Music Genre Categorization Using Machine Learning Techniques

A Major Project Report

Submitted in Partial Fulfilment for the Award of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

By

Rahul Mishra
(2012ECS01)

Sumit Jha
(2012ECS11)

Mahendra Kumar
(2012ECS53)

Under the Guidance of
Mr. Sanjay Sharma

To

SHRI MATA VAISHNO DEVI UNIVERSITY, J&K, INDIA

MAY, 2016

CERTIFICATE

This is to certify that we, Rahul Mishra (2012ECS01), Sumit Jha (2012ECS11), Mahendra Kumar (2012ECS53) have worked under the guidance of "Mr. Sanjay Sharma" on the project titled **"Music Genre Categorization using Machine Learnig Techniques"** in the School of Computer Science & Engineering, College of Engineering, Shri Mata Vaishno Devi University, Kakryal, Jammu & Kashmir from 2nd Jan 2016 to 23rd May 2016 for the award of Bachelor of Technology in Computer Science & Engineering.
The contents of this project, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Student 1 Signature	:	
Student 1 name	:	Rahul Mishra
Student 2 Signature	:	
Student 2 name	:	Sumit Jha
Student 3 Signature	:	
Student 3 name	:	Mahendra Kumar

This is to certify that the above student has worked for the project titled **"Music Genre Categorization using Machine Learnig Techniques"** under my supervision.

Mr. Sanjay Sharma
Guide Name & Signature

Date: ______________

ACKNOWLEDGEMENT

It is a moment of great pleasure and immense satisfaction for us to express our deepest sense of gratitude and indebtedness to all the people who have contributed in making of our major project a rich experience.

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly indebted to **Mr. Sanjay Sharma** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

We would like to express our gratitude towards members of **Shri Mata Vaishno Devi University** for their kind co-operation and encouragement which help us in completion of this project.

Finally, we would like to thank all our Teachers, Friends and Family Members who have supported us throughout and enlightened us to take the right path and reach there. The days we spent in this institute will be cherished forever and also be reckoned as a guiding factor in our career.

Rahul Mishra

Mahendra Kumar

Sumit Jha

B. Tech. 8th Semester (SCSE)

ABSTRACT

Music has some really powerful effect on our emotions, and Human ear is quite incredible at predicting the genre accurately, although music genre is a relatively complex concept that sometimes the music industry itself feels confused in assigning genre to some of the songs.
Classification of genre by Machine is one of such technical approach which has a major drawback of predicting genre accurately. Previously many attempts has been made to design system to categorize music but the prediction result was not overwhelming. So, the problem continued that can machine predict genres of songs with better accuracy results?

The Project discusses various Machine Learning concepts of classification and improving the accuracy of those classification techniques to be used for Genre based Music Categorization is the goal of our Project. In this project Million Song Dataset, made available by LabROSA, containing 30 summary features for each music files one of 10 genre, being used as training and testing data. The project focuses on implication of Classification algorithm like, Random Forest, Gaussian Naive Bayes, Support Vector Machine, Decision tree, K-Nearest Neighbor. Lyrical modelling concept has been the second approach improvising Bag of Words technique over the dataset.

We aimed to study and apply various machine learning concepts to solve a real world problem like Music Categorization based on Genre and come out with an improved result of **62% F1-score while using Multiclass classification technique and 47% F1-score accuracy while applying Lyrical Modelling concept**. For the fact that many songs are unlabeled and much more songs are being released daily, the classification technique predicting genre with an accuracy of 62% F1-score is quite of help.

List of Figures

Table of Contents

1. INTRODUCTION

Classifying Music in CD stores and in online digital media has genre categorization at its base because of its ease to differentiate music in varieties. It has been considered a tradition in CD stores to differentiate shelf on the basis of genre to guide consumers into specific album of specific singer. Apart from that, Online digital media and personal song collection also has genre as one of its main categorization criteria.

Humans are quite remarkable in distinguishing Music Genre. Usually a rough classification can be done by us after listening few seconds of music but with the improvement in Machine Learning algorithms, after 2010 many attempts has been made to design system for Automatic Music Genre Classification but the result were not very good. For those who have used the same dataset of Million Song, provided by LabROSA, comprised of 10 different music genres, has yielded an F1-sore of 58% in [1], which needs to be improved**.**

1.1. Motivation

Consumption of digital music is gaining popularity and its distribution over internet is increasing by significant amount day by day. Music genre are generally used to categorize digital music collection so as to facilitate navigation into it. Associating genres automatically to music has become so important firstly because of the increase in number of music unit and secondly because automatic music genre classification generates category independent of manual subjective category.

In this digital era, millions of songs are being accessed by consumers. Also, with the technological advancement artist and producer are able to release and distribute songs instantly. This democratization of accessing digital media has arisen the requirement to develop efficient categorization systems.

Humans have always been primary tool in attributing genre-tags to songs. Using machine to do the job is a more complex task, but is the requirement of present time. Since Machine learning excels majorly in handling complex data very well, so, this project addresses learning of various Machine learning algorithm to be used for automatically classifying music files with improved prediction accuracy.

Advantages of Music Genre Categorization

Automatic classification based on genre will help creating music database in such a way that a general description like 90's classical is given by user and software based on the classifier does the file selection allowing user to create playlist of his own selection criteria. Apart from its practical implication, music genre classification is interesting field of study as well. Rigorous study of the classification and machine based music processing will certainly enhance our knowledge about the human perception of music.

1.2.System Overview

Extracting feature from music file for classification is not that easy. Digital music is discretized sampled waveform, although categorization over frequency domain is preferred than the time domain because of the ability of humans to differentiate between frequencies range like, deep bass in hip-hop is Low frequency while talking, singing is mid-range frequency and sharp claps are considered high-pitched frequency.

The base of the project is Million Songs genre dataset that initially contains songs with 34 summary features given as 'genre, track id, artist name, title, loudness, tempo, time signature, key, mode, duration and 24 different timbre value for each segment'. Each of the song is categorized in one of the 10 genres, mainly classic pop and rock, classical, dance and electronica, folk, pop, hip-hop, jazz and blues, punk, metal & lastly soul and reggae. After that feature scaling has been applied over the dataset to obtain 30 summary feature namely 'loudness, tempo, time signature, key, mode, duration and 24 different timbre value for each segment' along with the 'genre' for each song. The resultant dataset has been modified into 9 genre collection by removing hip-hop. Before applying the machine learning algorithms over dataset it has been accessed to obtain almost equal number of features for each genre (approx. 2000 songs of each genre) type so as to provide a good training condition for Multiclass Classification algorithms. Some of the main classification algorithms applied are Decision Tree classifier, K Nearest Neighbor classifier, Random Forest and Naïve Bayes classifier, out of which Random Forest has given the best result of 62% F1-Score. The machine learning algorithm has been formulated using scikit learn library and numpy library of python programming.

Apart from that, the dataset has been modified into 2 summary feature dataset, one being genre and other the lyrics of that particular song, for lyrical modelling approach to be applied over it. Based on the availability of lyrics for each song in the 'MSD' dataset for only approx. 24700 songs has been formed that has been re-featured to a collection of approx. 7500 songs in 8 genres. In Lyrical Model the concept of 'bag of words' has been implemented for categorization.

