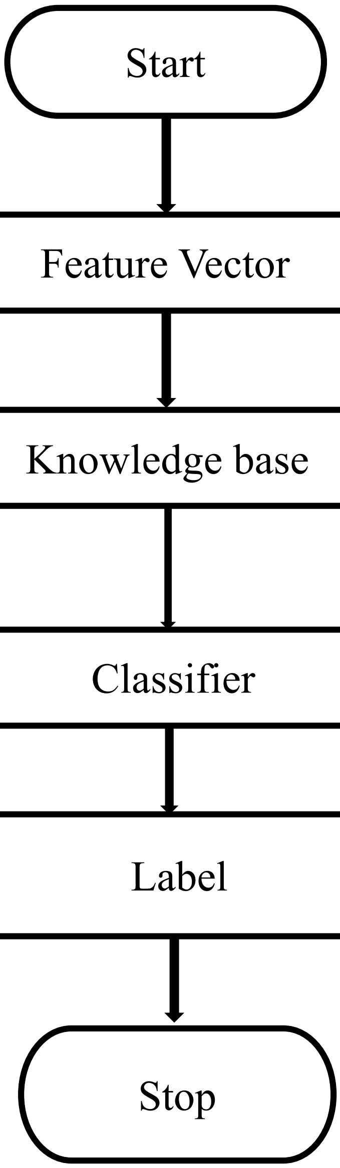

Fig 4.2 Flow chart diagram for classifier.

The flow chart diagram of fig 4.2 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.

