MUSIC RECOMMENDATION SYSTEM USING CONTENT BASED CLASSIFICATION

B.TECH PROJECT REPORT

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

THANGAL KUNJU MUSALIAR COLLEGE OF ENGINEERING

KOLLAM-691005

UNIVERSITY OF KERALA

MAY 2017

Music Recommendation system using Content Based Classification

Submitted in partial fulfilment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering

Submitted by

Faseru Oreoluwa Emmanuel (13403022)

Swaraj Krishnan R (13403052)

Under the guidance of

Prof. Niza A.K

Prof. Reena Mary George

Department of Computer Science and Engineering

T.K.M. COLLEGE OF ENGINEERING, KOLLAM
KERALA

MAY 2017

Department of Computer Science and Engineering

T.K.M. COLLEGE OF ENGINEERING, KOLLAM, KERALA

MAY 2017

C E R T I F I C A T E

This is to certify that the thesis entitled **"Music Recommendation system using Content Based Classification"** is a bonafide record of the major project done by **Faseru Oreoluwa Emmanuel (13403022), Swaraj Krishnan R (13403052)** under my supervision and guidance, in partial fulfilment for the award of Degree of Bachelor of Technology in Computer Science and Engineering from the University of Kerala for the year 2017.

Project Guide	Project Coordinator	Head of the Department
Prof.NIZA AK	**Dr. MANU J PILLAI**	**Dr. ANSAMMA JOHN**
Assistant Professor	Assistant Professor	Associate Professor & HOD
Computer Science	Computer Science	Computer Science
&Engineering	&Engineering	&Engineering

Place: Kollam

Date:

ACKNOWLEDGEMENT

I take the opportunity to express my deepest gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of this project.

First and foremost, I would like to express our heartfelt gratitude to Dr. Ansamma John, Associate Professor and Head of the Department, Computer Science and Engineering, for her valuable assistance provided during the course of the project.

I wish to place on records our ardent and earnest gratitude to our project coordinator and guide Dr. Manu J Pillai, and guides Prof.Niza A.K and Prof. Reena Mary George Dept. of Computer Science and Engineering. Their tutelage and guidance was the leading factor in translating our effort to fruition. Their prudent and perfective vision has shown light on our trail to triumph. They have played a major role in providing us with all facilities for the completion of this work.

I would like to take this opportunity to express my thanks towards the teaching and non-teaching staff in the Department of Computer Science & Engineering, TKMCE, Kollam, for their valuable help and support. I also thank all our family, friends and well-wishers who greatly helped as in our endeavour.

Faseru Oreoluwa Emmanuel Swaraj Krishnna R

ABSTRACT

With the abundance of personal computers, advances in high speed modems operating at 100 Mbps and GUI based peer-to-peer (P2P) file-sharing systems that make it simple for individuals without much computer knowledge to download their favorite music, there has been an increase of digitized music available on the Internet and on personal computers. As such, there is also a rising need to manage and efficiently search the large number of multimedia databases available online which is difficult using text searches alone.

Sound or music is a longitudinal wave phenomenon, where the pressure changes, caused by vibration of the sound source, propagate through media such as air or water. Music has been with human being in various fields serving religious, political, and/or social purposes, to name a few.Recently, many researches of modeling or measuring human feeling have been conducted to understand human emotions. However, researches on music-related human emotions have much difficulty due to the subjective perception of emotions. We selected low-level musical features along with some high level features which can help us to identify listening patterns of the user.

Widespread use of mp3 players and cell-phones and availability of music on these devices according to user demands increased the need for more accurate Music Information Retrieval (MIR) Systems. Music recommendation is one of the subtasks of MIR Systems and it involves finding music that suits a personal taste . Audioscrobbler , iRate, MusicStrands , and inDiscover are some of the music recommendation systems today.We implemented personalized music recommendation system based on the low-level features selected by the proposed method. This recommendation system eliminates scalability problem of tag-based music recommendation systems, by employing automatically extracted low-level features of music which trigger emotions. Several feature extraction methods including low-level parameters such as zero-crossing rate, signal bandwidth, spectral centroid, root mean square level, band energy ratio, delta spectrum, psychoacoustic features, MFCC and temporal envelopes have been employed for audio classification

Usually music recommendation systems follow a collaborative filtering or a content based approach. Collaborative filtering is the approach used in Amazon, a new item is rated by some users and the item is recommended to other users based on the rating of the previous users. The disadvantage of the collaborative approach is that when a new item arrives, it has to be rated by someone in order to be used for the other users. The recommendation system we build is content based, In the content-based approach, based on some form of distance between the items already rated by the user and a new item.

Contents

List of Figures

Chapter 1

INTRODUCTION

With the abundance of personal computers, advances in high speed modems operating at 100 Mbps and GUI based peer-to-peer (P2P) file-sharing systems that make it simple for individuals without much computer knowledge to download their favourite music, there has been an increase of digitized music available on the Internet and on personal computers. As such, there is also a rising need to manage and efficiently search the large number of multimedia databases available online which is difficult using text searches alone.
Sound or music is a longitudinal wave phenomenon, where the pressure changes, caused by vibration of the sound source, propagate through media such as air or water. Music has been with human being in various fields serving religious, political, and/or social purposes, to name a few. Recently, many researches of modeling or measuring human feeling have been conducted to understand human emotions. However, researches on music-related human emotions have much difficulty due to the subjective perception of emotions. We selected low-level musical features along with some high level features which can help us to identify listening patterns of the user. [1].

1.1 Motivation

While just a few years ago people were mainly looking for and buying music from local CD stores, the way people search for music and the way people consumer music has radically changed. Thanks to portable music players like the iPod and mobile Internet access music can be listened to everywhere and at any time. It is not only the way people consume music that has completely changed, but also the variety of songs that is available. Today even the most obscure type of music is available for download from modern music online platforms. Thus, the problem is no longer to store or transmit digital music, but to assist users and help them find music they like. Assisting users in their music search is the task of a music recommender system. Many of the popular music services today have a music recommender system attached. However, the majority of these systems is based on conventional collaborative filtering algorithms that analyze for example user ratings. This thesis, in contrast, focuses on another well-known approach in Music Information Retrieval (MIR), namely content-based

music recommendation. The idea of a content-based music recommender systems is that the machine itself can listen and understand music on its own[2].The ultimate goal would be that the machine can then act as a musical expert and assist the user in her search task and point her to new interesting songs like a good friend would do. Furthermore, the machine could then help to automatically organize and visualize a music collection according to the user's taste or mood. From a technical point of view such content-based music recommender systems significantly differ from other recommendation techniques. Instead of accumulating e.g. user ratings or information from the web, they directly analyze the music audio signals and generate recommendations based on the information extracted from the signals themselves.

1.2 Problem Definition

In this project, the problem is to recommend songs to users from an online database based on similarity to the songs in the users playlists. The required features will be extracted from the users songs and used for recommending new songs to the user. This approach will be better than other approaches where tags and mannual labor is used for recommending and classifying music.

1.3 Existing Work

1.3.1 Spotify

The current recommender system used at Spotify is based solely on artist data. Using data about related artists from All Music Guide and combining this with history of played tracks, other artists are recommended. This algorithm is very fast and easy to implement, but suffers from several drawbacks:

- The data from All Music Guide only covers a minor part of the set of artists in Spotify and is pure editorial data, not being possible to extend automatically over new items.
- No conclusions regarding structure of artist similarities are drawn by the system, which is completely relying on the data by All Music Guide.
- Recommendations are always artists, never albums or tracks.

1.3.2 Others

- Last.fm: is arguably the current state of the art collaborative filtering implementation for music, having its roots in the software Audoscrobbler. It seems to be entirely artist-based, not looking at individual tracks.
- Pandora: bases its recommendations on an extremely ambitious human-aided feature extraction. This feature extraction goes under the name Music Genome Project, where

trained people employed by Pandora work with the task of classifying each track in terms of several hundred variables.

- Genius: is the music recommendation service in Itunes, the music software from Apple. It is not clear whether it is based on collaborative filtering or feature classifications, but most likely the former.
- Mufin, a feature-based recommender system developed by the Fraunhofer Institute, also known as the inventors of the MP3 format. Also ZuKool deserves to be mentioned, as well as Musicovery, which has an unorthodox control panel where users are asked to input their mood as a point on a two-dimensional surface.

1.4 Propsed Work

The proposed application will have all the basic features of a music player:

- Loading music playlists.
- Mp3 track playback.
- Music album grouping.
- Music recommendations based on user preference.

We plan to develop an algorithm that should be able to classify any input audio clip into a musical genre, based on certain audio features and its training on a test-database. This algorithm falls under the class of content based identification (CBID) methods, wherein the identification and/or classification of data is done based on the characteristic content of the data, hereafter called features.

Characteristics or features of sound source can be classified into three layers as follows: Low-level acoustic feature layer, Mid-level audio signature layer which can be used to separate distinct signal objects and High-level semantic layer which can be used to classify different class of sound sources [1]. In our experiment, six high-level features such as rhythm, tonal, timbre, dynamics, fluctuation, and spectral, twenty-eight mid-level features such as RMS, peak, centroid, tempo, zero-cross, etc., and eight hundred-ninety low-level feature values are extracted from the songs in the database.

After feature selection, the training dataset is grouped according to increasing order of Euclidean distance to identify songs that are similar to one another. They are now labelled into the various classes.

Chapter 2

LITERATURE REVIEW

Low-level audio features[1] are measurements of audio signals that contain information about a musical work and music performance. They also contain extraneous information due to the difficulty of precisely measuring just a single aspect of music, so there is a tradeoff between the signal-level description and the high level music concept that is encoded. In general, low-level audio features are segmented[2] in three different ways: frame based segmentations (periodic sampling at 10 ms-1000 ms intervals), beat-synchronous segmentations (features are aligned to musical beat boundaries), and statistical measures that construct probability distributions out of features (bag of features models). Many low-level audio features are based on the short-time spectrum of the audio signal. Fig. 2 illustrates how some of the most widely used low-level audio features are extracted from a digital audio music signal using a windowed fast Fourier transform (FFT) as the spectral extraction step. Both frame-based and beat-based windows.

- Short-Time Magnitude Spectrum[3] Many low-level audio features use the magnitude spectrum as a first step for feature extraction because the phase of the spectrum is not as perceptually salient for music as the magnitude. This is generally true except in the detection of onsets and in phase continuation for sinusoidal components.

- Constant-Q/Mel Spectrum: The ears response to an acoustic signal is logarithmic in frequency and uses non-uniform frequency bands, known as critical bands, to resolve close frequencies into a single band of a given center frequency. Many systems represent the constant bandwidth critical bands using a constant-Q transform, where the Q is the ratio of bandwidth to frequency. It is typical to use some division of the musical octave for the frequency bands, such as a twelfth, corresponding to one semitone in Western music, but it is also common to use more perceptually motivated frequency spacing for band centers.

- Onset Detection: Musical events are delineated by onsets; a note has an attack followed by sustain and decay portions. Notes that occur simultaneously in music are often actually scattered in time and the percept is integrated by the ear-brain system. Onset detection is concerned with marking just the beginnings of notes.

- Mel/Log-Frequency Cepstral Coefficients: Mel-frequency cepstral coefficients (MFCC) take the logarithm of the Mel magnitude spectrum and de-correlate the resulting values using a Discrete Cosine Transform. This is a real-valued implementation of the complex cepstrum in signal processing. The effect of MFCCs is to organize sinusoidal modulation of spectral magnitudes by increasing modulation frequency in a real-valued array. Values at the start of the array correspond to long wave spectral modulation and therefore represent the projection of the log magnitude spectrum onto a basis of formant peaks. Values at the end of the MFCC array are the projection coefficients of the log magnitude spectrum onto short wavelength spectral modulation, corresponding to harmonic components in the spectrum. It is usual to use about 20 MFCC coefficients, therefore representing the formant peaks of a spectrum; this extraction corresponds, in part, to musical timbre the way the audio sounds other than its pitch and rhythm.

