

**B.E. PROJECT**

On

# **SONG – HIT PREDICTION**

Submitted by

**Payal Goel (302/CO/11)**

**Pramit Mallick (306/CO/11)**

**Rahul Duggal (311/CO/11)**

In partial fulfilment of B.E. (Computer Engineering) degree of  
Delhi University

Under the Guidance of

**Dr. Shampa Chakraverty**



Department Of Computer Engineering

Netaji Subhas Institute of Technology

University Of Delhi, Delhi

May 2015

## **Certificate**

This is to certify that the dissertation entitled “**Songs Hit Prediction**” being submitted by Payal Goel, Pramit Mallick and Rahul Duggal in the Department of Computer Engineering, Netaji Subhas Institute of Technology, Delhi, for the award of the degree of “Bachelor of Engineering” is a bona fide record of the work carried out by them. They have worked under my guidance and supervision and have fulfilled the requirements for the submission of this report, which has reached the requisite standard.

The results contained in this report have been submitted in part or in full, to any university or institute for the award of any degree or diploma.

**Dr. Shampa Chakraverty**

Professor and Head of Department

Computer Engineering Department

Netaji Subhas Institute of Technology

## Candidate's Declaration

This is to certify that the work which is being hereby presented by us in this project titled **“Songs Hit Prediction”** in partial fulfilment of the award of the Bachelor of Engineering submitted at the Department of Computer Engineering, Netaji Subhas Institute of Technology Delhi, is a genuine account of our work carried out during the period from January 2015 to May 2015 under the guidance of Prof. Shampa Chakraverty, Head of Department(COE), Netaji Subhas Institute of Technology, Delhi.

The matter embodied in the project report to the best of our knowledge has not been submitted for the award of any other degree elsewhere.

Dated:

Payal Goel (302/CO/11)

Pramit Mallick (306/CO/11)

Rahul Duggal (311/CO/11)

This is to certify that the above declaration by the students is true to the best of my knowledge.

Prof. Shampa Chakraverty

## Acknowledgements

Any Project indisputably plays one of the most important roles in an engineering student's life to make him a successful engineer. It provides the students with an opportunity to gain valuable experience on the practical application of their technical knowledge and also brings out and hones their technical creativity. Thus the need for one is indispensable.

We would like to express our deep gratitude towards our mentor **Dr. Shampa Chakraverty**, Head of Department, Computer Engineering Department, Netaji Subhas Institute of Technology, New Delhi under whose supervision we completed our work. Her invaluable suggestions, enlightening comments and constructive criticism always kept our spirits up during our work. She was always there to help whenever we faced any problems.

Our experience in working together has been wonderful. We hope that the knowledge, practical and theoretical, that we have gained through this term B.E. Project will help us in our future endeavours in the field.

We are grateful to all our friends for providing critical feedback and support whenever required.

We regret any inadvertent omissions.

Payal Goel

Pramit Mallick

Rahul Duggal

(302/CO/11)

(306/CO/11)

(311/CO/11)

# Table of Contents

I.	Certificate	2
II.	Candidate Declaration	3
III.	Acknowledgements	4
IV.	List of Figures	7
V.	Abstract	8
1.	Chapter 1: Introduction	9
2.	Chapter 2: Literature Survey	12
2.1.	Machine Learning	12
2.2.	K-means Clustering	15
2.3.	Support Vector Machine	15
2.4.	Linear Regression	16
2.5.	Logistic Regression	18
2.6.	EchoNest	20
2.7.	PCA	27
2.8.	Parrott's	29
2.9.	UMBC Semantic Similarity	31
3.	Chapter 3: Architecture & Methodology	36
3.1.	Data Set Collection	37
3.2.	Applying Classifiers and Analysis	49
4.	Chapter 4: Experiments & Results	54
4.1.	Plotting Locations of the Artists	54
4.2.	K –means Clustering (Genres)	55
4.3.	Detailed Analysis	56

4.3.1. PCA on features	56
4.3.2. Emotion Composition of song	58
4.3.3. Classification using Linear Regression, Logistic Regression and SVMs	59
5. Chapter 5: Conclusion & Future	64
6. References	65
7. Appendix A – Tools and Platforms used	67
8. Appendix B – Code	69

## List of figures

1. Fig 2.1 Output of the EchoNest API
2. Fig 2.2 Sample JSON Output of the API
3. Fig 2.3 Plot of the JSON
4. Fig 2.4. EchoNest features
5. Fig 2.5 Parrott's Classification I
6. Fig 2.6 Parrott's Classification II
7. Fig 3.1 Composition of phase 'Data Collection'
8. Fig 3.2 a sample of Billboard's Website
9. Fig 3.3 A sample of data collected through parsing for year 1947.
10. Fig 3.5 List of songs along with its acoustic features
11. Fig 3.6. Screenshot of the master data of year 1977
12. Fig 4.1 Plot of Artists location
13. Fig 4.2 Sample Result after clustering
14. Fig 4.4 PCA Analysis
15. Fig 4.5 Comparison of Audio & Lyrics Models
16. Fig 4.6 Only Audio Features
17. Fig 4.7 Only Lyrical Features
18. Fig 4.8 Audio & Lyrical Features
19. Fig 4.9 Composition of Emotions Decade-Wise
20. Fig 4.10 Checking the weights of the feature vectors

# Abstract

Record Companies invest billions of dollars in new talent each year. They would benefit tremendously from gaining insight into what actually makes a hit song. In our thesis, in order to tackle this problem, we extracted both acoustic and lyrical features of the songs using EchoNest API and mood classification on the lyrics. We separate the hit songs from the non-hits using classifiers like linear regression, logistic regression and Support Vector Machines. Experiments have been performed on over 3000 songs and it is found that a mix of acoustic features and lyrical features provide a sturdier metric for quantifying the popularity of a song. These acoustic and lyrical features can be used to mine large amount of information.

In a particular case, acoustic features gave an accuracy of around 58% whereas the lyric features gave an accuracy of around 51%. The case was the other way around in the 1960's song prediction where the lyrical features outperformed the acoustic ones.

We have also compared the performance of linear regression, logistic regression and support vector machines(SVM) classifiers. The results show that logistic regression very narrowly outperforms SVC for the classification task.



# Chapter 1: Introduction

Record Companies invest billions of dollars in new talent each year. They would benefit tremendously from gaining insight into what actually makes a hit song. On April 4, 1964 the Beatles accomplished what no band had achieved until then; indeed what no other band has achieved since. In addition to holding the No. 1 USA single with “Can’t Buy Me Love” the Beatles also held the No.2 slot and No.3 slot. In fact, Beatles songs occupied the first five positions in the charts. What exactly was it that fuelled the Beatles’ rise to fame? Is there any intrinsic quality in music that predisposes it to greatness?

This idea is the main drive behind the new research field referred to as “Hit Song Science” which Pachet [9] define as “emerging field of science that aims at predicting the success of songs before they are released in the market”. Many music information retrieval (MIR) systems have been developed for a range of different purposes such as automatic classification per genre [12] and composer [5], yet, as it appears, the use of MIR systems for hit prediction remains relatively unexplored. Dhanaraj and Logan [2] explored the use of support vector machine (SVM) and boosting classifiers to distinguish top 1 hits from other songs in various styles based on acoustic and Lyric- based features. Pachet and Roy [10] tried to develop an accurate classification model for low, medium or high popularity based on acoustic and human features but were unsuccessful. Borg and Hokkanen [1] draw similar conclusions as [10] when trying to predict the popularity of music videos based on their YouTube view count by training support vector machines but were unsuccessful. The experiment by Ni et al. [8]

has more optimistic results when predicting if a song would reach a top 5 position on the UK top 40 singles chart compared to top 30-40 position. Herremans.et.al [] focuses on predicting whether a song is a “top 10” dance hit versus a lower listed position using SVM classifier.

Although societal, cultural and other qualitative factors undoubtedly play a part in songs’ popularity, in this work; we search for some factor that can be quantified in songs which makes them more likely to reach the top of the charts. In this thesis, we use acoustic and lyrics based features and also some of the features related to songs like latitude and longitude of the place where the song was recorded; for making a prediction about the song hit. We extracted acoustic features like acoustiness, danceability, duration, energy, key, liveness, etc using the EchoNest API. We use tf-idf (term frequency – inverse document frequency) of each word along with its similarity to various emotions like joy, love, happiness, pain; etc to quantify the lyrics based features. Parrot’s classification is used to determine lyrics-based feature. Once the features are extracted, we use the classifiers like Support vector Machines, linear classifiers like linear regression and logical regression. Classifiers are used to classify the songs, whether they will be a perennial hit or not. After classification, a comparison is made among various classifiers and best classifier (which generates the best results i.e. results having minimum errors) is found. We also use unsupervised learning technique like k-means clustering to classify the songs into clusters on the basis of its acoustic and lyrical features and analyse its results. Latitude and longitude of the places where the songs were recorded, is also used to analyze the areas from where most of the hit songs originate. Thus, it is seen that extracting these features may help us to analyze large amount of information.

The remaining thesis is organized as follows. In chapter 2, we discuss about the previous works done in this area. Chapter 3 discusses about our contribution to this field of research. Chapter 4 provides an experimental evaluation of our proposed approach. Chapter 5 summarizes the thesis with conclusions and Future work.

## **Chapter 2: Literature Survey**

This chapter gives extensive background behind the concepts employed in this project. None of the work in this chapter is original; the ideas from each section have been cross referenced to indicate the source of the information presented, whenever needed. This study was very necessary from the point of view of getting the background information to help is proceed in applying the proposed techniques and obtaining the results.

### **2.1. Machine Learning**

Machine learning, a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviours based on empirical data, such as from sensor data or databases. Machine Learning is a scientific discipline that addresses the following question: 'How can we program systems to automatically learn and to improve with experience?' Learning in this context is not learning by heart but recognizing complex patterns and make intelligent decisions based on data. The difficulty lies in the fact that the set of all possible decisions given all possible inputs is too complex to describe. To tackle this problem the field of Machine Learning develops algorithms that discover knowledge from specific data and experience, based on sound statistical and computational principles.

The field of Machine Learning integrates many distinct approaches such as probability theory, logic, combinatorial optimization, search, statistics, reinforcement learning and control theory. The developed methods are at the

basis of many applications, ranging from vision to language processing, forecasting, pattern recognition, games, data mining, expert systems and robotics.

A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as examples that illustrate relations between observed variables. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviours given all possible inputs is too large to be covered by the set of observed examples (training data). Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new case. We have followed the same principle, where we have designed algorithms to generalize from the given examples and then produce a useful output. Machine learning algorithms are described as either 'supervised' or 'unsupervised'

### 2.1.1. Unsupervised Learning

In machine learning, unsupervised learning refers to the problem of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning and reinforcement learning. Unsupervised learning is closely related to the problem of density estimation in statistics. However unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data. Many methods employed in unsupervised learning are based on data mining methods used to preprocess data. Unsupervised learners are not provided with classifications. In fact, the basic task of unsupervised learning is to develop

classification labels automatically. Unsupervised algorithms seek out similarity between pieces of data in order to determine whether they can be characterized as forming a group. These groups are termed clusters, and there are whole families of clustering machine learning techniques. If we train directly using the unsupervised learning algorithms such as K-mean clustering we can divide the tweets into different classes but even then it would require us to have user interference to decide which class tweets is important. However it has advantage of being automatic classifier when the decision parameter is unknown and clustering of objects needs to be performed.

### 2.1.2. Supervised Learning

Supervised learning is the machine learning task of inferring a function from supervised (labeled) training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which is called a classifier (if the output is discrete, see classification) or a regression function (if the output is continuous, see regression). Supervised learning is when the data you feed your algorithm is "tagged" to help your logic make decisions. Example - Bayes spam filtering, where you have to flag an item as spam to refine the results.

Our classifying approach is a supervised learning method that uses labelled dataset of songs having acoustic and lyrical features as basis of classifier. Using some of the songs as training dataset, classifier is applied on testing dataset to identify the songs which are going to be a perennial hit.

## 2.2. K-Means Clustering

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means clustering aims to partition  $n$  observations into  $k$  clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

### Description

Given a set of observations  $(x_1, x_2, \dots, x_n)$ , where each observation is a  $d$ -dimensional real vector,  $k$ -means clustering aims to partition the  $n$  observations into  $k$  ( $\leq n$ ) sets  $S = \{S_1, S_2, \dots, S_k\}$  so as to minimize the within-cluster sum of squares (WCSS). In other words, its objective is to find:

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

Where  $\mu_i$  is the mean of points in  $S_i$ .

## 2.3. Support Vector Machines

In machine learning, support vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary

linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

In addition to performing linear classification, SVM's can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapped their inputs into high-dimensional feature spaces.

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high-or infinite dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training –data point of any class, since in general the larger the margin the lower the generalization error of the classifier

## 2.4. Linear Regression

In statistics, linear regression is an approach for modelling the relationship between a scalar dependent variable  $y$  and one or more explanatory variables (or independent variables) denoted  $X$ . The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regressions. In linear regression, data are modelled using linear predictor functions, and unknown model parameters are estimated from the data. Such models are called linear models. Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways,



such as by minimizing the “lack of fit” in some other norm or by minimizing a penalized version of the least squares loss function as in ridge regression and lasso.

## Description

Given a data set  $\{y_i, x_{i1}, \dots, x_{ip}\}_{i=1}^n$  of  $n$  statistical units, a linear regression model assumes that the relationship between the dependent variable  $y_i$  and the  $p$ -vector of regressors  $x_i$  is linear. This relationship is modeled through a disturbance term or error variable  $\varepsilon_i$  — an unobserved random variable that adds noise to the linear relationship between the dependent variable and regressors. Thus the model takes the form

$$y_i = \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i, \quad i = 1, \dots, n,$$

where  $T$  denotes the transpose, so that  $x_i^T \boldsymbol{\beta}$  is the inner product between vectors  $x_i$  and  $\boldsymbol{\beta}$ .

