EMOTION IDENTIFICATION OF SONGS

A PROJECT REPORT

Submitted By

K. Prem Nishanth	**31510104071**
S. Rahul	**31510104075**
N. Ramanathan	**31510104079**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

SSN COLLEGE OF ENGINEERING

KALAVAKKAM 603110

ANNA UNIVERSITY :: CHENNAI - 600025

March 2014

ANNA UNIVERSITY : CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report titled **"EMOTION IDENTIFICATION OF SONGS"** is the *bona fide* work of "**K. Prem Nishanth (31510104071)**, **S. Rahul (31510104075)** and **N. Ramanathan (31510104079)**" who carried out the project work under my supervision.

Dr. Chitra Babu
Head of the Department
Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110

Dr. R. S. Milton
Supervisor
Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110

Place:
Date:

Submitted for the examination held on. . . . . .

Internal Examiner **External Examiner**

ACKNOWLEDGEMENTS

We thank GOD, the almighty for giving us strength and knowledge to do this project.

We would like to thank and express our deep sense of gratitude to our guide **Dr. R. S. MILTON**, Professor, Department of Computer Science and Engineering, for his valuable advice and suggestions as well as his continued guidance, patience and support that helped us shape and refine our work.

Our sincere thanks to **Dr. CHITRA BABU**, Professor and Head of the Department of Computer Science and Engineering, for her words of advice and encouragement and we would like to thank our project Coordinator **Dr. S. SHEERAZUDDIN**, Associate Professor, Department of Computer Science and Engineering for his valuable suggestions throughout this project.

We express our deep respect to the founder **Dr. SHIV NADAR**, Chairman, SSN Institutions. We also express our appreciation to **Dr. S. SALIVAHANAN**, our Principal, for all the help he has rendered during this course of study.

We would like to extend our sincere thanks to **Dr. J. J. THIAGARAJAN** and **Mr. K. N. RAMAMURTHY** for their immense help with this project.

We would also like to thank all the teaching and non-teaching staffs of our department who have contributed directly and indirectly during the course of our project work. Finally, we would like to thank our parents and friends for their patience, cooperation and moral support throughout our lives.

K. Prem Nishanth **S. Rahul** **N. Ramanathan**

ABSTRACT

Music has become an integral part in the lives of everyone. Every song portrays a different emotion and people listen to songs that reflect their emotions. The main aim of this project is to classify the emotion of songs as happy or sad. Emotion identification has wide applications in the music industry and can be incorporated as extensions to the existing music players. A fixed dataset that contains audio-related features such as tempo (the speed of the song measured in beats per minute), energy (which is obtained by integrating over the power spectral density), mode (indicating if a song is played in major or minor mode), key (identifying which of the 12 keys the song has been played in) along with its lyrics were used for the classification. Different classification algorithms were used for predicting the emotion using the aforementioned features. Random forests was used to classify the songs based on audio-related features and Naive Bayes Classifier for classifying the songs based on their lyrics. Finally, a combination of both the feature spaces was taken for the classification with Multi-Layer Graphs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Music is integral to our lives. Every song portrays a different emotion. Humans listen to different songs based on their emotions. A system that can classify songs based on emotions in real time by considering various features of the songs and then allows the selection of songs based on emotions can have wide applications in the music industry. The development of such a system is the objective of this project.

1.1 Problem statement

This project aims at designing a system that can classify the emotion of songs as happy or sad. It analyses various features of songs extracted from the audio, and the lyrics.

The system takes as the input is a dataset containing the audio-related features of the song like tempo, energy, mode, key, loudness, harmony and lyrics of the song, and produces as the output contains the predicted emotion of the song. *The objective is to identify the emotion of the song with high accuracy of the prediction.*

1.2 Dataset

We have used a subset of the Million Song Dataset (MSD), which is a freely-available collection of audio-related features and metadata for a million popular music tracks. The subset contains the audio-related features of 10,000 songs. The lyrics were extracted for the songs present in the dataset from different popular lyrics websites. To aid the process, we have also used the dataset provided by `MusixMatch`, which contains the important words present in a song along with the number of occurrences of each of them.

1.3 Organization of the report

The report is organized as follows. Chapter 2 discusses about a few existing systems for identifying the emotion of songs. The first system uses audio-related features and the second uses lyrics for the classification. This chapter also presents the novel approach taken by us against the background of the existing systems that are mentioned. Chapter 3 contains a detailed design of the systems. We have explained about the features and the methods we have used for the classification. Chapter 4 explicates the implementation of the system and presents the results obtained.

CHAPTER 2

LITERATURE SURVEY

Systems for identifying the emotion of songs have been implemented that help users in finding songs based on their emotions. Most of the systems perform the analysis by using audio-related features like tempo, pitch, mode, etc. while some of them perform the classification based on the lyrical content and the meaning of the songs. There are very few systems that incorporate both. Some of the existing systems are given below.

2.1 Audio-related features

Jose Padiel and Ashish Goel, in their project report [4], have developed a music mood classifier that classifies the songs as happy or sad. They have used only the audio-related features for classification and their final dataset comprised 223 songs (137 sad and 86 happy). The audio-related features used by them are energy, tempo, key, mode and harmony. They have used SVM for the classification. SVMs are supervised learning models with associated learning algorithms that analyze data and recognize patterns that can be used for classification purposes. Given a set of training examples, each labelled as belonging to one of two categories (supervised), an SVM training algorithm builds a model that assigns new examples into one category or the other.

2.2 Lyrics

Dang Trung Thanh and Kiyoaki Shirai [5], have developed a system that classifies songs based on lyrics. They use the following three methods for classification.

1. SVM classifier
2. Naive Bayes classifier
3. Graph based method

2.2.1 SVM classifier

The SVM classifier uses the following three features for the analysis of the song.

- Word features
- Word sentiment features
- Artist feature

Word features: All words in the lyrics play a role and hence each word in the lyrics is considered a feature for classification.

Word sentiment feature: There are certain words that reflect the emotion of the song directly. These words are called *sentiment words*. They determine the emotional polarity of the song. They can either be strengthened or weakened by adding words like “very much” or “don’t” respectively. These are called the modifier words. For example, “I love you” implies a positive meaning whereas “I don’t love you” implies a negation. On the other hand, “I love you very much” implies a stronger positive meaning. This implies the three types, namely

1. SW (Sentiment word)
2. NEG-SW (Sentiment word with negation)
3. MOD-SW (Sentiment word with modifier)

Artist feature: Certain artists tend to make songs of a particular genre more so than others. This information aids in the classification. For example, the band Coldplay mostly makes sad, slow and meaningful songs whereas an artist called Blake Shelton makes happy country songs.

2.2.2 Naive Bayes classifier

Basic Model: A Naive Bayes classifier is a probabilistic classifier based on applying Bayes' theorem with strong independent assumptions. It assumes that the presence or absence of a particular feature is unrelated to the presence or absence of any other feature. For example, the lyrics of a song contains the title, introduction, verse and chorus. According to Naive Bayes classifier, each of these features contributes independently to the classification regardless of the presence or absence of the other features.

Significance of chorus and title: The lyrics of each song is divided into several parts as

1. Title
2. Introduction
3. Verse
4. Chorus

Among these, the title and chorus play an important role in providing information about the emotion of the song. The title in general depicts the main theme. The chorus repeats at least twice with slight variations. This again reflects the theme of the song. Hence, the classifier adds more weight to the words from the title and chorus portions of the song.

2.2.3 Graph based methods

A graph consists of nodes and edges. Each node signifies a data item under analysis and an edge between the two data items (or nodes) describes the relation between them. In our problem scenario, each song will be a node and the edge between two songs (or nodes) will be the artist who produced both the songs. A graph of test data is first built and the Naive Bayes model is used in classifying them.

2.3 Novel approach

We have incorporated a few variations in this project: we have included an extra audio feature, *loudness*, and devised a new method to calculate the harmony features. We have also devised a new approach to combine the audio and lyrics feature spaces into a single feature space and perform classification based on it.

CHAPTER 3

DESIGN

The analysis of the song is performed with two sets of features, namely its audio-related features and lyrics . We have used three approaches to identify the song's emotion. The first approach considers only the audio-related features, the second uses only the lyrics, and the third combines both sets of features into a single feature space.

3.1 Audio features module

The dataset contains the following audio-related features:

- Tempo (the speed of the song measured in beats per minute)
- Energy (obtained by integrating over the power spectral density)
- Mode (indicating if a song is played in major or minor mode)
- Key (identifying which of the 12 keys the song has been played in)
- Loudness (general loudness of the track)
- Harmony (the combination of simultaneously sounded musical notes to produce a pleasing effect)

3.1.1 LIBSVM

LIBSVM is a library for Support Vector Machines (SVM). SVMs are supervised learning models with associated learning algorithms that analyze data and

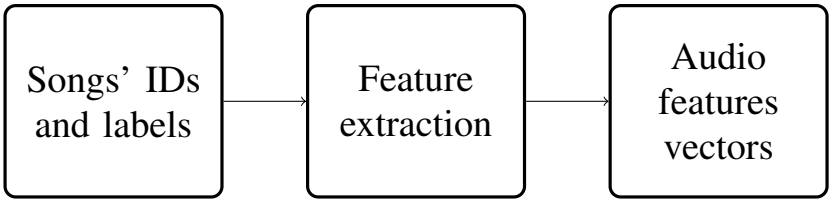

FIGURE 3.1: Audio Features Module 1

recognize patterns that can be used for classification purposes. Given a set of training examples, each marked as belonging to one of two categories (supervised), an SVM training algorithm builds a model that assigns new examples into one category or the other. We have used the Java version of LIBSVM. The feature extraction has been performed as shown in Figure 3.1. The pipeline for the classification using LIBSVM is shown in Figure 3.2

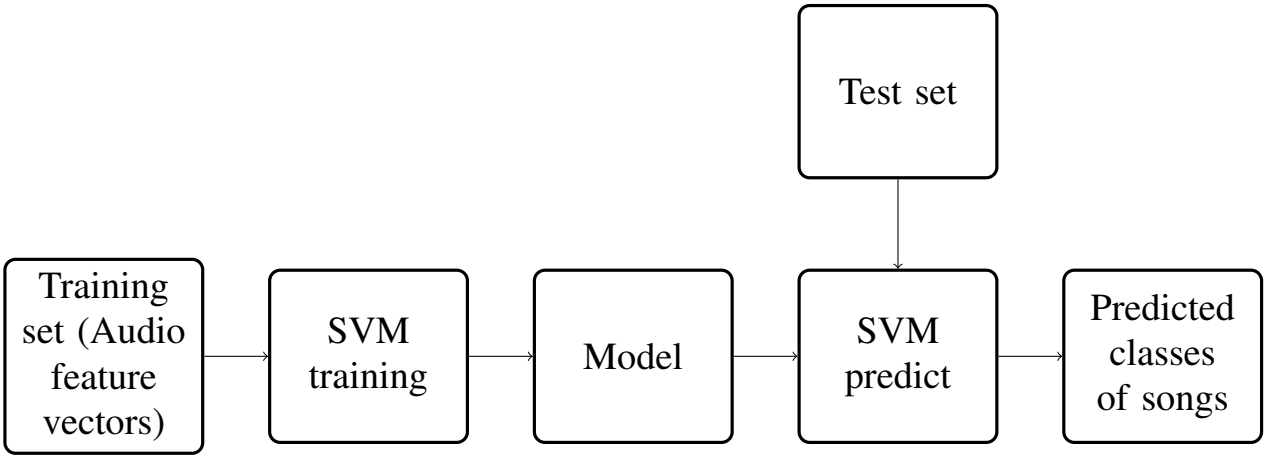

FIGURE 3.2: Audio Features Module 2

The initial input consists of the songs' labels and their track IDs, which is sent to the feature extraction module. This module is a Python program that extracts the required audio features of the songs from the Million Song Dataset and generates the training set in the required format, as shown in Figure 3.3.