Altogether, 28 such features are extracted and used for the music classification process. After feature extraction, the training data has to be classified according to their similarity. Unlike in classification, the class label (or group ID) of each element is unknown. You need to discover these groupings. Given a large number of songs and many attributes describing the mp3 files, it can be very costly or even infeasible to have a human study the data and manually come up with a way to partition the songs into strategic groups. You need a clustering tool to help.Clustering is the process of grouping a set of data objects into multiple groups or clusters so that objects within a cluster have high similarity, but are very dissimilar to objects in other clusters. Dissimilarities and similarities are assessed based on the attribute values describing the objects and often involve distance measures[2].

2.1 Music Cluster Analysis

Cluster analysis or simply clustering is the process of partitioning a set of data objects (or observations) into subsets. Each subset is a cluster, such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters. The set of clusters resulting from a cluster analysis can be referred to as a clustering. In this context, different clustering methods may generate different clusterings on the same data set. The partitioning is not performed by humans, but by the clustering algorithm. Hence, clustering is useful in that it can lead to the discovery of previously unknown groups within the data.

As a data mining function, cluster analysis can be used as a standalone tool to gain insight into the distribution of data, to observe the characteristics of each cluster, and to focus on a particular set of clusters for further analysis. Alternatively, it may serve as a preprocessing step for other algorithms, such as characterization, attribute subset selection, and classification, which would then operate on the detected clusters and the selected attributes or features.

The simplest and most fundamental version of cluster analysis is partitioning, which organizes the objects of a set into several exclusive groups or clusters. To keep the problem specification concise, we can assume that the number of clusters is given as background knowledge. This parameter is the starting point for partitioning methods. Formally, given a data set, D,

of n objects, and k, the number of clusters to form, a partitioning algorithm organizes the objects into k partitions .k $_n$/, $where each partition represents a cluster. The clusters are formed to optimize an obje$

k-Means

he k-means algorithm defines the centroid of a cluster as the mean value of the points within the cluster. It proceeds as follows. first, it randomly selects k of the objects in D, each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the Euclidean distance between the object and the cluster mean. The k-means algorithm then iteratively improves the within-cluster variation. For each cluster, it computes the new mean using the objects assigned to the cluster in the previous iteration. All the objects are then reassigned using the updated means as the new cluster centers. The iterations continue until the assignment is stable, that is, the clusters formed in the current round are the same as those formed in the previous round.

Algorithm: k-means. The k-means algorithm for partitioning, where each clusters center is represented by the mean value of the objects in the cluster. Input: k: the number of clusters, D: a data set containing n objects. Output: A set of k clusters. Method:

1. arbitrarily choose k objects from D as the initial cluster centers.
2. repeat.
3. (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster.
4. update the cluster means, that is, calculate the mean value of the objects for each cluster.
5. untill no change.

After clustering, each cluster is labelled.

2.2 Support Vector Machines

The support vector machine (SVM) is a supervised classification system that minimizes an upper bound on its expected error. It attempts to find the hyper-plane separating two classes of data that will generalize best to future data. Such a hyper-plane is the so called maximum margin hyper-plane, which maximizes the distance to the closest points from each class. More concretely, given data points (x0...xn) and class labels -1,1, any hyper-plane separating the two data classes has the form :

$$\vec{w}.\vec{x_i} - b \geq 1 \; if \; class = 1 \tag{2.1}$$

$$\vec{w}.\vec{x_i} - b \leq -1 \; class = -1 \tag{2.2}$$

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. The basic idea of SVM is finding the best separator function (classifier/hyperplane) to separate the two kinds of objects. The best hyperplane is the hyperplane which is located in the middle of two object. Finding the best hyperplane is equivalent to maximize margin between two different sets of object.

In real world, most classification tasks are not that simple to solve linearly. More complex structures are needed in order to make an optimal separation, that is, correctly classify new objects (test cases) on the basis of the examples that are available (train cases). Kernel method is one of method to solve this problem. Kernels are rearranging the original objects using a set of mathematical functions. The process of rearranging the objects is known as mapping (transformation). The kernel function, represents a dot product of input data points mapped into the higher dimensional feature space by transformation.

Chapter 3

MATERIALS AND METHODOLOGY

For any project, the tools and methods are considered as the vital components for its making. For a clear understanding of the project different tools and algorithms that have been used in this project are discussed below:

3.1 Python

Python is a general purpose, high level programing language whose design philosophy emphasizes good readability. It supports both functional and object oriented programming styles. Pythons syntax allows programmers to express concepts in fewer lines of code than would be possible in languages like C and the language provides constructs intended to enable clear programs on both a small and large scale. Python is a programing language that lets you work more quickly and integrate your system more effectively. Python in small and has clean source codes. Standard library is full of useful modules.

3.1.1 Features of Python

- *Simple*

  Python is a simple and minimalistic language. Reading a good python program feels almost like reading English, although very strict English. This pseudo code nature of python one of its greatest strengths. It allows you to concentrate on the solution to the problem than to the language itself. Python has an extraordinary simple syntax.

- *High Level Language*

  When you write programs in python, you never need to bother about the low level details such as managing the memory used by your program etc.

- *Powerful*

Python is a very powerful programming language, it has a wide variety of data types, functions, control statements, decision making statements, etc.

- *Object oriented Programming language*

  Python is an object oriented programming language which supports polymorphism, inheritance, encapsulation, abstraction.

- *Interpreted*

  Internally, python converts the source code into an intermediate form called byte code and then translates this into the native language of your computer and then runs it. All this, actually, makes python much easier since you dont have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes your python programs much more portable, since you can just copy your python program on to another computer and it just works.

- *Compiler based* If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use it from your python program.

3.2 Essentia

Essentia is an open-source C++ library with Python bindings for audio analysis and audio-based music information retrieval. It is released under the Affero GPLv3 license and is also available under proprietary license upon request. The library contains an extensive collection of reusable algorithms which implement audio input/output functionality, standard digital signal processing blocks, statistical characterization of data, and a large set of spectral, temporal, tonal and high-level music descriptors.

Essentia is not a framework, but rather a collection of algorithms (plus some infrastructure) wrapped in a library. It is designed with a focus on the robustness, performance and optimality of the provided algorithms, including computational speed and memory usage, as well as ease of use. The flow of the analysis is decided and implemented by the user, while Essentia is taking care of the implementation details of the algorithms being used.

3.2.1 Features Of Essentia

- Flexible and easily extendable algorithms for common audio analysis processes and audio and music descriptors.
- Linux, Mac OS X, Windows, iOS, Android, and JavaScript.
- Pythons scientific environment and command-line audio analysis tools.
- Optimized for computational speed.

3.3 Scikit-learn

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

3.4 Pyqt

PyQt is a set of Python v2 and v3 bindings for The Qt Company's Qt application framework and runs on all platforms supported by Qt including Windows, OS X, Linux, iOS and Android. PyQt5 supports Qt v5. PyQt4 supports Qt v4 and will build against Qt v5. The bindings are implemented as a set of Python modules and contain over 1,000 classes. Qt is more than a GUI toolkit. It includes abstractions of network sockets, threads, Unicode, regular expressions, SQL databases, SVG, OpenGL, XML, a fully functional web browser, a help system, a multimedia framework, as well as a rich collection of GUI widgets.

3.5 SQL

Structured Query Language is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS). Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

3.5.1 Features Of SQL

- High Speed:-SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently.
- Well Defined Standards Exist:-SQL databases use long-established standard,which is being adopted by ANSI ISO. Non-SQL databases do not adhere to any clear standard.
- No Coding Required:-Using standard SQL it is easier to manage database systems without having to write substantial amount of code.
- Emergence of ORDBMS:-Previously SQL databases were synonymous with relational database. With the emergence of Object Oriented DBMS, object storage capabilities are extended to relational databases.

Chapter 4

DESIGN AND IMPLEMENTATION

In this work, the focus was on identifying features that can establish a relationship between different songs present in user's library. The project consists of three phases extracting the features, clustering the songs and recommending from an online database based on the clusters. We experimented with different algorithms and compared those results to choose the best algorithm.

4.1 Implentation Of Feature Extraction

The set of features extracted is critical as they need to be strong enough to clearly separate the classes of signals. This procedure requires perceptual features that model the human auditory system. A library called essentia was used in our project to provide an abstraction of feature extraction process.

Essentia is an open-source C++ library with Python bindings for audio analysis and audio-based music information retrieval. It is released under the Affero GPLv3 license and is also available under proprietary license upon request. The library contains an extensive collection of reusable algorithms which implement audio input/output functionality, standard digital signal processing blocks, statistical characterization of data, and a large set of spectral, temporal, tonal and high-level music descriptors.

The following algorithms from essentia were used in the feature extraction:

- Audio file input/output: ability to read and write nearly all audio file formats (wav, mp3, ogg, flac, etc).
- Statistical descriptors: median, mean, variance, power means, raw and central moments, spread, kurtosis, skewness, flatness.
- Time-domain descriptors: duration, loudness, LARM, Leq, Vickers' loudness, zero-crossing-rate, log attack time and other signal envelope descriptors.
- Spectral descriptors: Bark/Mel/ERB bands, MFCC, GFCC, LPC, spectral peaks, complexity, rolloff, contrast, HFC, inharmonicity and dissonance.

- Tonal descriptors: Pitch salience function, predominant melody and pitch, HPCP (chroma) related features, chords, key and scale, tuning frequency.
- Rhythm descriptors: beat detection, BPM, onset detection, rhythm transform, beat loudness

From the above features a number of features were selected and was used in clustering and classification process.

4.2 Implementation Of Recommendation System

In this section, we present a personalized music recommendation system, implemented using the proposed low-level features. This recommendation system eliminates scalability problem of tag-based music recommendation systems, by employing automatically extracted low-level features of music.

4.2.1 Algorithm

The features extracted were made into a vector of size 68 by including their mean, standard variance values. The Overall Allgortithm folowed is:

1. Start
2. Find the folder having users music files.
3. Use essentia to extract features.
4. Normalize the extracted values.
5. Use k-means algorith to classify the songs in to k clusters.
6. Assign a class label to each cluster created. Train the SVM model with clusters created.
7. Fetch song descriptions from online database.
8. For each data item in the online database classify them to a cluster using SVM.
9. Read a Class Label as input and return all the songs in database that are having the same class label.
10. If the user requests for a song from the online database, stream that song from dropbox.
11. Stop