1.3.Scope and Result of Project

Overview

This project involves evaluation of the current state of automatic genre based classification of music and represents the methods of field with improvement. The work doesn't include the audio signal analysis process for feature extraction but shows the importance and usage of the feature provided in dataset for prediction process. The projected focuses specially on the algorithms applied on the features of song, its training, fitting of data and predicting the genre based on the learned data.

Structure

The project is represented in an organized way given as: The starting has included the overview of Music genre, the Million Song dataset used and the feature detailing of the available features in the dataset. It has also enlisted the human perception of music and study of genre categorization by human. The next Section consists of the Basics of Machine Learning followed by various algorithm of multiclass classification and lyrical model approach that had been used for the categorization purpose. The following section shows dataset and machine learning technique correlation for fitting of data, learning from data and predicting the genre for the provided data. After that the next section provides the experimental overview of the project, defining the implication and result for each of the used algorithm, their advantages, drawbacks and comparison with one another to highlight the best model for classification purpose. The next is all the codes written followed by Conclusion and lastly references that has been of great support in the whole project.

Result

Multiclass classification Algorithms prediction accuracy has resulted in 62% F1-score, an increase of 4% from the previous best F1 Score of 58% in [1], with the best model of Multiclass classification approach being Random Forest

An increase of 5% with Lyrical Modelling concept with F1-score of 47%.

2. Requirement Specification

2.1.Preprocessing the Data

Before working on the dataset the data needs to be processed and formatted in suitable formats and for that we require following steps to be followed before we can work on dataset.

- First of all, feature scaling needs to be performed over the dataset so that the features are scaled between values (-1 to 1). So, that a single feature doesn't dominates the prediction process only because it has higher ranges of value as compared to other features.
- All the features are needed to be converted to floating types and not mixed type, because if latter is the case numpy arrays generated by us, are converted to string type and that could lead to unexpected results.
- In the Bag-of-word scheme, stemming should done before the bagging process as it could reduce the number of bags and could produce more meaningful results.
- Another important preprocessing step that needs to be done is Principal Component Analysis (PCA) for dimension reduction in bag-of-words so that we only consider important words only while classification.
- Splitting the dataset into training and testing set is little tricky but is quite well handled by sklearn library in a single line split them in 80:20 percentage respectively that too the elements are taken randomly to form the two sets so that no biasing occurs.

2.2.Python Libraries

The few important libraries to be checked before running code on local environment.

- ➢ Sklearn (scikit-learn)
- ➢ Numpy
- ➢ Scipy
- ➢ Matplotlib
- ➢ Beautifulsoup
- ➢ Request
- ➢ Pickel

3. Music Genre Classification

A common definition of terms is necessary in every field of science and that too is applicable for Music genre classification. However, such definition doesn't happen to exist in our field of study, even the literature disagrees on the basic terminologies. Two of the main reasons identified for this are:

- Music being a part of daily life, has terms having an intuitive meaning, like for the features of music song such as tempo, rhythm, pitch.
- Human perceive sound depending on its personal, cultural and emotional aspect and so does happens for classification.

This lack of simple basic hinders the progress of audio signal classification resulting in degradation in feature extraction on which the whole classification process majorly relies.

Music genre classification is a subfield of the larger field of audio signal classification, defined as working with the extracted features of audio signal (obtaining relevant features from a sound and based on those identify the set of class the sound is most likely to be classified) to be used by the classification algorithms for genre . Automatic genre categorization refers to achieve this task with machines using machine learning techniques.

3.1.Genres and Feature vector

Conventional classification approach that identifies music belonging to a particular tradition or set of conventions is **Music genre** [2]. Now days genre has become the very basic categorization criteria and so the music are categorized into different genres but because of the artistic nature of music these criteria of classification is considered subjective and it might happen that some genres may overlap.

Although the project doesn't focuses on the feature extraction technique but the study is really the need for the feature selection to be done from dataset so as to achieve better prediction result. Apart from that based on the features availability the fitting of dataset for training purpose of the classifiers used throughout the project.

3.1.1. Genre

History

Initially genre analysis has been used for teaching and learning of English for special purpose, but the digitalization of communication and domination of Internet after 90's has extended genre study to the digital field. Crowsten and William (1998) were some of the first to study genre for digital media. They identified the importance of study of genre for analysing digital world of internet because of ease of access [3]. They identified 48 various genre after studying random 100 web pages.

The term 'Cybergenre' was first coined by Shepherd and Watters in 2001, denoting digital genre which was further termed in two categories based on the dependency on the media by many authors.

- Extant subgenre: genre of some media that migrated to another computer environment and replicated without exploiting the capabilities of the new media, e.g. newspaper, dictionaries, research articles and biographies.
- Novel genres: genres totally dependent on one medium and couldn't exist in another, e.g. virtual games and homepages.

Genre in Music

The Latin word 'genus' meaning 'class or kind' was the origin for word genre that we use today. It might be a considered a category that can be defined by structural or functional criteria. Sometimes criteria like musical techniques, style, the cultural context or the content of the music also plays important role to define a music genre or subgenre. Geographical origin also sometimes play important criteria to classify a music genre like ‘jazz and blues’, though a single geographic category might include many subgenres.

Genre classification is always been a subjective term in respect of both listener and socio-cultural environment. Hence a definition for music genre can be stated as: ‘*A music genre is a specific class of music consisting of set of some common properties that an average listener perceive so as to distinguish music of that class from other songs’.* A specific genre has unique characteristic feature of rhythmic structure as well as the instrumentation property for that particular kind of music, although there are certain more features used human for manual categorization of music genre. The major challenge of automatic genre classification is to find out those factors and extracting those features from the music file.

3.1.2. Feature Extraction

This includes the very basic key concept of feature extraction keeping signal processing in its background. The project doesn’t implement any technique for feature extraction but prior knowledge of features of songs that gets extracted from song is needed before going to the implementation part of machine learning classifier algorithm so as to learn the relative feature to be used in the training process. The main difficulty of music genre categorization at first is to differentiate between music styles, i.e. to obtain feature vector for various genre song. It is similar to the fact of comparing two object (in our case it is music songs) for their similarity and dissimilarity when they can’t be directly compared. Although our project directly access the million song dataset that already contains extracted features i.e. 34 summary feature that can be directly accessed only after separating the required features from dataset and applying feature scaling.

Feature Extraction is a process of transforming the data in order to make it accessible for essential information retrieval by computing a numerical representation for a segment of audio signal. For classification *feature extraction* is one of the two commonly used pre-processing technique. It applies one or more transformation over raw data to generate new features. The other one is *feature selection* – that identifies a feature subset within the raw data to be used for effective classification which has been implemented within the project. The Feature selection process can be applied over the input dataset or over the output provided by Feature extraction process. A classification system might use both or either of the two techniques depending on the accuracy the feature extraction is giving as shown in *Fig* 1. Our project uses only the Feature selection pre-processing as the Million songs dataset initially contains the extracted features of songs in it. The Feature selection pre-processing technique has been elaborated further in the project.