The next step is to partition the data into training and testing datasets for cross-validation. Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.

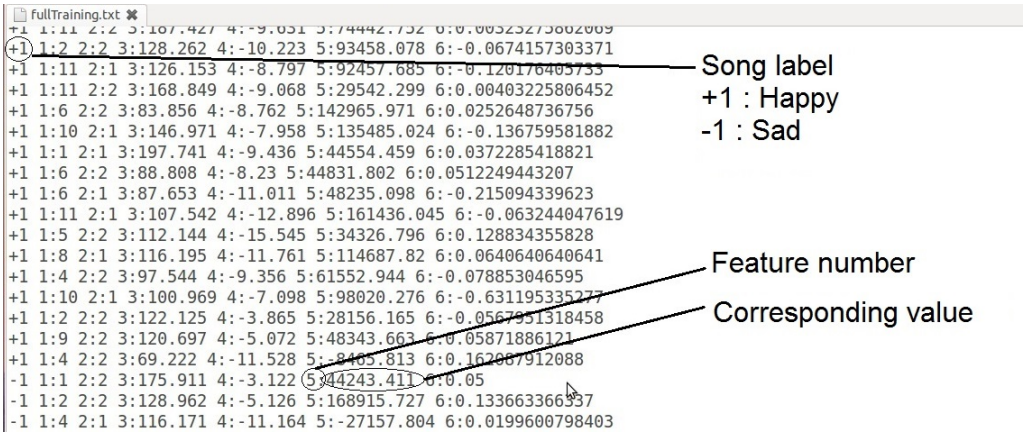

FIGURE 3.3: LIBSVM Input Format

A common type of cross-validation is k-fold cross-validation where the original dataset is randomly split into k equal sized subsets. Of the k subsets, one is chosen to be the test set, and the remaining $k-1$ subsets are used for training. The process is then repeated k time (k fold), with each of the k subsets used exactly once as the test set. The k results can be averaged to calculate the prediction.

We have performed 10-fold Cross-validation on the dataset. The various sets are given as input to the SVM training module which generates a model file which, along with the test sets, is given to the SVM predicting module that generates the output file containing the predicted classes.

3.1.2 LIBLINEAR

LIBLINEAR implements linear SVM. Linear SVM can be used when the training data are linearly separable, i.e., we can find two hyper planes such that they separate the data and there are no points between them. In geometry, hyperplane is a generalization of the two-dimensional plane into an arbitrary number of dimensions. LIBLINEAR can be used for large datasets and is faster than

LIBSVM. The training and test sets for LIBLINEAR are the same as the ones used for LIBSVM.

3.1.3 WEKA toolkit

WEKA stands for Waikato Environment for Knowledge Analysis. It is a freely available toolkit used for classification purposes. The input file has to be given in a format known as `arff`, as shown in Figure 3.4. The name `arff` stands for (Attribute Relation File Format). Once the input training and testing files are given, WEKA automatically applies the chosen algorithm to provide the classification results. We experimented with different algorithms and decided to work with the algorithm Random Forests as it produced the best results.

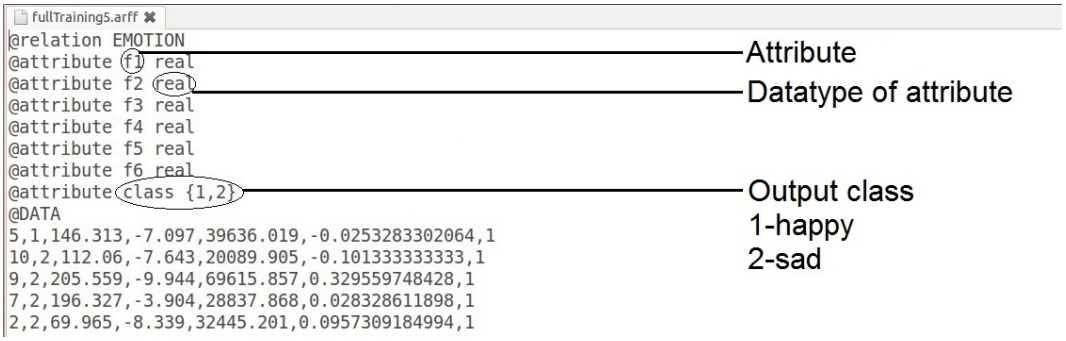

FIGURE 3.4: WEKA Input Format

3.1.4 Random forests

”Random Forests” is an ensemble learning method (uses multiple learning algorithms) for classification that operates by constructing a multitude of decision trees at training time and producing as output the class that is the mode (most

frequently occurring) of the classes output by individual trees. We have implemented the same in Java.

3.2 Lyrics module

We then analysed the lyrics of the song using Naive Bayes classifier.

3.2.1 Naive Bayes classifier

The Naive Bayes module is shown in Figure 3.5

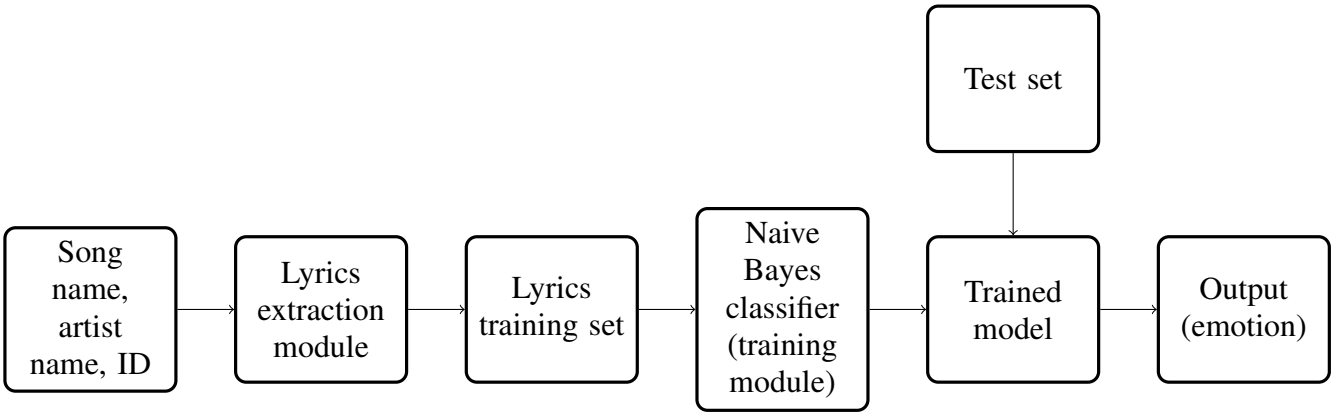

FIGURE 3.5: Lyrics

Basic model: A Naive Bayes classifier is a probabilistic classifier based on applying Bayes' theorem with strong independent assumptions. The Naive Bayes classifier assumes the presence or absence of a particular feature is unrelated to the presence or absence of any other feature. For example, the lyrics of a song contain title, introduction, verse and chorus. Naive Bayes classifier assumes that each of these features contributes independently to classification of the song based on its emotion regardless of the presence or absence of the other features.

A Python program was written to extract lyrics of the song from two of the most popular lyrics websites, namely `www.azlyrics.com` and `www.metrolyrics.com` given the song's name, artist and track ID. The lyrics were kept in two folders, `Happy` and `Sad`, depending on its emotion.

Significance of the chorus and title: The lyrics of each song is divided into several parts as

1. Title
2. Introduction
3. Verse
4. Chorus

Among these, the title and chorus play an important role in providing clue about the emotion of the song. The title of the song in general depicts the main theme of the song. The chorus repeats at least twice in the song with slight variation in some characteristic words. This again reflects the theme of the song. Naive Bayes classifier hence adds more weight to these words from the title and the chorus parts of the song.

3.2.2 NLTK

NLTK stands for Natural Language Toolkit. It is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

The implementation of Naive Bayes found in NLTK was used to classify the songs based on the lyrics.