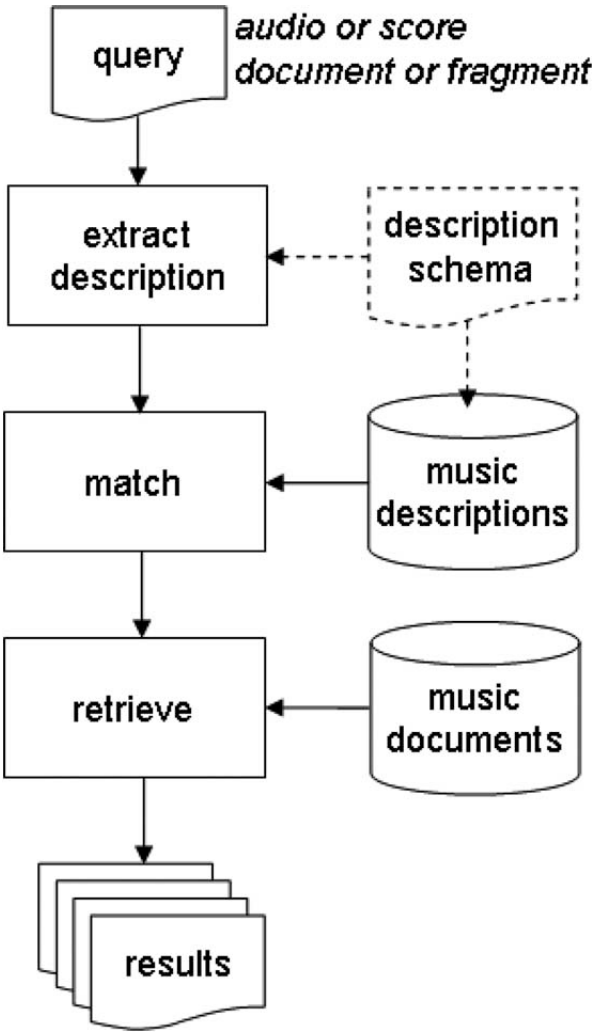


Figure 4.1: music recommendation system

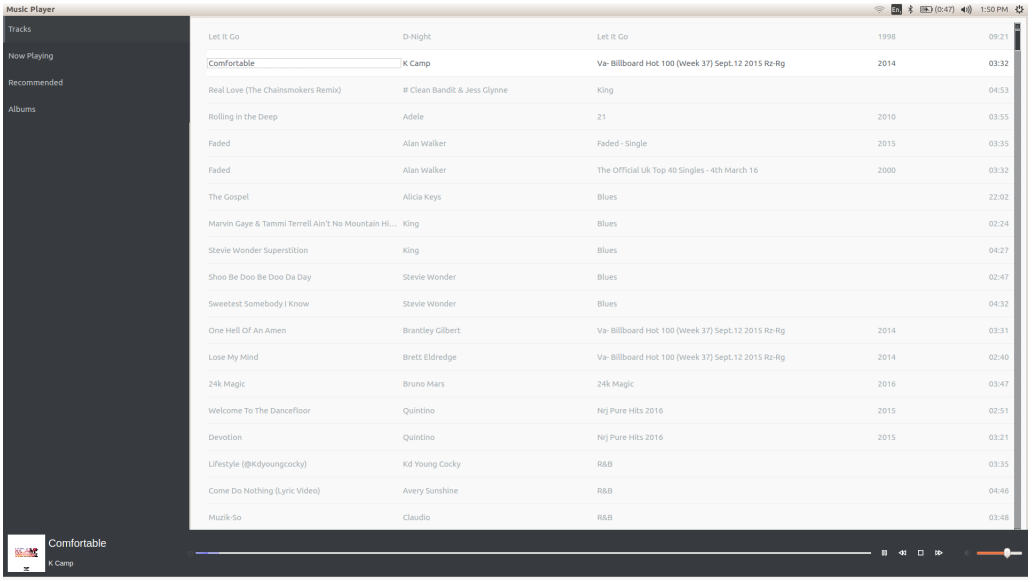
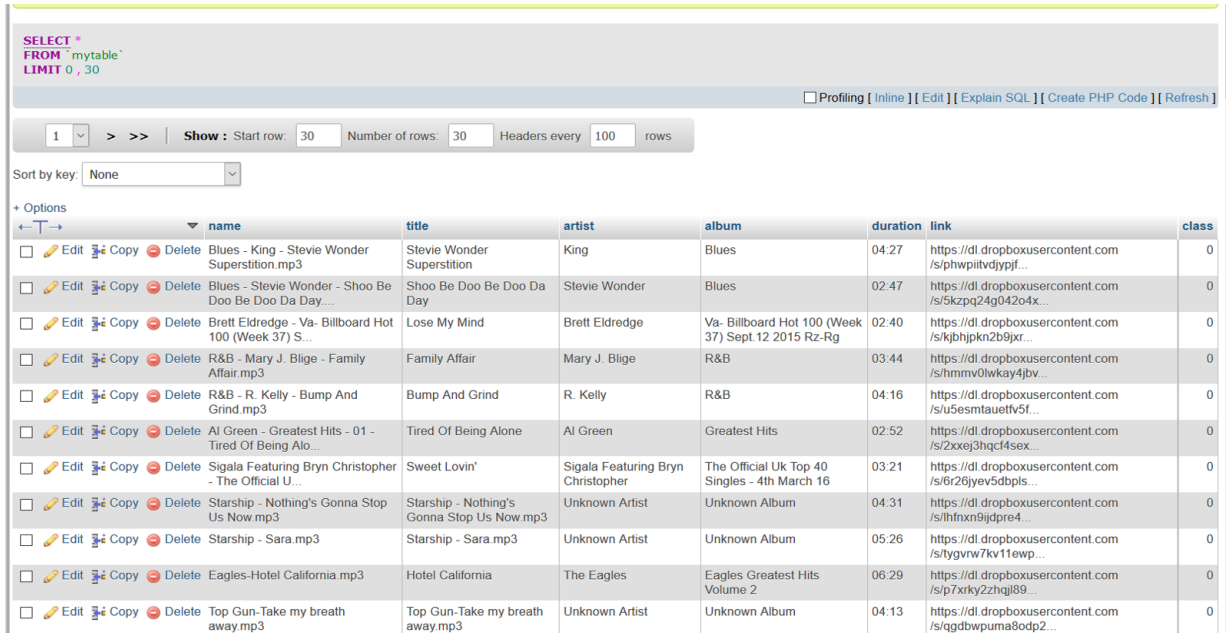


Figure 4.2: Music Player Interface

4.3 Implementation Of Database

A dropbox account is created for storage of all songs. A shareable link is generated for each song to allow for streaming in the music application. Some attributes of the songs are extracted from the metadata for display. These attributes are song title, artist name, album, duration and cluster number. All these attributes are collected and stored in a music table in a mysql database.

When a query is made to recommend, songs from the corresponding cluster can be loaded into the player and streamed for listening.



```
SELECT *
FROM `mytable`
LIMIT 0 , 30
```

Profiling [Inline] [Edit] [Explain SQL] [Create PHP Code] [Refresh]

1 > >> | Show : Start row: 30 Number of rows: 30 Headers every 100 rows

Sort by key: None

+ Options

	name	title	artist	album	duration	link	class
Edit Copy Delete	Blues - King - Stevie Wonder Superstition.mp3	Stevie Wonder Superstition	King	Blues	04:27	https://dl.dropboxusercontent.com/s/phwpiitvdjypjf...	0
Edit Copy Delete	Blues - Stevie Wonder - Shoo Be Doo Be Doo Da Day....	Shoo Be Doo Be Doo Da Day	Stevie Wonder	Blues	02:47	https://dl.dropboxusercontent.com/s/5kzpq24g042o4x...	0
Edit Copy Delete	Brett Eldredge - Va- Billboard Hot 100 (Week 37) S...	Lose My Mind	Brett Eldredge	Va- Billboard Hot 100 (Week 37) Sept.12 2015 Rz-Rg	02:40	https://dl.dropboxusercontent.com/s/kjbhjpkn2b9jxr...	0
Edit Copy Delete	R&B - Mary J. Blige - Family Affair.mp3	Family Affair	Mary J. Blige	R&B	03:44	https://dl.dropboxusercontent.com/s/hmmv0lwkay4jbv...	0
Edit Copy Delete	R&B - R. Kelly - Bump And Grind.mp3	Bump And Grind	R. Kelly	R&B	04:16	https://dl.dropboxusercontent.com/s/u5esmtauetfv5f...	0
Edit Copy Delete	Al Green - Greatest Hits - 01 - Tired Of Being Alo...	Tired Of Being Alone	Al Green	Greatest Hits	02:52	https://dl.dropboxusercontent.com/s/2xxej3hqcf4sex...	0
Edit Copy Delete	Sigala Featuring Bryn Christopher - The Official U...	Sweet Lovin'	Sigala Featuring Bryn Christopher	The Official Uk Top 40 Singles - 4th March 16	03:21	https://dl.dropboxusercontent.com/s/6r26jyev5dbpls...	0
Edit Copy Delete	Starship - Nothing's Gonna Stop Us Now.mp3	Starship - Nothing's Gonna Stop Us Now.mp3	Unknown Artist	Unknown Album	04:31	https://dl.dropboxusercontent.com/s/lhfnxn9ijdpre4...	0
Edit Copy Delete	Starship - Sara.mp3	Starship - Sara.mp3	Unknown Artist	Unknown Album	05:26	https://dl.dropboxusercontent.com/s/tygvrw7kv11ewp...	0
Edit Copy Delete	Eagles-Hotel California.mp3	Hotel California	The Eagles	Eagles Greatest Hits Volume 2	06:29	https://dl.dropboxusercontent.com/s/p7xrky2zhqjl89...	0
Edit Copy Delete	Top Gun-Take my breath away.mp3	Top Gun-Take my breath away.mp3	Unknown Artist	Unknown Album	04:13	https://dl.dropboxusercontent.com/s/qgdbwpuma8odp2...	0

Figure 4.3: Online Database

Chapter 5

RESULT AND DISCUSSION

The main aim of this project is to implement a music recommendation system that gives a high accuracy and efficiency in the results provided to users. For this reason, the performance of several machine learning algorithms will be compared to determine which would be best.

5.1 Evaluvation Techniques

To test the three algorithms, svm, nave bayes and knn, a process known as k-fold cross validation is used. In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

5.2 Results

Both Kmeans and Kmedois were able to seperate songs into clusters based on there characteristics.

A series of accuracy tests were conducted on the clusters created with k-means and k-mediod clustering approach separately. Accuracy was determined by the ratio of correct predictions to the total no of samples. The total accuracy percentage is calculated and the average error was documented for each test. Each list of rated songs used for training was put through 5-fold cross validation.

From the results gotten from the validation performed, it was determined that using svm with the k-means cluster dataset had the best accuracy. This was implemented in the music player application for recommending the songs to users.

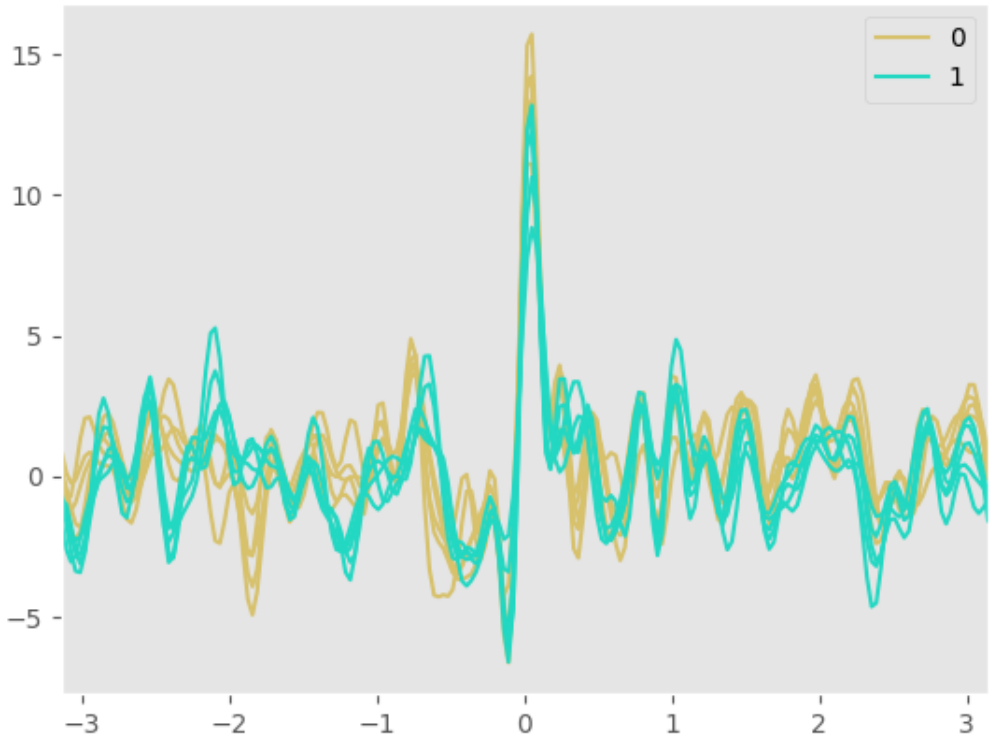


Figure 5.1: Results Two clusters one indicating metal songs and other indicating blues