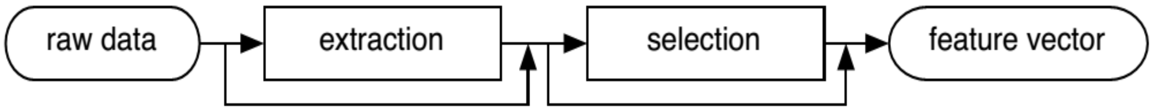


Figure 1. Schematic overview of feature extraction and feature selection

Most of the information of audio signal is encoded in its frequency phase and the spectral component's amplitude. So, the information can be retrieved by examining the signal's *frequency spectrum* that is similar to what happens in human auditory system. '*Fourier Analysis*' is a mathematical technique used for the above. *Fourier transform, Short-Time Fourier Transform* are some of Fourier analysis implementation method. For more detailing, one can consult books on signal processing.

3.1.3. Common Music Genres

From rock to pop, from classical to hip-hop, music ranges in various type and styles. There is a very rich history of music genres that even contain enlightening geographical significance. This section introduces to some of the common genres that has been used for the classification purpose in the project.

- **Classical pop and rock**

  Classical rock is a radio format that has its origin in 1980s from the album-oriented rock format. It contains minimal voice component and is traditionally built on simple unsyncopated or unmodified rhythms.

  Pop music or 'popular music' (terms being used interchangeably) has originated from the western world during 1950s and 60s. Music of this genre is generally featured by a consistent rhythmic element. Although rock and pop often have overlaps. Classical music do not contain repetitive and heavily pronounced rhythms and has less bass.

- **Dance and electronica**

  Also referred as Electronic Dance Music (EDM) is a kind of club music having broad range of persuasive electronic music. It has some subgenres namely Disco, Techno, House, Ambient, jungle, drum and bass and Electro. It has consistent bass rhythms i.e. all the quarter notes.

- **Folk**

  It is combination of both traditional music and the folk revival of 20th century that evolved from the traditional music itself. It is mainly geographically originated genre and is often related to a nation's culture. A folk music has a short instrumental piece called tune, a melody and repetitive word section.

- **Hip-hop**

  Hip-hop music also termed ‘rap’ has origin in 1970s in the United States. This genre consist of stylized rhythmic music i.e. a rhythmic and rhyming speech usually accompanies rapping. It is generally associated with deep, rhythmically repetitive bass.

- **Jazz and blues**

  Jazz and blues music was originated from African American communities during late 19th and early 20th century. This genre can refer either to blues containing advanced harmonies and rhythms or to jazz having rhythms real simple. This categorization genre have soft volume and excludes repetitive rhythms.

- **Metal**

  Metal or mostly called heavy metal is a genre of rock music have origin in UK and USA in between 1960s and 70s. Loud distorted guitars is metal genre’s traditional characteristics. It is often characterized by emphatic music dense bass & drum sounds along with energized vocals.

- **Punk**

  This is a loud, fast and kind of distorted genre category that has its origin in US during 1970s. It hard edged melodies and stripped down instrumentation.

- **Soul and reggae**

  Reggae music genre has its origin during 1960s in Jamaica. The bass sound is often thick and heavy in this genre. The bass sound is equalized too so that the upper frequencies are removed and emphasized lower frequency is obtained.

3.2.Schematic Model

As stated previously, Music genre categorization is all about classification of songs into some predefined categories. The goal is therefore to design algorithms that classifies the input digital music file into specific genre given the features of the song is provided with the song initially.

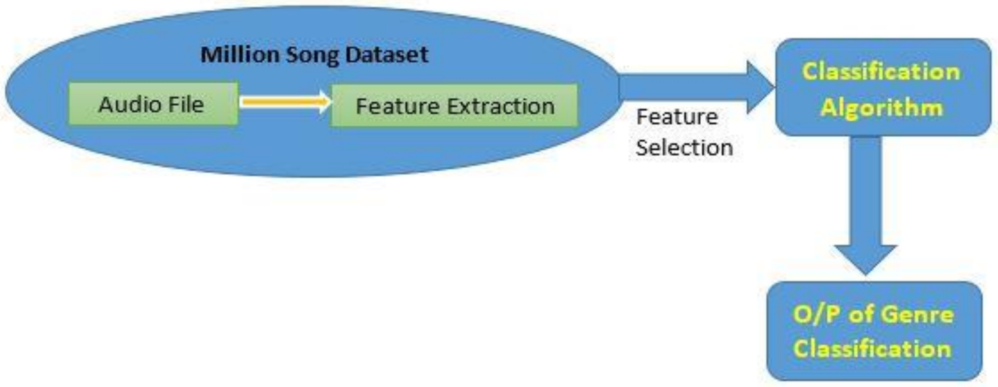


Figure 2. Schematic Overview of music genre Classification

The oval shape is the Million Song dataset that already contains the extracted featured of the songs and so the process of extraction over audio file is not required. However, the process has been discussed in the section **2.1.2.** The only requirement to achieve goal is to do apply pre-processing technique feature selection (Section 2.1.2.) and then applying the classifier over the selected feature to do the final classification of songs into genres. The classification has been well deduced in Section 3. *Figure 2.* Shows the most simple representation of the classification been done.

3.3.Difficulty

Music genre classification is a tough task to perform with machine being the classifying tool. This is mostly because of some of following reasons:

First of all, it is a fact that music is a perceptual phenomenon. Although human ear receives the sound well but how it is actually perceived by us is still a major topic of discussion and has no clear results. The knowledge we have about perception is not sufficient enough to build a system that perfectly simulates like human for music perception.

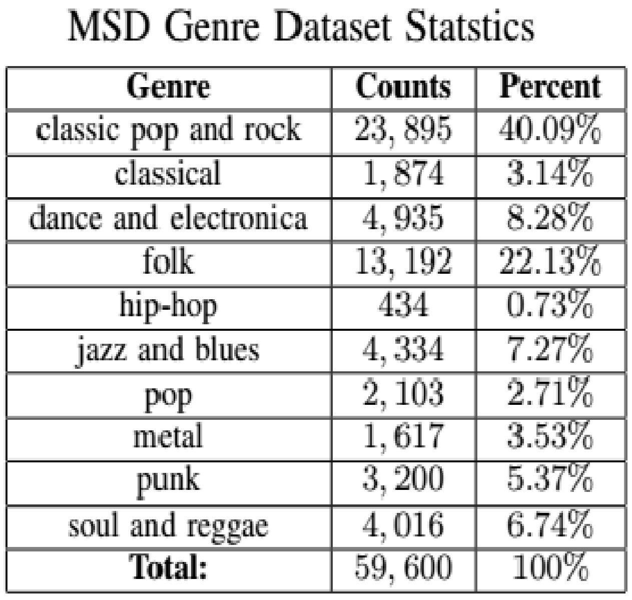
Secondly, the human perception of a song genre is not always correct and hence the music genre classification by human doesn't always result correct. So, it is difficult to train a system for classification purpose in presence of such ambiguity.

So, to model a classification system, it might be a need to check if the system too does the mistakes in classifying particular genre of song that can be classified into more than one genre, similarly like the humans do.

3.4.The Dataset and Features

For this project the Million Song Genre Dataset has been used, that is been made available by LabROSA at Columbia university [4], for educational study purpose.

This dataset consists of about million songs of various genres, to be precise of 10 genres, namely classic pop and rock, classical, dance and electronica, folk, pop, hip-hop, jazz and blues, punk, metal & lastly soul and reggae. The distribution of songs into various genres can be recognized in Figure 3.