3.3 Using consensus from multi-layer graphs

Combining multiple features to perform unsupervised and supervised learning is becoming an increasingly popular approach. We considered the problem of emotion classification songs based on the audio features and their lyrical content. The diverse information contained in the audio features and lyrics can enable the design of more robust and effective classifiers. An approach to create a unified representation for data samples described by multiple descriptors is to construct individual graphs and generate a common representation using spectral techniques (e.g. multi-layer graph clustering)[1]. We have adopted the graph-based approach and developed a novel algorithm for classification. The pipeline for the algorithm is shown in Figure 3.6

3.3.1 Feature extraction

- **Audio features**: The same audio features used in random forests were used and appropriate normalization is performed.
- **Lyrics**: The bag of words model of the lyrics has been used. The bag-of-words model is a simplifying representation used in natural language processing and information retrieval. A document is represented as the multi-set of its words, disregarding grammar and word order but keeping multiplicity (number of occurrences of each word)

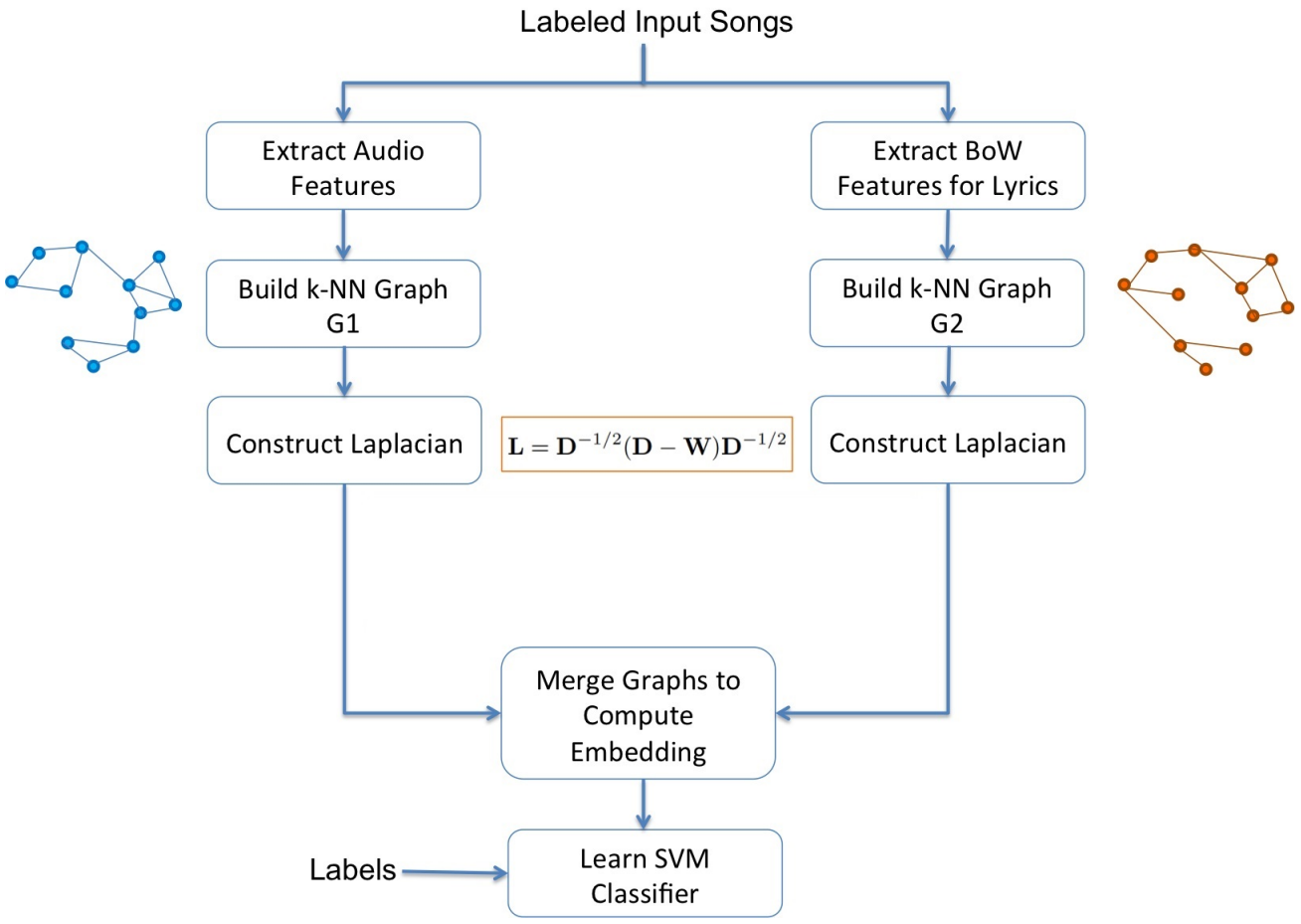

FIGURE 3.6: Training

3.3.2 Multi-layer graph construction

In several scenarios, multiple features are required to describe relationships between entities. For example, to identify songs belonging to the same emotion, we need to use both the audio features and the lyrical content. In such cases, we need to model the relationships for the different features independently, and combine the information for higher level tasks such as classification. Graphs are natural candidates for describing entity relationships. In our case, we have a multi-layer graph where each layer contains the same set of nodes (songs) and the edges in each layer are different, based on the feature set considered.

The first step in the algorithm is to construct undirected, weighted and unsupervised graphs for each layer. Though different graph construction strategies can be employed, we have used a simple K-Nearest Neighbour approach with suitable distance functions for the two modalities (audio features and lyrics). We fix the number of neighbours, K, and for each node we find the K nearest neighbours and create edges between them. The neighbours are found by using L_1 distance function (absolute difference between features) and L_2 distance function (Euclidean distance) between nodes. Since the graph is weighted, the adjacency matrix for the graph, which is the matrix depicting the neighbours of each node, contains the corresponding weights.

For layer 1, we construct the graph using the audio features. Given two features f_i and f_j , we measure the edge weight using the formula

$$W_{ij} = \exp(-\gamma \| f_i - f_j \|_2)$$

if the vertex v_j is one of the K nearest neighbours of v_i. We have used ℓ_2 distance function to calculate this.

Computing embeddings

Let us consider a weighted and undirected graph $G = \{V, E\}$, where V is the set of vertices and E is the set of edges. The adjacency matrix W of the graph is a symmetric matrix whose entry W_{ij} represents the edge weight if there is an edge between vertex v_i and v_j, or 0 otherwise. The degree of a vertex is defined as the sum of the weights of all the edges incident to it in the graph, and the degree matrix D is defined as the diagonal matrix containing the degrees of each vertex along its

diagonal. The normalized graph Laplacian matrix $\mathbf{L}$ is then defined as

$$\mathbf{L} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}. \tag{3.1}$$

The Eigen values and Eigen vectors of the normalized Laplacian matrix can be used to obtain a low-dimensional graph embedding, where nodes that have a high edge weight are projected close to each other while the nodes with a small edge weight are placed far from each other. The spectral clustering algorithm [2], groups the vertices V into disjoint subsets by solving the following minimization problem :

$$\min_{\mathbf{U}} \text{ trace } (\mathbf{U}^T\mathbf{L}\mathbf{U}) \text{ s.t. } \mathbf{U}^T\mathbf{U} = \mathbb{I}, \tag{3.2}$$

where the matrix $\mathbf{U}$ of size $N \times K$ (N vertices and K groups) is the embedding and $\mathbb{I}$ is the identity matrix.

Now extending this to our case, we have two undirected graphs $G_1 = \{V_1, E_1\}$ and $G_2 = \{V_2, E_2\}$. Let us denote the corresponding Laplacians and Embeddings as $\mathbf{L}_1$, $\mathbf{L}_2$ and $\mathbf{U}_1$, $\mathbf{U}_2$ respectively. Using an approach similar to [1], we infer a consensus embedding $\mathbf{U}$ that is close to both $\mathbf{U}_1$ and $\mathbf{U}_2$. This resulting embedding captures the relationships described by both the graphs.

3.3.3 Computing projection for novel test samples

For a test sample, described by both audio features and a lyric model, we need to project it onto the embedding space estimated using the training data. We use an approach based on sparse coding to achieve this. Sparse coding aims to represent a data sample as a linear combination of pre-defined basis functions. Given a set

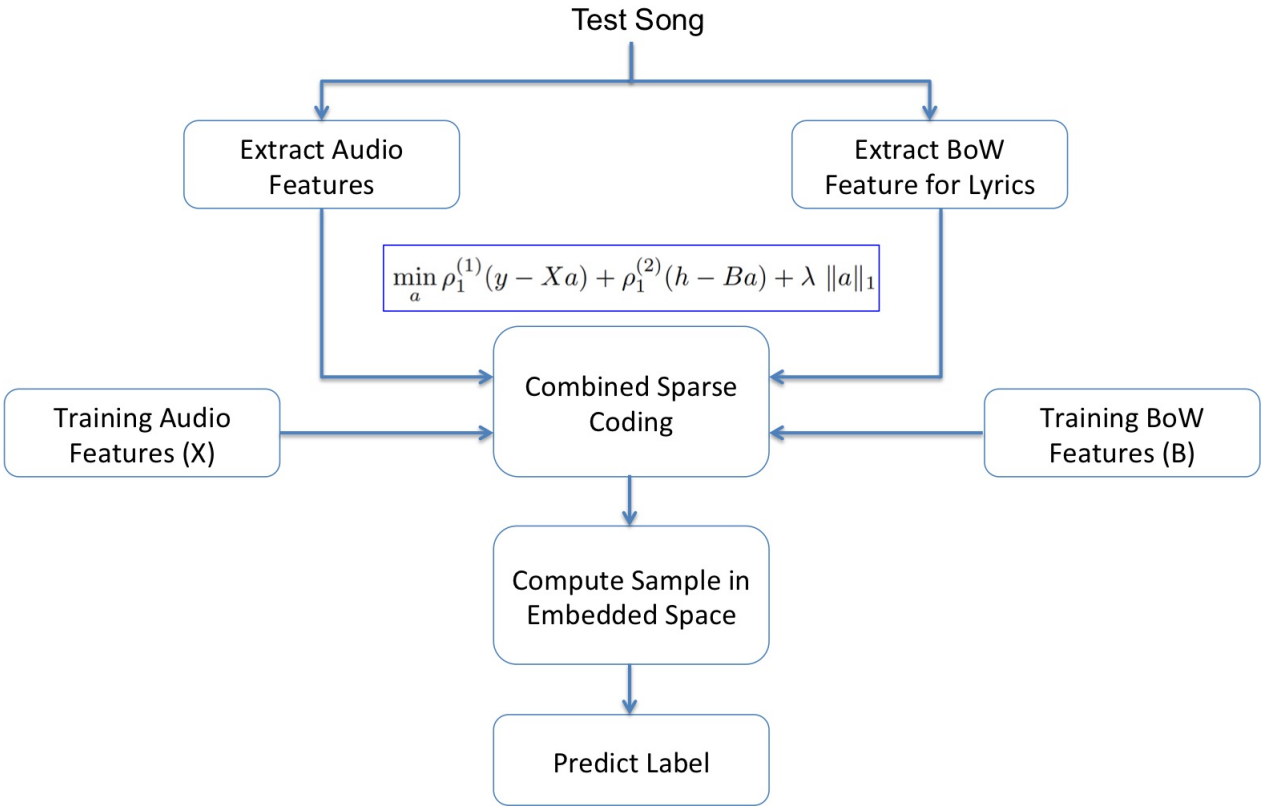

FIGURE 3.7: Testing

of basis functions stored in a dictionary D, we can sparsely code a data sample y as

$$\min_a \rho_1(y - Da) + \lambda\rho_2(a) \tag{3.3}$$

where ρ_1 is the *loss function* that measures the distance between y and Da, ρ_2 is the sparsity *regularizer* on a, and λ is the regularization penalty that controls the trade-off between loss and regularization. In our case, we use the training data samples as the dictionary. Since we have two different modalities (audio and lyrics), we propose to obtain a common representation using the two dictionaries as

$$\min_a \rho_1^{(1)}(y - Xa) + \rho_1^{(2)}(h - Ba) + \lambda \|a\|_1, \tag{3.4}$$

This is not straightforward to solve Equation 3.4. We use the algorithm proposed

in [3] to obtain the representation a. Now, we compute the embedding for the test sample as

$$\mathbf{u}_{test} = \mathbf{U}a. \tag{3.5}$$

3.3.4 Classification

For the training data, we learn a $2-$class linear SVM classifier using the embeddings. Now, for a test sample we compute $\mathbf{u}_{test}$ and predict the class using the SVM classifier.