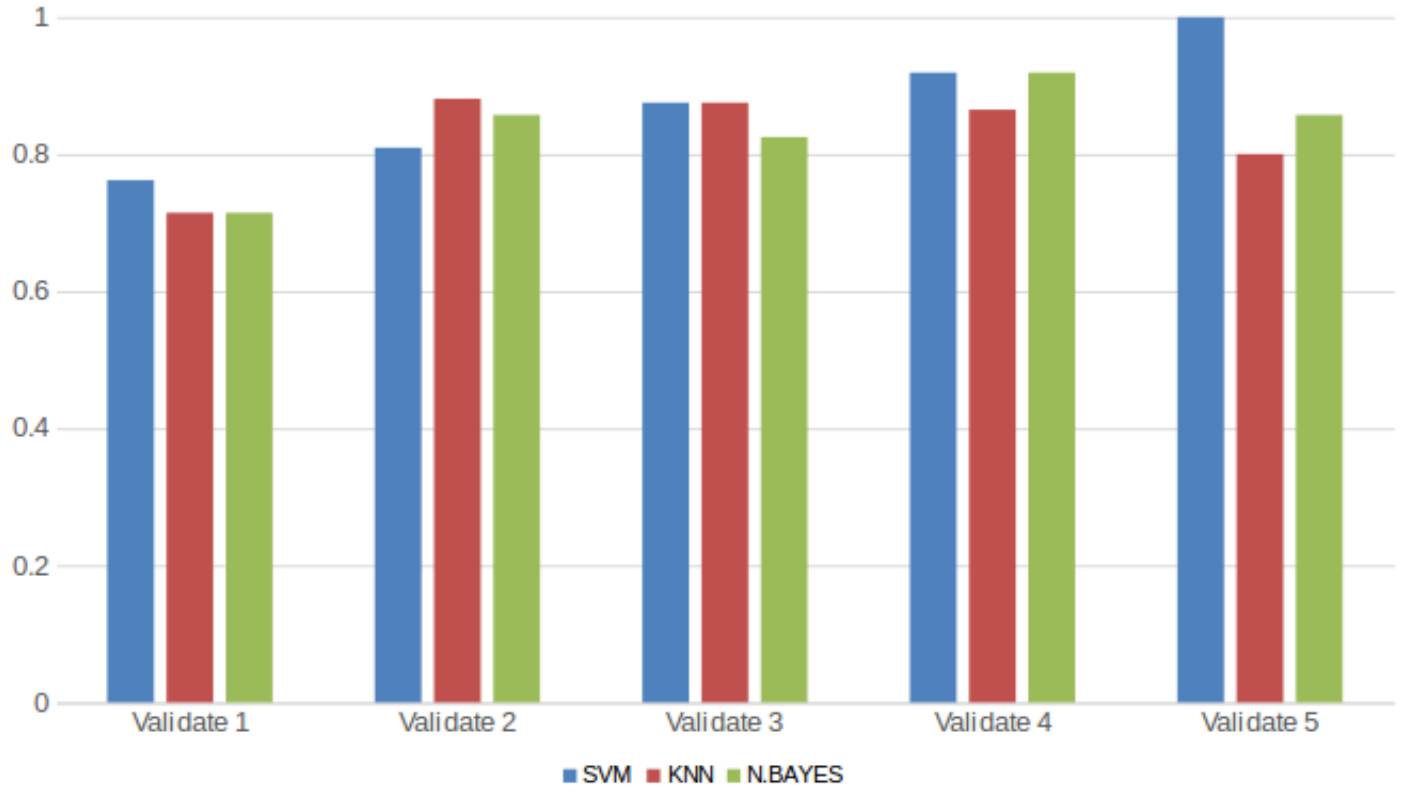


Figure 5.2: Results K-mean

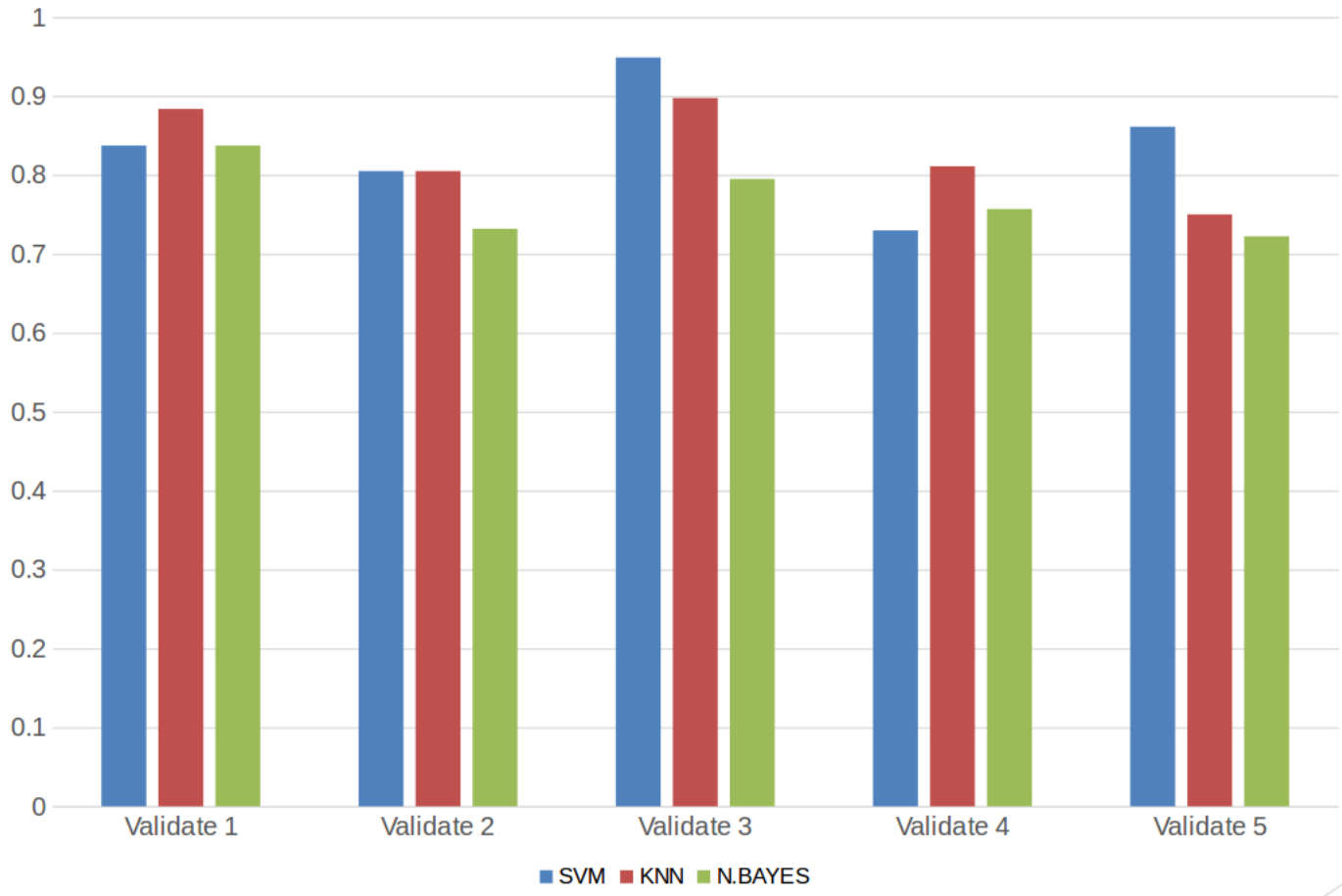


Figure 5.3: Results K-medoid

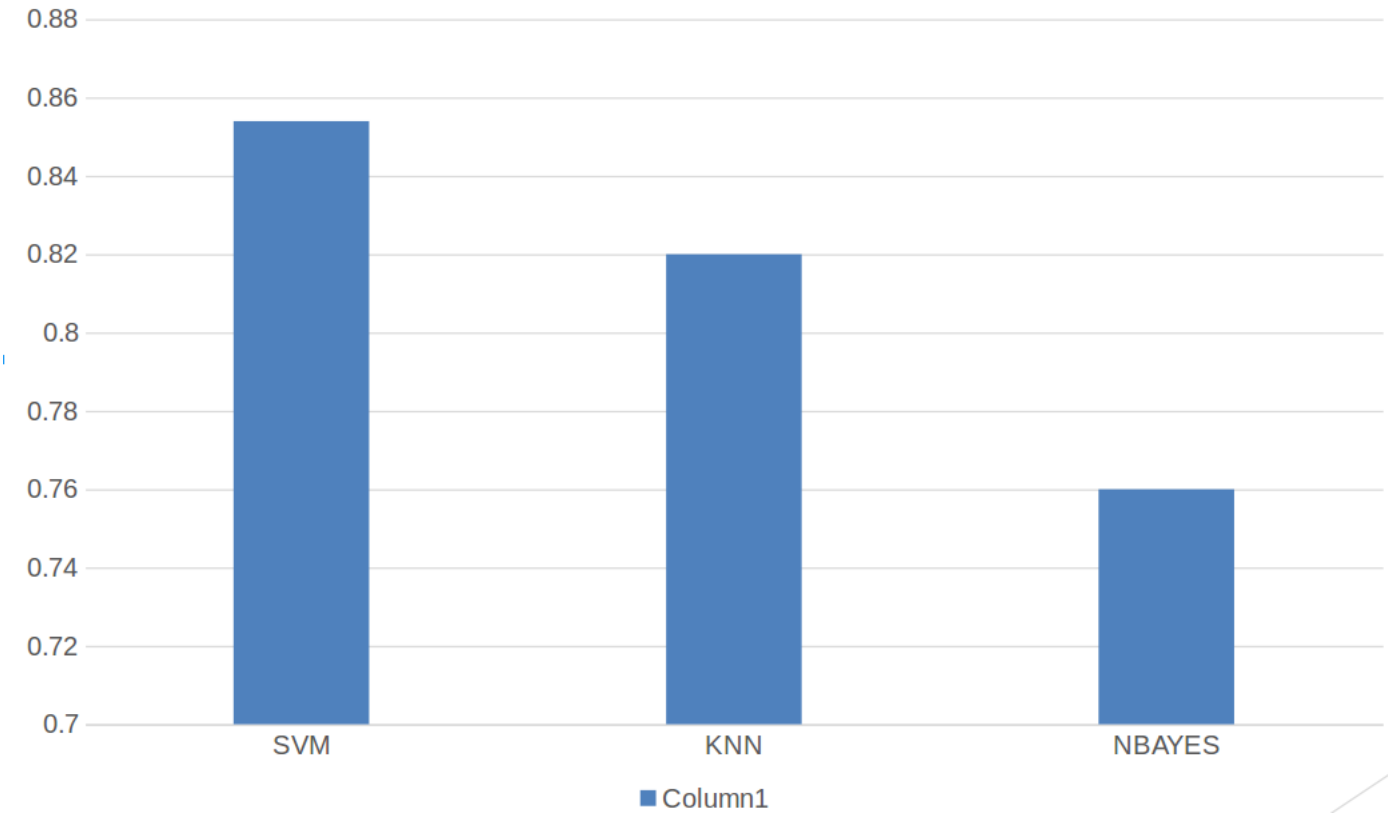


Figure 5.4: Overall Results

Chapter 6

CONCLUSION AND FUTURE WORK

We conclude this report with an account of the limitations to this project as well as possible future improvements and additions. The main drawback to using content based recommendation alone is that the cultural features of the songs, as a result songs from different languages cant be differentiated. To counter this issue the meta-data of the songs can be used to filter results.

Through the work weve done, a classification and recommendation system framework has been designed, implemented and evaluated. The online music database and the recommendation system were fully implemented. In addition, multiple recommendation strategies were used to return accurate predictions. These strategies include: Support vector machine methods and Clustering

References

[1] Kyoungro Yoon, Jonghyung Lee, and Min-Uk Kim "Music Recommendation System Using Emotion Triggering Low-level Features" IEEE Transactions on Consumer Electronics, Vol. 58, No. 2, May 2012

[2] S. Esmaili, S. Krishnan and K. Raahemifar "CONTENT BASED AUDIO CLASSIFICATION AND RETRIEVAL USING JOINT TIME-FREQUENCY ANALYSIS" ICASSP, 2004

[3] M. Zentner, D. Grandjean, and K. Scherer, Emotions evoked by the sound of music: Characterization, classification, and measurement, Emotion, vol. 8, no. 4, pp. 494521, 2008.