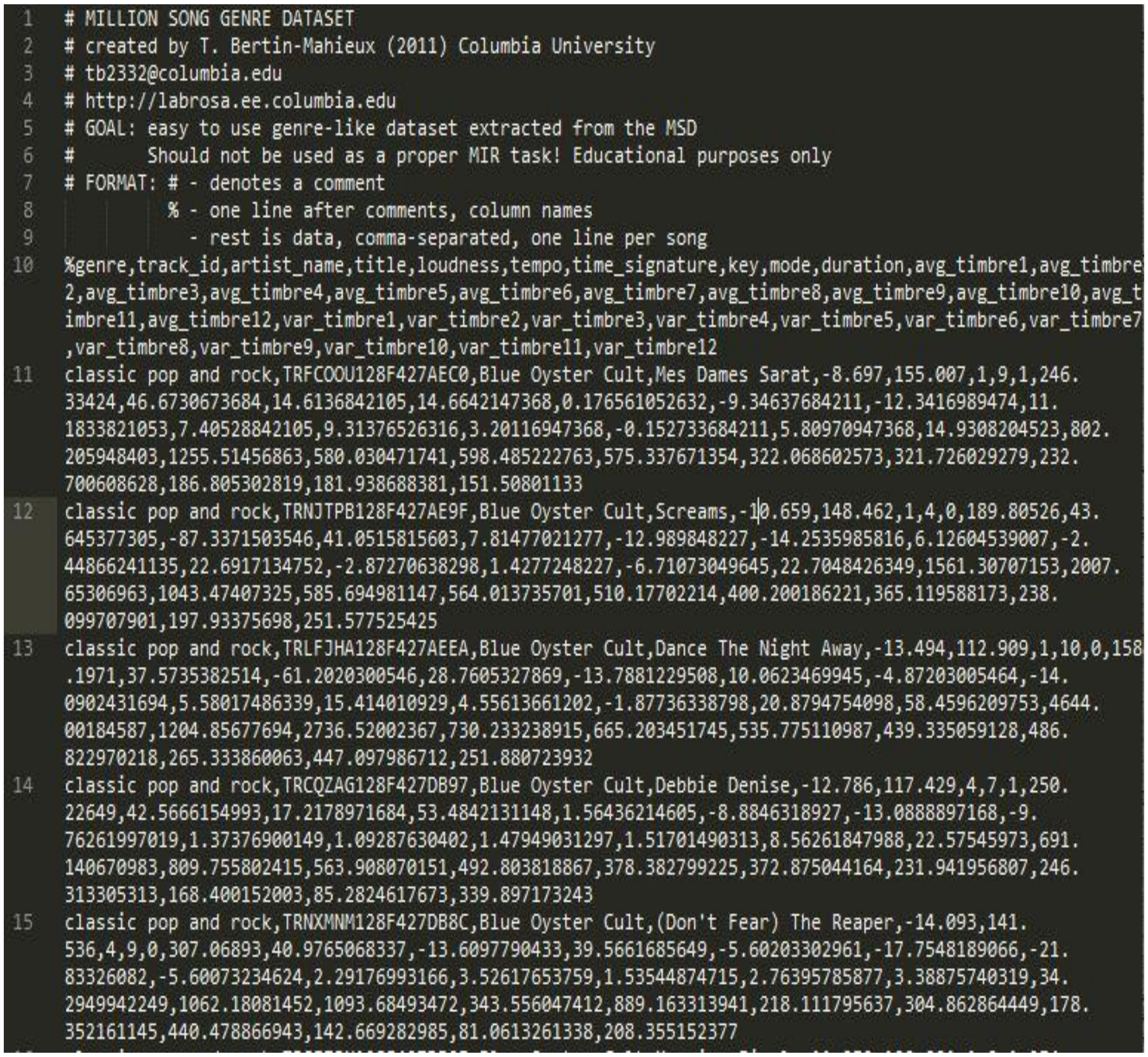
MSD Genre Dataset Statstics

Genre	Counts	Percent
classic pop and rock	23, 895	40.09%
classical	1, 874	3.14%
dance and electronica	4, 935	8.28%
folk	13, 192	22.13%
hip-hop	434	0.73%
jazz and blues	4, 334	7.27%
pop	2, 103	2.71%
metal	1, 617	3.53%
punk	3, 200	5.37%
soul and reggae	4, 016	6.74%
Total:	59, 600	100%

Figure 3. Dataset Composition

This dataset is a well-organized classification dataset as it consist the features of each of the song in the dataset too. The Million Song Dataset has about million songs with related metadata and audio analysis features, in total 34 summary features given as ‘genre, track id, artist name, title, loudness, tempo, time signature, key, mode, duration and 24 different timbre value for each segment’. The Feature representation and value of each feature for some of the songs can be viewed in the *Figure 4*.

In this dataset, we have each song as a training example containing 34 summary features along with the genre of the track, which would be of use during training as well as testing the classifiers for predicting the genre of particular combination of features.



```
# MILLION SONG GENRE DATASET
# created by T. Bertin-Mahieux (2011) Columbia University
# tb2332@columbia.edu
# http://labrosa.ee.columbia.edu
# GOAL: easy to use genre-like dataset extracted from the MSD
#       Should not be used as a proper MIR task! Educational purposes only
# FORMAT: # - denotes a comment
          % - one line after comments, column names
            - rest is data, comma-separated, one line per song
%genre,track_id,artist_name,title,loudness,tempo,time_signature,key,mode,duration,avg_timbre1,avg_timbre
2,avg_timbre3,avg_timbre4,avg_timbre5,avg_timbre6,avg_timbre7,avg_timbre8,avg_timbre9,avg_timbre10,avg_t
imbre11,avg_timbre12,var_timbre1,var_timbre2,var_timbre3,var_timbre4,var_timbre5,var_timbre6,var_timbre7
,var_timbre8,var_timbre9,var_timbre10,var_timbre11,var_timbre12
classic pop and rock,TRFCOOU128F427AEC0,Blue Oyster Cult,Mes Dames Sarat,-8.697,155.007,1,9,1,246.
33424,46.6730673684,14.6136842105,14.6642147368,0.176561052632,-9.34637684211,-12.3416989474,11.
1833821053,7.40528842105,9.31376526316,3.20116947368,-0.152733684211,5.80970947368,14.9308204523,802.
205948403,1255.51456863,580.030471741,598.485222763,575.337671354,322.068602573,321.726029279,232.
700608628,186.805302819,181.938688381,151.50801133
classic pop and rock,TRNJTPB128F427AE9F,Blue Oyster Cult,Screams,-10.659,148.462,1,4,0,189.80526,43.
645377305,-87.3371503546,41.0515815603,7.81477021277,-12.989848227,-14.2535985816,6.12604539007,-2.
44866241135,22.6917134752,-2.87270638298,1.4277248227,-6.71073049645,22.7048426349,1561.30707153,2007.
65306963,1043.47407325,585.694981147,564.013735701,510.17702214,400.200186221,365.119588173,238.
099707901,197.93375698,251.577525425
classic pop and rock,TRLFJHA128F427AEEA,Blue Oyster Cult,Dance The Night Away,-13.494,112.909,1,10,0,158
.1971,37.5735382514,-61.2020300546,28.7605327869,-13.7881229508,10.0623469945,-4.87203005464,-14.
0902431694,5.58017486339,15.414010929,4.55613661202,-1.87736338798,20.8794754098,58.4596209753,4644.
00184587,1204.85677694,2736.52002367,730.233238915,665.203451745,535.775110987,439.335059128,486.
822970218,265.333860063,447.097986712,251.880723932
classic pop and rock,TRCQZAG128F427DB97,Blue Oyster Cult,Debbie Denise,-12.786,117.429,4,7,1,250.
22649,42.5666154993,17.2178971684,53.4842131148,1.56436214605,-8.8846318927,-13.0888897168,-9.
76261997019,1.37376900149,1.09287630402,1.47949031297,1.51701490313,8.56261847988,22.57545973,691.
140670983,809.755802415,563.908070151,492.803818867,378.382799225,372.875044164,231.941956807,246.
313305313,168.400152003,85.2824617673,339.897173243
classic pop and rock,TRNXMNM128F427DB8C,Blue Oyster Cult,(Don't Fear) The Reaper,-14.093,141.
536,4,9,0,307.06893,40.9765068337,-13.6097790433,39.5661685649,-5.60203302961,-17.7548189066,-21.
83326082,-5.60073234624,2.29176993166,3.52617653759,1.53544874715,2.76395785877,3.38875740319,34.
2949942249,1062.18081452,1093.68493472,343.556047412,889.163313941,218.111795637,304.862864449,178.
352161145,440.478866943,142.669282985,81.0613261338,208.355152377
```