CHAPTER 4

IMPLEMENTATION

This chapter describes the implementation of the system. First, we have discuss about features of songs. Then we explain the classification based on audio features, and that using the lyrics of the songs. We implement the final classification using multi-layer graphs.

4.1 Audio features extraction from songs

We have used the features of the songs available in the Million Song Dataset. The following audio feature extraction tools were used to extract the audio features of the songs given as input.

1. **YAAFE** (Yet Another Audio Feature Extraction): We experimented with the YAAFE tool. But many of the required features mentioned were not available for extraction.

2. **MusicBrainz server**: MusicBrainz is an open music encyclopedia that collects music metadata. It has metadata available for all the songs in the Million Song Dataset (MSD). A local musicbrainz server, 27 GB, in size was downloaded and set up but connection to its database failed.

4.2 Manual labelling of songs

Over 800 songs by familiar and unfamiliar artists from the dataset were manually labelled. Each song was classified with the following labels in mind:

- *Happy or Sad*
- *Fast or Slow*
- *Heavy or Light*

The classified file looked as shown in Figure 4.1

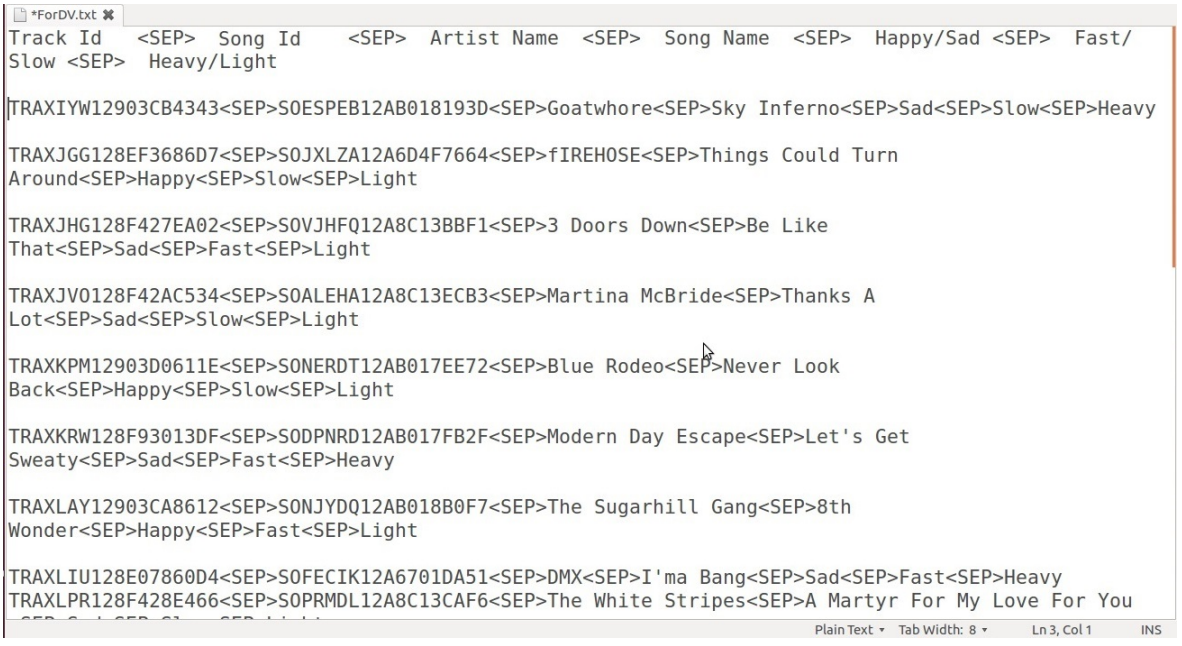

FIGURE 4.1: Classified Songs

We have used only the emotion of the song (happy/sad) for classification. The other labels can be used in the future. Some songs were found to be neither happy nor sad and they had to be omitted. The final dataset comprised 500 songs.

Key Name	Key Number	Frequency(Hz)
A0	1	27.500
A0#	2	29.135
B0	3	30.868
C1	4	32.703
C1#	5	34.648
D1	6	36.708
D1#	7	38.891
E1	8	41.203
F1	9	43.654
F1#	10	46.249
G1	11	48.999
G1#	12	51.913

TABLE 4.1: 12 Keys and Frequencies

4.3 Audio features used

The following audio features were used.

1. **Key**: It identifies which of the 12 keys (labelled as 1 to 12) the song has been played in. The 12 keys and their corresponding frequencies are given below.
2. **Mode**: Mode defines the set of musical notes used in a song. There are predominantly two types, minor and major.
3. **Tempo**: Tempo refers to the speed of the song (measured in beats per minute).
4. **Energy**: Energy is the work done to produce a tone at a particular frequency. The SegmentsTimbre table in the MSD consists of 12 columns each representing a key. The song is split into 0.3 seconds long segments. Each segment along with its 12 corresponding frequency values makes up a row. For example, a song that is 6 minutes long (360 seconds) is divided

into 1200 segments thus resulting in a 1200×12 table for that song. The sum of all values in the matrix is taken as the overall energy of the song.

5. **Loudness**: It refers to the general loudness of the track and is the perception of amplitude.
6. **Harmony**: Musically speaking, most happy-sounding harmonies contain the root note of the song (key) and the note that follows it after four semitones, while most sad-sounding harmonies contain the root note and the note that follows it after three semitones. A semitone is defined as the smallest interval used in music, equal to half a tone; a half step.

   We have taken this as the basis of our approach in calculating the harmony features. The "SongsPitch" table in the MSD contains the SongSegmentNumber along with the contribution of each one of the 12 notes in each segment. We take the number of the segments in which the key, the third and the fourth semitones dominate the segment and store the values. These will be used as the harmony features.

4.4 Analyzing the features and their combinations

All the features listed above are stored in the Million Song Dataset as HDF5 files. HDF stands for Hierarchical Data Format. It is designed to store and organize large amounts of numerical data. We wrote a small tool in Python that takes the song's name as input and extracts the required audio-related features from the dataset. This program makes use of `h5py`, a Python package that provides an interface to the HDF5 data format. The HDF5 file is shown in Figure 4.2 and Figure 4.3

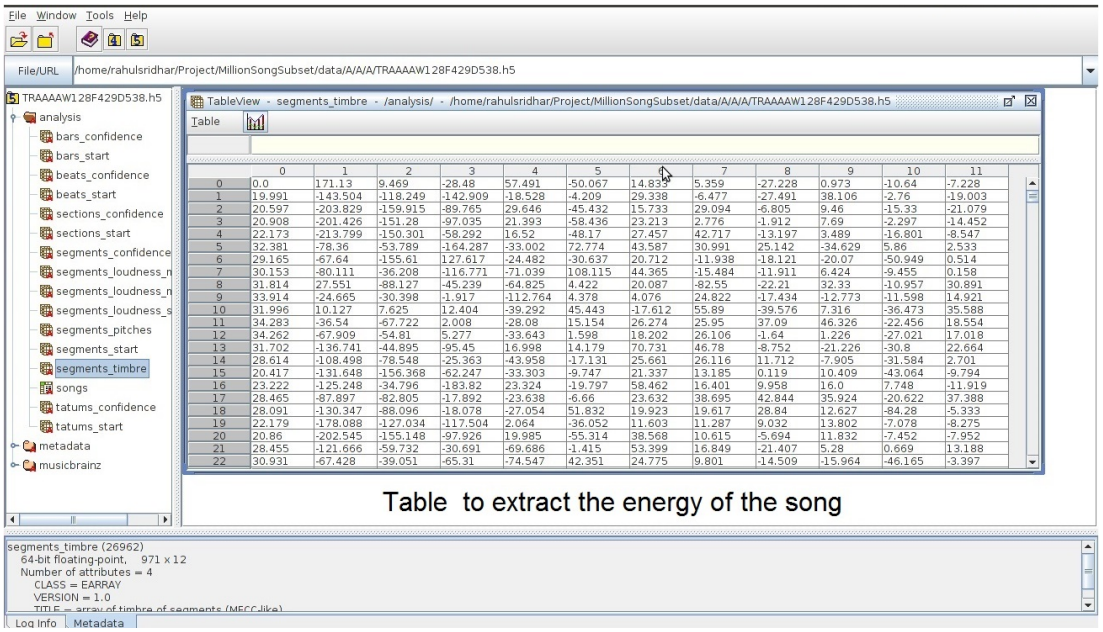

FIGURE 4.2: HDF Dataset, SegmentsTimbre Table

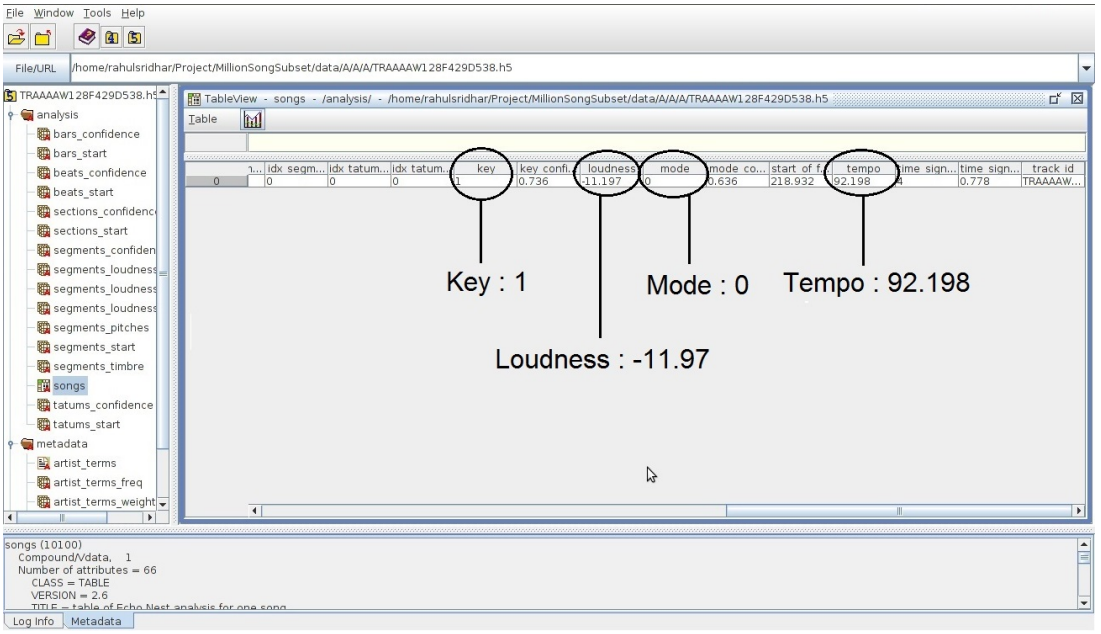

FIGURE 4.3: HDF Dataset, Songs Table

Each feature will contribute to predict the emotion of the song. Energy and tempo were taken as the main features and combinations with other features (mode, key, loudness and harmony) were tried out and the results of the analysis were noted.