4.4 Flow chart of Web App

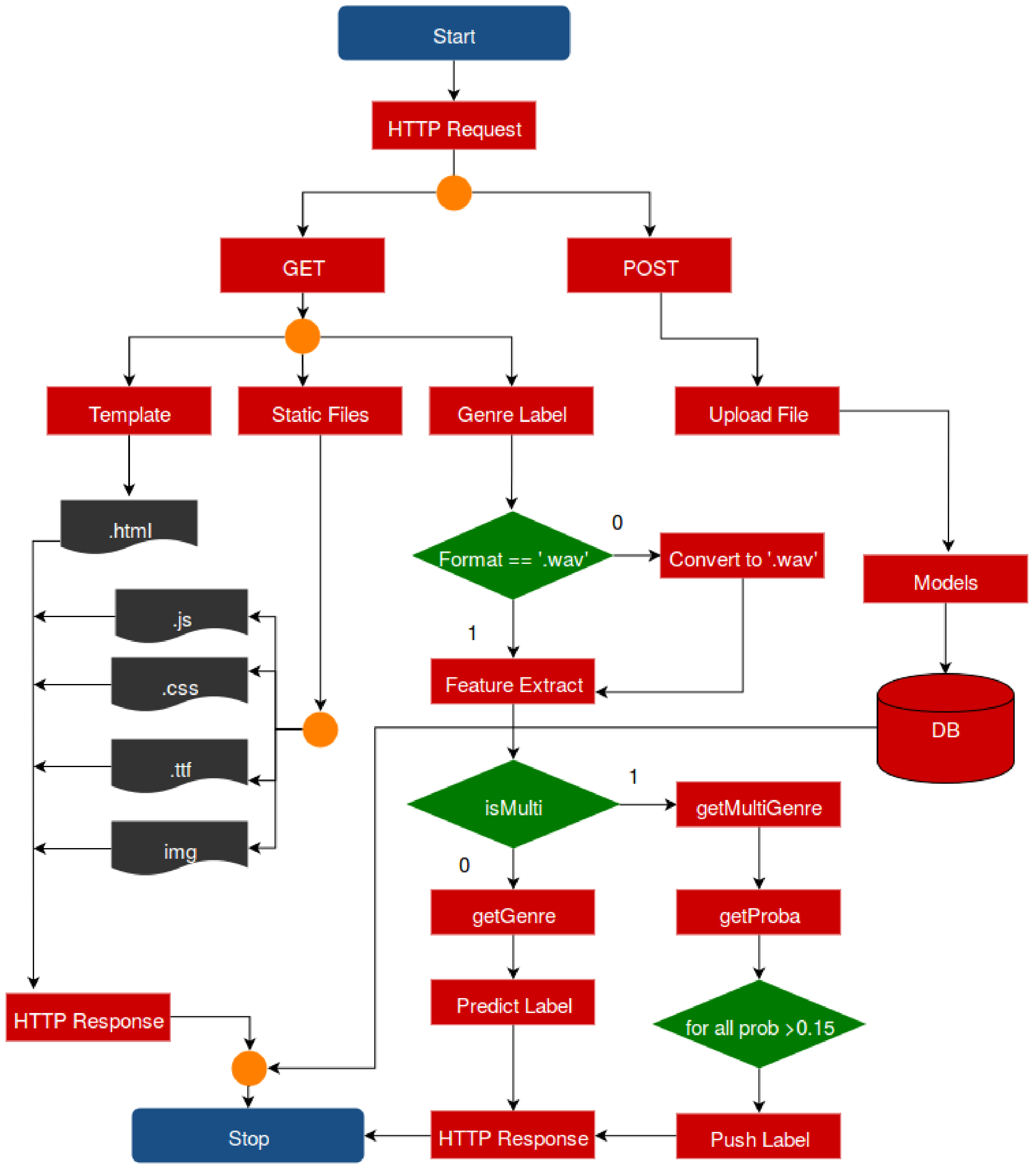

Fig 4.3 Flow chart diagram for Web App.

Figure 4.3 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.

4.5 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 manages 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.