APPENDIX

FEature Extratcion)

```
//PERFORM MUSIC FEATURE EXTRACTION AND  kmedoids
 #!/usr/bin/python
from essentia.standard import MonoLoader
from essentia.standard import LowLevelSpectralEqloudExtractor
from essentia.standard import LowLevelSpectralExtractor
from essentia.standard import Loudness
from essentia.standard import RhythmExtractor2013
from essentia.standard import KeyExtractor
from essentia.standard import LogAttackTime
import essentia.streaming
import numpy as np
from math import sqrt
import re
from os import listdir
from multiprocessing import Pool,Lock
from random import randint
import csv
from suggest1 import smain

dataset = {}
lock = Lock()


#LOAD THE CSV WITH DATA ALREADY BEEN EXTRACTED SO WE DONT HAVE TO EXTRACT THEM AGAIN
def load():
    print "\n\tLoading Files....."
    line = csv.reader(open("data.csv","r"))
    datas  =  list(line)
    global dataset
    for i in datas:
        try:
            dataset[i[0]] = [float(x) for x in i[1:]]
        except ValueError:
            print "error"

    print "\tLoading Complete...."
    return dataset
```

```
#FIND DISTANCE BETWEEN SONGS
def euclidieanDistance(listOne,listTwo):
distance = 0
length = len(listOne)
for i in range(length):
distance += (listOne[i]-listTwo[i])*(listOne[i]-listTwo[i])
return sqrt(distance)

#EXTRACT FEATURES
def lowLevel(songName):
global dataset
global lock
print songName
#REMOVE ; AND , FROM SONGNAMES
key = re.sub(r',',"",songName.split('/')[-1])
key = re.sub(r';',"",key)
#DONT HAVE TO EXTRACT IF IT IS ALREADY EXTRACTED
if key in dataset.keys():
feature = dataset[key]
return feature
else:
loader = MonoLoader(filename = songName)
audio = loader()
extractor = LowLevelSpectralEqloudExtractor()
feature =list(extractor(audio))
del feature[1];del feature[1]
extractor =LowLevelSpectralExtractor()
featureTwo = list(extractor(audio))
del  featureTwo[0]
del featureTwo[-2]
featureTwo[4] = feature[4][1]
feature.extend(featureTwo)
extractor = Loudness()
feature.append(extractor(audio))
extractor = LogAttackTime()
feature.append(extractor(audio)[0])
extractor = KeyExtractor()
feature.append(extractor(audio)[2])
extractor = RhythmExtractor2013()
data = extractor(audio)
feature.append(data[0])
feature.append(data[2])
for x in  range(len(feature)) :
```

```
if type(feature[x]) is  np.ndarray :
#feature[x] = avg(feature[x])
mean,std =stdDev(feature[x])
feature[x] = mean
feature.append(std)
arr = key+","+str(feature)[1:-1]+"\n"
  f=  open('data.csv','a')
  lock.acquire()
  f.write(arr)
  lock.release()
  f.close()
return feature

def avg(arr):
    return sum(arr)/len(arr)

#Find standard deviation
def stdDev(arr):
mean = avg(arr)
diff = sum([(x-mean)**2 for x in arr])
return mean,sqrt(diff/len(arr))


#TO RANGE 0.1 and 100
def normalize(val,norm):
  for i in range(len(val)):
     val[i] = ((val[i]- norm[i][1])/(norm[i][0]-norm[i][1]))*(1-0.1)+0.1
     return val

#COST FOR CLUSTERS
def findCost(medoid,names,distance):
cost = 0
for i in range(len(names)):
temp = distance[medoid[0]][i]
for x in medoid:
if distance[x][i] < temp:
temp = distance[x][i]
cost += temp
return cost

#FIND NEW MEDOIDS
def recluster(medoid,names,distance,numCluster,clusters):
medNum = 0
```

```
newMedoids = []
newMedoids.extend(medoid)
newCost = 0
rMedoid = []
while(medNum < len(medoid)):
arr=clusters[medoid[medNum]]
for start in range(len(arr)):
newMedoids[medNum] = arr[start]
cost = findCost(newMedoids,names,distance)
if  cost < newCost or newCost == 0:
newCost = cost
rMedoid = []
rMedoid.extend(newMedoids)
start += 1
for x in range(len(newMedoids)):
newMedoids[x] = medoid[x]
medNum += 1
return newCost,rMedoid

#SONGS ARE CLUSTERED BASED ON MEDOIDS
def cluster(medoid,names,distance,numCluster):
clusters = {}
for med in medoid:
clusters[med] = []
cost=0;
for i in range(len(names)):
temp = distance[medoid[0]][i]
closestMedoid = medoid[0]
for x in medoid:
if distance[x][i] < temp:
temp = distance[x][i]
closestMedoid = x
cost += temp
clusters[closestMedoid].append(names[i])
newCost,newMedoids = recluster(medoid,names,distance,numCluster,clusters)
if newCost < cost:
cost,clusters = cluster(newMedoids,names,distance,numCluster)
return cost,clusters

#OPEN FOLDER AND SCAN FOR mp3
def findMusic(myPath):
print "Enter The path : example /home/swaraj/Music \n"
#myPath = raw_input()
```

```
#myPath = "/home/oem/Music"
#myPath = "/media/orefash/New Volume/down/Bruno Mars - 24K Magic/uk top"
files = [myPath+"/"+file for file in listdir(myPath) if  re.match(r'(.*)mp3',file)]
return files

def pool(path):
global dataset
dataset = load()
names = []
extract = Pool(8)
featureList = {}
songList = findMusic(path)
extractedList = extract.map(lowLevel,songList)
norm = []
for i in zip(*extractedList):
norm.append([max(i),min(i)])
for x in range(len(songList)):
randomTemp = re.sub(r',","",songList[x].split("/")[-1])
names.append(re.sub(r';","",randomTemp))
featureList[names[-1]] =  normalize(extractedList[x],norm)
distance = {}
#Find euclidiean Distances
for song in names:
distance[song] = []
current = featureList[song]
for songitem in  names:
if songitem == song :
distance[song].append(0)
else:
value = featureList[songitem]
distance[song].append(euclidieanDistance(current,value))
#CLUSTERING
bestCluster = {}
bestCost = 0
numCluster = 1
#CHOOSE MEDOIDS RANDOMLY FOR THE FIRST TIME
medoid = []
while(len(medoid) < numCluster):
temp = randint(0,len(names)-1)
if names[temp] not in medoid:
medoid.append(names[temp])
#CALL CLUSTER METHOD
bestCost,bestCluster = cluster(medoid,names,distance,numCluster)
```

```
#PRINTING THE OUTPUT
print "\n\n",bestCost
# WRITE TO CSV FILE
className = 0
result = str()
for key,value in bestCluster.iteritems():
print "\n",key,":",value
#ADD EACH VALUE TO FILE
for itr in value:
index = names.index(itr)
result += itr
for item in extractedList[index]:
result += "," +str(item)
result += "," + str(className)+"\n"
className += 1
with open("storage.csv","wb") as fp:
fp.write(result)

k = smain()
return k
#pool("/media/orefash/700EF6AE0EF66C8A/Users/orefa/Downloads/Billboard 2016 Year End Hot 1
```

kmeans.py to do kmeans

```
//PERFORM KMEANS AND THE ACCURACY IN EACH CLASSIFIER

import  numpy as np
import  pandas as pd
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB


def classifying(label):
    df = pd.read_csv("storage.csv",header=None)
    model = KMeans(n_clusters=10, random_state=0)
    model.fit(df.iloc[:,1:-1])
    print("labels: ")
    #write clustters to file
    result = str()
    for i in range(df.shape[0]):
        result += df.iloc[i,0]+","+str(model.labels_[i])+"\n"
    with open("kmeancluster.csv","w") as fp:
        fp.write(result)
    from sklearn.svm import SVC
    svm = SVC(kernel='linear')
    print("Cross Validation SVM 5 FOld: ")
    print(cross_val_score(svm, df.iloc[:, 1:-1],model.labels_, cv=5))
    gnb = GaussianNB()
    print("Cross Validation Of naive")
    print(cross_val_score(gnb,df.iloc[:, 1:-1],model.labels_, cv=5))
    knn = KNeighborsClassifier(n_neighbors=3)
    print("Cross Validation Of KNN")
    print(cross_val_score(knn, df.iloc[:, 1:-1], model.labels_, cv=5))
    #ouptput songs
    svm.fit(df.iloc[:, 1:-1],model.labels_)
    test = pd.read_csv("normtest.csv", header=None)
    result = []
    for i in range(test.shape[0]):
        if svm.predict([test.iloc[i, 1:]])[0] == label:
            result.append(test.iloc[i, 0])
    return result
```

Recommendation Part

```
// The part which perform recommendation
import  pandas as pd
from sklearn.svm import  SVC

def classifier(label):
    "returns list of songs in that class"
    #LOAD train and test
    train = pd.read_csv("main2.csv",header= None)
    test = pd.read_csv("storage.csv",header=None)
    #Train the SVC
    model = SVC(kernel="linear")
    model.fit(train.iloc[:,1:-1],train.iloc[:,-1])
    #Update The list
    result = []
    for i in range(test.shape[0]):
             if model.predict([test.iloc[i,1:-1]])[0] == label:
                k = []
                k.append(test.iloc[i,0])
                k.append(label)
                result.append(k)
    return result

#Call with a class label and watch the magic


def smain():
    k = []
    for i in range(0,14):
        for j in classifier(i):
            k.append(j)
    return k
```