Separate approaches were used for classification based on audio-related features

and lyrics: Random Forests was chosen to classify the songs based on audio features and the implementation of Naive Bayes found in NLTK (Natural Language ToolKit) was used to classify based on lyrics.

4.5 Classification based on audio features

This section explains how the songs are classified using the audio features alone.

4.5.1 LIBSVM

The training set file was prepared in the appropriate format and given as input to the `SVMtrain` Java program. This creates a model file. The model file, along with the test set, is given as input to the `SVMpredict` program. The final output file generated by `SVMpredict` contains the predicted classes of the test set. 10-fold Cross-validation was performed and the prediction accuracy in each case was noted (for an initial set of 110 songs), as given in Table 4.2. The cross-validation results, along with the training set and test set, are given in Figure 4.4 and Figure 4.5 respectively. The output is shown in Figure 4.6

4.5.2 LIBLINEAR

Initially, when the dataset was small (110 songs), the results obtained from LIBSVM were satisfactory. However, when the size of the dataset was increased to 500 songs, the results were biased towards the major class in the training set ('sad'). Hence we opted for LIBLINEAR.

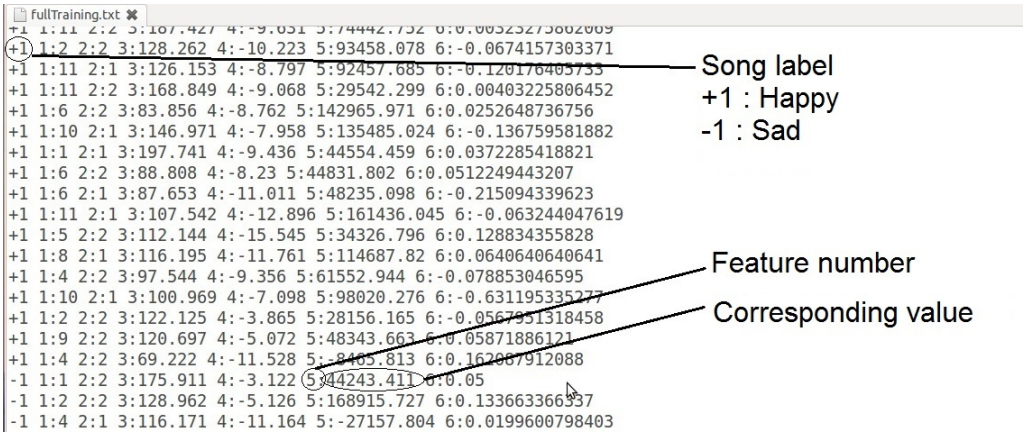

FIGURE 4.4: LIBSVM Training Set

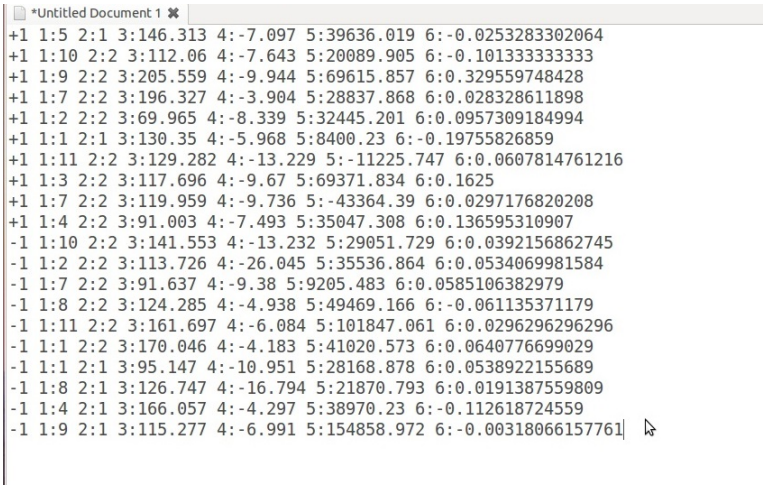

FIGURE 4.5: LIBSVM Test Set

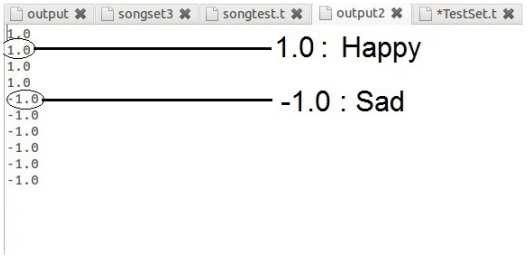

FIGURE 4.6: LIBSVM Output

LIBLINEAR implements linear SVM. It can be used for large datasets and is faster than LIBSVM. The input training set and test set for LIBLINEAR is similar to that of LIBSVM. The accuracy of classification obtained during cross-validation was 56%. The LIBLINEAR output is shown in Figure 4.7

Test set number	Accuracy (%)
1	50
2	70
3	50
4	50
5	60
6	60
7	50
8	50
9	50
10	50
11	90
Average	57.2

TABLE 4.2: Cross-validation results

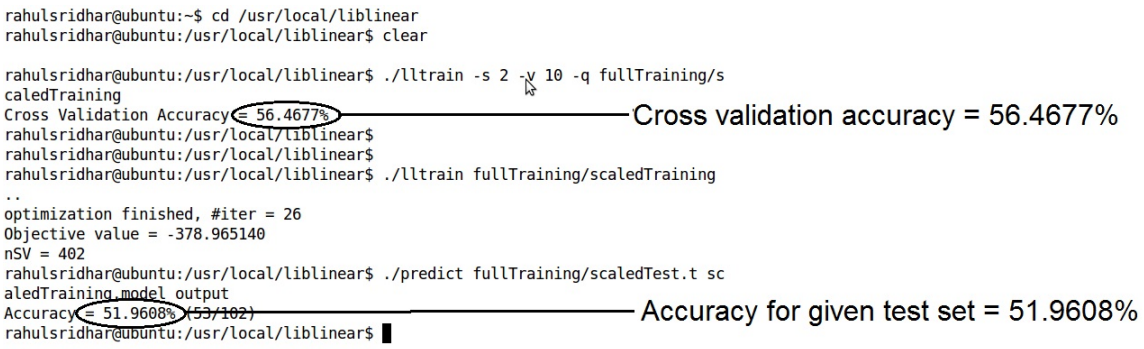

FIGURE 4.7: LIBLINEAR output

4.5.3 WEKA toolkit

The input file for the WEKA toolkit was prepared in the required arff format. Cross-validation sets were created using a Python program. This program takes as

input the entire set of 500 songs and creates training and test sets to facilitate 10-fold cross-validation (as shown in Figures 4.8, 4.9 and 4.10). Training and testing was done using various classifiers in the toolkit and the results were noted. The performance of Random forests was found to be the best amongst all. Hence, we chose Random forests for classification based on audio features.