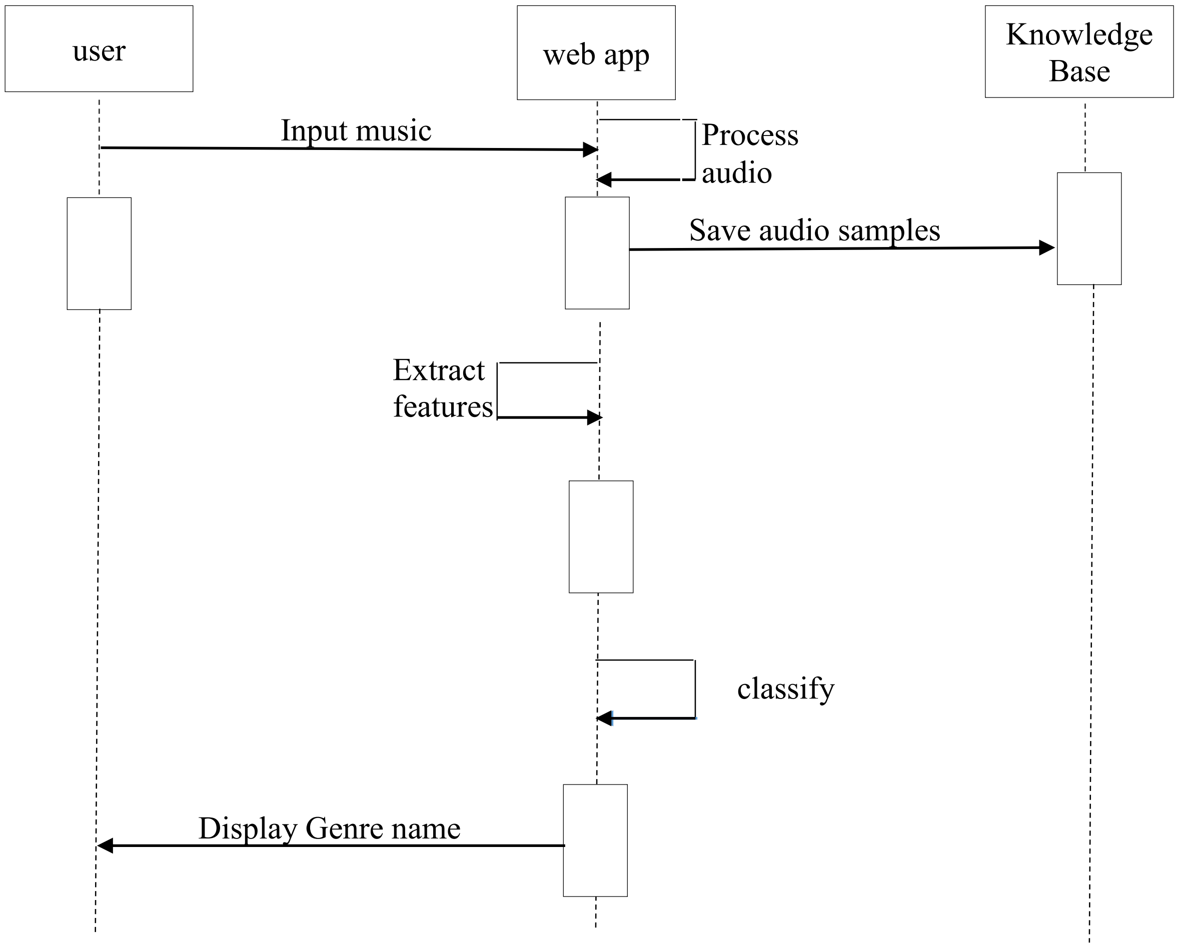

Fig 4.4. Sequence diagram for overall system of music genre classification.

4.6 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-5

IMPLEMENTATION

5.1 Programming Language Selected

5.1.1 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.

5.1.2 Matlab

We have chosen Matlab for trying out various machine learning algorithms. It helps us to choose the best algorithm for our Web App. Matlab's rich library and functional programing makes it ideal to implement classification algorithms and making prototypes.

5.2 Platform Selected

5.2.1 Linux

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

5.3 Version Control

5.3.1 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.

5.4 Prototype

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

5.4.1 Feature Extraction

We chose to extract MFCC from the audio files as the feature. For finding MFCC in

Matlab, we have used HTK MFCC MATLAB toolkit. The output will be a matrix of 13*n dimensional vector. Where n depends on the total duration of the audio. 13*(100*sec).

While feature extraction we were getting ‘nan’(not a number) and infinity in the output. This is usually caused be a division by zero or a very small number closed to 0 resulting in infinity/nan in the output. This could be a regular result or some algorithm or implementation error in the MFCC toolkit. To overcome this situation, we have set nan or infinity entries in the feature array to 0.

5.4.2 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 90% 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;
```

5.4.3 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
```

```
ff = mf;
  for i=0:(size(mm,1)-1)
   ff = [ff;diag(cf,i)]; %use diagonals
  end
t(:,end+1) = ff(:,1);
```

5.4.4 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.

  $$\{(x1, y1), (x2, y2) \ldots (xn, yn)\}$$

  Where (xi, yi) represents data pair i.

  $xi -$ feature vector

  $yi -$ target class

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

5.5 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.

5.5.1 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.

5.5.1.1 *feature:*

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.

- *flattern(x) :* Flatterns a numpy array.

5.5.1.2 *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.

5.5.1.3 *acc*

Module for finding the accuracy.