gui

```
import sip
sip.setapi('QString', 2)
import eyeD3
from eyed3.id3 import ID3_V1_0, ID3_V1_1, ID3_V2_3, ID3_V2_4
from mutagen import File
from os import listdir
```

```
import os.path
from os.path import isfile, join
import sys, re
from string import digits
from fing import FingerTabWidget
from PyQt4 import QtCore, QtGui
from PyQt4.Qt import *
from PyQt4.QtWebKit import *
from album import gmain
from PyQt4.phonon import Phonon
from t1 import rec
from pool import pool


try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)


#for music artcover popup
class MyPopup(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.pic = QLabel()
        self.pixmap = QPixmap("img/image1.jpg")
        pixmap2 = self.pixmap.scaledToWidth(500)
        self.pic.setPixmap(pixmap2)
        self.pic.show()

#for recommendation table pop-up
class RecPopup(QWidget):
    def __init__(self,rec_list):
        QWidget.__init__(self)
```

```
head = ( "Title", "Artist", "Album", "Time") #table header labels
print rec_list
tabstyle2 = """
    QTableWidget{
    padding: 10px 10px 10px 30px;
    background: #f9f9f9;
    color: #a5aeb2;
    selection-color: white;
    font-size: 15px;
    margin:0px;

    }
    QTableWidget::item{
    padding: 15px 0px;
    color: #a5aeb2;
    border-top: 1px solid #eeeded;

    }
    QTableWidget::item:selected{
    color: #646a6f;
    background: white;
    border-top: 1px solid #eeeded;
    }

"""
#recommendation table definition
self.recTable = QtGui.QTableWidget(0, 4)
self.recTable.setHorizontalHeaderLabels(head)
self.recTable.setShowGrid(False)
self.recTable.setSelectionMode(QtGui.QAbstractItemView.SingleSelection)
self.recTable.setSelectionBehavior(QtGui.QAbstractItemView.SelectRows)
self.recTable.setAutoFillBackground(True)
self.recTable.setAlternatingRowColors(False)
self.recTable.horizontalHeader().setDefaultSectionSize(200)
self.recTable.horizontalHeader().setVisible(False)
self.recTable.verticalHeader().setVisible(False)
self.recTable.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)
self.recTable.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Expa
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.recTable.sizePolicy().hasHeightForWidth())
self.recTable.setSizePolicy(sizePolicy)
```

```
        self.recTable.setStyleSheet(tabstyle2)

        headr = self.recTable.horizontalHeader()
        headr.setResizeMode(0, QtGui.QHeaderView.Stretch)
        headr.setResizeMode(1, QtGui.QHeaderView.Stretch)
        headr.setResizeMode(2, QtGui.QHeaderView.Stretch)
        headr.setResizeMode(3, QtGui.QHeaderView.ResizeToContents)
        titleItem = QtGui.QTableWidgetItem("Title")
        titleItem.setFlags(titleItem.flags() ^ QtCore.Qt.ItemIsEditable)

        artistItem = QtGui.QTableWidgetItem("Artist")
        artistItem.setFlags(artistItem.flags() ^ QtCore.Qt.ItemIsEditable)
        artistItem.setTextAlignment(QtCore.Qt.AlignLeft)

        albumItem = QtGui.QTableWidgetItem("Albumm")
        albumItem.setFlags(albumItem.flags() ^ QtCore.Qt.ItemIsEditable)

        timeItem = QtGui.QTableWidgetItem("Genre")
        timeItem.setFlags(timeItem.flags() ^ QtCore.Qt.ItemIsEditable)
        timeItem.setTextAlignment(QtCore.Qt.AlignCenter)

        currentRow = 0
        self.recTable.insertRow(currentRow)
        self.recTable.setItem(currentRow, 0, titleItem)
        self.recTable.setItem(currentRow, 1, artistItem)
        self.recTable.setItem(currentRow, 2, albumItem)
        self.recTable.setItem(currentRow, 3, timeItem)

        self.recTable.setRowHeight(currentRow, 50)
        self.recTable.show()


#main application window start
class MainWindow(QtGui.QMainWindow):
    def __init__(self):
        super(QtGui.QMainWindow, self).__init__()
        self.setWindowIcon(QtGui.QIcon('img/Musicplayer-256.png'))
        self.setWindowTitle("Test Music Player")
        self.audioOutput = Phonon.AudioOutput(Phonon.MusicCategory, self)
        self.mediaObject = Phonon.MediaObject(self)
        self.metaInformationResolver = Phonon.MediaObject(self)
        self.mediaObject.setTickInterval(1000)
        #self.mediaObject.tick.connect(self.tick)
```

```
        self.mediaObject.currentSourceChanged.connect(self.sourceChanged)
        self.mediaObject.aboutToFinish.connect(self.aboutToFinish)

        self.mainMenu = self.menuBar()
        self.count = 0
        self.tclicked = 0
        self.buttons = {}

        Phonon.createPath(self.mediaObject, self.audioOutput)
        self.setupActions()
        self.setupUi()
        self.sources = []
        self.albumsource = []
        self.resource = []
        self.files = []
        self.a_files = []
        self.album_list = []
        self.rec_list = []
        self.index = 0
        self.current_source = "a"
        self.fulltable =[]

    def setupActions(self):
        #application menu bar definition
        self.FileAction = QtGui.QAction("&Open Folder", self)
        self.FileAction.setShortcut("Ctrl+O")
        self.FileAction.setStatusTip('Select Music')
        self.FileAction.triggered.connect(self.ButAction)
        self.playAction = QtGui.QAction(QtGui.QIcon('img/play.png'), "Play",self, shortcut
        self.playAction.triggered.connect(self.PlayAction)

        self.pauseAction = QtGui.QAction(
                self.style().standardIcon(QtGui.QStyle.SP_MediaPause),
                "Pause", self, shortcut="Ctrl+A", enabled=False,
                triggered=self.mediaObject.pause)

        self.stopAction = QtGui.QAction(QtGui.QIcon('img/stop.png'), "Stop",
                self)
        self.stopAction.triggered.connect(self.StopAction)

        self.nextAction = QtGui.QAction(QtGui.QIcon('img/forward.png'),
                "Next", self, shortcut="Ctrl+N")
        self.nextAction.triggered.connect(self.next_track)
```

```
        self.previousAction = QtGui.QAction(QtGui.QIcon('img/rewind.png'),
                "Previous", self, shortcut="Ctrl+R")
        self.previousAction.triggered.connect(self.previous_track)
        self.exitAction = QtGui.QAction("E&xit", self, shortcut="Ctrl+X",
                triggered=self.close)

    def albumL(self,line):
        self.alist = line
        a_dict = {}
        k = len(self.album_list)

    def popup(self, pos):#main table right-click action
        for i in self.musicTable.selectionModel().selection().indexes():
            print i.row(), i.column()
        menu = QMenu()
        quitAction = menu.addAction("Quit")
        recommendAction = menu.addAction("Recommend")
        action = menu.exec_(self.mapToGlobal(pos))
        if action == quitAction:
            qApp.quit()
        else:
            self.reclist()


    def ButAction(self, event):#music load to table function
        self.current_source = "main"
        dirs = QtGui.QFileDialog.getExistingDirectory(self, 'Select Folder')
        files = [dirs+"/"+file for file in listdir(dirs) if  re.match(r'(.*)mp3',file)]
        self.files.extend(files)#add all song path to file list
        for path in files:
            self.sources.append(Phonon.MediaSource(path))
            self.populate_table(path)#load songs into table
        icon_1 = QtGui.QIcon()
        icon_1.addPixmap(QtGui.QPixmap('img/pause.png'))
        self.playAction.setIcon(icon_1)
        self.playAction.setToolTip('Pause')
        self.album_list = gmain(dirs)#get all albums for album tab
        self.rec_list = pool(dirs)#extract features from songs
        self.album_test(self.album_list)#load songs to album tab

    def titleResolve(self, path):
        pap = path.split("/")[-1]
```

```
        return pap

    def tagGet(self,path):#get music tags to load into music table
        taglist = []
        tag = eyeD3.Tag()

        try:#to get music play duration
            trackInfo = eyeD3.Mp3AudioFile(path)
            Duration = trackInfo.getPlayTimeString()
        except:
            Duration = "Unknown"
        else:
            Duration = trackInfo.getPlayTimeString()

        tag.link(path)

        try:#get music genre
            g = tag.getGenre()
        except:
            Genre = "Unknown Genre"#tag.getGenre()
        else:
            if g == None:
                Genre = "Unknown Genre"
            else:
                junk = tag.getGenre()
                Genre = junk
        g = str(Genre)
        g = g.replace("(","")
        g = g.replace(")","")
        Genre = g.translate(None,digits)

        #get music arist, album, title, year tags
        if tag.getArtist() == "":
            Artist = "Unknown Artist"
        else:
            Artist = tag.getArtist()

        if tag.getAlbum() == "":
            Album = "Unknown Album"
        else:
            Album = tag.getAlbum()

        if tag.getTitle() == "":
```

```
            Title = self.titleResolve(path)
        else:
            Title = tag.getTitle()
        if tag.getYear() == None:
            Year = " "
        else:
            Year = tag.getYear()
        taglist.extend([Title,Artist,Album,Duration,Year,Genre])
        return taglist


    def album_test(self, line):#function to load album tab in app
        print "In album test"
        self.alist = line
        #print line
        a = len(self.album_list)
        print "a: "+str(a)
        l = 0
        n = 0
        if a/6 < 1:
            c = 1
        else:
            c = a/6 +1
        print "c: "+str(c)
        for i in range(c):
            for j in range(6):
                if l==a:
                    print "break"
                    break
                if l<a:
                    name = 'album/'+str(l)+'.jpg'
                    if os.path.isfile(name):#create album tab buttons
                        self.buttons[(i, j)] = QtGui.QToolButton(self)
                        self.buttons[(i, j)].setToolButtonStyle(QtCore.Qt.ToolButtonTextUn
                        metrics = QtGui.QFontMetrics(self.font())
                        t = self.alist[l][0]
                        st = t
                        width = metrics.width(t)
                        if width > 240:
                            m = 240.0/width
                            k = m * len(t)
                            m = int(k)
```

```
                        p = t.rfind(" ",0,m)
                        st = t[:p] + '\n' + t[p+1:]
                        #print st
                    self.buttons[(i, j)].setText(st)
                    self.buttons[(i, j)].setIcon(QtGui.QIcon(name))
                    self.buttons[(i, j)].setIconSize(QtCore.QSize(240,240))
                    self.buttons[(i, j)].setFixedSize(240,285)
                    print "l: "+str(l)
                    self.buttons[(i, j)].clicked.connect(self.album_table(line[l]))#lo
                    self.buttons[(i, j)].setStyleSheet("padding:1px; background: white
                        border: 2px solid grey")
                    self.verticalLayoutScroll.addWidget(self.buttons[(i, j)], i, j)
                    l = l+1

                else:
                    self.buttons[(i, j)] = QtGui.QToolButton(self)
                    self.buttons[(i, j)].setToolButtonStyle(QtCore.Qt.ToolButtonTextUn
                    metrics = QtGui.QFontMetrics(self.font())
                    t = self.alist[l][0]
                    st = t
                    width = metrics.width(t)
                    if width > 240:
                        m = 240.0/width
                        k = m * len(t)
                        m = int(k)
                        p = t.rfind(" ",0,m)
                        st = t[:p] + '\n' + t[p+1:]
                        #print st
                    self.buttons[(i, j)].setText(st)
                    self.buttons[(i, j)].setIcon(QtGui.QIcon("img/Musicplayer-256.png'
                    self.buttons[(i, j)].setIconSize(QtCore.QSize(240,240))
                    self.buttons[(i, j)].setFixedSize(240,285)
                    print "l: "+str(l)
                    self.buttons[(i, j)].clicked.connect(self.album_table(line[l]))
                    self.buttons[(i, j)].setStyleSheet("padding:1px; background: white
                        border: 2px solid grey")
                    self.verticalLayoutScroll.addWidget(self.buttons[(i, j)], i, j)
                    l = l+1
            else:
                print "l::"+str(l)
                break
    if(a < 6):
        self.verticalLayoutScroll.setColumnStretch(a,1)
```

```
    def album_table(self, l):#fill albumtable with songs
        def fill():
            print "triggered"
            a = l[1:]#list of songs in album
            self.a_files = a
            self.albumsource = []
            for i in a:
                self.albumsource.append(Phonon.MediaSource(i))
            print "list"
            self.albumTable.setRowCount(0)
            self.albumTable.clearContents()
            currentRow = 0
            for i in range(1,len(l)):
                path = l[i]
                line = self.tagGet(path)#get song tags

                titleItem = QtGui.QTableWidgetItem(line[0])
                titleItem.setFlags(titleItem.flags() ^ QtCore.Qt.ItemIsEditable)

                artistItem = QtGui.QTableWidgetItem(line[1])
                artistItem.setFlags(artistItem.flags() ^ QtCore.Qt.ItemIsEditable)
                artistItem.setTextAlignment(QtCore.Qt.AlignLeft)

                albumItem = QtGui.QTableWidgetItem(line[2])
                albumItem.setFlags(albumItem.flags() ^ QtCore.Qt.ItemIsEditable)

                timeItem = QtGui.QTableWidgetItem(str(line[3]))
                timeItem.setFlags(timeItem.flags() ^ QtCore.Qt.ItemIsEditable)
                timeItem.setTextAlignment(QtCore.Qt.AlignCenter)


                self.albumTable.insertRow(currentRow)
                self.albumTable.setItem(currentRow, 0, titleItem)
                self.albumTable.setItem(currentRow, 1, artistItem)
                self.albumTable.setItem(currentRow, 2, albumItem)
                self.albumTable.setItem(currentRow, 3, timeItem)

                self.albumTable.setRowHeight(currentRow, 50)
                currentRow = currentRow + 1
        return fill

    def populate_table(self, path):#function to load songs into main table
```

```
        #adds songs to table
        line = self.tagGet(path)
        self.fulltable.append(line)

        titleItem = QtGui.QTableWidgetItem(line[0])
        titleItem.setFlags(titleItem.flags() ^ QtCore.Qt.ItemIsEditable)

        artistItem = QtGui.QTableWidgetItem(line[1])
        artistItem.setFlags(artistItem.flags() ^ QtCore.Qt.ItemIsEditable)
        albumItem = QtGui.QTableWidgetItem(line[2])
        albumItem.setFlags(albumItem.flags() ^ QtCore.Qt.ItemIsEditable)

        timeItem = QtGui.QTableWidgetItem(str(line[3]))
        timeItem.setFlags(timeItem.flags() ^ QtCore.Qt.ItemIsEditable)
        timeItem.setTextAlignment(QtCore.Qt.AlignCenter)

        yearItem = QtGui.QTableWidgetItem(str(line[4]))
        yearItem.setFlags(yearItem.flags() ^ QtCore.Qt.ItemIsEditable)
        yearItem.setTextAlignment(QtCore.Qt.AlignCenter)

        currentRow = self.musicTable.rowCount()
        self.musicTable.insertRow(currentRow)
        self.musicTable.setItem(currentRow, 0, titleItem)
        self.musicTable.setItem(currentRow, 1, artistItem)
        self.musicTable.setItem(currentRow, 2, albumItem)
        self.musicTable.setItem(currentRow, 3, yearItem)
        self.musicTable.setItem(currentRow, 4, timeItem)
        self.musicTable.setRowHeight(currentRow, 50)

        if not self.musicTable.selectedItems():
            self.musicTable.selectRow(0)
            self.mediaObject.setCurrentSource(self.metaInformationResolver.currentSource()
        self.index = self.index + 1
        if len(self.sources) > self.index:
            self.metaInformationResolver.setCurrentSource(self.sources[index])
        else:
            if self.musicTable.columnWidth(0) > 300:
                self.musicTable.setColumnWidth(0, 300)


    def tableClicked(self, row, column):#when main table is clicked
        print "In click"
        self.current_source = "main"
```

```
        print self.current_source
        self.tclicked = 1
        self.count = self.musicTable.currentRow()
        self.mediaObject.stop()
        self.mediaObject.setCurrentSource(self.sources[self.count])
        self.mediaObject.play()


    def atableClicked(self, row, column):#when album table is clicked
        print "In click"
        self.current_source = "album"
        print self.current_source
        self.tclicked = 1
        self.count = self.albumTable.currentRow()
        print self.count
        self.mediaObject.stop()
        self.mediaObject.setCurrentSource(self.albumsource[self.count])
        self.mediaObject.play()

    def rtableClicked(self, row, column):#when recommend table is clicked
        print "In click"
        self.current_source = "recommend"
        print self.current_source
        self.tclicked = 1
        self.count = self.recTable.currentRow()
        print self.count
        self.mediaObject.stop()
        self.mediaObject.setCurrentSource(self.resource[self.count])
        self.mediaObject.play()

    def sourceChanged(self, source):#current playing song is changed
        a = self.current_source
        if a == "main":#main music table
            self.musicTable.selectRow(self.count)
        elif a == "album":
            self.albumTable.selectRow(self.count)
        else:
            self.recTable.selectRow(self.count)
        print 'kk'
        self.setArt()#load song cover art
        self.setLabel()#load song labels

    def reclist(self):#recommendation function
```

```
        self.count = self.musicTable.currentRow()
        self.w = RecPopup(self.rec_list)
        self.w.setGeometry(QRect(0, 0,900,900))



    def on_clicked(self):
        print "recommend"
        x = rec()
        for i in self.fulltable:
            print i
        #line = x

        currentRow = 0
        for line in x:
            titleItem = QtGui.QTableWidgetItem(line[1])