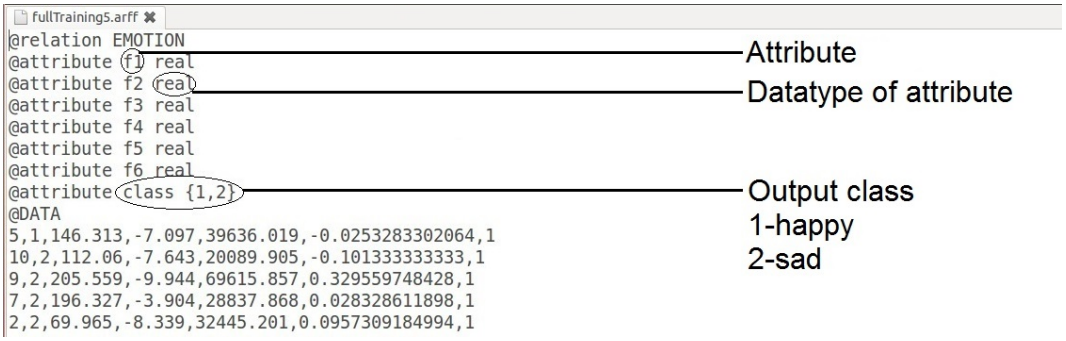

FIGURE 4.8: WEKA training set

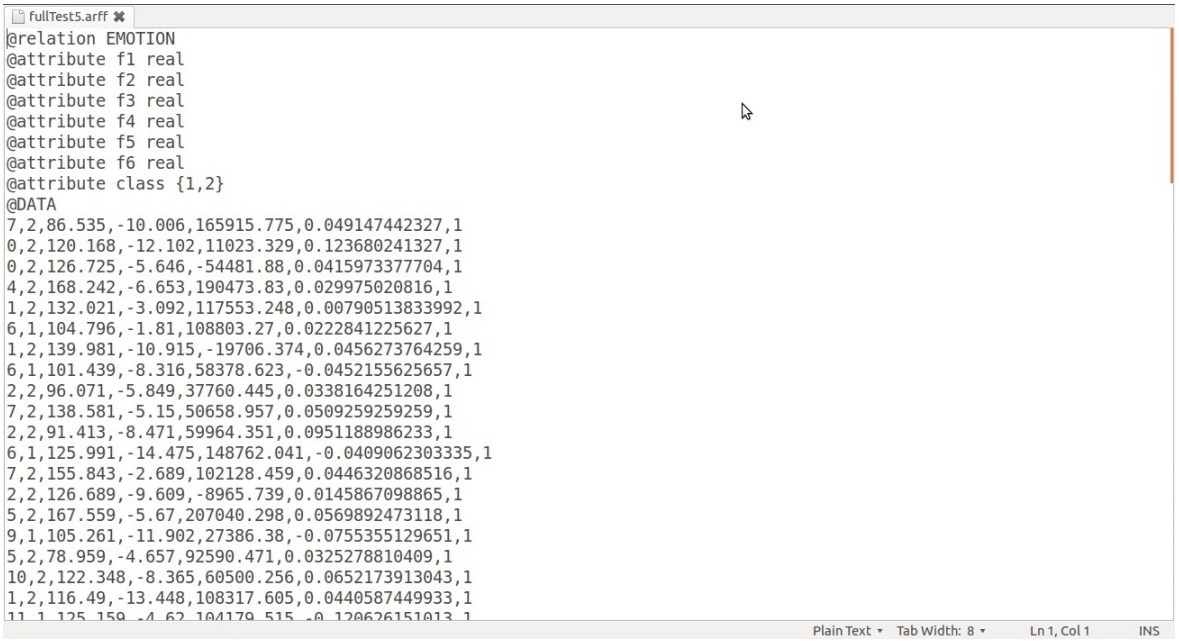

FIGURE 4.9: WEKA test set

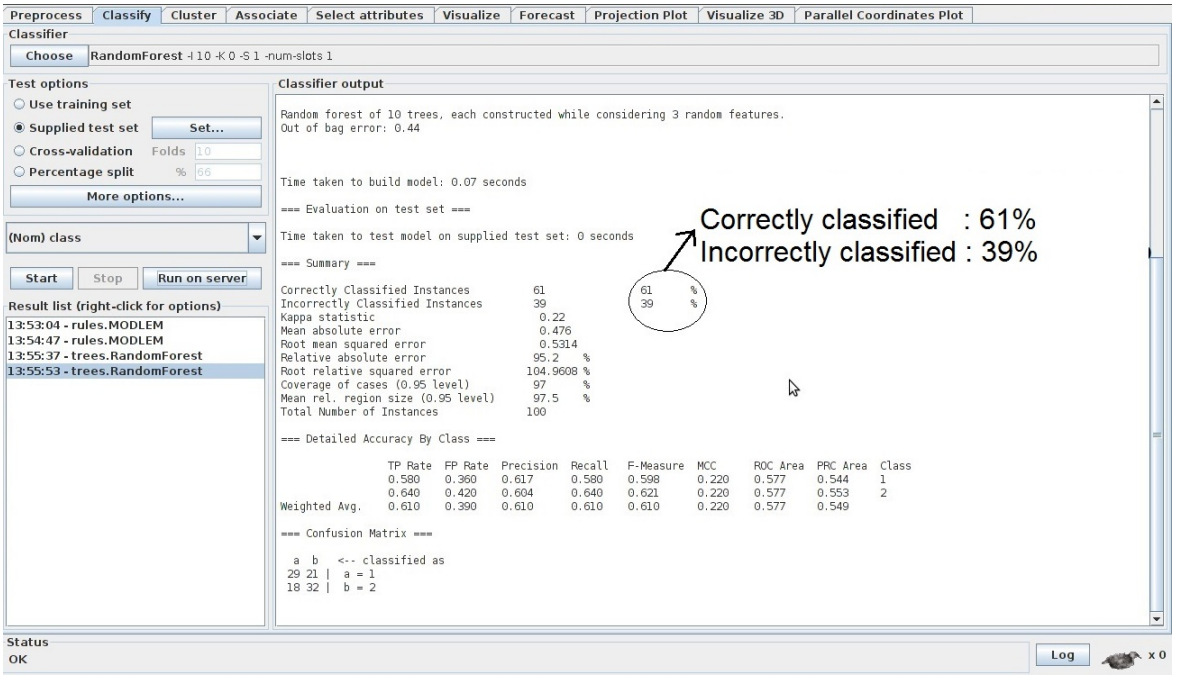

FIGURE 4.10: Result of Random forest

4.5.4 Random forests

We have implemented Random forests and modified it to our specifications. The coding was done in Java with NetBeans as the frontend. Several decision trees were created using a combination of all audio features (energy, tempo, mode, key, loudness and harmony). Each node in the tree represents a feature and has 2 child nodes, one node to represent feature values greater than the corresponding feature's average value and one to represent feature values lesser than the average. The average value was calculated based on the following formula:

$$\text{avg} = \text{avgT} - \frac{|\,\text{avgH} - \text{avgS}\,| * |\,\text{countH} - \text{countS}\,|}{\text{countT}}$$

where

avg: weighted average value

avgT: average of all songs

avgH: average of happy songs

avgS: average of sad songs

countH: count of happy songs

countS: count of sad songs

countT: count of all songs

The trees are of different depths, depending on the number of features used in creating them. Two sample trees are given in Figure 4.11 and Figure 4.12. Figure 4.13 describes the random forests module.

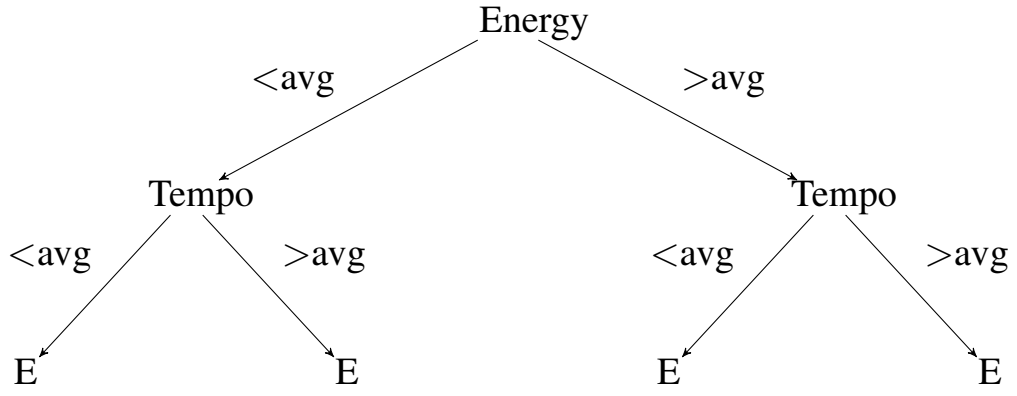

FIGURE 4.11: Random Forest - Decision Tree 1

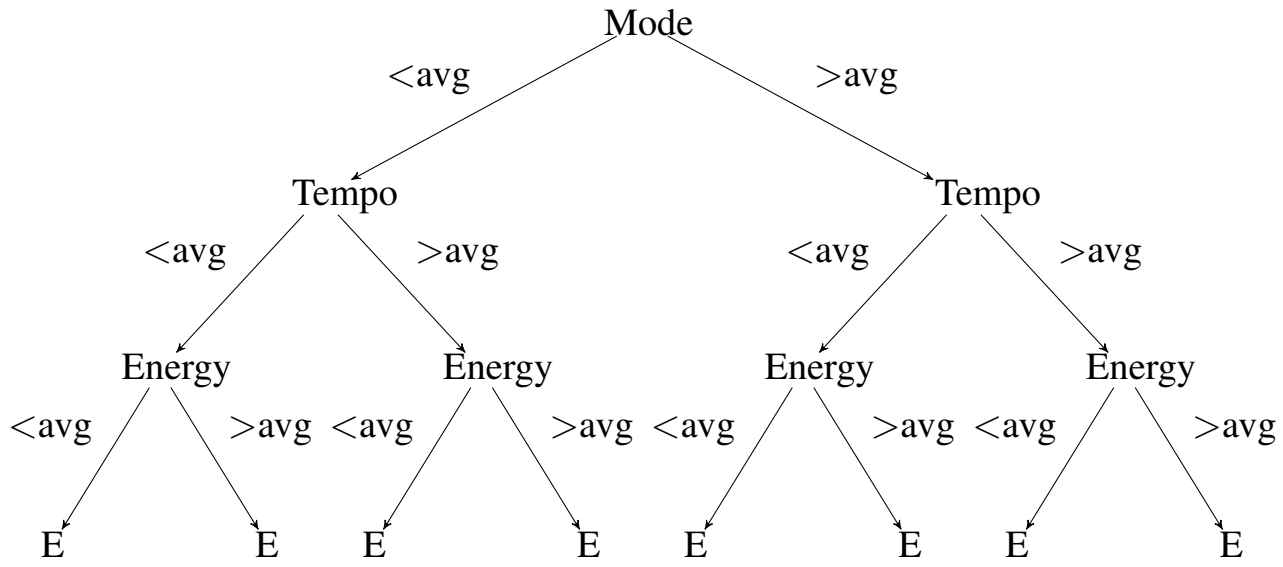

FIGURE 4.12: Random Forest - Decision Tree 2

avg: weighted average

E: predicted emotion

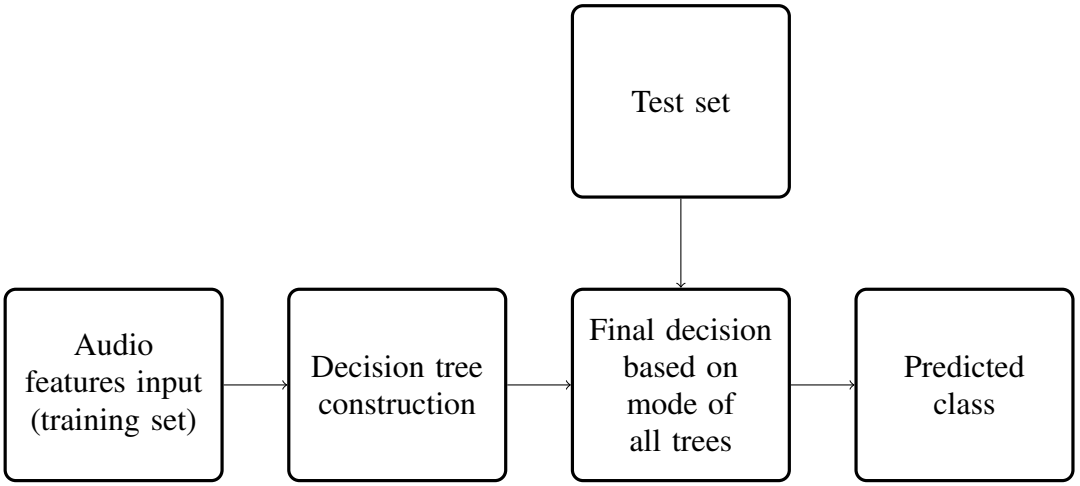

FIGURE 4.13: Random Forests module

The training set, prepared in the appropriate format was then given as input to the program which created decision trees based on it. Testing was done by comparing the test data with all decisions trees and storing each tree's prediction. Finally, the

mode of all predictions were taken and compared with the actual emotion of the song.

The results were noted, with and without including the new features, namely loudness and harmony. When they were not included the average accuracy was 56.25% but when they were included the average accuracy came up to 61%, thus justifying our decision to include those features. The results from random forests are shown in Figures 4.14 and 4.15.