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

5.6 Pseudocode for each Module

5.6.1 Code for classification in KNN

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

5.6.2 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('\nTraining 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
```

5.6.3 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) .^ 2)
J = 1 / m * sum(-y .* log(htheta) - (1 - y) .* log(1 - htheta)) + lambda / (2 * m) * sum(theta(2:end) .^ 2);
theta = [0;theta(2:end)];
grad = (1/m) * (X' * (htheta - y) + lambda * theta);
```

5.6.4 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
```

5.6.5 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
```

```
        mm = 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("loooping----",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)
```

5.6.6 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_l = 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_l):
            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)
```

5.6.7 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},
                            xsrfHeaderName: 'X-CSRFToken',
                            xsrfCookieName: '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},
                    xsrfHeaderName: 'X-CSRFToken',
                    xsrfCookieName: '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');
                }
```

5.6.8 Back End

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

```
// Url Dispacher
// 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 HttpResponse
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'
        return response
    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)
```

5.7 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-6

SYSTEM TESTING AND EXPERIMENTATIONS

6.1 Testing

Software testing plays an important role in modifying the errors and refining the quality of the system software. Initially the program should be written and documented and also associated data structures should be designed. After all these processes are coded, software testing is started. In system testing the products functionality is checked once it is finished.

6.2 Unit Testing

Unit testing is a level of software testing where individual units/components of software are tested. The purpose is to validate that each unit of the software performs as designed.

Test cases are built in order to test and make sure that all the components within the system interact properly. The goal is to detect the error in each module. The various test cases for the modules of the project are listed below.

Testing modules are described below:

- Audio/Video conversion
- Feature extraction
- Classifier

Sl. No. Test case	1
Name of the test	Unit testing for audio converter
Sample input	Any audio/video media files except .mkv
Expected output	Media files in .wav format
Actual output	Same as expected output
Remark	Successful

Table 6.1: Unit testing of the sensor for successful test

Table 6.1 illustrate the testing for audio conversion. The audio file is successfully converted to .wav format file. The test has been conducted for successful case.

Sl. No. Test case	2
Name of the test	Unit testing for feature extraction
Sample input	.wav audio file
Expected output	Features in the dimension of 1×104 matrix
Actual output	Same as expected output
Remark	Successful

Table 6.2: Unit testing for feature extraction

Table 6.2 illustrate the testing of the feature extraction of input .wav audio file. The features are extracted from the .wav audio file. The test has been conducted for successful case.

Sl. No. Test case	3
Name of the test	Unit testing for classifier
Sample input	Extracted feature vector
Expected output	Label the genre
Actual output	Same as expected output
Remark	Successful

Table 6.3: Unit testing for classification

Table 6.2 illustrates the testing of the extracted features and labels the genres.

6.3 Experimentations

We have taken of 1000 audio tracks each 30 seconds long. There are 10 genres represented, each containing 100 tracks. Initially we have chosen ten genres for our project: blues, classical, country, disco, hip-hop, jazz, metal, pop and reggae. Our total data set was 1000 songs, of which we used 90% for training and 10% for testing and measuring results and also chosen songs randomly from dataset for random cross validation.

6.3.1 Experiment Results for KNN

6.3.1.1 Results for classification knn

Predictors Name: {1*156 cell}

Response Names: ‘y’

Class Names: [1,2,3,4,5,6,7,8,9,10]

Score Transform: ‘none’

Num Observations: 1000

Distance: ‘Euclidean’

Num Neighbors: 5

Result: Accuracy 53.009%

6.3.2 Experiment Results for Logistic Regression

Training Accuracy	75.7778
Testing Accuracy	54.000

Table 6.4: Experiments results for logistic regression

Training accuracy is the accuracy got when we used 90% of the dataset to train the classifier. Testing accuracy is the accuracy got when we used the rest 10% of the dataset as test samples.

6.3.3 Experiment Results for SVM

TRAINING	**TESTING**	**FOLD**	**TRAINING ACCURACY**	**TESTING ACCURACY**
70	30	5	99.88	62.4
90	10	5	99.86	64.00
50	50	5	99.9	60.30

Table 6.4: Experiment results for random cross validation for ten genres

The table 6.4 illustrates the accuracy on experimenting while choosing random songs from dataset.