Often these  $n$  equations are stacked together and written in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Where  $y_i$  is called the regress and, endogenous variable, response variable, measured variable, criterion variable, or dependent variable.  $x_{i1}, x_{i2}, \dots, x_{ip}$  are regressors, exogenous variables, explanatory variables, covariates, input variables, predictor variables, or independent variables.  $\beta$  is a  $p$ - dimensional parameter vector. Its elements are also called effects, or regression coefficients.  $\epsilon_i$  is called the error term, disturbance term, or noise. This variable captures all other factors which influence the dependent variable  $y_i$  other than the regressors  $x_i$ . The relationship between the error term and the regressors, for example whether they are correlated, is a crucial step in formulating a linear regression model, as it will determine the method to use for estimation.

## 2.5. Logistic Regression

In statistics, logistic regression or logit regression or logit model is a direct probability model that was developed by statistician D.R. Cox in 1958. The binary logistic model is used to predict a binary response based on one or more predictor variables (features). That is, it is used in estimating the parameters of a qualitative response model. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables, which are usually continuous, by estimating probabilities.

### 2.5.1. Definition of logistic function

An explanation of logistic regression begins with an explanation of the logistic function. The logistic function is useful because it can take an input with any value from negative to positive infinity, whereas the output always takes values

between zero and one[14] and hence is interpretable as a probability. The logistic function  $\sigma(t)$  is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}},$$

If  $t$  is viewed as a linear function of an explanatory variable  $x$  (or of a linear combination of explanatory variables), then we express  $t$  as follows:

$$t = \beta_0 + \beta_1 x$$

And the logistic function can now be written as:

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Note that  $F(x)$  is interpreted as the probability of the dependent variable equalling a "success" or "case" rather than a failure or non-case. It's clear that the response variables  $Y_i$  are not identically distributed:  $P(Y_i = 1 | X)$  differs from one data point  $X_i$  to another, though they are independent given design matrix  $X$  and shared with parameters  $\beta$

## 2.6. Echo Nest API

The Echo Nest API allows you to call methods that respond in JSON or XML. We must have API key to make use of the Echo Nest API. API calls for getting data about songs. Some of the most powerful features of the Echo nest API are the deep understanding of musical attributes about songs. API is used to return the audio summary data associated with that song. Here are some of the audio\_summary results found using API.

- **Tempo:** represents the BPM of the song.
- **Danceability:** A number that ranges from 0 to 1, representing how danceable the Echo Nest thinks this song is.
- **Duration:** Length of the song, in seconds.
- **Energy:** A number that ranges from 0 to 1, representing how energetic The Echo Nest think the song is.
- **Key:** The key that The Echo Nest believes the song is in. Key signatures start at 0(C) and ascend the chromatic scale. In this case, a key of 1 represents a song in D-flat.
- **Loudness:** The overall loudness of a track in decibels (dB).
- **Mode:** Number representing whether the song is in minor (0) or major (1) key. Use this in conjunction with key.
- **Time\_signature:** Time signature of the key; how many beats per measure.

A more detailed description of some of the features are mentioned below:

**Danceability:**

Describes how suitable a track is for dancing using a number of musical elements (the more suitable for dancing, the closer to 1.0 the value). The combination of musical elements that best characterize danceability include tempo, rhythm stability, beat strength, and overall regularity.

**Energy:**

Represents a perceptual measure of intensity and powerful activity released throughout the track. Typical energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

**Speechiness:**

Detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.

**Liveness:**

Detects the presence of an audience in the recording. The more confident that the track is live, the closer to 1.0 the attribute value. Due to the relatively small population of live tracks in the overall domain, the threshold for detecting liveness is higher than for speechiness. A value above 0.8 provides strong likelihood that the track is live. Values between 0.6 and 0.8 describe tracks that may or may not be live or contain simulated audience sounds at the beginning or end. Values below 0.6 most likely represent studio recordings.

**Acousticness:**

Represents the likelihood a recording was created by solely acoustic means such as voice and acoustic instruments as opposed to electronically such as with synthesized, amplified, or effected instruments. Tracks with low acousticness include electric guitars, distortion, synthesizers, auto-tuned vocals, and drum machines, whereas songs with orchestral instruments, acoustic guitars, unaltered voice, and natural drum kits will have acousticness values closer to 1.0.

**Valence:**

Describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g., happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry). This attribute in combination with energy is a strong indicator of acoustic mood, the general emotional qualities that may characterize the track's acoustics. Note that in the case of vocal music, lyrics may differ semantically from the perceived acoustic mood.

The EchoNest returns some meta data about the song and then using signal processing techniques, it calculates the various parameters

---

## Output Data

- **meta data:** analyze, compute, and track information.

- **track data**

- **time signature:** an estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
- **key:** the estimated overall key of a track. The key identifies the tonic triad, the chord, major or minor, which represents the final point of rest of a piece.
- **mode:** indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived.
- **tempo:** the overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
- **loudness:** the overall loudness of a track in decibels (dB). Loudness values in the Analyzer are averaged across an entire track and are useful for comparing relative loudness of segments and tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude).
- **duration:** the duration of a track in seconds as precisely computed by the audio decoder.
- **end of fade in:** the end of the fade-in introduction to a track in seconds.
- **start of fade out:** the start of the fade out at the end of a track in seconds.
- **codestring, echoprintstring:** these represent two different audio fingerprints computed on the audio and are used by other Echo Nest services for song identification. For more information on Echoprint, see <http://echoprint.me>.
- **synchstring:** a synchronization code that allows a client player to synchronize the analysis data to the audio waveform with sample accuracy, regardless of its decoder type or version. See Synchstring section.

- 
- **rhythmstring:** a representation of spectro-temporal transients as binary events. This temporal data distributed on 8 frequency channels aims to be independent of timbre and pitch representations. See Rhythmstring section.
  - **timbre, pitch, and loudness** are described in detail as part of the *segments* interpretation below.
  - **sequenced data:** the Analyzer breaks down the audio into musically relevant elements that occur sequenced in time. From smallest to largest those include:
    - **segments:** a set of sound entities (typically under a second) each relatively uniform in timbre and harmony. Segments are characterized by their perceptual onsets and duration in seconds, loudness (dB), pitch and timbral content.
      - **loudness\_start:** indicates the loudness level at the start of the segment
      - **loudness\_max\_time:** offset within the segment of the point of maximum loudness
      - **loudness\_max:** peak loudness value within the segment
    - **tatums:** list of tatum markers, in seconds. Tatums represent the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events (segments).
    - **beats:** list of beat markers, in seconds. A beat is the basic time unit of a piece of music; for example, each tick of a metronome. Beats are typically multiples of tatums.
    - **bars:** list of bar markers, in seconds. A bar (or measure) is a segment of time defined as a given number of beats. Bar offsets also indicate downbeats, the first beat of the measure.
    - **sections:** a set of section markers, in seconds. Sections are defined by large variations in rhythm or timbre, e.g. chorus, verse, bridge, guitar solo, etc. Each section contains its own descriptions of tempo, key, mode, time\_signature, and loudness.

Fig 2.1 Output of the EchoNest API

## JSON Schema Example

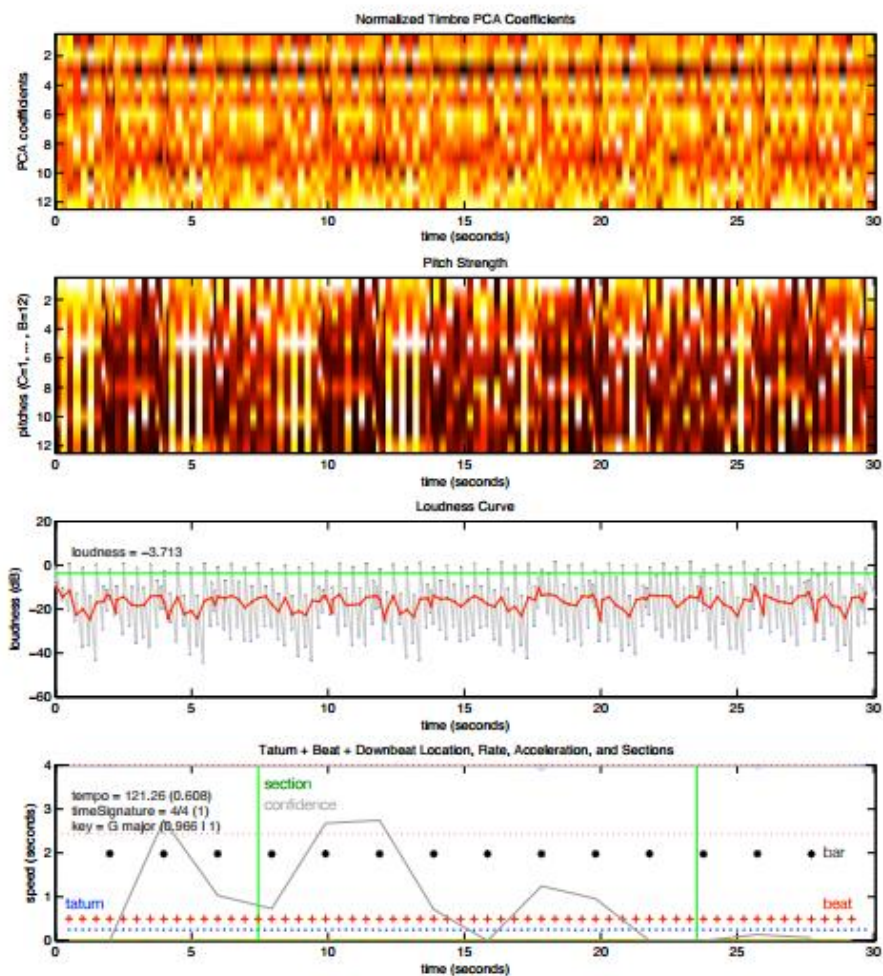
```
{
  "meta":
  {
    "analyzer_version": "3.08b", "detailed_status": "OK", "filename": "/Users/Jim/
    Desktop/file.mp3", "artist": "Michael Jackson", "album": "Thriller",
    "title": "Billie Jean", "genre": "Rock", "bitrate": 192, "sample_rate": 44100,
    "seconds": 294, "status_code": 0, "timestamp": 1279120425, "analysis_time":
    3.83081
  },
  "track":
  {
    "num_samples": 6486072, "duration": 294.15293,
    "sample_md5": "0a84b8523c00b3c8c42b2a0eaabc9bcd", "decoder": "mpg123",
    "offset_seconds": 0, "window_seconds": 0, "analysis_sample_rate": 22050,
    "analysis_channels": 1, "end_of_fade_in": 0.87624, "start_of_fade_out":
    282.38948, "loudness": -7.078, "tempo": 117.152, "tempo_confidence": 0.848,
    "time_signature": 4, "time_signature_confidence": 0.42, "key": 6,
    "key_confidence": 0.019, "mode": 1, "mode_confidence": 0.416,
    "codestring": "eJwdk8U7m4Rz9Pej...tbtSnk8U7m4Rz980uF", "code_version": 3.15,
    "echoprintstring": "eJzFnQuyrTquZbsENjamOf5A_5t...pDF6eF__7eH_D9MWE8p",
    "echoprint_version": 4.12, "synchstring": "eJxlWwmWJCSOu0ocIWz2-1-
    ssSRDZP...nUf0aeyz4=", "synch_version": 1.00, "rhythmstring":
    "eJxVnAmyHdCM...9v71b_92-5V-fmWuX2n", "rhythm_version": 1.00

  },
  "bars":
  [{ "start": 1.49356, "duration": 2.07688, "confidence": 0.037}, ...],
  "beats":
  [{ "start": 0.42759, "duration": 0.53730, "confidence": 0.936}, ...],
  "tatums":
  [{ "start": 0.16563, "duration": 0.26196, "confidence": 0.845}, ...],
  "sections":
  [{ "start": 0.00000, "duration": 8.11340, "confidence": 1.000,
    "loudness": -15.761, "tempo": 135.405, "tempo_confidence": 0.938,
    "key": 0, "key_confidence": 0.107, "mode": 1, "mode_confidence": 0.516,
    "time_signature": 4, "time_signature_confidence": 1.000}, ...],
  "segments":
  [{
    {
      "start": 0.00000, "duration": 0.31887, "confidence": 1.000,
      "loudness_start": -60.000, "loudness_max_time": 0.10242,
      "loudness_max": -16.511, "pitches": [0.370, 0.067, 0.055, 0.073, 0.108, 0.082,
      0.123, 0.180, 0.327, 1.000, 0.178, 0.234], "timbre": [24.736, 110.034, 57.822,
      -171.580, 92.572, 230.158, 48.856, 10.804, 1.371, 41.446, -66.896, 11.207]
    }, ...]
  }
}
```

Fig 2.2 Sample JSON Output of the API



## Interpretation



Plot of the JSON data for a 30-second excerpt of "around the world" by Daft Punk.

Fig 2.3 Plot of the JSON

### Rhythm

Beats are subdivisions of bars. Tatums are subdivisions of beats. That is, bars always align with a beat and ditto tatums. Note that a low confidence does not necessarily mean the value is inaccurate. Exceptionally, a confidence of -1 indicates "no" value: the corresponding element must be discarded. A track may result with no bar, no beat, and/or no tatum if no periodicity was detected. The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4. A value of -1 may indicate no time signature, while a value of 1 indicates a rather complex or changing time signature.