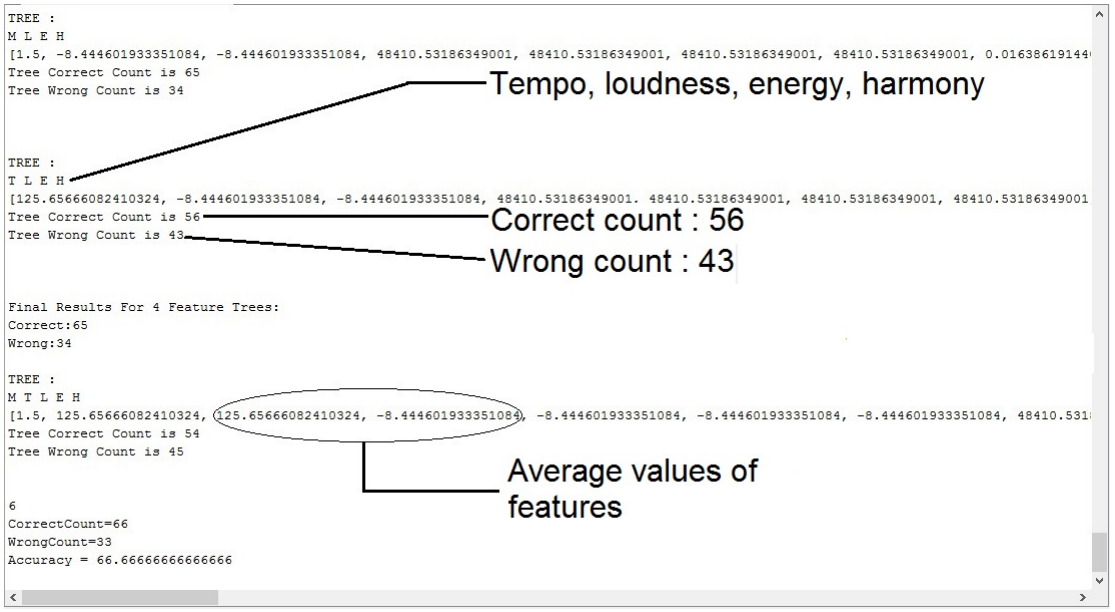

FIGURE 4.14: Output of Random Forest

A front-end was designed using NetBeans IDE and it allows the user to select a dataset from the given 5 datasets and correspondingly choose the happy or sad songs in them.

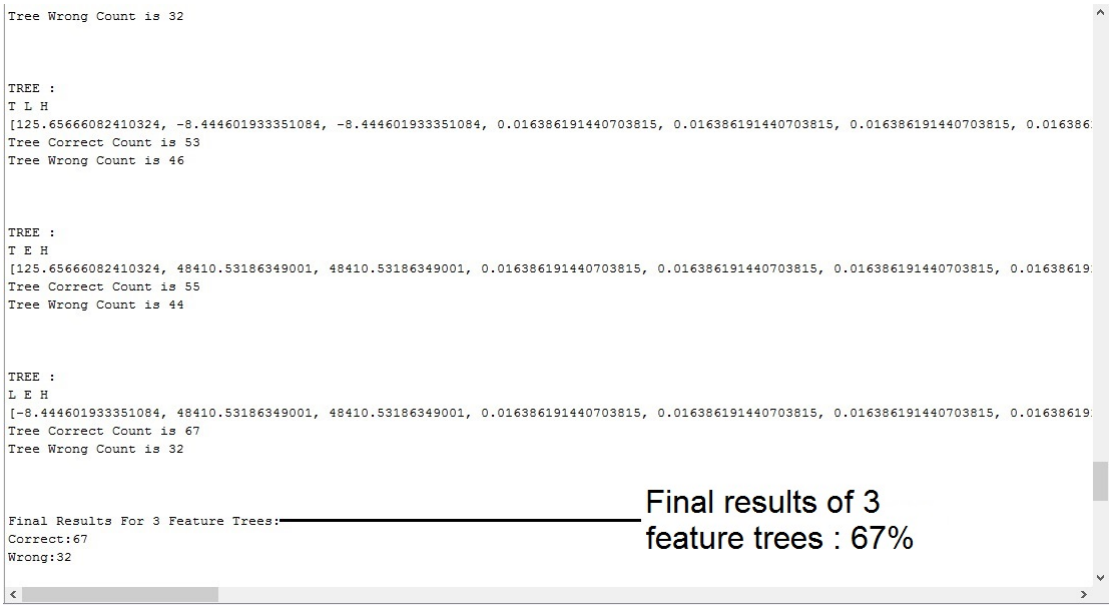

FIGURE 4.15: Output of Random Forest

4.6 Classification based on lyrics

We wrote a small utility in Python to extract lyrics automatically from two popular lyrics websites, which takes the song name and artist name as input. The program formats the input accordingly for each website and extracts the lyrics. The websites used are

1. AZlyrics.com: The appropriate format
   `www.azlyrics.com/ArtistName/SongName.html`

2. Metrolyrics.com: The appropriate format
   `www.metrolyrics.com/SongName-lyrics-ArtistName.html`

The extracted lyrics were all stored in two separate folders, one for happy songs and the other for sad. This comprises the dataset for lyrics. A sample lyrics file is shown in Figure 4.16.

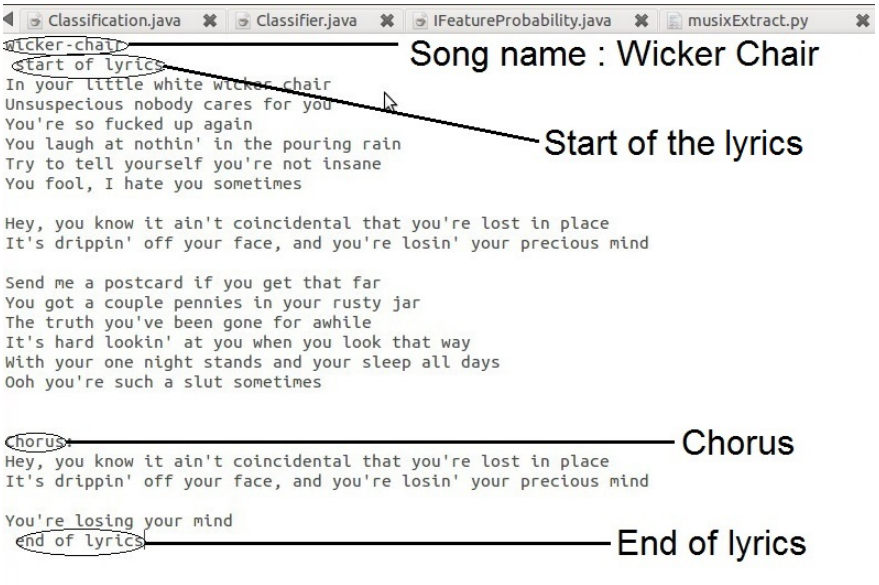

FIGURE 4.16: Song lyrics

4.6.1 Word list

A 6800 strong word list containing two sets of words viz positive words and negative words were used. Each song was classified by counting the number of times a word from each of these list appears in its lyrics. The emotion was predicted by a numerical comparison of the positive and negative counts. A prediction accuracy of 58% was obtained. This is acceptable since the word list is not exhaustive and can be improved.

4.6.2 Bag of words

The dataset provided by musiXMatch was used. It contains a set of the most frequently occurring words along with their frequencies for 2 lakh songs present in the million song dataset. A Python program was written to extract the bag of words for the 500 songs in our dataset. The bag of words representation of the

lyrics is as shown in Figure 4.17. LIBLINEAR was used to classify the same. A prediction accuracy of 53% was obtained.

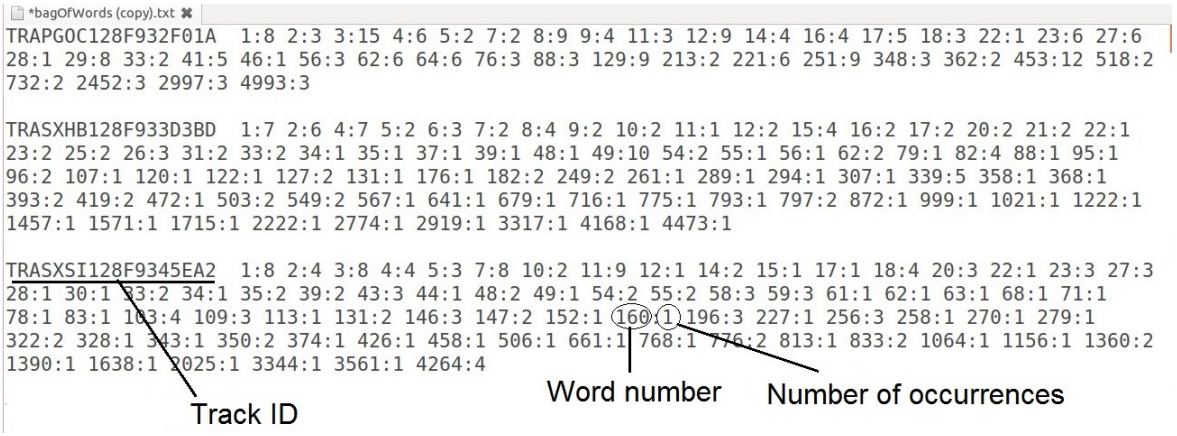

FIGURE 4.17: Bag of Words

4.6.3 Naive Bayes classifier

An existing implementation of Naive Bayes classifier in Java was initially used to classify lyrics. Separate lyric sets are used for training and testing. The output is predicted either as positive, representing happy, or as negative, representing sad. The classification results are shown in Figure 4.18

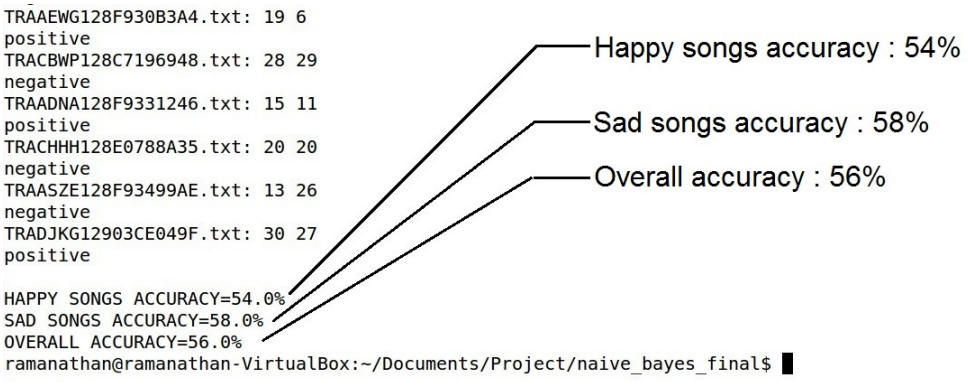

FIGURE 4.18: Output of Naive Bayes classifier

4.6.4 NLTK

NLTK stands for Natural Language Toolkit. It is a suite of libraries and programs for Natural Language Processing (NLP) for Python programming language. The implementation of Naive Bayes found in NLTK was used to classify the songs based on lyrics. The results obtained were accurate up to a maximum of 75% in certain cases (as shown in Figures 4.19 and 4.20).