Figure 4. Small part of Million Songs Genre Dataset

3.4.1. Terminologies

The dataset contains various features that needs little description.

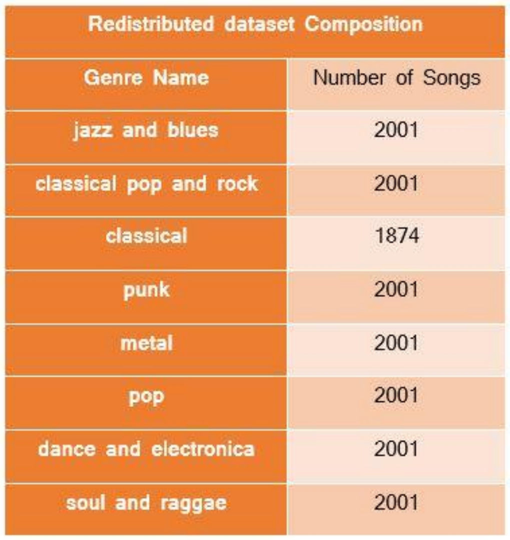
genre:	is the music category of the particular track in the dataset.
track_id:	is the musicmatch track id of the track
artist_name:	is the song's artist name
title:	title of the song
loudness:	is the property of a sound that is primarily a psychological correlation of physical strength (amplitude).
tempo:	The tempo is the speed of the underlying beat for a piece of music. Tempo is measured in BPM, or Beats/Minute. The top number represents how many beats there are in a measure, and the bottom number represents the note value which makes one beat.
key:	The key of a piece is a group of pitches, or scale upon which a music composition is created.
mode:	Refers to a type of scale, coupled with a set of characteristic melodic behaviors.
duration:	song duration (in Secs).
avg_timber1 - 12:	Timbre is then a general term for the distinguishable characteristics of a tone.
var_timber1 – 12:	Timbre is mainly determined by the harmonic content of a sound and the dynamic characteristics.

3.4.2. Feature Selection and implementation

For Multiclass Classification

The given dataset composition of songs is unbalanced as shown in *Figure 3*, that classifies genres with 40% prediction rate except for 'classical rock and pop', since this genre is 40.09% of the whole dataset, which is quite a large composition, that enables the machine to train better over 'classical rock and pop' and give a good testing result.

So, we redistributed the dataset to obtain a balanced composition of songs given in *Figure 5*.



Redistributed dataset Composition	
Genre Name	Number of Songs
jazz and blues	2001
classical pop and rock	2001
classical	1874
punk	2001
metal	2001
pop	2001
dance and electronica	2001
soul and raggae	2001

Figure 5. Balanced Dataset Composition

Apart from that, the feature selection process has been implemented, to select important features out of the given 34 summary feature in the dataset corresponding to each track, and speparating genre form the features. In the project we have used 30 summary features namely 'loudness, tempo, time signature, key, mode, duration and 24 different timbre value for each segment' appended in a single vector termed feature vector along with genre for each track in the balanced dataset.

Code Implementation

```
import numpy as np
import math
import sys
import matplotlib
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.cross_validation import train_test_split
```