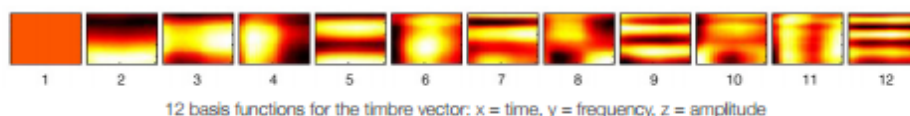
### Pitch

The key is a track-level attribute ranging from 0 to 11 and corresponding to one of the 12 keys: C, C#, D, etc. up to B. If no key was detected, the value is -1. The mode is equal to 0 or 1 for "minor" or "major" and may be -1 in case of no result. Note that the major key (e.g. C major) could more likely be confused with the minor key at 3 semitones lower (e.g. A minor) as both keys carry the same pitches. Harmonic details are given in segments below.

### Segments

Beyond timing information (start, duration), segments include *loudness*, *pitch*, and *timbre* features.

- *loudness* information (i.e. attack, decay) is given by three data points, including dB value at onset (*loudness\_start*), dB value at peak (*loudness\_max*), and segment-relative offset for the peak loudness (*loudness\_max\_time*). The dB value at onset is equivalent to the dB value at offset for the preceding segment. The last segment specifies a dB value at offset (*loudness\_end*) as well.
- *pitch* content is given by a "chroma" vector, corresponding to the 12 pitch classes C, C#, D to B, with values ranging from 0 to 1 that describe the relative dominance of every pitch in the chromatic scale. For example a C Major chord would likely be represented by large values of C, E and G (i.e. classes 0, 4, and 7). Vectors are normalized to 1 by their strongest dimension, therefore noisy sounds are likely represented by values that are all close to 1, while pure tones are described by one value at 1 (the pitch) and others near 0.
- *timbre* is the quality of a musical note or sound that distinguishes different types of musical instruments, or voices. It is a complex notion also referred to as sound color, texture, or tone quality, and is derived from the shape of a segment's spectro-temporal surface, independently of pitch and loudness. The Echo Nest Analyzer's *timbre* feature is a vector that includes 12 unbounded values roughly centered around 0. Those values are high level abstractions of the spectral surface, ordered by degree of importance. For completeness however, the first dimension represents the average loudness of the segment; second emphasizes brightness; third is more closely correlated to the flatness of a sound; fourth to sounds with a stronger attack; etc. See an image below representing the 12 basis functions (i.e. template segments). The actual timbre of the segment is best described as a linear combination of these 12 basis functions weighted by the coefficient values:  $\text{timbre} = c_1 \times b_1 + c_2 \times b_2 + \dots + c_{12} \times b_{12}$ , where  $c_1$  to  $c_{12}$  represent the 12 coefficients and  $b_1$  to  $b_{12}$  the 12 basis functions as displayed below. Timbre vectors are best used in comparison with each other.



### Confidence Values

Many elements at the track and lower levels of analysis include confidence values, a floating-point number ranging from 0.0 to 1.0. Confidence indicates the reliability of its corresponding attribute. Elements carrying a small confidence value should be considered speculative. There may not be sufficient data in the audio to compute the element with high certainty.

Fig 2.4. EchoNest features

## 2.8. PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components. The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric.

PCA is mathematically defined[3] as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Consider a data matrix,  $X$ , with column-wise zero empirical mean (the sample mean of each column has been shifted to zero), where each of the  $n$  rows represents a different repetition of the experiment, and each of the  $p$  columns gives a particular kind of datum (say, the results from a particular sensor).

Mathematically, the transformation is defined by a set of  $p$ -dimensional vectors of weights or loadings  $\mathbf{W}_{(k)} = (w_1, \dots, w_p)_{(k)}$  that map each row vector  $\mathbf{X}_{(i)}$  of  $\mathbf{X}$  to a new vector of principal component scores  $\mathbf{t}_{(i)} = (t_1, \dots, t_p)_{(i)}$ , given by

$$t_{k(i)} = \mathbf{X}_{(i)} \cdot \mathbf{W}_{(k)}$$

in such a way that the individual variables of  $\mathbf{t}$  considered over the data set successively inherit the maximum possible variance from  $\mathbf{x}$ , with each loading vector  $\mathbf{w}$  constrained to be a unit vector.

## 2.9. Parrot's classification of Emotions chart

The contrasting and categorisation of emotions describes how emotions are thought to relate to each other. Various recent proposals of such groupings are described in the following sections.

W. Gerrod Parrott devised a 2001 theory consisting of a tree type structured list leading to the classification of deeper emotions. This categorisation list includes over 100 emotions – classified as shown in the chart below:

Primary emotion	Secondary emotion	Tertiary emotions
Love	Affection	Adoration, affection, love, fondness, liking, attraction, caring, tenderness, compassion, sentimentality
	Lust	Arousal, desire, lust, passion, infatuation
	Longing	Longing
Joy	Cheerfulness	Amusement, bliss, cheerfulness, gaiety, glee, jolliness, joviality, joy, delight, enjoyment, gladness, happiness, jubilation, elation, satisfaction, ecstasy, euphoria
	Zest	Enthusiasm, zeal, zest, excitement, thrill, exhilaration
	Contentment	Contentment, pleasure
	Pride	Pride, triumph
	Optimism	Eagerness, hope, optimism
	Enthrallment	Enthrallment, rapture
	Relief	Relief
Surprise	Surprise	Amazement, surprise, astonishment
Anger	Irritation	Aggravation, irritation, agitation, annoyance, grouchingness, grumpiness
	Exasperation	Exasperation, frustration
	Rage	Anger, rage, outrage, fury, wrath, hostility, ferocity, bitterness, hate, loathing, scorn, spite, vengefulness, dislike, resentment
	Disgust	Disgust, revulsion, contempt
	Envy	Envy, jealousy
	Torment	Torment
Sadness	Suffering	Agony, suffering, hurt, anguish
	Sadness	Depression, despair, hopelessness, gloom, glumness, sadness, unhappiness, grief, sorrow, woe, misery, melancholy
	Disappointment	Dismay, disappointment, displeasure
	Shame	Guilt, shame, regret, remorse
	Neglect	Alienation, isolation, neglect, loneliness, rejection, homesickness, defeat, dejection, insecurity, embarrassment, humiliation, insult
	Sympathy	Pity, sympathy
Fear	Horror	Alarm, shock, fear, fright, horror, terror, panic, hysteria, mortification
	Nervousness	Anxiety, nervousness, tenseness, uneasiness, apprehension, worry, distress, dread

Fig 2.5 Parrott's Classification

Here Parrott classified emotions into 6 primary emotions – Love, Joy, Surprise, Anger, Sadness, Fear.



## 2.10 UMBC EBIQUITY-CORE: Semantic Textual Similarity Systems API

There is a simple Web API that can be used to query the UMBC Semantic Similarity service. Given two words or phrases it returns a string representing a number between 0.0 and 1.0. Call it using a URL like the following

```
http://swoogle.umbc.edu/SimService/GetSimilarity?operation=api&phrase1=XXX&phrase2=YYY
```

Where XXX and YYY are the urlencoded words or short noun or verb phrases one wants to compare. For example, to compare the words car and bike with the URL -

```
http://swoogle.umbc.edu/SimService/GetSimilarity?operation=api&phrase1=car&phrase2=bike
```

This returns the following number: 0.38658366.

We used this number as a measure of similarity between two words.

This service is a consequence of a research done by University of Maryland and Baltimore County. In this research, the authors developed semantic text similarity system that combined LSA (latent semantic analysis) word similarity and WordNet knowledge.

In order to produce a reliable word co-occurrence statistics, a very large and balanced text corpus is required. After experimenting with several corpus choices including Wikipedia, Project Gutenberg eBooks (Hart, 1997), ukWaC (Baroni et al., 2009), Reuters News stories (Rose et al., 2002) and LDC gigawords, the Web corpus from the Stanford WebBase project (Stanford, 2001) was selected. The February 2007 crawl was used, which is one of the largest collections and contains 100 million web pages from more than 50,000 websites. The WebBase project did an excellent job in extracting textual content from HTML tags but still has abundant text duplications, truncated text, non-English text and strange characters. The collection was processed to remove undesired sections and produce high quality English paragraphs. Paragraphs were detected using heuristic rules and only those were retained whose length was at least two hundred characters. Non-English texts were eliminated by checking the first twenty words of a paragraph to see if they were valid English words. The percentage of punctuation characters in a paragraph was used as a simple check for typical text. Finally, a three billion words corpus of good quality English was obtained, which is available at (Han and Finin, 2013).

POS tagging and lemmatization on the WebBase corpus was performed using the Stanford POS tagger (Toutanova et al., 2000). Word/term co-occurrences are counted in a moving window of a fixed size that scans the entire corpus<sup>1</sup>. A context window of  $\pm 4$  words allowed to compute semantic similarity between words with different POS. The word co-occurrence models were based on a predefined vocabulary of more than 22,000 common English words and noun phrases. More than 2,000 verb phrases extracted from WordNet were also added. The final dimensions of our word co-occurrence matrices are  $29,000 \times 29,000$



when words are POS tagged. The vocabulary includes only open-class words (i.e. nouns, verbs, adjectives and adverbs). There are no proper nouns in the vocabulary with the only exception of country names. Stop-word list consisting of only the three articles “a”, “an” and “the” was used.

SVD Transformation Singular Value Decomposition (SVD) has been found to be effective in improving word similarity measures (Landauer and Dumais, 1997). SVD is typically applied to a word by document matrix, yielding the familiar LSA technique. In this case we apply it to our word by word matrix. In literature, this variation of LSA is sometimes called HAL (Hyperspace Analog to Language) (Burgess et al., 1998). Before performing SVD, the raw word co-occurrence count  $f_{ij}$  was transformed to its log frequency  $\log(f_{ij} + 1)$ . The 300 largest singular values were used and reduced the 29K word vectors to 300 dimensions. The LSA similarity between two words is defined as the cosine similarity of their corresponding word vectors after the SVD transformation.

Combining with WordNet Knowledge Statistical word similarity measures have limitations. Related words can have similarity scores as high as what similar words get. Word similarity is typically low for synonyms having many word senses since information about different senses are mashed together (Han et al., 2013). By using WordNet, we can reduce the above issues.

Boosting LSA similarity using WordNet, we increase the similarity between two words if any of the following relations hold.

- They are in the same WordNet synset.
- One word is the direct hypernym of the other.

- One word is the two-link indirect hypernym of the other.
- One adjective has a direct similar to relation with the other.
- One adjective has a two-link indirect similar to relation with the other.
- One word is a derivationally related form of the other.
- One word is the head of the gloss of the other or its direct hypernym or one of its direct hyponyms.
- One word appears frequently in the glosses of the other and its direct hypernym and its direct hyponyms.

We use the algorithm described in (Collins, 1999) to find a word gloss header.

We require a minimum LSA similarity of 0.1 between the two words to filter out noisy data when extracting WordNet relations. We define a word's "significant senses" to deal with the problem of WordNet trivial senses. The word "year", for example, has a sense "a body of students who graduate together" which makes it a synonym of the word "class". This causes problems because "year" and "class" are not similar, in general.

A sense is significant, if any of the following conditions are met:

- (i) it is the first sense;
- (ii) its WordNet frequency count is not less than five; or
- (iii) its word form appears first in its synset's word form list and it has a WordNet sense number less than eight.

We assign path distance of zero to the category 1, path distance of one to the category 2, 4 and 6, and path distance of two to the other categories. The new similarity between word  $x$  and  $y$  by combining LSA similarity and WordNet relations is shown in the following equation

$$\text{sim}\oplus(x, y) = \text{simLSA}(x, y) + 0.5e^{-\alpha D(x,y)} \quad (1)$$

where  $D(x, y)$  is the minimal path distance between  $x$  and  $y$ . Using the  $e^{-\alpha D(x,y)}$  to transform simple shortest path length has been demonstrated to be very effective according to (Li et al., 2003). The parameter  $\alpha$  is set to be 0.25, following their experimental results. The ceiling of  $\text{sim}\oplus(x, y)$  remains 1.0 and we simply cut the excess.

**Dealing with words of many senses** For a word  $w$  with many WordNet senses (currently ten or more), we use its synonyms with fewer senses (at most one third of that of  $w$ ) as its substitutions in computing similarity with another word. Let  $S_x$  and  $S_y$  be the sets of all such substitutions of the words  $x$  and  $y$  respectively. The new similarity is obtained using Equation 2.

$$\text{sim}(x, y) = \max(\max_{s_x \in S_x \cup \{x\}} \text{sim}\oplus(s_x, y), \max_{s_y \in S_y \cup \{y\}} \text{sim}\oplus(x, s_y)) \quad (2)$$

An online demonstration of a similar model developed for the GOR project is available (UMBC, 2013b), but it lacks some of this version's features.

It is this online version that we have used.

# **Chapter 3: Architecture &**

## **Methodology**

Irrespective of the process model employed, software development comprises several tasks. Requirements acquisitions, conceptual modelling, risk analysis, database design, coding, testing and software maintenance are some of the tasks involved. Further, each task entails a specific set of skills for its accomplishment. For example, the ability to prepare test suites is important for testing while coding requires a good practice of programming skills.

The strength of a team is derived from the skills its members possess. According to the members' skills, each team can accomplish different in detail about the objectives and motivations behind the conceptualizations in the previous sections.

In this section, we are going to discuss about the modular high level architecture describing various modules and interaction among these modules.

Architecture of our development cycle can be mainly divided into three stages namely:-

1. Dataset Collection

2. Applying the dataset to classifiers

- 2.1. PCA (Principle Component Analysis)

- 2.2. Classifier: Linear regression

- 2.3. Classifier: Logistic regression

## 2.4. Classifier: Support Vector Machines

3. Comparing the results of various classifiers and determining the percentage of error.

## 3.1. Dataset Collection

This phase is one of the most important phases of the project development cycle as it is only dataset which defines the level of accuracy and stability that will be achieved and reflected in results. Like every other project this one also starts with collection of database which comprises of various songs name along with their ranks based on yearly basis.