NUMBER OF CLASSES	MAX ACCURACY	TOTAL COMBINATION CLASSES	BEST COMBINATION
2	99	44	[0,6] [1,3] [1,4] [1,6] [1,9] [2,6] [4,5] [6,8]
3	98.66	119	[4,5,6] [5,6,8]
4	92.5	209	[0,6,7,8] [1,2,4,6] [1,2,5,6] [1,4,6,7] [1,5,6,7] [1,5,6,9] [1,6,7,8] [2,5,6,8] [4,5,6,7]
5	88.0	251	[0,1,2,4,6] [0,1,2,6,7] [0,1,3,5,6] [0,1,4,6,7] [0,1,6,7,8]
6	85.0	209	[0,2,5,6,7,8] [1,2,4,6,6,7] [1,4,5,6,7,8]
7	79.7142	119	[0,1,2,4,5,6,7] [0,1,2,4,5,7,8] [0,1,2,5,6,7,8] [0,1,3,5,6,7,8] [1,2,3,5,6,7,8] [1,2,4,5,6,7,8] [1,2,4,5,6,7,9]
8	73.75	44	[0,1,2,3,4,5,6,7] [0,1,2,3,4,5,6,9] [0,1,2,3,5,6,7,8] [0,1,2,4,5,6,7,9] [0,1,4,5,6,7,8,9] [1,2,3,4,5,6,7,8]
9	70.44	9	[0,1,2,3,4,5,6,7,8] [0,1,2,3,4,5,6,7,9] [0,1,2,4,5,6,7,8,9]

Table 6.5: Experiment results for best combinations for 9:1 ratio with 5 fold

The table 6.5 illustrates the accuracy for best combination among genres by taking 90 and 10 ratios for training, testing respectively with 5 folds.

NUMBER OF CLASSES	MAX ACCURACY	TOTAL COMBINATION CLASSES	BEST COMBINATION
2	98.66	44	[1,3] [1,4] [1,6] [2,6] [6,8]
3	94.44	119	[1,6,8] [5,6,7]
4	90.167	209	[0,1,6,7] [1,2,4,6] [1,2,5,6] [1,2,6,7] [1,4,6,7] [1,5,6,8] [1,6,7,8]
5	87.068	251	[1,2,5,6,8] [1,5,6,7,8] [0,1,6,7,8]
6	83.221	209	[0,1,5,6,7,8] [1,2,4,5,6,7] [1,2,5,6,7,8]
7	76.952	119	[0,1,2,4,5,6,7] [0,1,2,5,6,7,8] [1,2,3,5,6,7,8]
8	72.916	44	[0,1,2,3,5,6,7,8] [0,1,2,4,5,6,7,8] [0,1,2,5,6,7,8,9]
9	67.85	9	[0,1,2,3,4,5,6,7,8] [0,1,2,4,5,6,7,8,9] [0,1,3,4,5,6,7,8,9] [1,2,3,4,5,6,7,8,9]

Table 6.6: Experiment results for best combinations for 7:3 ratios with 5 fold

The table 6.6 illustrates the maximum accuracy on experimenting for best combinations by taking 70 and 30 ratios for training, testing respectively with 5 folds.

6.4 Summary

This chapter deals with the unit testing, experimentations and results for each module.

Chapter-7

RESULTS

7.1 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.