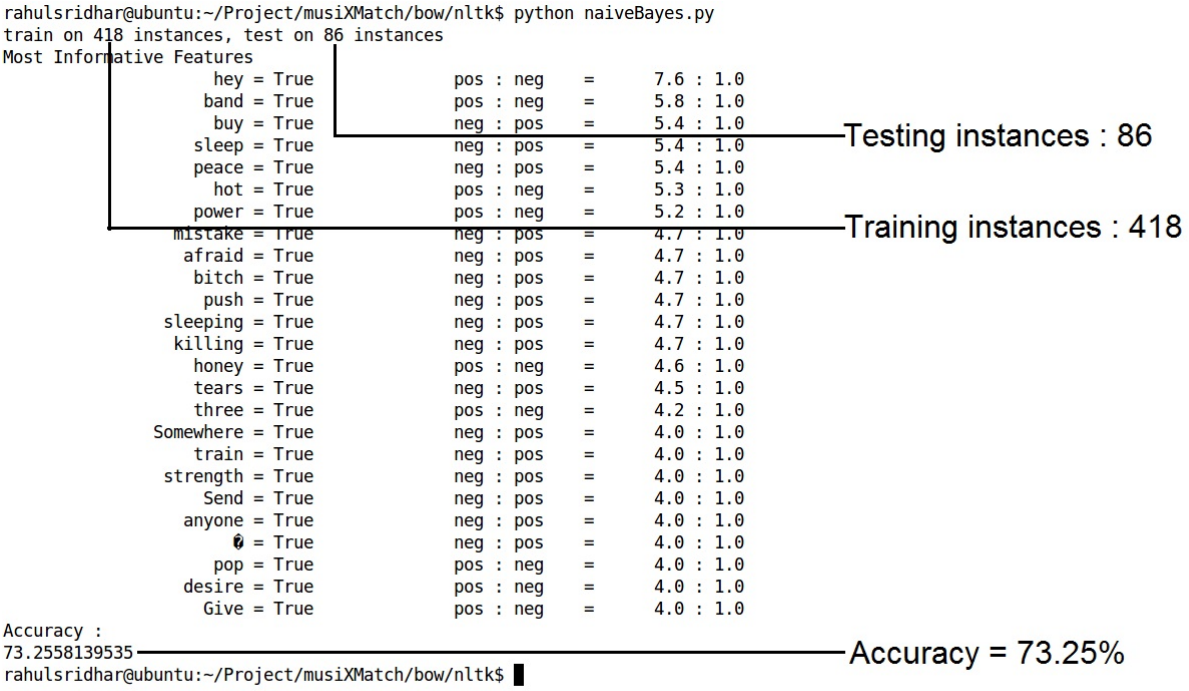

FIGURE 4.19: Output from the Naive Bayes classifier of NLTK for a training set

```
rahulsridhar@ubuntu:~$ cd Project/musiXMatch/bow/nltk
rahulsridhar@ubuntu:~/Project/musiXMatch/bow/nltk$ clear
rahulsridhar@ubuntu:~/Project/musiXMatch/bow/nltk$ python naiveBayes.py
train on 380 instances, test on 56 instances
Most Informative Features
                    hey = True              pos : neg    =      7.0 : 1.0
                 pretty = True              pos : neg    =      6.3 : 1.0
                  worth = True              neg : pos    =      6.3 : 1.0
                 moment = True              neg : pos    =      5.7 : 1.0
                    kid = True              pos : neg    =      5.7 : 1.0
                  sleep = True              neg : pos    =      5.4 : 1.0
                  tears = True              neg : pos    =      5.0 : 1.0
                 afraid = True              neg : pos    =      5.0 : 1.0
               sleeping = True              neg : pos    =      5.0 : 1.0
                   tear = True              neg : pos    =      5.0 : 1.0
                 Always = True              pos : neg    =      5.0 : 1.0
                    hot = True              pos : neg    =      4.6 : 1.0
                   move = True              pos : neg    =      4.4 : 1.0
                  cross = True              neg : pos    =      4.3 : 1.0
                   read = True              pos : neg    =      4.3 : 1.0
                 desire = True              pos : neg    =      4.3 : 1.0
                    buy = True              neg : pos    =      4.3 : 1.0
                    guy = True              pos : neg    =      4.3 : 1.0
                     uh = True              pos : neg    =      4.3 : 1.0
                  stone = True              pos : neg    =      4.3 : 1.0
                  peace = True              neg : pos    =      4.3 : 1.0
                    Did = True              neg : pos    =      4.3 : 1.0
                   foot = True              pos : neg    =      4.3 : 1.0
                      G = True              pos : neg    =      4.3 : 1.0
                  alive = True              pos : neg    =      4.3 : 1.0
Accuracy :
71.4285714286 ——————————————————————————————————————————— Accuracy = 71.42%
rahulsridhar@ubuntu:~/Project/musiXMatch/bow/nltk$
```

FIGURE 4.20: Output from the Naive Bayes classifier of NLTK for a different training set

4.7 Using consensus from multi-layer graphs

All the modules have been implemented in Python. The audio features and bag of words are stored in separate matrices. These are given as input to the main driver module. KNN graphs are constructed for both audio features and bag of words. The `compute embedding` module creates an embedding which is a combined representation of both the graphs. This represents the training data. The `sparse code` module creates a sparse representation of the test data and returns training samples closest to it. This makes up the test data.

The training data and test data are given as input to SVMTrain and SVMPredict which outputs the prediction accuracy as shown in Figure 4.21.

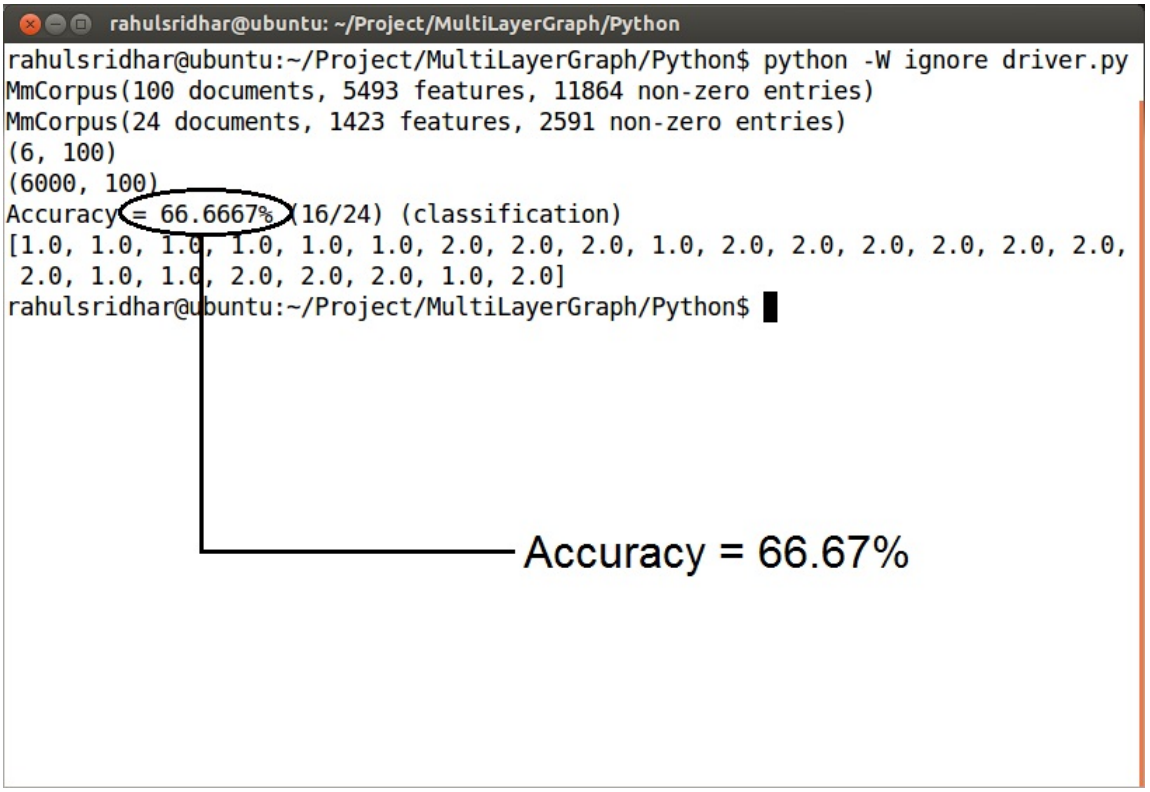

FIGURE 4.21: Multi Layer Graph

CHAPTER 5

CONCLUSION AND FUTURE WORK

We implemented a system to identify the emotion of the songs and discussed the detailed design and implementations of the algorithms used in the report.

The classification based on audio-related features was initially done using LIBSVM. It worked quite decently for small datasets. However, on increasing the size of the dataset, the prediction became biased towards the major class in the training set. Similar problems were encountered while using LIBLINEAR. We then experimented with different classifiers from the WEKA toolkit. Random forests produced the best results. We implemented the same in Java, modifying it according to our specifications and used it to classify the songs.

For classification based on lyrics, the word list approach and the bag of words approach were tried individually. The results were not satisfactory. Finally, we zeroed in on the Naive Bayes classifier. The implementation of Naive Bayes in NLTK was used for the classification based on lyrics.

For classification based on a combination of the feature spaces, multi-layer graphs was used. The results obtained from testing on smaller datasets look promising. It can be tested on bigger datasets in the future.

REFERENCES

1. X. Dong, P. Frossard, P. Vandergheynst and N. Nefedov. (2009) *Clustering on multi-layer graphs via subspace analysis on Grassmannian manifolds*

2. A. Ng, M. Jordan, and Y. Weiss. (2001) *On spectral clustering - analysis and an algorithm*, Advances in NIPS.

3. K.N. Ramamurthy, A.Y. Aravkin and J.J. Thiagarajan. (2014) *Beyond L2 loss functions for learning sparse models*, ACM KDD. (Submitted)

4. Jose Padial, Ashish Goel. (2011) *Music Mood Classification*, Carneggie Mellon University.

5. Dang Trung Thanh, Kiyoaki Shirai. (2009) *Machine Learning Approaches for Mood Classification of Songs toward Music Search Engine*, Japan Advanced Institute of Science and Technology.

6. Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman and Paul Lamere. (1983) *The Million Song Dataset*, In Proceedings of the 12th International Society for Music Information Retrieval Conference.