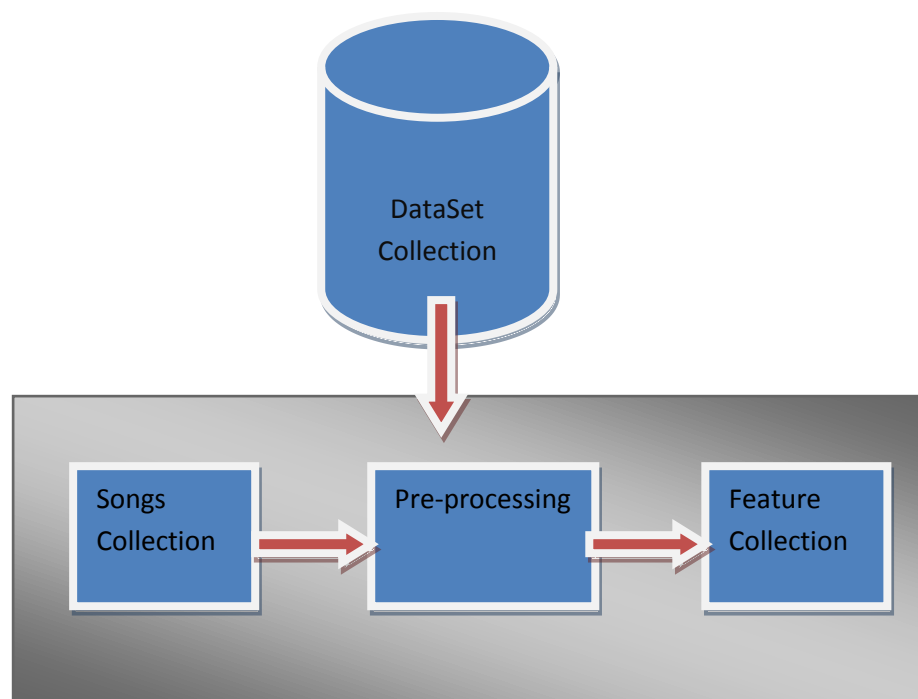


Fig 3.1 Composition of phase 'Data Collection'

### 3.1.1. Songs Collection

Songs are collected from [billboards.com](http://billboards.com). Billboard Year-End charts are a cumulative measure of a single or album's performance in the United States. Year-end charts were calculated by an inverse-point system based solely on a title's performance (for example a single appearing on the Billboard Hot 100 would be given one point for a week spent at position 100, two points for a week spent at position ninety-nine, and so forth, up to 100 points for each week spent at number one). Other factors including the total weeks a song spent on the chart and at its peak position were calculated into its year-end total.

We scrapped the data/song-rank list from this site: <http://www.bobborst.com/popculture/top-100-songs-of-the-year/?year=1964>.

This site basically compiles the results from Billboard in a more presentable format.

Previous Year < Year of 2014

PLAY THE CHART











1		<b>HAPPY</b> Pharrell Williams	WATCH 
2		<b>DARK HORSE</b> Katy Perry Featuring Juicy J	
3		<b>ALL OF ME</b> John Legend	 WATCH 
4		<b>FANCY</b> Iggy Azalea Featuring Charli XCX	
5		<b>COUNTING STARS</b>	

Fig. 3.2 a sample of Billboard's Website

In Fig. 3.2, A sample of Billboard's website is shown containing the list of top 5 songs of year 2014. From this website, using the script in python, we scrape the content of the HTML page using python's 'pattern' library to generate the list of top 100 songs of year 1947-2013 along with their ranks.

1	Francis Craig	Near You
2	Harmonicats	Peg O' My Heart
3	Ted Weems	Heartaches
4	Ray Noble and Buddy Clark	Linda
5	Tex Williams	Smoke, Smoke, Smoke That Cigarette
6	Vaughn Monroe	I Wish I Didn't Love You So
7	Three Suns	Peg O' My Heart
8	Al Jolson	Anniversary Song
9	Larry Green	Near You
10	Sammy Kaye	That's My Desire
11	Vaughn Monroe	Ballerina
12	Art Lund	Mam'selle
13	Freddy Martin	Managua, Nicaragua
14	Perry Como	Chi-Baba Chi-Baba
15	Red Ingle and Jo Stafford	Temptation (Tim-tayshun)
16	Ted Weems and Perry Como	I Wonder Who's Kissing Her Now

Fig. 3.3 A sample of data collected through parsing for year 1947.

Fig. 3.3 shows the sample data parsed from the HTML page. The sample data consists of the list of top 16 songs of year 1947, along with their artist names.

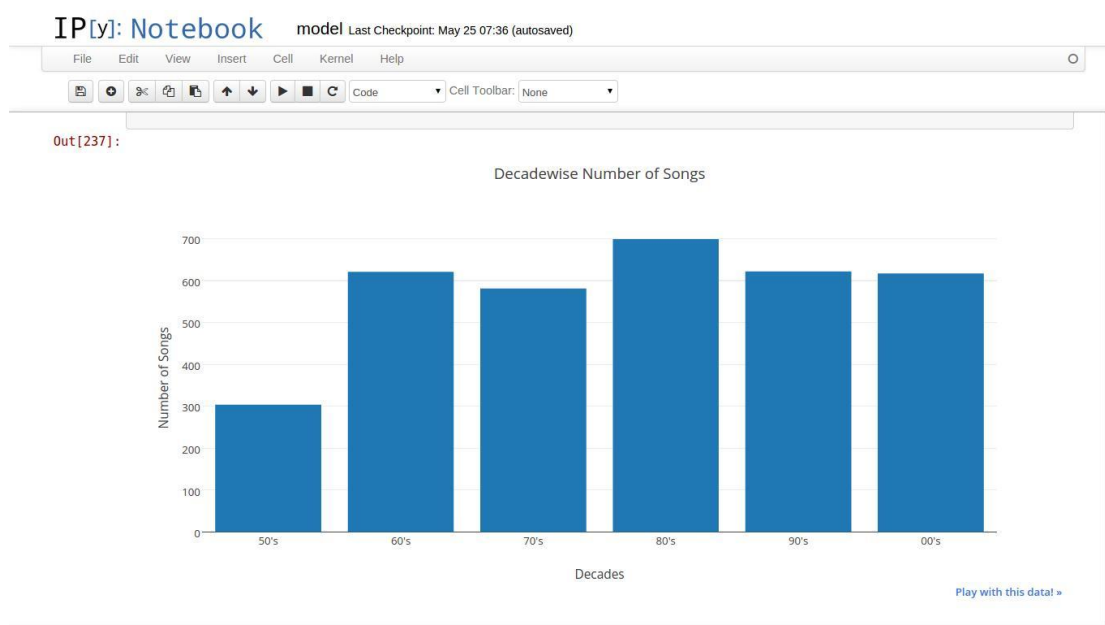


Fig 3.4 A brief of the number of songs obtained for each decade



### 3.1.2. Extracting audio features using Echonest API

Echonest API provides a comprehensive API which lets the user explore the audio features of a song. Using the API, we gathered data about each of the song in our list. The data that we used were:

1. Acousticness
2. Danceability
3. Duration
4. Energy
5. Key
6. Liveness
7. Loudness
8. Mode
9. Speechiness
10. Time-signature
11. Valence

rank	song_name	artist	latitude	longitude	acousticness	danceability	duration	energy	key	liveness	loudness	mode	speechiness	tempo	time	valence
1	Hey Jude	Beatles	53.4098	-2.97848	0.275356	0.464355	300.06907	0.462577	5	0.106562	-10.116	1	0.024347	148.13	4	0.527583
2	Love Is Blue	Paul Mauriat	43.3	5.4	0.120112	0.380633	294.33288	0.526747	2	0.789219	-15.552	1	0.048691	111.93	4	0.382058
3	Honey	Bobby Goldsboro	30.7754	-85.2268	0.103549	0.473622	229.53333	0.176127	7	0.096748	-18.802	1	0.028081	96.161	4	0.447784
4	(Sittin' On) The D	Otis Redding	32.804382	-83.617554	0.684492	0.768707	163.75578	0.367428	2	0.080979	-11.226	1	0.031158	103.62	4	0.541401
5	People Got To Be	Rascals	40.7146	-74.0071	0.412402	0.666801	179.70621	0.559847	10	0.144432	-11.73	1	0.03501	126.34	4	0.962275

Fig. 3.5 List of songs along with its acoustic features

Fig. 3.5 explains about the list of the songs along with its acoustic features extracted using the EchoNest API.

A brief of the features are described here :

[http://developer.echonest.com/raw\\_tutorials/faqs/faq\\_04.html](http://developer.echonest.com/raw_tutorials/faqs/faq_04.html)

We searched the songs in the Echonest's database using pyechonest (a library in python for using echonest API)

Ex -

```
>>>from pyechonest import song
```

```
>>>rkp_results = song.search(artist='The Beatles', title='Let it Be')
```

This returns the results of the query to the variable rkp\_results and we access the required fields by –

```
rkp_results[0].audio_summary['acousticness'],
rkp_results[0].audio_summary['danceability'],
rkp_results[0].audio_summary['duration'],
rkp_results[0].audio_summary['energy'],    rkp_results[0].audio_summary['key'],
rkp_results[0].audio_summary['liveness'],
rkp_results[0].audio_summary['loudness'],
rkp_results[0].audio_summary['mode'],
rkp_results[0].audio_summary['speechiness'],
rkp_results[0].audio_summary['tempo'],
rkp_results[0].audio_summary['time_signature'],
rkp_results[0].audio_summary['valence']]
```

Echonest has a large database of songs and has already stored these precalculated features on their servers, but there are always some caveats. We were able to extract the features of most of the songs, but some of them could not be found.

### 3.1.3. Extracting Lyrics using shell script

We used the python package ‘songtext-0.1.3’ and its LyricsWiki API to get the lyrics of the songs in a rankwise fashion. It was required to export the LYRICSNMUSIC\_API\_KEY environment variable to the shell startup file for the API to work. We could then query from the command line to get an output like this:

```
$ songtext --api lyricsnmusic london grammar nightcall  
33 track(s) matched your search query.  
London Grammar: Nightcall  
-----  
I'm giving you a nightcall  
To tell you how I feel  
I'm gonna drive you through the night  
Down the hills  
I'm gonna tell you something  
You don't want to hear  
I'm gonna show you where it's dumped  
But have no fear
```

We then wrote a shell script to direct the command line output to a file and ran a loop over all the songs of all the years to get the lyrics of each of the songs. Some songs weren't identified correctly by the API and for some the lyrics couldn't be found, but overall, we found the lyrics of most of the songs in our list.

### 3.1.4. Preprocessing

After gather the basic data, that is the song list along with their ranks (yearwise) and the lyrics of these songs, we decided to take a new approach towards understanding the emotional quotient of a song using Parrott's emotion – classification. But first this required us to calculate the TD-IDF of the words in the song.

#### 3.1.4.1. TF-IDF

TF-IDF is a short for 'Term Frequency – Inverse Document Frequency'.

This technique is used to quantify the importance of a word in a document (in our case, lyrics).

**tf-idf**, short for **term frequency-inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a [document](#) in a collection or [corpus](#).<sup>[1]:8</sup> It is often used as a weighting factor in [information retrieval](#) and [text mining](#). The tf-idf value increases [proportionally](#) to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

TF is calculated by counting the number of times a word occurs (raw frequency) in a document and normalising it with the number of words in that document.

$$\text{tf}(t, d) = \frac{f(t, d)}{\max\{f(w, d) : w \in d\}}$$

IDF is calculated using the inverse of the log of the number of documents in which that particular word occurs.

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Finally, TD-IDF is calculated as the product of these two terms:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

So, the TF-IDF of each of the words in each of the songs of a year were calculated, which denoted the importance of that word in that lyrics (in that song that particular year).

We then used the TF-IDF to rank the most important words in a lyrics of a song and used the top 20 words of that lyrics in our next analysis.

### 3.1.4.2. Calculating the emotion vector of songs

For this technique, we used an API provided by a research called the UMBC Top-N Similarity Service.

Computing semantic similarity between words/phrases has important applications in natural language processing, information retrieval, and artificial intelligence. There are two prevailing approaches to computing word similarity, based on either using of a thesaurus (e.g., WordNet ) or statistics from a large corpus. UMBC provides a hybrid approach combining the two methods.

The statistical method is based on distributional similarity and Latent Semantic Analysis (LSA). It is further complemented with semantic relations extracted from WordNet. The whole process is automatic and can be trained using different corpora. It is assumed that the semantics of a phrase is compositional on its component words and an algorithm is applied to compute similarity between two phrases using word similarity.

Using this service, we calculated how similar a word is to the 6 basic/primary emotions as classified by Parrott, namely – Love, Joy, Fear, Surprise, Sadness, Anger.

We decided to denote every song with a 6 float vector to denote the degree of emotions and to quantify it in some manner. The 6 elements of the vector correspond to the 6 emotions. We believe that a song might have degrees of certain emotions and should not be classified as a single valued emotion.

So, we iterate over each song in the list of a top 100 in a year, and for every year, we pick up the top 20 words (ranked using TF-IDF) and we find out its similarity with each of the primary emotions. Then we take the max similarity between the word and the 6 emotions and multiply it with the word's TF-IDF and add this value to the particular emotion's index in the song's emotion vector.