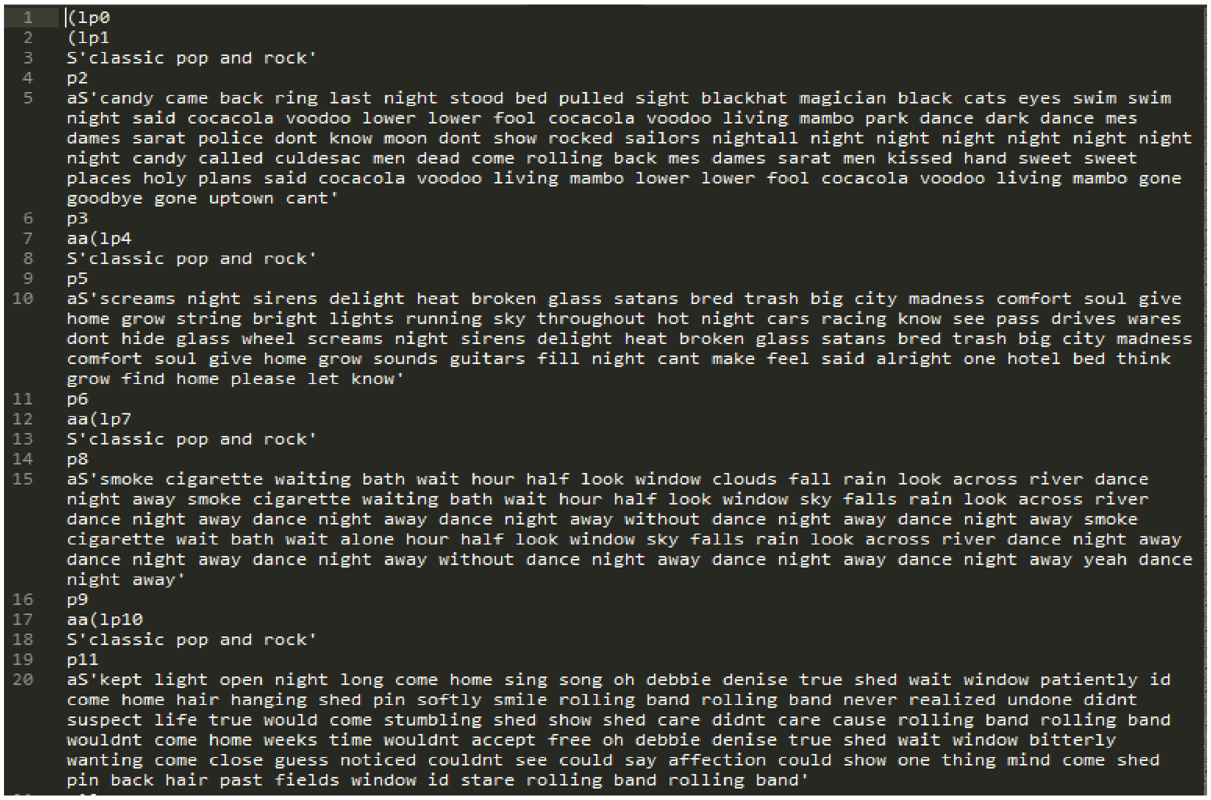
```
import time
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import RandomizedPCA
#sys.stdout = open('output.txt', 'w')

def HPOptimizationGridSearch(feature_train,label_train):
    print "Fitting the classifier to the training set"
    t0 = time.time()
    param_grid = {
             'min_samples_split': range(1,41),
              'min_samples_leaf': range(1,20),
              'n_estimators': range(1,500)
              }
    clf = GridSearchCV(OneVsRestClassifier(RandomForestClassifier()), param_grid)
    clf = clf.fit(feature_train,label_train)
    print "done in %0.3fs" % (time.time() - t0)
    print "Best estimator found by grid search:"
    print clf.best_estimator_

def processItem(datapoint):

    info=[datapoint[0]]
    for feature in datapoint[4:]:
        info.append(float(feature))
    return info

def targetFeatureSplit( data ):
    """
        given a numpy array like the one returned from
        featureFormat, separate out the first feature
        and put it into its own list (this should be the
        quantity you want to predict)

        return targets and features as separate lists

        (sklearn can generally handle both lists and numpy arrays as
        input formats when training/predicting)
    """
    target = []
    features = []
    for item in data:
        target.append( item[0] )
        features.append( item[1:] )

    return target, features

genres={}
data_list=[]
def done():
    line_num=0  #line number in text file
    with open('msd_genre_dataset.txt','r') as f:
        for line in f:
            if line_num<10:
```

```
            pass
        else:
            temp=line.split(',')
            try:
                if(genres[temp[0]] and genres[temp[0]]<2000):
                    if(temp[0]=="hip-hop"):
                        temp[0]='pop'
                    genres[temp[0]]+=1
                    data_list.append(processItem(temp))
            except:
                genres[temp[0]]=1
        line_num+=1
print "*************************************************************************************"
print "No of songs: ",line_num-10
print "*************************************************************************************"

print "Dataset composition:"
for type in genres.keys():
    #print "\""+ type+"\",",
    print type,genres[type]
print "*************************************************************************************"
print "Loading data in numpy arrays."
t=time.time()
np_data=np.array(data_list)
print "Time elapsed:",time.time()-t,"secs."
t=time.time()
scaler=MinMaxScaler()
np_data[:,1:]=scaler.fit_transform(np_data[:,1:])
print "Splitting data into target and features"
labels,features=targetFeatureSplit(np_data)
print "Time elapsed:",time.time()-t,"secs."
```

```

    print "Creating training set and Test set"
    t=time.time()
    feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)
    print "Time elapsed:",time.time()-t,"secs."
    print"Fitting and predicting the data"
    t=time.time()

    #HPOptimizationGridSearch(feature_train,label_train)

    clf=OneVsRestClassifier(RandomForestClassifier(min_samples_split=2,min_samples_leaf=1,n_estimators=300))
    clf.fit(feature_train,label_train)
    pred=clf.predict(feature_test)
    print "Time elapsed:",time.time()-t,"secs."
    print "accuracy_score=",accuracy_score(label_test,pred)
    t=time.time()
    print "confusion_matrix:"
    print(classification_report(label_test,pred))
    print "classification_report:"
    print classification_report(label_test,pred)

done()
```

For Lyrical Analysis

The technique works on the lyrics of tracks. So, based on the availabilty of songs [5], dataset for approx 24600 was built which again was baanced based on lyrics availabilty for each genre and a new dataset composition was achieved. From the balacned dataset lyrics for each track is used as feature along with the genre, so as to apply bag of words technique. The dataset overview after creating bag of words is shown in *Fig 6.*



```
(lp0
(lp1
S'classic pop and rock'
p2
aS'candy came back ring last night stood bed pulled sight blackhat magician black cats eyes swim swim
night said cocacola voodoo lower lower fool cocacola voodoo living mambo park dance dark dance mes
dames sarat police dont know moon dont show rocked sailors nightall night night night night night night
night candy called culdesac men dead come rolling back mes dames sarat men kissed hand sweet sweet
places holy plans said cocacola voodoo living mambo lower lower fool cocacola voodoo living mambo gone
goodbye gone uptown cant'
p3
aa(lp4
S'classic pop and rock'
p5
aS'screams night sirens delight heat broken glass satans bred trash big city madness comfort soul give
home grow string bright lights running sky throughout hot night cars racing know see pass drives wares
dont hide glass wheel screams night sirens delight heat broken glass satans bred trash big city madness
comfort soul give home grow sounds guitars fill night cant make feel said alright one hotel bed think
grow find home please let know'
p6
aa(lp7
S'classic pop and rock'
p8
aS'smoke cigarette waiting bath wait hour half look window clouds fall rain look across river dance
night away smoke cigarette waiting bath wait hour half look window sky falls rain look across river
dance night away dance night away dance night away without dance night away dance night away smoke
cigarette wait bath wait alone hour half look window sky falls rain look across river dance night away
dance night away dance night away without dance night away dance night away dance night away yeah dance
night away'
p9
aa(lp10
S'classic pop and rock'
p11
aS'kept light open night long come home sing song oh debbie denise true shed wait window patiently id
come home hair hanging shed pin softly smile rolling band rolling band never realized undone didnt
suspect life true would come stumbling shed show shed care didnt care cause rolling band rolling band
wouldnt come home weeks time wouldnt accept free oh debbie denise true shed wait window bitterly
wanting come close guess noticed couldnt see could say affection could show one thing mind come shed
pin back hair past fields window id stare rolling band rolling band'
```

Figure 6. Pickel of Lyrical Dataset

Code for Obtaining Lyrics for the tracks in the dataset

```
import requests
from bs4 import BeautifulSoup, Comment, NavigableString
import sys, codecs, json


def getLyrics(singer, song):
    #Replace spaces with _
    singer = singer.replace(' ', '_')
    song = song.replace(' ', '_')
    r = requests.get('http://lyrics.wikia.com/{0}:{1}'.format(singer,song))
    s = BeautifulSoup(r.text,'lxml')
    #Get main lyrics holder
    lyrics = s.find("div",{'class':'lyricbox'})
```

```
        if lyrics is None:
            #raise ValueError("Song or Singer does not exist or the API does not have Lyrics")
            return None
        #Remove Scripts
        [s.extract() for s in lyrics('script')]

        #Remove Comments
        comments = lyrics.findAll(text=lambda text:isinstance(text, Comment))
        [comment.extract() for comment in comments]

        #Remove unecessary tags
        for tag in ['div','i','b','a']:
            for match in lyrics.findAll(tag):
                match.replaceWithChildren()
        #Get output as a string and remove non unicode characters and replace <br> with newlines
        lyrics= str(lyrics).replace('\n','').replace('<br/>',' ')
        output=lyrics[22:-6:]
        try:
            return output
        except:
            return output.encode('utf-8')

genres={}
lyrics_found=0
import codecs
y=codecs.open("songLyrics1.txt","w")
line_num=0
with codecs.open('msd_genre_dataset.txt','r') as f:
    for line in f:
        if line_num<=27432:
            pass
        else:
            temp=line.split(',')
            #print "artist name=",temp[2],"Title=",temp[3]
            lyrics=getLyrics(temp[2],temp[3])
            if(lyrics!=None):
                if(lyrics.split(' ')[0]!='<span'):
                    try:
                        genres[temp[0]]+=1
                    except:
                        genres[temp[0]]=1
                    lyrics_found+=1
                    y.write(temp[0]+'**/**'+lyrics)
                    y.write('\n')
            if(line_num%100==0):
                print "At present progress\n"
                print "Songs processed: "+ str(line_num)
                print "Lyrics saved upto now:" +str(lyrics_found)
                for gen in genres.keys():
                    print gen,":",genres[gen]
        line_num+=1
y.close()
```