            titleItem.setFlags(titleItem.flags() ^ QtCore.Qt.ItemIsEditable)

            artistItem = QtGui.QTableWidgetItem(line[2])
            artistItem.setFlags(artistItem.flags() ^ QtCore.Qt.ItemIsEditable)
            artistItem.setTextAlignment(QtCore.Qt.AlignLeft)

            albumItem = QtGui.QTableWidgetItem(line[3])
            albumItem.setFlags(albumItem.flags() ^ QtCore.Qt.ItemIsEditable)

            timeItem = QtGui.QTableWidgetItem(str(line[4]))
            timeItem.setFlags(timeItem.flags() ^ QtCore.Qt.ItemIsEditable)
            timeItem.setTextAlignment(QtCore.Qt.AlignCenter)

            self.resource.append(Phonon.MediaSource(line[5]))
            #self.resource.append(line[5])

            self.recTable.insertRow(currentRow)
            self.recTable.setItem(currentRow, 0, titleItem)
            self.recTable.setItem(currentRow, 1, artistItem)
            self.recTable.setItem(currentRow, 2, albumItem)
            self.recTable.setItem(currentRow, 3, timeItem)

            self.recTable.setRowHeight(currentRow, 50)
            currentRow = currentRow + 1

    def aboutToFinish(self):#to load next song for playing
        print 'finish'
```

```
        index = self.musicTable.currentRow()
        if index < len(self.sources)-1:
            index += 1
        else:
            index = 0
        self.count = index
        self.mediaObject.enqueue(self.sources[index])


    def PlayAction(self):
        icon_1 = QtGui.QIcon()
        icon_1.addPixmap(QtGui.QPixmap('img/pause.png'))
        icon_2 = QtGui.QIcon()
        icon_2.addPixmap(QtGui.QPixmap('img/play.png'))
        if self.mediaObject.state() == Phonon.PlayingState:
            self.mediaObject.pause()
            self.playAction.setIcon(icon_2)
            self.playAction.setToolTip('Play')
        elif self.mediaObject.state() == Phonon.PausedState or self.mediaObject.state() ==
            self.mediaObject.play()
            self.playAction.setIcon(icon_1)
            self.playAction.setToolTip('Pause')

    def StopAction(self):
        icon_2 = QtGui.QIcon()
        icon_2.addPixmap(QtGui.QPixmap('img/play.png'))
        self.mediaObject.stop()
        self.playAction.setIcon(icon_2)
        self.playAction.setToolTip('Play')

    def getArt(self):#get cover art
        if self.current_source == "main":
            i = self.musicTable.currentRow()
            filepath = self.files[i]
        else:
            i = self.albumTable.currentRow()
            filepath = self.a_files[i]
        file = File(filepath) # mutagen can automatically detect format and type of tags
        try:
            artwork = file.tags['APIC:'].data
        except TypeError:
            p = 0
        except KeyError:
```

```
            p = 0
        else:
            p = 1
            with open('img/image1.jpg', 'wb') as img:
                img.write(artwork)
        return p

    def setArt(self):#set art in gui
        print 'king'
        p = self.getArt()

        if p is 1:
            try:
                self.pix = QtGui.QPixmap("img/image1.jpg")
                print 'in'
            except:
                self.pix = QtGui.QPixmap("img/Musicplayer-256.png")
            else:
                print 'a'
        else:
            self.pix = QtGui.QPixmap("img/Musicplayer-256.png")

        self.pixy = self.pix.scaled(70,70)
        self.pixy1 = self.pix.scaledToWidth(600)
        self.pic.setPixmap(self.pixy)
        self.bigpic.setPixmap(self.pixy1)


    def artcover(self, event):#popup cover art
        self.w = MyPopup()
        self.w.setGeometry(QRect(0, 0,500,500))

    def setLabel(self):#set song labels in lower portion of ui
        if self.current_source == "main":
            i = self.musicTable.currentRow()
            path = self.files[i]
        else:
            i = self.albumTable.currentRow()
            path = self.a_files[i]
        tag = eyeD3.Tag()
        tag.link(path)
        if tag.getArtist() == "":
            Artist = "Unknown Artist"
```

```
        else:
            Artist = tag.getArtist()

        if tag.getTitle() == "":
            Title = self.titleResolve(path)
        else:
            Title = tag.getTitle()
        self.artist.setText(Artist)#set artist name in ui
        self.title.setText(Title)#set song name in ui

    def next_track(self, source):#for next button
        if self.current_source == "main":
            index = self.musicTable.currentRow()
            if index < len(self.sources)-1:
                index += 1
            else:
                index = 0 # start from the beginning of the list
            self.count = index
            self.mediaObject.setCurrentSource(self.sources[index])
        else:
            index = self.albumTable.currentRow()
            if index < len(self.albumsource)-1:
                index += 1
            else:
                index = 0 # start from the beginning of the list
            self.count = index
            self.mediaObject.setCurrentSource(self.albumsource[index])
        self.mediaObject.play()


    def previous_track(self):#for previous button
        if self.current_source == "main":
            index = self.musicTable.currentRow()
            if index >0:
                index -= 1
            else:
                index = len(self.sources)-1 # start from the beginning of the list
            self.count = index
            self.mediaObject.setCurrentSource(self.sources[index])
        else:
            index = self.albumTable.currentRow()
            if index >0:
                index -= 1
```

```
            else:
                index = len(self.albumsource)-1 # start from the beginning of the list
            self.count = index
            self.mediaObject.setCurrentSource(self.albumsource[index])
        self.mediaObject.play()


    def onResize(self, event):
        print "resized"

    def setupUi(self):#set up music gui
        self.resize(1123, 759)
        self.centralwidget = QtGui.QWidget(self)
        self.fileMenu = self.mainMenu.addMenu('&File')
        self.fileMenu.addAction(self.FileAction)

        self.HLayout = QtGui.QHBoxLayout(self)

        self.tabs = QtGui.QTabWidget()#for tabs in ui
        self.tabs.setTabBar(FingerTabWidget(width=350,height=50))
        self.tabs.setTabPosition(QtGui.QTabWidget.West)

        styles = """
            QTabBar {color: #c2c5c9;
                      font-size: 15px;
                      padding:0px
                      }
            QTabBar::tab:selected {background: #3c4043;
                                   border-left: 4px solid #646a6f;
                                   }
            QTabBar::tab {background: #373b3f;
                          color: white;
                          }

            """
        tabstyle = """
            QTableWidget{
            padding: 10px 10px 10px 30px;
            background: #f9f9f9;
            color: #a5aeb2;
            selection-color: white;
            font-size: 15px;
            margin:0px;
```

```
    }
    QTableWidget::item{
    padding: 15px 0px;
    color: #a5aeb2;
    border-top: 1px solid #eeeded;

    }
    QTableWidget::item:selected{
    color: #646a6f;
    background: white;
    border-top: 1px solid #eeeded;
    }

"""
tabstyle2 = """
    QTableWidget{
    padding: 10px 10px 10px 30px;
    background: #f9f9f9;
    color: #a5aeb2;
    selection-color: white;
    font-size: 15px;
    margin:0px;

    }
    QTableWidget::item{
    padding: 15px 0px;
    color: #a5aeb2;
    border-top: 1px solid #eeeded;

    }
    QTableWidget::item:selected{
    color: #646a6f;
    background: white;
    border-top: 1px solid #eeeded;
    }

"""
self.tabs.setStyleSheet(styles)#style music table
self.TLayout = QtGui.QHBoxLayout(self)
bar = QtGui.QToolBar()#create music ui buttons
bar.addAction(self.playAction)
bar.addAction(self.previousAction)
```

```
        bar.addAction(self.stopAction)
        bar.addAction(self.nextAction)

        headers = ( "Title", "Artist", "Album", "Year", "Time")#main table header labels

        self.musicTable = QtGui.QTableWidget(0, 5)#create main music table in ui
        self.musicTable.setHorizontalHeaderLabels(headers)
        self.musicTable.setShowGrid(False)
        self.musicTable.setSelectionMode(QtGui.QAbstractItemView.SingleSelection)
        self.musicTable.setSelectionBehavior(QtGui.QAbstractItemView.SelectRows)
        self.musicTable.setAutoFillBackground(True)
        self.musicTable.setAlternatingRowColors(False)
        self.musicTable.horizontalHeader().setDefaultSectionSize(200)
        self.musicTable.horizontalHeader().setVisible(False)
        self.musicTable.verticalHeader().setVisible(False)
        self.musicTable.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)
        self.musicTable.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
        self.musicTable.cellDoubleClicked.connect(self.tableClicked)
        sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Expa
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.musicTable.sizePolicy().hasHeightForWidth())
        self.musicTable.setSizePolicy(sizePolicy)
        self.musicTable.setStyleSheet(tabstyle)
        self.musicTable.setContextMenuPolicy(Qt.CustomContextMenu)
        self.musicTable.customContextMenuRequested.connect(self.popup)

        self.tabs.addTab(self.musicTable, 'Tracks')#add to tracks tab in ui

        header = self.musicTable.horizontalHeader()#format music table columns
        header.setResizeMode(0, QtGui.QHeaderView.Stretch)
        header.setResizeMode(1, QtGui.QHeaderView.Stretch)
        header.setResizeMode(2, QtGui.QHeaderView.Stretch)
        header.setResizeMode(3, QtGui.QHeaderView.Stretch)
        header.setResizeMode(4, QtGui.QHeaderView.ResizeToContents)

        self.TLayout.addWidget(self.tabs)

        self.seekSlider = Phonon.SeekSlider(self)#initialize play slider
        self.seekSlider.setMediaObject(self.mediaObject)
        self.seekSlider.setStyleSheet(self.stylesheet())

        self.volumeSlider = Phonon.VolumeSlider(self)#volume slider
```

```
self.volumeSlider.setAudioOutput(self.audioOutput)
self.volumeSlider.setSizePolicy(QtGui.QSizePolicy.Maximum,
        QtGui.QSizePolicy.Maximum)
volumeLabel = QtGui.QLabel()

self.pic = QtGui.QLabel(self)#initialize music cover icon in lower ui
self.pix = QtGui.QPixmap('img/Musicplayer-256.png')
self.pixmap = self.pix
self.pixmap2 = self.pixmap.scaled(70,70)
self.pic.setPixmap(self.pixmap2)
self.pic.setStyleSheet("background: grey")
self.pic.mousePressEvent = self.artcover

self.nplay = QtGui.QWidget()#initialize now pplaying tab
self.nplay.setStyleSheet("background: white;")
self.tabs.addTab(self.nplay, 'Now Playing')
self.cover_layout = QtGui.QGridLayout()
self.cover_layout.setColumnStretch(0, 1)
self.cover_layout.setColumnStretch(3, 1)
self.cover_layout.setRowStretch(0, 1)
self.cover_layout.setRowStretch(3, 1)
self.bigpic = QtGui.QLabel(self)
self.cover_layout.addWidget(self.bigpic, 1, 1, 2, 2)
self.nplay.setLayout(self.cover_layout)

self.rectab = QtGui.QWidget()
self.rectab.setStyleSheet("background: white;")
self.tabs.addTab(self.rectab, 'Recommended')
self.mLayout = QtGui.QVBoxLayout(self)

head = ( "Title", "Artist", "Album", "Time")

self.recTable = QtGui.QTableWidget(0, 4)
self.recTable.setHorizontalHeaderLabels(head)
self.recTable.setShowGrid(False)
self.recTable.setSelectionMode(QtGui.QAbstractItemView.SingleSelection)
self.recTable.setSelectionBehavior(QtGui.QAbstractItemView.SelectRows)
self.recTable.setAutoFillBackground(True)
self.recTable.setAlternatingRowColors(False)
self.recTable.horizontalHeader().setDefaultSectionSize(200)
self.recTable.horizontalHeader().setVisible(False)
self.recTable.verticalHeader().setVisible(False)
self.recTable.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)
```

```
self.recTable.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
self.recTable.cellDoubleClicked.connect(self.rtableClicked)
sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Expa
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.recTable.sizePolicy().hasHeightForWidth())
self.recTable.setSizePolicy(sizePolicy)
self.recTable.setStyleSheet(tabstyle2)

#self.tabs.addTab(self.recTable, 'Tracks')

headr = self.recTable.horizontalHeader()
headr.setResizeMode(0, QtGui.QHeaderView.Stretch)
headr.setResizeMode(1, QtGui.QHeaderView.Stretch)
headr.setResizeMode(2, QtGui.QHeaderView.Stretch)
headr.setResizeMode(3, QtGui.QHeaderView.ResizeToContents)

self.inLayout = QtGui.QHBoxLayout(self)
self.inLayout.addStretch()
self.button = QtGui.QPushButton("Recommend", self)
self.button.clicked.connect(self.on_clicked)
self.inLayout.addWidget(self.button)
self.mLayout.addLayout(self.inLayout)
self.mLayout.addWidget(self.recTable)


self.rectab.setLayout(self.mLayout)

self.album1 = QtGui.QWidget()#album tab in ui
self.album1.setStyleSheet("background: white;")
self.tabs.addTab(self.album1, 'Albums')
self.albumLayout = QtGui.QVBoxLayout(self)
self.albumGrid = QtGui.QHBoxLayout(self)
self.albumList = QtGui.QHBoxLayout(self)
self.Xlay = QtGui.QVBoxLayout(self)
self.albumLayout.addLayout(self.albumList)

self.album1.setLayout(self.albumLayout)

head = ( "Title", "Artist", "Album", "Time")

self.albumTable = QtGui.QTableWidget(0, 4)#initialize album table
self.albumTable.setHorizontalHeaderLabels(head)
```

```
self.albumTable.setShowGrid(False)
self.albumTable.setSelectionMode(QtGui.QAbstractItemView.SingleSelection)
self.albumTable.setSelectionBehavior(QtGui.QAbstractItemView.SelectRows)
self.albumTable.setAutoFillBackground(True)
self.albumTable.setAlternatingRowColors(False)
self.albumTable.horizontalHeader().setDefaultSectionSize(200)
self.albumTable.horizontalHeader().setVisible(False)
self.albumTable.verticalHeader().setVisible(False)
self.albumTable.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)
self.albumTable.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
self.albumTable.cellDoubleClicked.connect(self.atableClicked)
sizePolicy = QtGui.QSizePolicy(QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Expa
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.albumTable.sizePolicy().hasHeightForWidth())
self.albumTable.setSizePolicy(sizePolicy)
self.albumTable.setStyleSheet(tabstyle2)
#self.tabs.addTab(self.albumTable, 'Tracks')

headr = self.albumTable.horizontalHeader()
headr.setResizeMode(0, QtGui.QHeaderView.Stretch)
headr.setResizeMode(1, QtGui.QHeaderView.Stretch)
headr.setResizeMode(2, QtGui.QHeaderView.Stretch)
headr.setResizeMode(3, QtGui.QHeaderView.ResizeToContents)

self.albumGrid.addWidget(self.albumTable)
self.scrollArea = QtGui.QScrollArea(self)
self.scrollArea.setWidgetResizable(True)
self.scrollArea.setFixedHeight(245)
self.scrollArea.setStyleSheet("background: grey")
self.scrollAreaWidgetContents = QtGui.QWidget(self.scrollArea)
self.scrollArea.setWidget(self.scrollAreaWidgetContents)

self.Xlay.addWidget(self.scrollArea)

self.verticalLayoutScroll = QtGui.QGridLayout(self.scrollAreaWidgetContents)
self.albumLayout.addWidget(self.scrollArea)
self.albumLayout.addLayout(self.albumGrid)#end of album tab in ui

bottom = QtGui.QWidget(self)#bottom widget in ui
bottom.setStyleSheet("background: #e83c00; margin:0px; padding:0px;")

self.title = QtGui.QLabel(" ",self)
```

```
        self.artist = QtGui.QLabel(" ",self)
        self.title.setStyleSheet("color: #fff; font-size:20px; font-family:Arial;\
         border:0px; padding:0px; margin:0px")
        self.artist.setStyleSheet("color: #fff; font-size:13px; font-family:Arial;\
            border:0px; padding:0px; margin:0px")
        self.title.setFixedWidth(252)
        self.artist.setFixedWidth(252)
        square = QtGui.QWidget(self)
        square.setStyleSheet("background: #373b3f")
        self.BLayout = QtGui.QHBoxLayout(self)
        self.KLayout = QtGui.QHBoxLayout(self)
        square.setLayout(self.KLayout)
        self.TabLayout = QtGui.QHBoxLayout(self)
        self.labLayout = QtGui.QVBoxLayout(self)
        self.labLayout.addWidget(self.title)
        self.labLayout.addWidget(self.artist)
        self.KLayout.addLayout(self.TabLayout)
        self.TabLayout.addWidget(self.pic)
        self.TabLayout.addLayout(self.labLayout)
        self.TabLayout.addWidget(self.seekSlider)
        self.TabLayout.addWidget(bar)
        self.TabLayout.addWidget(volumeLabel)
        self.TabLayout.addWidget(self.volumeSlider)
        self.BLayout.addWidget(square)
        mainLayout = QtGui.QVBoxLayout()
        mainLayout.setContentsMargins(0, 0, 0, 0)
        mainLayout.addLayout(self.TLayout)
        mainLayout.addLayout(self.BLayout)
        mainLayout.insertSpacing(1,-30)

        widget = QtGui.QWidget()#add all widgets to main widget
        widget.setStyleSheet("background: #373b3f")
        widget.setLayout(mainLayout)
        widget.setMaximumHeight(1050)
        widget.resizeEvent = self.onResize
        self.setCentralWidget(widget)
    def stylesheet(self):#style sliders
        return """
            QSlider::groove:horizontal {
                background: #f2f2f2;
                height: 3px;
            }
```

```
            QSlider::sub-page:horizontal {
                background: qlineargradient(x1: 0, y1: 0,    x2: 0, y2: 1,
                    stop: 0 #66e, stop: 1 #bbf);
                background: qlineargradient(x1: 0, y1: 0.2, x2: 1, y2: 1,
                    stop: 0 #bbf, stop: 1 #55f);
                height: 40px;
            }

            QSlider::handle:horizontal {
                background: #bbf;
                border: 1px solid #bbf;
                height: 19px;
                width: 19px;
                margin: -2px 0;
                margin-top: 0px;
                margin-bottom: 0px;
                border-radius: 9px;
            }
        """

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    app.setApplicationName("Music Player")
    app.setQuitOnLastWindowClosed(True)

    window = MainWindow()
    window.show()

    sys.exit(app.exec_())
```