Ex – Suppose a song has the word 'lover' and the emotion vector of the song be  $[0,0,0,0,0,0]$  (0-for each of the emotions love, joy, sadness, anger, surprise, fear)

And 'lover' gives the following degrees of similarities with the primary emotions:

Love-0.7506343

Joy-0.17573524

Sadness-0.040953867

Anger-0.013633692

Surprise-0.06152932

Fear-0.09170411

So, here we see that 'lover' gives maximum similarity with the emotion Love, so we multiply 0.7506343 with 'lover' 's TF-IDF (say 0.1) and add it to the emotion vector.

So, the emotion vector becomes [0.07506343,0,0,0,0,0]

This way, we calculate the emotion vector for each of the songs

At the end, in as our master data, we collected and computed the following 17 characteristics of each song:

'song\_rank','acousticness','danceability','duration','energy','key','liveness','loudness','mode','speechiness','time\_signature','valence','love','joy','surprise','anger','sadness','fear'.

Example:

rank	song_name	artist	latitude	longitude	acousticness	danceability	duration	energy	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	love	joy	surprise	anger	sadness	fear
1	Tonight's	Rod Stevie	51.5073	-0.12783	0.05916	0.56357	215.967	0.2867	11	0.12853	-23.099	1	0.05305	135.384	3	0.6141	0.02807	0	0.0155	0.00511	0.00266	0.0378
2	Just War	Andy Gibb	-27.2333	153.117	0.06469	0.60427	225.093	0.75812	9	0.15393	-4.89	1	0.03609	96.804	4	0.88711	0.08406	0.02927	0.02991	0.00434	0.00398	0.0279
3	Best Of M	Emotions	41.85	-87.6501	0.09974	0.83858	102.243	0.67053	7	0.0479	-8.739	1	0.03472	114.452	4	0.98167	0	0	0	0	0	0
4	Love Them	Barbra Str	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Angel in N	Hot	0.09294	0.00706	0.07484	0.01914	0	0.05668	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	Like Dree	Kenny Nol	37.1679	-95.845	0.42956	0.39678	212.067	0.50539	6	0.0666	-11.49	1	0.03435	136.864	4	0.48911	0.04171	0.02814	0.00091	0.00655	0.00637	0.0073
7	Don't Less	Theima Hc	33.4041	-90.8988	0.23335	0.66369	247.84	0.49041	0	0.31229	-11.24	0	0.03619	120.044	4	0.33922	0.17671	0.03642	0.02396	0.00394	0	0.0515
8	Your Love	Rita Cooli	36.5277	-86.0256	0.04385	0.4091	209.84	0.77149	1	0.81942	-8.426	1	0.06136	137.895	4	0.74308	0.18367	0.00354	0.04854	0.0341	0.06612	0
9	Undercovi	Alan O'De	34.0535	-118.245	0.31746	0.671	217.44	0.52915	5	0.0757	-9.976	1	0.03682	103.143	4	0.79821	0.17909	0.00876	0.02047	0	0	0.0544
10	Torn Betw	Mary Mac	43.8438	-82.6514	0.83865	0.55723	230.623	0.26182	10	0.09483	-17.264	1	0.0405	133.657	4	0.35698	0.19611	0	0.00243	0	0.03646	0.0076
11	I'm Your	B K.C. and T	0.14996	0.016	0.02784	0.00547	0.00584	0.06402	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	Dancing C	ABBA	1.3582	0.06837	0.03293	0.04016	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	You Make	Leo Sayer	50.8322	-0.27467	0.02216	0.53652	256.193	0.90062	8	0.81308	-5.356	0	0.153	102.555	4	0.63515	0.02219	0.1017	0.0032	0.00744	0	0.0555
14	Margrita	Jimmy Buf	30.3653	-88.5564	0.5456	0.57227	274.32	0.93194	2	0.79084	-4.479	1	0.04537	125.715	4	0.86817	0.04989	0	0.01169	0.00238	0	0.0738
15	Telephone	Electric Li	52.4814	-1.89807	0.73651	0.37316	279.146	0.43775	9	0.11264	-6.775	1	0.02955	72.17	4	0.15637	0.03206	0.00727	0.02148	0.00346	0.03431	0.0072
16	Whatcha	Pablo Cru	37.7796	-122.42	0.02768	0.72805	200.829	0.38419	0	0.12297	-18.022	1	0.03648	110.073	4	0.96038	0.08498	0.00545	0.01106	0.00265	0.00368	0.0430
17	Do You W	Peter McC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	Sir Duke	Stevie Wo	42.347	-83.0602	0.15575	0.61484	232.533	0.54064	6	0.12941	-8.893	1	0.10676	106.863	4	0.96051	0.02173	0.0159	0.00489	0	0.00137	0.1860
19	Hotel Cali	Eagles	34.0535	-118.245	0.00665	0.55539	372.307	0.50469	2	0.14244	-10.541	1	0.02664	147.594	4	0.59321	0	0	0	0	0	0
20	Got To Gi	Marvin Ge	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	Theme Fro	Bill Conti	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	Southern	Glen Camy	34.0314	-93.5028	0.00885	0.64036	177.987	0.82579	11	0.08031	-8.19	0	0.03109	95.243	4	0.80676	0.07941	0.09586	0.01827	0	0.02461	0.0784
23	Rich Girl	Darryl Hall	0.15317	0	0.01172	0.00343	0	0.09175	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	When I Ne	Leo Sayer	50.8322	-0.27467	0.7273	0.432	260.84	0.52053	11	0.91906	-7.019	1	0.04181	113.334	3	0.34322	0.07193	0.00348	0.00347	0.00263	0.02018	0.0119
25	Hot Line	Sylvers	33.9395	-118.243	0.77908	0.84146	181.733	0.63572	4	0.19643	-17.434	0	0.04097	132.948	4	0.97445	0	0	0	0	0	0
26	Car Wash	Rose Royc	34.0535	-118.245	0.22022	0.78154	310.08	0.62237	9	0.04731	-6.702	0	0.06218	114.372	4	0.90599	0.03465	0.01921	0.02483	0.00387	0.00111	0.0497
27	You Don't	Marilyn N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig 3.6.Screenshot of the master data of year 1977



## 3.2. Applying Classifiers and Analysis

We have a dataset of over 4000 entries and an attribute vector of 17 features.

First we attempt to minimise the number of features using PCA

### 3.2.1. PCA (Principle Component Analysis)

PCA is mathematically defined[3] as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Consider a data matrix,  $X$ , with column-wise zero empirical mean (the sample mean of each column has been shifted to zero), where each of the  $n$  rows represents a different repetition of the experiment, and each of the  $p$  columns gives a particular kind of datum (say, the results from a particular sensor).

Mathematically, the transformation is defined by a set of  $p$ -dimensional vectors of weights or loadings  $\mathbf{w}_{(k)} = (w_1, \dots, w_p)_{(k)}$  that map each row vector  $\mathbf{x}_{(i)}$  of  $X$  to a new vector of principal component scores  $\mathbf{t}_{(i)} = (t_1, \dots, t_p)_{(i)}$ , given by

$$t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(k)}$$

in such a way that the individual variables of  $\mathbf{t}$  considered over the data set successively inherit the maximum possible variance from  $\mathbf{x}$ , with each loading vector  $\mathbf{w}$  constrained to be a unit vector.

We used the powerful python package tool called Scikit to help us understand the feasibility of reduction in the number of attributes and its implications.

The package helps in Linear dimensionality reduction using Singular Value Decomposition of the data and keeping only the most significant singular vectors to project the data to a lower dimensional space.

This implementation uses the `scipy.linalg` implementation of the singular value decomposition. It only works for dense arrays and is not scalable to large dimensional data.

The time complexity of this implementation is  $O(n^3)$  assuming  $n \sim n_{\text{samples}} \sim n_{\text{features}}$ .

An example of the code using `pca` in `scikit` is:

```
>>> import numpy as np

>>> from sklearn.decomposition import PCA

>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])

>>> pca = PCA(n_components=2)

>>> pca.fit(X)

PCA(copy=True, n_components=2, whiten=False)

>>> print(pca.explained_variance_ratio_)

[ 0.99244...  0.00755...]
```

We reduced our 17 vector to 2 dimensions for easier visualization in a 2D plane and later also checked out the reduction to 5 dimensions. But the results were not encouraging as the samples were getting distributed equally into each of the reduced dimensions thereby offering no new leads or easier visualisation or help in classification. So in our next and further classifications, we take the entire 17 feature vectors into account instead of the reduced features.

The graphs of the PCA analysis are attached in Chapter 4.

### 3.2.2. Classifier: Linear Regression

For all of the classifiers, we created 2 matrices X and Y where X is a 2D ( $n \times 17$ ) matrix with each row representing a song and its 17 features and Y ( $n \times 1$ ) where the row of a Y represents its rank.

Then we divide the entire dataset into training data set and test data set for evaluation. We take the size of the training set to be 80% and that of the test set to 20% of the master data.

We plan to divide the data into 2 classes:

1. All those that rank above 50 (Class 1)
2. All those that rank below 50 (Class 0)

Using linear regression we get the prediction for the test data and we take a threshold value of 0.5, i.e if the prediction of the value is  $<0.5$ , then we classify it as Class 1 otherwise Class 0.

We then compute the accuracy percentage of the test data by calculating it as follows:

Accuracy Percentage = (Number of correctly Classified songs/Total number of songs).

### 3.2.3. Classifier: Logistic Regression

Similarly we train the classifier on the training data set and calculate the accuracy of the system on the test set. Here the module returns the probability of the test sample lying in Class 1 or Class 0 and we categorise using the probability being greater than 0.5 or not.

### 3.2.3. Classifier: Support Vector Machines

Scikit package in Python offers a variety of options for implementing SVMs.

The SVMs require that the features be scaled for both the training and test data, i.e the mean of the features be 0 and the standard deviation be 1. So, we had to pre-process our data before feeding it to the SVM.

We used the default kernel that was the non-linear *Radial Basis Function* (RBF) kernel described by the equation:

- $\text{rbf: } \exp(-\gamma|x - x'|^2)$ .  $\gamma$  is specified by keyword `gamma`, must be greater than 0.

This kernel gave the maximum accuracy percentage.

When training an SVM with the *Radial Basis Function* (RBF) kernel, two parameters must be considered: C and gamma. The parameter C, common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. gamma defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected

We take the value of  $c = 1$  for our classification.

## Chapter 4: Experiments & Results

### 4.1. Analysis done by plotting the locations of all the Artists

We performed an exploratory analysis on the data considering the latitude and longitude features of all the songs. We use Google maps API and javascript to plot the artists location on a heatmap to identify the locations from where most of the hit songs originate.

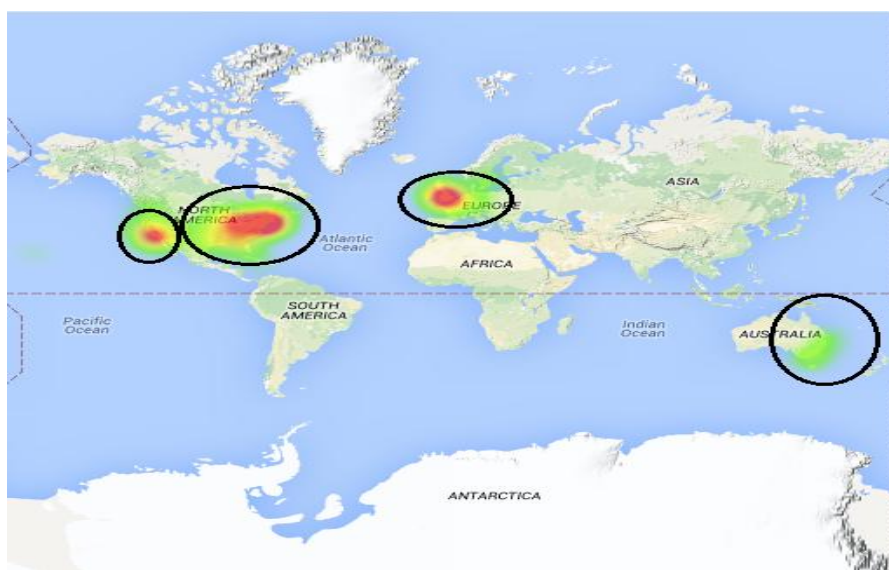


Fig. 4.1 Plot of Artists location

Circles in Fig. 4.1 shows that majority of artists belongs to areas like east and west coast of USA, United Kingdom and Eastern coast of Australia.

We realise that most of the music industry is based out of America.

## 4.2. Analysis done using K-means Clustering

We noticed from the EchoNest extracted features that songs of similar genres had similar attribute values. This led us to do some other exploratory analysis on the data set.

We use K-means clustering algorithm provided by the Scikit Package in python, to classify the songs into clusters on the basis of its acoustic features and analyze its results.

```
print data_param[(data_param['artist']=='Carpenters') & ()][['song_name','artist','cluster']]
```

	song_name	artist	cluster
1	(They Long To Be) Close To You	Carpenters	6
49	We've Only Just Begun	Carpenters	4
100	Superstar	Carpenters	6
104	For All We Know	Carpenters	6
107	Rainy Days And Mondays	Carpenters	6
212	Hurting Each Other	Carpenters	1
284	Sing	Carpenters	4
294	Yesterday Once More	Carpenters	1
356	Top Of The World	Carpenters	1
433	Please Mr. Postman	Carpenters	1
485	Only Yesterday	Carpenters	4

Fig. 4.2 Sample Result after clustering

In the above figure, it is seen that all the songs by 'Carpenters' can be classified only in three categories.

This confirmed that songs of the same genres have similar features.

Thus, the type of songs an artist record can now be easily found which can give us the reason of popularity of that artist among all other artists.

## 4.3. Detailed Analysis

Our detailed analysis can be broken down into-

1. PCA analysis
2. Emotion composition of hit songs in a year
3. Classification using Linear, Logistic and SVMs and their comparisons

### 4.3.1. PCA

We tried to reduce the number of features from 17 to something lesser for easier visualisation and use in the classifiers, but as the figure below represents, the dimensionality reduction didn't produce any encouraging results as it distributed the data points equally over the entire range.