Code for obtaining bag of words from Pickel Data

```
from __future__ import division
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
import unicodedata
import codecs
import string
import pickle
```

```
stemmer = SnowballStemmer('english')
cachedStopWords = stopwords.words("english")
data=[]
with codecs.open('finalLyricsList2.txt','r') as f:
    lines=f.readlines()
    count=0
    for line in lines:

        labels,features=line.split('**/**')
        features = unicode(features, "utf-8")
        features = unicodedata.normalize('NFKD',features).encode('ascii','ignore')
        features=features.translate(None, string.punctuation)
        features=features.lower()
        features=[word for word in features.split() if word not in cachedStopWords]
        for word in features:
            stemmer.stem(word)
        features=" ".join(features)
        data.append([labels,features])
        count+=1
        if(count%500==0):
            print "completed:",count/len(lines)*100

    genres={}
    for line in lines:
        labels,features=line.split('**/**')
        try:
            genres[labels]+=1
        except:
            genres[labels]=0
    print "File composition"
    for g in genres.keys():
        print g+": ",genres[g]

#saving into serialized object
stem_data=open("stemmed_text",'wb')
pickle.dump(data,stem_data)
stem_data.close()



from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer()
email=[]
bow=vectorizer.fit(email)
bow=vectorizer.transform(email)
vectorizer.vocabulary_.get("great")
```

4. Machine Learning

4.1.Role of Machine Learning

Machine learning is a subpart of computer science and engineering that developed and emerged from the study of pattern recognition and computational learning theory of programmable machines in AI (Artificial Intelligence). In late 20th century, Arthur Samuel coined the term of machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed". Machine learning techniques identifies and researches the study and development of principles and theorem that can learn from and make prognostication on data. Such procedures work by building a standard from example inputs in order to make data-driven prognostication or decisions expressed as outputs, more or less than following factually stagnant program instructions.

Machine learning is meticulously related to and usually overhang with computational statistics; a branch of statistics which also focuses in foreseeing through the use of computers. It has strong connection to mathematical optimization, which delivers mechanism, concept and operation domains to the field. Machine learning procedures are deployed in a variety of calculation tasks and problems where designing, developing and programming explicit procedures is not possible. Some of the major applications of machine learning algorithms includes computer vision (object identification), spam filtering (text classification), optical character recognition (OCR), search engines (search Engine Optimization) and lots of other applications in near future. Machine learning is often amalgamated with data mining, where the latter is a branch targets more on preparatory data analysis and is known as unsupervised learning.

Within the field of data analytics, machine learning is a way that is used to develop complicated standards and procedure that lend themselves to foretelling the output. These analytical standards allows different communities of researchers, data scientists, engineers, and analysts to "produce dependable, recurring decisions and outputs" and uncover "hidden details " through learning from chronicled relationships and trends in the data.

A more vague terminology for machine learning by Tom M. Mitchell is stated as "*For some performance value P and T being the value for some class of task, any machine is said to learn from experience E, if for varying experience value E, its performance at task T improves with E for measurement parameter P"*. This statement is quite important for explaining machine learning in fundamentally operational rather than cognitive terms, thus this was followed by Alan Turing's idea in his paper "Computing Machinery and Intelligence" that the argument "Can machine think?" be exchanged by the argument "Can machines do what we can do".
In this project report we will be majorly discussing three classification algorithms for categorizing various songs according to genre and then in the latter half of the report we will be using and implementing Lyrical analysis for classification using bag of words strategy.

4.2.Classification Algorithm

4.2.1. Naïve Bayes Classifier

Naïve Bayes classifier is one of the common classifiers in machine learning that is based on Bayes theorem under the strong assumption that the features involved in the classification process are totally independent to each other's occurrence.
Naive Bayes has been under thorough study since the middle of 20th century. It was introduced under a distinct name into the text retrieval community in the latter half of 1950s, and stay a popular (basic standard) procedure for text classification, the problem of deciding documents as belonging to one category or the other (such as spam or important, sports ,fashion, politics, etc.) with word count as the features. With suitable pre-processing, it is emulous in this territory with more advanced procedures

including SVMs (Support Vector Machines). The Naïve Bayes classifier also finds application in automatic medical diagnosis that is being used widely at various popular research Institutes.

Following are some important points in favour of working with Naïve Bayes classifier:

- Naive Bayes classifiers are easy to implement.
- They are scalable to large extend, needing a number of parameters linear in the number of variables (features/predictors) in a learning problem and there by solving problems rather very fast as compared to other algorithms.
- Maximum-likelihood training with Naïve Bayes Algorithm can be done by calculating a closed-form expression (which is a mathematical expression that can be evaluated in a finite number of operations) that takes asymptotically linear time, instead of computationally expensive monotonous nearness as used for many other types of classifiers.

Naive Bayes procedures are known by many names, including simple Bayes and independence Bayes in various statistics and computer science literature. All these names give hint the regarding the use of Bayes' theorem in the classifier's decision rule, but naive Bayes is not typically a Bayesian method.

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive independence assumption that

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y),$$

For all$i$, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

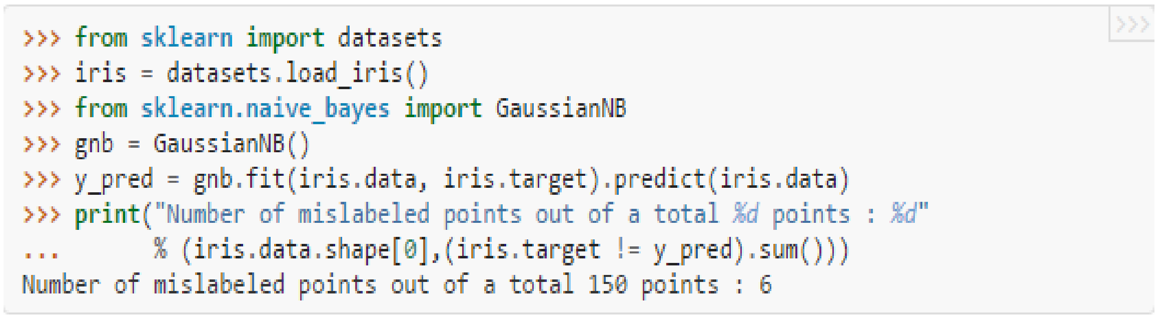
$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

And we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different types of Naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of conditional probabilities hence, there accuracy can differ drastically depending on the nature of problem at hand.

Now, we have the simple implementation of Naïve Bayes Classifier in sklearn



```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(iris.data, iris.target).predict(iris.data)
>>> print("Number of mislabeled points out of a total %d points : %d"
...       % (iris.data.shape[0],(iris.target != y_pred).sum()))
Number of mislabeled points out of a total 150 points : 6
```

In the above code,

Line Number

1. Import the dataset module from sklearn library.
2. Loads the iris dataset from the module and saves the data in numpy array iris.
3. Imports the Gaussian Naïve Bayes classifier from sklearn.naive_bayes.