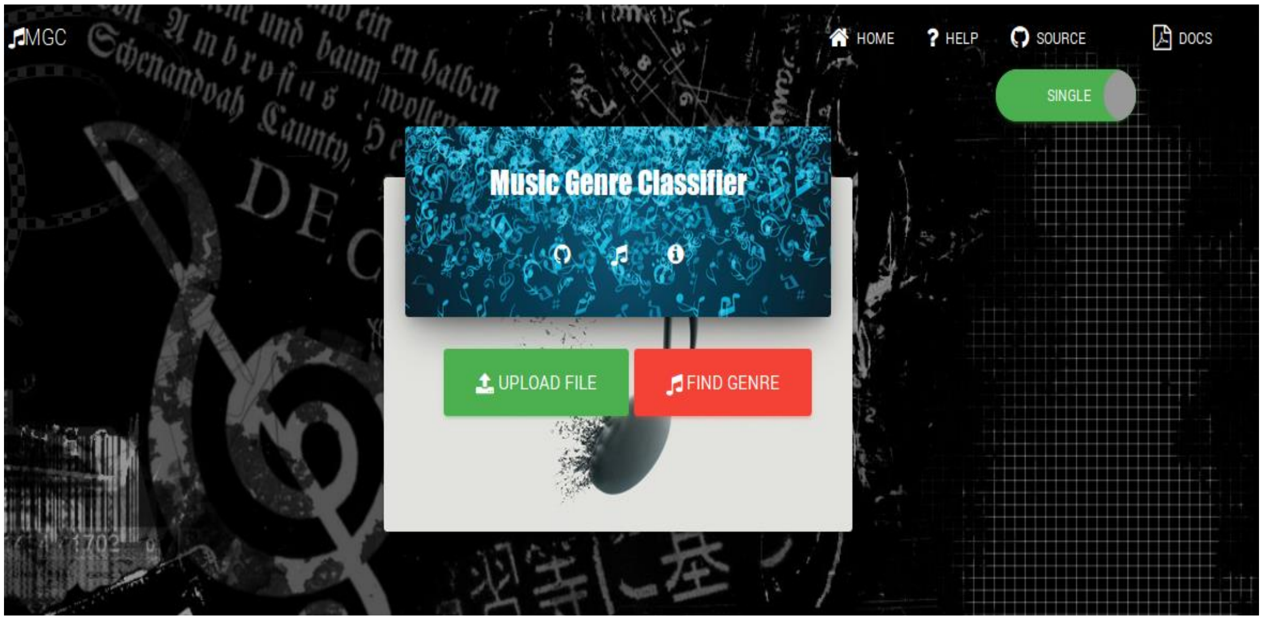

Fig 7.1: Front face of Web App

Figure 7.1 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.