Here we have reduced the number of components to 2 (X-axis is PCA component1 and Y-axis is PCA component2) and tried linear regression on it to find the best fit curve.

But as the figure suggests, the line lies almost horizontally with almost equal number of songs distributed all over the graph, yielding in no real or substantive change in classification.

It was this result and other iterations of the PCA (with number of components  $>2$  but  $<17$ ) that we decided to go ahead with the classification using all the 17 features.



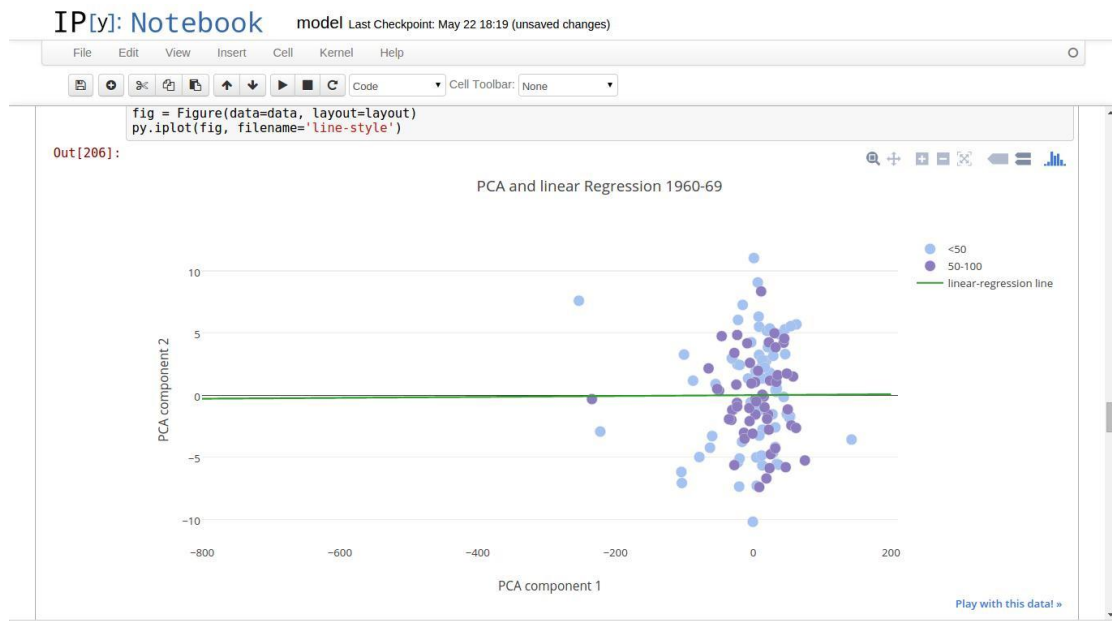


Fig 4.4 PCA Analysis

### 4.3.2. Summary of Classifier accuracy

Here we summarise the average accuracy of each of the classifiers. For comparison, we model each of the classifiers using only audio, only lyrics and using both features.

Models\Features	Audio Features Only	Lyrical Features Only	Combined (Audio + Lyrical Features)
Linear	0.513788098694	0.503628447025	0.510885341074
Logistic	0.515239477504	0.502177068215	0.502177068215
SVM	0.505079825835	0.518142235123	0.500725689405

Table 4.1 Accuracy Features

Here we see that most of the average accuracy of the models don't seem to provide any real distinction, i.e they aren't really able to classify properly.

This lead us to think that maybe popularity and the accuracy of our prediction also has temporal relations. Songs that were popular in the 70s might be of a different genre and about something different than songs in the 80s.

Our next analysis then focused on the temporal relationships (we took it decade-wise)

### 4.3.2. Decade–Wise Analysis

After recognising the temporal nature of predictions, we did a decade wise analysis of the data.

We fed each of the models decade –wise data and calculated the average of the accuracies of each of these decades. Below we have shown a graph which compares the average of the decade wise accuracy of each of the models.

The linear model doesn't show any improvement, but the logistic and SVM models show improvement when we include both the audio and lyrical features.

This supports our assumption of temporal dependencies and audio+lyrical features in prediction of our models.

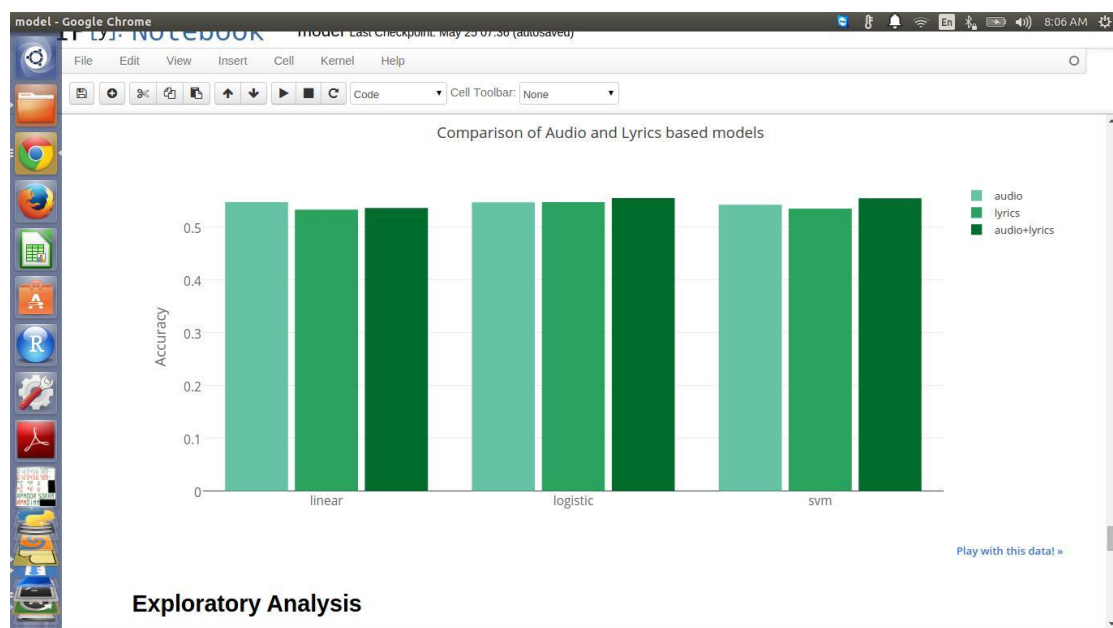


Fig 4.5 Comparison of Audio & Lyrics Models

Next we show the complete decade wise graphs of each of the models with audio features only, lyrical features only and combined audio and lyrical features.

### 4.3.2.1. Audio Features only –

Here we take into account, the audio features only, for all three classifiers.

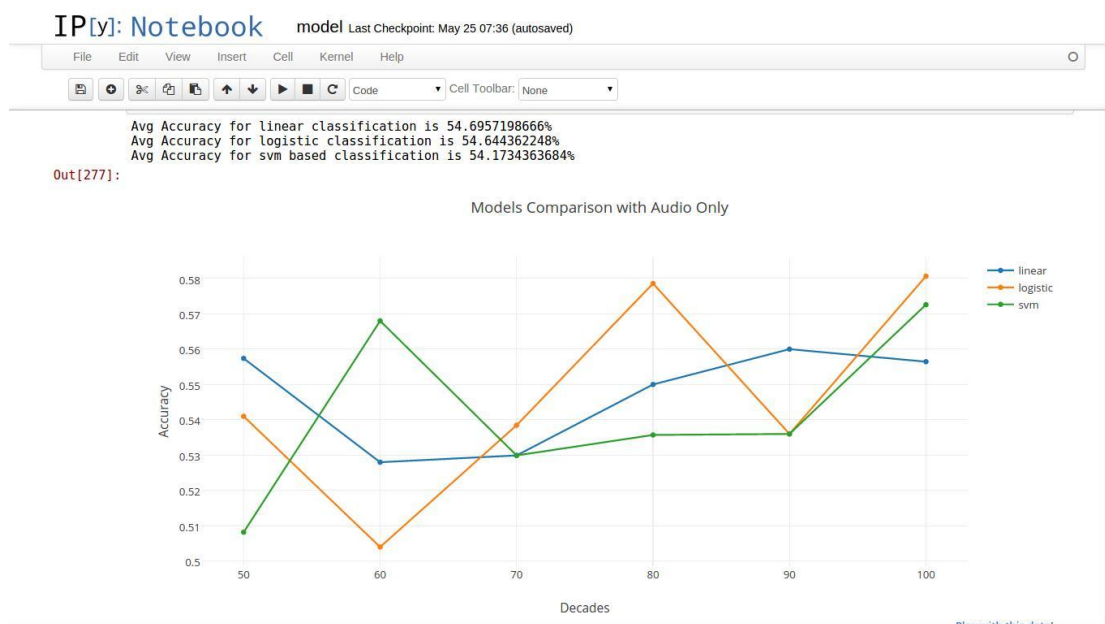


Fig 4.6 Only Audio Features

### 4.3.2.1. Lyrical Features only –

Here we take into account, the lyrical features only, for all three classifiers.

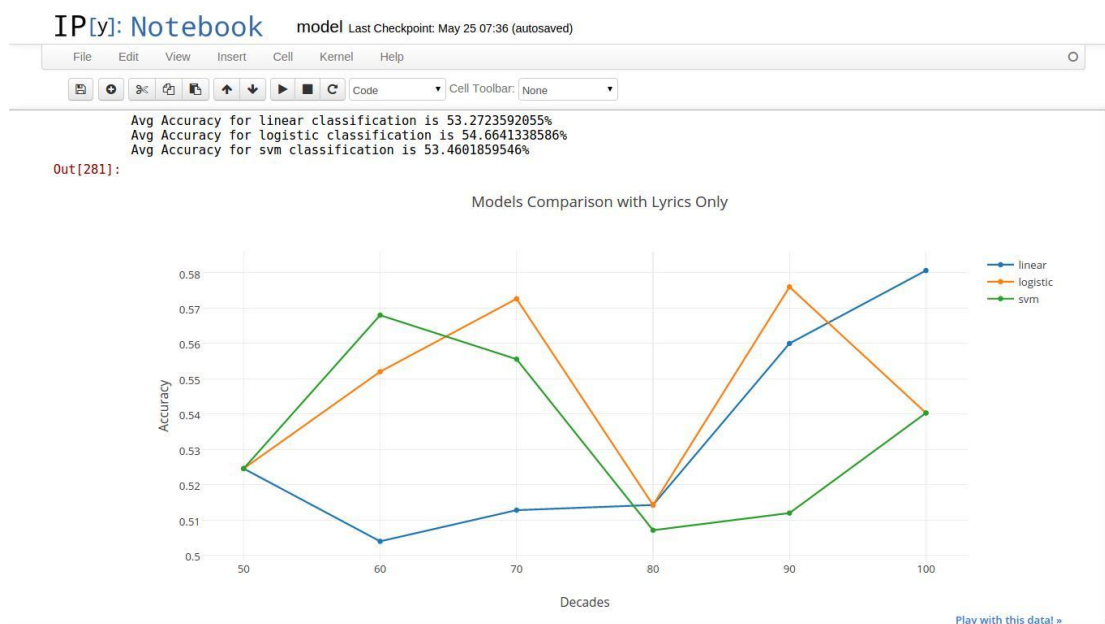


Fig 4.7 Only Lyrical Features

#### 4.3.2.1. Audio + Lyrical Features only –

Here we take into account, the audio and lyrical features both, for all three classifiers.

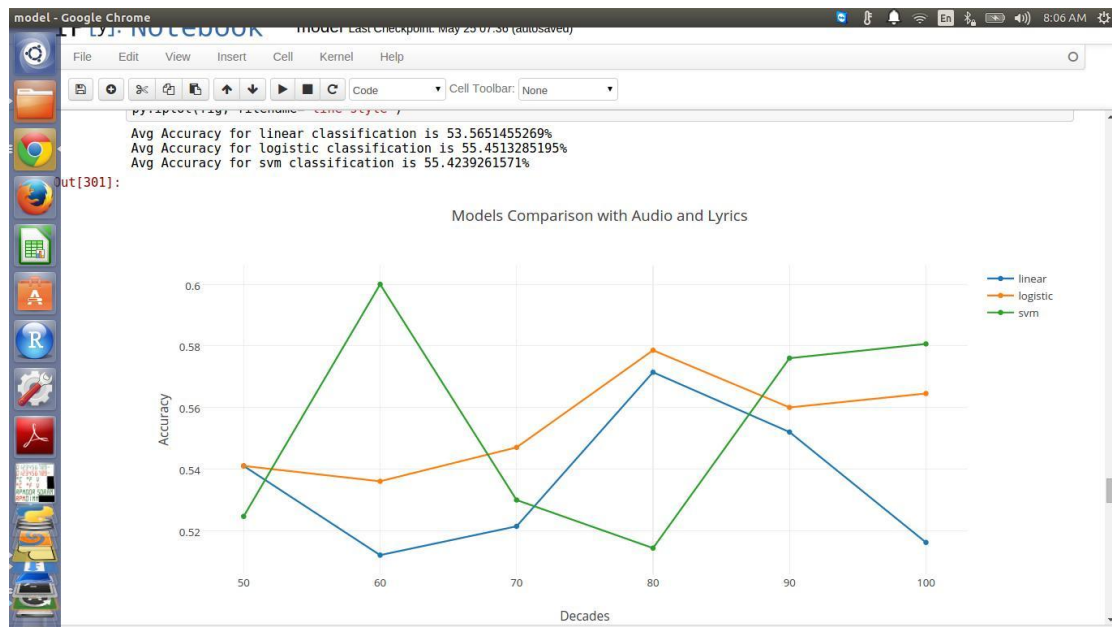


Fig 4.8 Audio & Lyrical Features

### 4.3.3. Emotional Analysis

We plotted the degrees of emotional quotient of each of the primary emotions, i.e

Love / Joy / Surprise / Anger / Sadness / Fear, in each of the decade.