Interact WIth mysql

```
import eyeD3
from eyed3.id3 import ID3_V1_0, ID3_V1_1, ID3_V2_3, ID3_V2_4
from mutagen import File
from os import listdir
import os.path
from os.path import isfile, join
import sys, re
import ntpath
import csv
```

```
import MySQLdb

def path_leaf(path):
    head, tail = ntpath.split(path)
    return tail or ntpath.basename(head)

def titleResolve(path):
    pap = path.split("/")[-1]
    key = re.sub(r',',"",pap.split('/')[-1])
    key = re.sub(r';',"",key)
    return pap

def get(path):
    a = path_leaf(path)
    taglist = []
    tag = eyeD3.Tag()
    try:
        trackInfo = eyeD3.Mp3AudioFile(path)
        Duration = trackInfo.getPlayTimeString()
    except:
        Duration = "Unknown"
    else:
        Duration = trackInfo.getPlayTimeString()

    tag.link(path)

    if tag.getArtist() == "":
        Artist = "Unknown Artist"
    else:
        Artist = tag.getArtist()

    if tag.getAlbum() == "":
        Album = "Unknown Album"
    else:
        Album = tag.getAlbum()

    if tag.getTitle() == "":

        Title = titleResolve(path)
    else:
        Title = tag.getTitle()
    try:
        g = tag.getGenre()
```

```
    except:
        Genre = "Unknown"#tag.getGenre()
    else:
        if g == None:
            Genre = "Unknown"
        else:
            #bdir = dirs.encode('utf-8')
            junk = tag.getGenre()
            Genre = junk
    #print str(Genre)+"1"
    taglist.extend([a,Title,Artist,Album,Duration,str(Genre)])
    #print taglist[5]
    return taglist

def findMusic(path):
    myPath =  path#"/home/orefash/Music/music"#raw_input()
    files = [myPath+"/"+file for file in listdir(myPath) if  re.match(r'(.*)mp3',file)]
    return files

def rec():
    print 'a'
    #SELECT col1 FROM tbl ORDER BY RAND() LIMIT 10
    #connect to db
    #db = MySQLdb.connect("localhost","id1191920_orefash","AllHail1Me","id1191920_main_db'
    try:
        #db = MySQLdb.connect("johnny.heliohost.org","orefash","AllHail1Me","orefash_music
        db = MySQLdb.connect("localhost","root","AllHail1Me","musicdb" )
        cursor = db.cursor()
        print "yes"

        #insert to table
        try:
            #print i[0]
            #print "here is %s"%(i[0])
            print "f"
            #show table
            #a = []
            b = []
            cursor.execute("""SELECT * FROM mytable ORDER BY RAND() LIMIT 10""")

            for row in cursor.fetchall():
                a =[]
                for i in row:
```

```
                a.append(i)
            b.append(a)

            #print row[0]

        #cursor.execute("""#INSERT INTO music_tab(name,title,artist,album,duration) VA
        #db.commit()
        print len(b)

    except Exception, e:
        print "error: %s" %e
        db.rollback()

    #print a[0][2]
    #show table
    #cursor.execute("""SELECT * FROM anooog1;""")

    #print cursor.fetchall()
    db.close()
    return b
except Exception, e:
    print "Unable to connect to database: %s" %e
#db = MySQLdb.connect("localhost","orefash","AllHail1Me","musicdb" )
#setup cursor
```