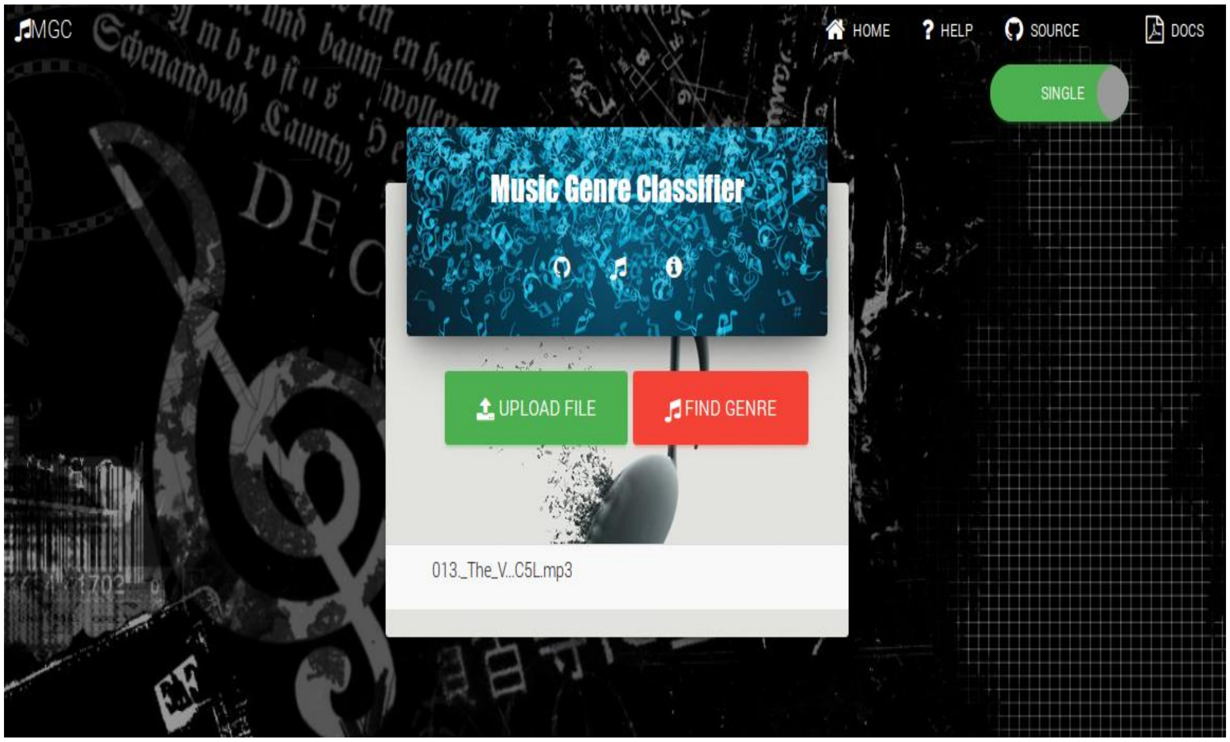

Fig 7.2: Upload the music

Figure 7.2 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.

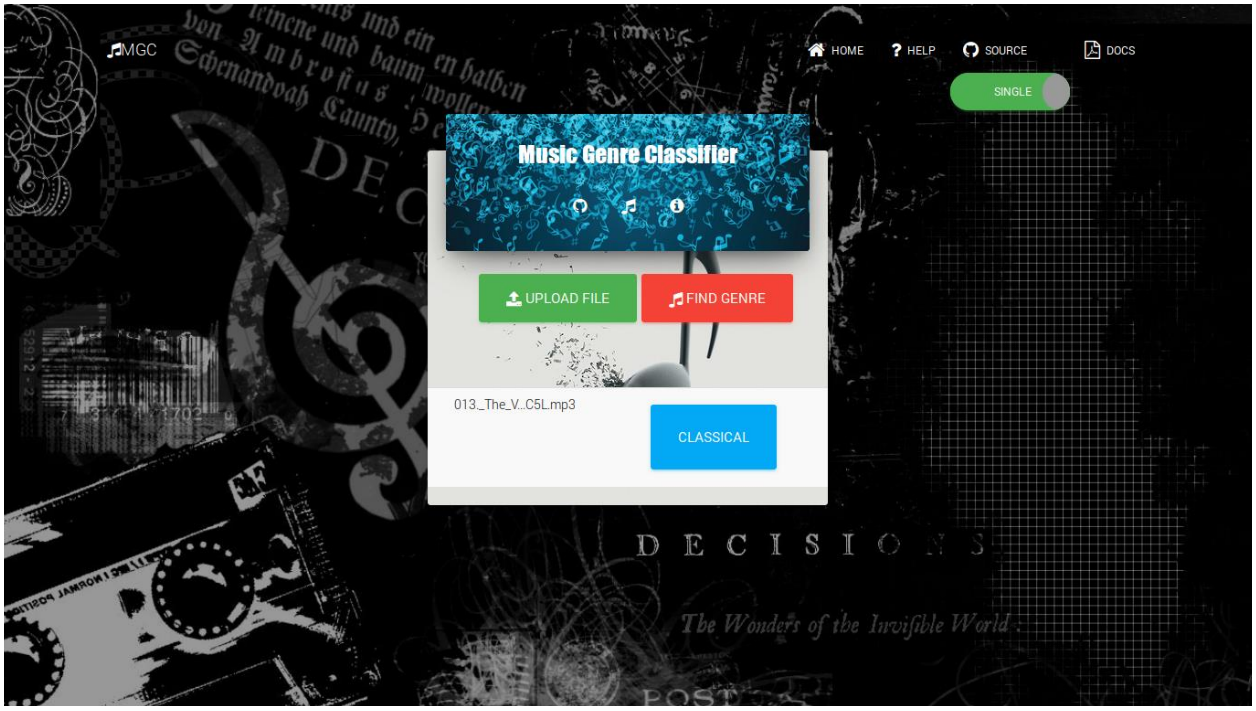

Fig: 7.3 finding the genre.