As we expected, the most sung-about emotion is ‘Love’ followed by either ‘Surprise’ or ‘Fear’

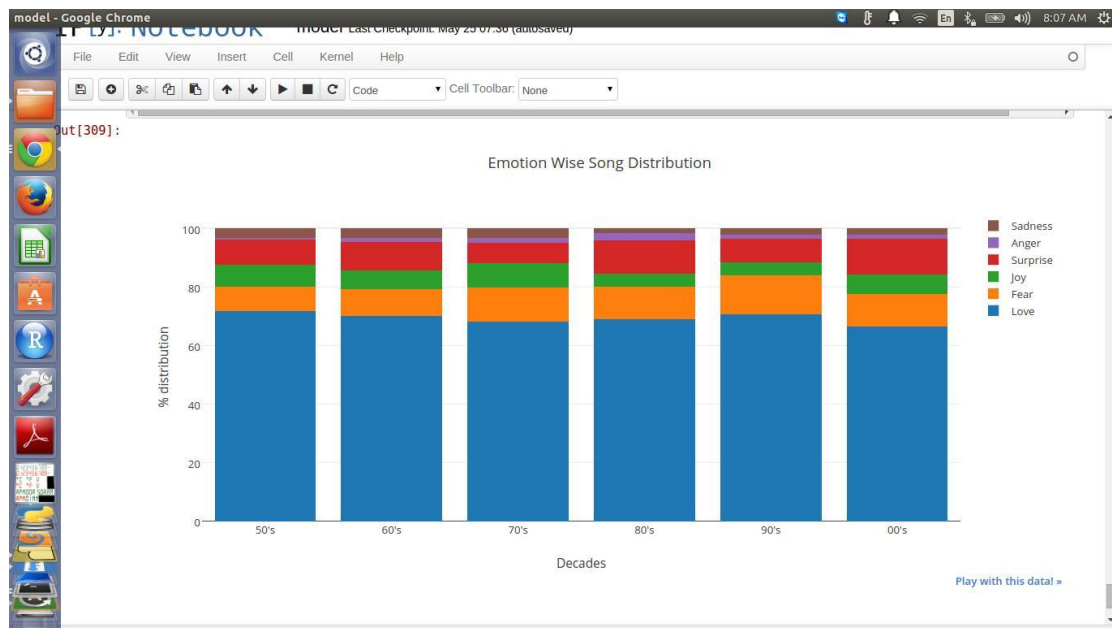


Fig 4.9 Composition of Emotions Decade-Wise

#### 4.3.4. Finding out which Feature contributes the most (Principality of a feature) using the weights of the Linear regression as a metric

We thought about finding out which of the features contribute the most to the song's popularity. The weights of the features of the trained Linear Regression model give us some clues. The feature with the maximum (mod value) will contribute the most towards the prediction.

According to this, we see that anger is the one emotion and feature that contributes the most to the popularity of a song

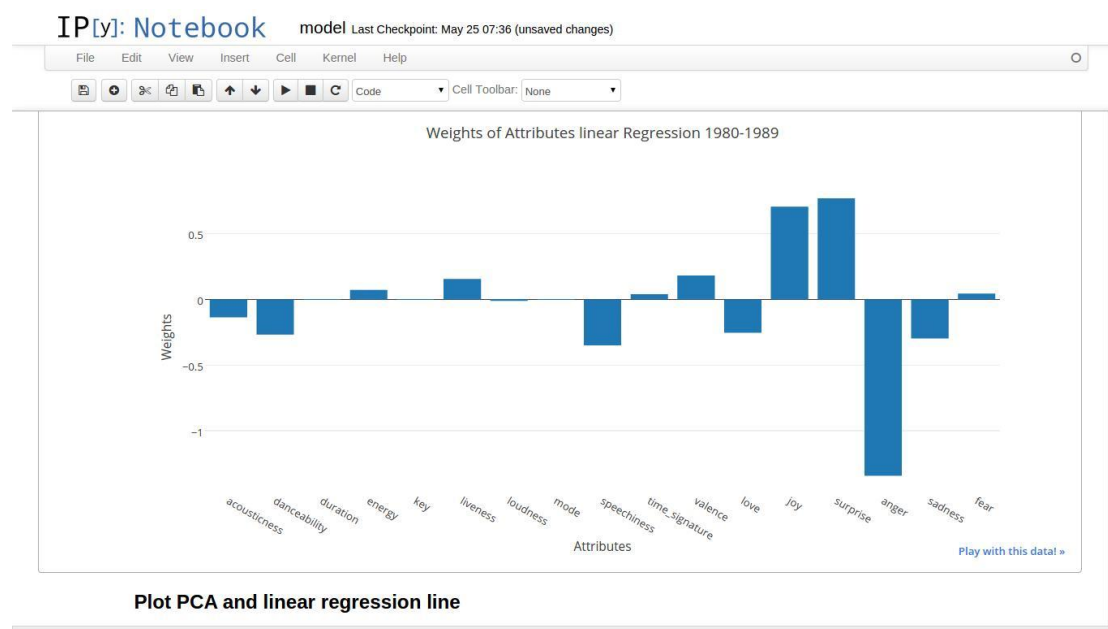


Fig 4.10 Checking the weights of the feature vectors

## **Chapter 5: Conclusion and Future**

### **Work**

In our experiments, we have done temporal analysis considering ‘decades’ of songs such as the 60’s which consist of years from 1960-69 and so on. Future work suggests several other temporal divisions can be experimented with.

For Parrot’s classification, we have considered only the primary level/division of emotions such as love, joy, etc. We found that most of the songs in the top-100 Billboards list can be mapped to the emotion love. It would be interesting to see how the songs would distribute into a classification considering secondary and tertiary levels of Parrot’s emotion classification.

More dimension such as genre can be added to the dataset to further improve the model. Both user defined genres such as those in last.fm as well as standard genre classifications such as pop, blues, etc can be used for this purpose.

We have only visually seen that the top-100 songs are also region specific, such as many of the top-100 songs are produced by artists from locations such as New York, etc. Location of the artist can be factored into the model to further improve the classification process.



## References

- [1] R. Dhanaraj and B. Logan. Automatic prediction of hit songs. In Proceedings of the International Conference on Music Information Retrieval, pages 488{91, 2005.
- [2] F. Pachet and P. Roy. Hit song science is not yet a science. In Proc. of the 9th International Conference on Music Information Retrieval (ISMIR 2008), pages 355{360, 2008.
- [3] Ni, Y., Santos-Rodriguez, R., Mcvicar, M., & De Bie, T.: The T Hit Song Science Once Again a Science? In: 4th International Workshop on Machine Learning and Music Learning from Musical Structure, (2011)
- [4] C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery, 2(2):121–167, 1998.
- [5] B. Logan, A. Kositsky, and P. Moreno. Semantic analysis of song lyrics. In ICME 2004, 2004.
- [6] G. Tzanetakis and P. Cook. Musical genre classi\_cation of audio signals. Speech and Audio Processing, IEEE transactions on, 10(5):293{302, 2002.
- [7] T. Jehan and D. DesRoches. EchoNest Analyzer Documentation, 2012. URL [developer.echonest.com/docs/v4/\\_static/AnalyzeDocumentation.pdf](http://developer.echonest.com/docs/v4/_static/AnalyzeDocumentation.pdf).
- [8] D. Herremans, K. S□brensen, and D. Martens. Classi\_cation and generation of

composer speci\_c music. Working paper - University of Antwerp, 2013.

[9] van Zaanen, Menno; Kanters, P.H.M Automatic mood classification using tf\*idf based on lyrics. 11th International Society for Music Information Retrieval Conference (ISMIR 2010)

[10] Lushan Han, Abhay Kashyap, Tim Finin, James Mayfield and JonathanWeese. UMBC EBIQUITY-CORE: Semantic Textual Similarity Systems.

[11] <http://msaprilshowers.com/emotions/parrotts-classification-of-emotions-chart/attachment/parrotts-chart-of-emotions/>

[12] <http://www.billboard.com/charts/year-end/2014/hot-100-songs>

[12] <http://swoogle.umbc.edu/SimService/api.html>

[13] [http://en.wikipedia.org/wiki/Support\\_vector\\_machine](http://en.wikipedia.org/wiki/Support_vector_machine)

[14] [http://en.wikipedia.org/wiki/Linear\\_regression](http://en.wikipedia.org/wiki/Linear_regression)

[15] [http://en.wikipedia.org/wiki/Logistic\\_regression](http://en.wikipedia.org/wiki/Logistic_regression)

[16] [http://en.wikipedia.org/wiki/K-means\\_clustering](http://en.wikipedia.org/wiki/K-means_clustering)

[17] <http://en.wikipedia.org/wiki/rough%20sets>

[18] [http://en.wikipedia.org/wiki/Machine\\_learning](http://en.wikipedia.org/wiki/Machine_learning)

[19] [http://en.wikipedia.org/wiki/Supervised\\_learning](http://en.wikipedia.org/wiki/Supervised_learning)

[20] [http://en.wikipedia.org/wiki/Unsupervised\\_learning](http://en.wikipedia.org/wiki/Unsupervised_learning)

## **Appendix A – Tools & Platforms Used**

### **Python-**

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

### **Scikit-**

scikit-learn (formerly scikits.learn) is an open source machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, logistic regression, naive Bayes, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

### **NLTK-**

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for the Python programming language. NLTK includes graphical demonstrations and sample data. It is accompanied by a book that

explains the underlying concepts behind the language processing tasks supported by the toolkit, plus a cookbook.

NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning. NLTK has been used successfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research systems.

## Appendix B – Code

### GatherData.py

```

from pattern.web import URL, DOM
from pattern import web
import requests
import csv
from pyechonest import song
from pyechonest import track
import time
from pyechonest import config

config.ECHO_NEST_API_KEY="ZZYYO7TXKOLXEUN9N"

base_url = 'http://www.bobborst.com/popculture/top-100-songs-of-the-year/?year='

song_count = 0

for year in xrange(1954,2014):

    file_name = str(year) + '.csv'
    fp = open(file_name,'w')
    f = csv.writer(fp, delimiter=',')

    url = base_url + str(year)
    r = requests.get(url)
    dom = web.Element(r.text)

    big_list = []

    for tr in dom.by_class('sortable alternate songtable')[0].by_tag('tbody')[0].by_tag('tr'):
        td = tr.by_tag('td')

        small_list = []
        small_list.append(td[0].content)
        small_list.append(td[1].content)
        small_list.append(td[2].content)

        song_count = song_count + 1

        big_list.append(small_list)

    f.writerows(big_list)

    print 'year ' + str(year)

print song_count

for y in xrange(1958,2014):
    write_file_name = 'param/' + str(y) + '_param.csv'

```

```

fp = open(write_file_name,'w')
f = csv.writer(fp, delimiter=',')

headings = ['song_name', 'artist', 'acousticness', 'danceability', 'duration', 'energy', 'key',
'liveness', 'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valence']
f.writerow(headings)

read_file_name = str(y) + '.csv'

with open(read_file_name,'rb') as csvfile:
    song_reader = csv.reader(csvfile, delimiter = ',')

    for row in song_reader:
        try:
            print row[1],row[2]
            rkp_results = song.search(artist=row[1], title=row[2])
            f.writerow([row[2], row[1], rkp_results[0].audio_summary['acousticness'],
            rkp_results[0].audio_summary['danceability'], rkp_results[0].audio_summary['duration'],
            rkp_results[0].audio_summary['energy'], rkp_results[0].audio_summary['key'],
            rkp_results[0].audio_summary['liveness'], rkp_results[0].audio_summary['loudness'],
            rkp_results[0].audio_summary['mode'], rkp_results[0].audio_summary['speechiness'],
            rkp_results[0].audio_summary['tempo'], rkp_results[0].audio_summary['time_signature'],
            rkp_results[0].audio_summary['valence']])
            time.sleep(0.5)

        except:
            f.writerow([row[2], row[1]])
            print 'error'
            continue

```

## LyricsScrapping.sh

```
#!/bin/bash

year=1947

for file in *.csv
do
    echo "===== $year ====="

    mkdir "./yearly_lyrics/$year"

    song_count=0

    while IFS=, read -ra arr
    do
        #echo -e "${arr[0]} ${arr[1]} ${arr[2]} \n"
        song_count=$((song_count+1))

        txt=".txt"
        path="./yearly_lyrics/$year/$song_count$txt"

        songtext -a ${arr[1]} -t ${arr[2]} > $path

        line_count=$(wc -l $path | cut -f1 -d' ')

        if (( line_count < 10 ))
        then
            echo "deleting $path $line_count"
            rm -r $path
        fi

        #echo "${arr[0]} ${arr[2]}"

    done <./$file

    year=$((year+1))
done

echo -e "count is $count \n"
```

## LyricsAnalysisBatchWise.py

```

from pattern.web import URL, DOM
from pattern import web
import requests
from requests import get
import csv
import time
import gensim
import re
import nltk
from nltk.corpus import wordnet as wn
from nltk.tokenize import RegexpTokenizer
from nltk import bigrams, trigrams
import math
import sklearn
import collections
from collections import Counter
from pprint import pprint
import operator

stopwords = nltk.corpus.stopwords.words('english')
tokenizer = RegexpTokenizer("[\w]+", flags=re.UNICODE)

def sss(s1, s2, type='relation', corpus='webbase'):
    try:
        response = get(sss_url,
            params={'operation':'api','phrase1':s1,'phrase2':s2,'type':type,'corpus':corpus})
        ret = float(response.text.strip())
        if ret>0:
            return ret
        else:
            return 0.0
    except:
        print 'Error in getting similarity for',s1,'and',s2
        return 0.0

def getting_emotion_vector_lyrics(yr):
    base_dir='data/billboard list/'
    complete_lyrics = []
    for y in xrange(yr,yr+1):
        song_file_name = base_dir + str(y) + '/' + str(y) + '.csv'
        rfn = base_dir + str(y) + '/lyrics/'
        with open(song_file_name) as csvfile:
            song_reader = csv.reader(csvfile, delimiter = ',')
            for row in song_reader:
                try:
                    lyrics_file_name = rfn + row[0] + '.txt'
                    lyr = open(lyrics_file_name,'r')
                    lyrics = lyr.read()
                    t = lyrics.index('-')
                    while(lyrics[t]!='-'):
                        t=t+1
                    complete_lyrics.append(lyrics[t+2:(len(lyrics))])
                except:
                    print 'Lyrics for song ' + row[2] + ' not found, rank '+row[0]

```



```

        complete_lyrics.append(' ')
    pprint(complete_lyrics)
    y=yr
    lyrics = complete_lyrics
    idf_words = []
    tf = {}
    for i,l in enumerate(lyrics):
        token = tokenizer.tokenize(l.lower())
        token = [t for t in token if t not in stopwords and len(t)>2]
        total=len(token)
        tf_count = Counter(token)
        tf_count={s:float(tf_count[s])/total for s in tf_count.keys()}
        tf[i] = tf_count
        idf_words.extend([word for word in set(token)])
    idf_count = Counter(idf_words)
    tfidf = {}
    for song in tf.keys():
        tfidf[song] = {}
        for word in tf[song].keys():
            tfidf[song][word] = tf[song][word] * math.log(float(len(lyrics))/idf_count[word])
        sorted_tfidf = sorted(tfidf[song].items(), key=operator.itemgetter(1), reverse=True)
        tfidf[song] = dict(sorted_tfidf[:min(20,len(sorted_tfidf))])
    pprint(tfidf)
    sss_url = "http://swoogle.umbc.edu/SimService/GetSimilarity"
    emotion_vector = {}
    basic_emotions = ['love','joy','surprise','anger','sadness','fear']
    for song in tf.keys():
        emotion_vector[song] = {'love':0,'joy':0,'surprise':0,'anger':0,'sadness':0,'fear':0}
        for word in tfidf[song].keys():
            emotion_for_word = [sss(word,emotion) for emotion in basic_emotions]

    emotion_vector[song][basic_emotions[emotion_for_word.index(max(emotion_for_word))]]+=((
    max(emotion_for_word))*tfidf[song][word])
        print song.':',word.':',emotion_vector[song]
    pprint(emotion_vector)
    i=1
    for song in tf.keys():
        maxval = max(emotion_vector[song].iteritems(), key=operator.itemgetter(1))[1]
        keys = [k for k,v in emotion_vector[song].items() if v==maxval]
        print 'predominant emotion in',song,'is/are',keys
    write_emotion_vector = 'data/billboard list/'+str(y)+'/'+str(y)+'_emotion_vectors.csv'
    fp = open(write_emotion_vector,'w')
    f = csv.writer(fp, delimiter=',')
    headings = ['song_rank','love','joy','surprise','anger','sadness','fear']
    f.writerow(headings)
    for rank in emotion_vector.keys():

r=[rank+1,emotion_vector[rank]['love'],emotion_vector[rank]['joy'],emotion_vector[rank]['surpr
ise'],emotion_vector[rank]['anger'],emotion_vector[rank]['sadness'],emotion_vector[rank]['fear']
]
        f.writerow(r)
        print r

    print 'Year ' + str(yr) + ' done'
    return 0

for i in xrange(1947,2014):
    getting_emotion_vector_lyrics(i)

```

**FinalAnalysis.py**

```

import csv
import pandas as pd
from sklearn import svm
from sklearn.linear_model import SGDClassifier
from sklearn import linear_model
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn import preprocessing
from sklearn import linear_model
from sklearn import decomposition

import plotly.plotly as py
from plotly.graph_objs import *

dfs = []

for year in range(1947,2014):

    try:
        df = pd.DataFrame.from_csv(str(year)+''.csv')
        df['song_rank'] = range(1,len(df)+1)
        df = df.dropna()
        df = df[(df['love']>0.) | (df['anger']>0.) | (df['fear']>0.) | (df['joy']>0.) | (df['sadness']>0.) |
(df['surprise']>0.)]

        df['class'] = df['song_rank'].apply(lambda x: int((x-0.1)/50))
        df['year'] = [int(year)] * len(df)

        dfs.append(df)
    except:
        continue

df = pd.concat(dfs)

X =
df[['acousticness','danceability','duration','energy','key','liveness','loudness','mode','speechiness','
time_signature','valence','love','joy','surprise','anger','sadness','fear']].as_matrix()
#X =
df[['acousticness','danceability','energy','liveness','speechiness','valence','love','joy','surprise','ang
er','sadness','fear']].as_matrix()

#X =
df[['acousticness','danceability','duration','energy','key','liveness','loudness','mode','speechiness','
time_signature','valence']].as_matrix()
#X = df[['love','joy','surprise','anger','sadness','fear']]

Y = [a[0] for a in df[['class']].values]

a_train, a_test, b_train, b_test = train_test_split(X, Y, test_size=0.20, random_state=42)

a_train_scaled = preprocessing.scale(a_train)
scaler = preprocessing.StandardScaler().fit(a_train_scaled)
a_test_scaled = scaler.transform(a_test)

```

```

#a_train_sc_no = preprocessing.normalize(a_train_scaled, norm='l2')

#print np.array(a_train_scaled)

print a_train_scaled[0][5]

#print a_test

pca = decomposition.PCA(n_components=2)
pca.fit(a_test)
a_train = pca.transform(a_train)
a_test = pca.transform(a_test)

#clf = svm.SVC(C=1)
#clf = linear_model.LogisticRegression()
clf = linear_model.LinearRegression()
clf.fit(a_train,b_train)

prediction = [0 if val<0.5 else 1 for val in clf.predict(a_test)]
# prediction = clf.predict(a_test)
print pd.DataFrame({'ans':b_test,'prediction':prediction})

# b_test_not = [1^b for b in b_test]
# print max(clf.score(a_test, b_test), clf.score(a_test,b_test_not))
c = 0
for i,val in enumerate(prediction):
    if prediction[i]==b_test[i]:
        c=c+1

print max(float(c) / len(prediction), 1 - (float(c) / len(prediction)))

data = Data([
    Bar(

x=['acousticness','danceability','duration','energy','key','liveness','loudness','mode','speechiness','
time_signature','valence','love','joy','surprise','anger','sadness','fear'],
    y=clf.coef_
    )
])
py.iplot(data, filename='basic-bar')

#print
len(['acousticness','danceability','duration','energy','key','liveness','loudness','mode','speechiness
','time_signature','valence','love','joy','surprise','anger','sadness','fear']), len(clf.coef_)

x_zero = [a_test[(i,ans)[0],0] for i,ans in enumerate(b_test) if ans==0]
y_zero = [a_test[(i,ans)[0],1] for i,ans in enumerate(b_test) if ans==0]

x_one = [a_test[(i,ans)[0],0] for i,ans in enumerate(b_test) if ans==1]
y_one = [a_test[(i,ans)[0],1] for i,ans in enumerate(b_test) if ans==1]

m,c = clf.coef_

```

```

x1 = -800
y1 = m*x1+c

x2 = 200
y2 = m*x2+c

print m,c

import plotly.plotly as py
from plotly.graph_objs import *

trace_zero = Scatter(
    x=x_zero,
    y=y_zero,
    mode='markers',
    name='<50',
    #text=['United States', 'Canada'],
    marker=Marker(
        color='rgb(164, 194, 244)',
        size=12,
        line=Line(
            color='white',
            width=0.5
        )
    )
)

trace_one = Scatter(
    x=x_one,
    y=y_one,
    mode='markers',
    name='50-100',
    #text=['United States', 'Canada'],
    marker=Marker(
        color='rgb(142, 124, 195)',
        size=12,
        line=Line(
            color='white',
            width=0.5
        )
    )
)

trace_line = Scatter(
    x=[x1,x2],
    y=[y1,y2],
    name='linear-regression line',
    mode='lines'
)

data = Data([trace_zero, trace_one, trace_line])
layout = Layout(
    title='PCA and linear Regression',
    xaxis=XAxis(
        title='PCA component 1',
        showgrid=False,
        zeroline=False

```

```

    ),
    yaxis=YAxis(
        title='PCA component 2',
        showline=False
    )
)
fig = Figure(data=data, layout=layout)
py.ipplot(fig, filename='line-style')

def classify(start_year, end_year, classifier, lyrics=True, audio=True):
    dfs = []

    for year in range(start_year, end_year):

        try:
            df = pd.DataFrame.from_csv(str(year) + '.csv')
            df['song_rank'] = range(1, len(df) + 1)
            df = df.dropna()
            df = df[(df['love'] > 0.) | (df['anger'] > 0.) | (df['fear'] > 0.) | (df['joy'] > 0.) | (df['sadness'] > 0.) |
                    (df['surprise'] > 0.)]

            df['class'] = df['song_rank'].apply(lambda x: int((x - 0.1) / 50))
            df['year'] = [int(year)] * len(df)

            dfs.append(df)
        except:
            continue

    df = pd.concat(dfs)

    if lyrics and audio:
        X =
df[['acousticness', 'danceability', 'duration', 'energy', 'key', 'liveness', 'loudness', 'mode', 'speechiness',
time_signature', 'valence', 'love', 'joy', 'surprise', 'anger', 'sadness', 'fear']].as_matrix()
    elif audio:
        X =
df[['acousticness', 'danceability', 'duration', 'energy', 'key', 'liveness', 'loudness', 'mode', 'speechiness',
time_signature', 'valence']].as_matrix()
    elif lyrics:
        X = df[['love', 'joy', 'surprise', 'anger', 'sadness', 'fear']]
        Y = [a[0] for a in df[['class']].values]

    a_train, a_test, b_train, b_test = train_test_split(X, Y, test_size=0.20, random_state=42)

    if classifier == 'svm':
        clf = svm.SVC(C=1)
    elif classifier == 'logistic':
        clf = linear_model.LogisticRegression()
    elif classifier == 'linear':
        clf = linear_model.LinearRegression()

    clf.fit(a_train, b_train)

    prediction = clf.predict(a_test)

    if classifier == 'linear':
        prediction = [0 if val < 0.5 else 1 for val in clf.predict(a_test)]
        c = 0

```

```

    for i,val in enumerate(prediction):
        if prediction[i]==b_test[i]:
            c=c+1

    return max(float(c) / len(prediction), 1 - (float(c) / len(prediction)))

b_test_not = [1^b for b in b_test]
return max(clf.score(a_test, b_test), clf.score(a_test,b_test_not))

print
classify(1960,1970,'logistic',lyrics=True,audio=False),classify(1980,1985,'logistic',lyrics=False,audio=True),classify(1980,1985,'logistic',lyrics=True,audio=True)
# print classify(1980,1985,'logistic')
# print classify(1980,1985,'svm')

count = np.zeros(6)
emotion = df[(df['love']>0.) | (df['anger']>0.) | (df['fear']>0.) | (df['joy']>0.) | (df['sadness']>0.) | (df['surprise']>0.)][['year','love','joy','surprise','anger','sadness','fear']]

emotion['dominant_emotion'] =
emotion[['love','joy','surprise','anger','sadness','fear']].idxmax(axis=1)

e_65_70 = emotion[(emotion['year']>=1965) &
(emotion['year']<1970)][['dominant_emotion'].value_counts()
e_70_75 = emotion[(emotion['year']>=1970) &
(emotion['year']<1975)][['dominant_emotion'].value_counts()
e_75_80 = emotion[(emotion['year']>=1975) &
(emotion['year']<1980)][['dominant_emotion'].value_counts()

e_75_80 = e_75_80.map(lambda x: (float(x)/e_75_80.sum()) * 100)

print e_75_80

#print emotion[emotion['dominant_emotion']=='anger']

import plotly.plotly as py
from plotly.graph_objs import *

trace1 = Bar(
    x=['1960-65', '1965-70', '1970-75'],
    y=[e_65_70['love'], e_70_75['love'], e_75_80['love']],
    name='Love'
)
trace2 = Bar(
    x=['1960-65', '1965-70', '1970-75'],
    y=[e_65_70['fear'], e_70_75['fear'], e_75_80['fear']],
    name='Fear'
)
trace3 = Bar(
    x=['1960-65', '1965-70', '1970-75'],
    y=[e_65_70['joy'], e_70_75['joy'], e_75_80['joy']],
    name='Joy'
)
trace4 = Bar(
    x=['1960-65', '1965-70', '1970-75'],
    y=[e_65_70['surprise'], e_70_75['surprise'], e_75_80['surprise']],

```

```
    name='Surprise'
)
trace5 = Bar(
    x=['1960-65', '1965-70', '1970-75'],
    y=[e_65_70['anger'], e_70_75['anger'], e_75_80['anger']],
    name='Anger'
)

trace6 = Bar(
    x=['1960-65', '1965-70', '1970-75'],
    y=[e_65_70['sadness'], e_70_75['sadness'], e_75_80['sadness']],
    name='Sadness'
)
data = Data([trace1, trace2, trace3, trace4, trace5, trace6])
layout = Layout(
    barmode='stack'
)
fig = Figure(data=data, layout=layout)
py.ipplot(fig, filename = 'stacked-bar')
#plot_url = py.plot(fig, filename='stacked-bar')
```