Figure 7.3 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.

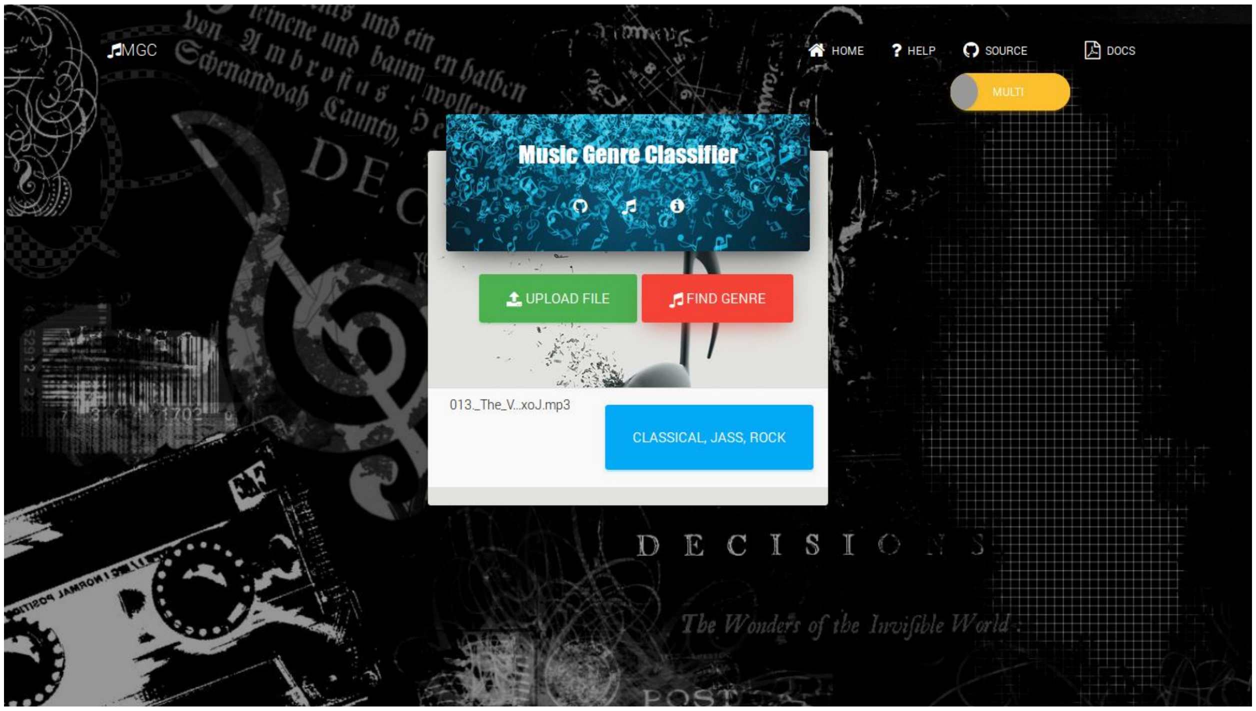

Fig: 7.4 finding multiple genres

The snapshot 7.3 illustrates the result of finding multiple genres

7.2 Summary

This chapter discusses the result of the project with snapshots.

Chapter 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 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 65% by using poly kernel SVM for 10 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.

8.2 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.

8.3 Source Code

This project is released under GPL v2.0 license. The source code is available on Github: https://github.com/indrajithi/mgc-django/

References

1. http://marsyasweb.appspot.com/download/data_sets/
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
8. https://github.com/sigurdga/django-jquery-file-upload
9. https://github.com/blueimp/jQuery-File-Upload