4. Defines the NBclassifier.
5. Fit the data and makes prediction over it , saves the result into variable y_pred
6. Prints the Number of mislabeled points out of total points.

Hyper parameter tuning in Naïve Bayes Classifier

Now as we can see from the line number 4, there are no hyper parameters to be tuned to increase accuracy of the prediction hence, we don't have to apply cross validation schemes like GridSearchCV to find tuning parameters for the above algorithm. And for the same reason the algorithm generally has significantly lesser accuracy but is very easy to implement and is fast to run over a large Dataset like Million Song Dataset.

Actual Code Implementation

```
# importing necessary libraries
import numpy as np
import math
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
import pickle as pkl
import matplotlib.pyplot as plt
#Loading labels and features from the pickle file
f=open('features','r')
labels=pkl.load(f)
features=pkl.load(f)
f.close()
```

```
print type(labels),type(features)
#spliting data into training and testing data
print "splitting the data into training and testing set"
feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)
print "Defining the classifier"
#############################################
#Gaussian naive_bayes
#############################################
from sklearn.naive_bayes import GaussianNB
clf = OneVsRestClassifier(GaussianNB())
x=[]
y=[]
print len(feature_train)
print len(label_train)
for a in range(1,50):
   pred = clf.fit(feature_train[:len(feature_train)/a],label_train[:len(feature_train)/a]).predict(feature_test)
   x.append(len(feature_train)/a)
   y.append(accuracy_score(label_test,pred))
plt.plot(x,y)
plt.title('Gaussian Naive Bayes classifier')
plt.xlabel('No. of Training Samples')
plt.ylabel('Accuracy Score')
plt.show()
```

Executing the Algorithm

After running the algorithm over different sizes of training and testing data set we found out following trends which are plotted in the graph below with help of matplotlib.

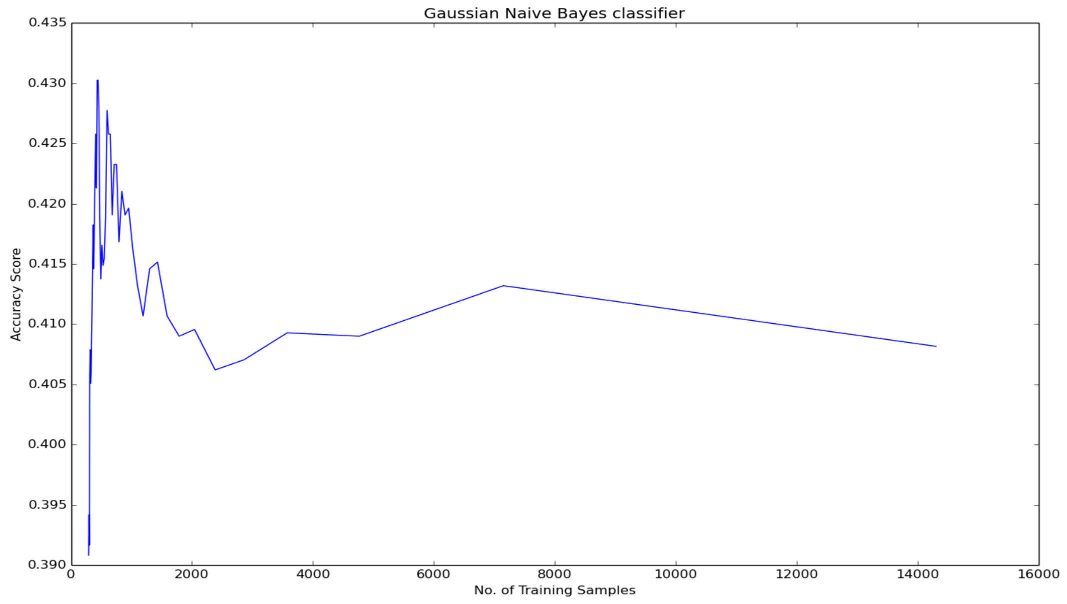


Figure 7. Gaussian Naive Bayes Classifier

Observation

It is clear from the graph that after certain size of training data the algorithm breaks means the F1-score (accuracy) gradually decreases with increase in training size. Generally it is the case that the accuracy increases with the increase in the size of training set, but this case clearly suffers from the drawback of the Naïve Bayes classifier, that is the features in the training data have high co-relation that is causing the algorithm to break.

Further Analysis

We can further analyse the results we got from the graph above. We generated the confusion matrix that is obtained.

Genre name	precision	recall	F1-score	support
classical pop and rock	0.26	0.25	0.26	389
classical	0.59	0.78	0.67	355
dance and electronica	0.39	0.35	0.37	427
folk	0.44	0.28	0.34	426
jazz and blues	0.51	0.27	0.36	386
metal	0.46	0.84	0.60	404
pop	0.29	0.45	0.36	395
punk	0.25	0.09	0.13	414
soul and reggae	0.42	0.43	0.43	381

Results:

The highest obtained F1-score for the Naïve Bayes Classifier is 0.43 or 43% at most at 750 training samples. Then, it starts to decrease with increase with the increase of the size of training set .In other words the algorithm broke after that point.

4.2.2. **Decision Tree Classifier**

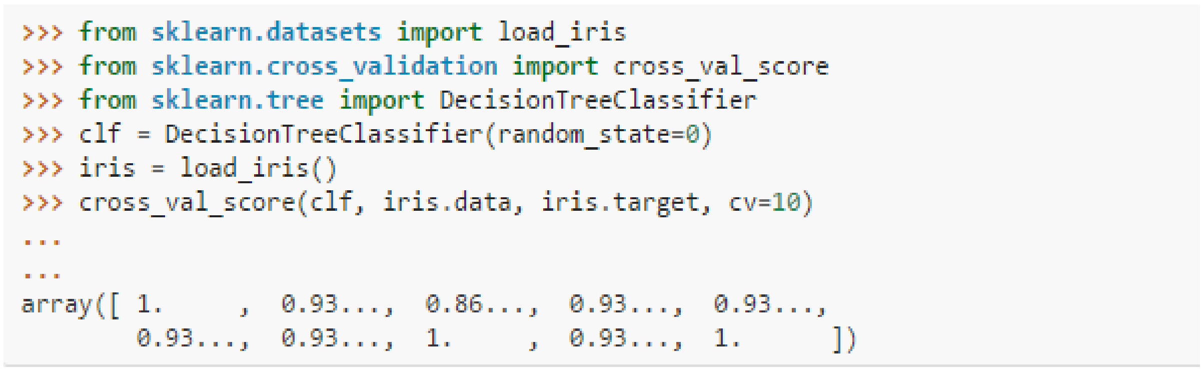
Decision Tree classifier, sometimes called as regression trees, capitalizes a decision tree as a predictive standard which drawing out observations about a feature to find about the item's labelled value. Decision tree Classifier (Tree based method) is one of the predictive modelling techniques used in tree models where the label variable can take a finite set of values and are known as classification trees. In Decision trees, what we really care about is maximizing the information gain, when we split into branches according to certain features.

Few things to keep in mind while using this algorithm is concept of entropy or purity of a branch. As we already discussed about maximizing information gain, so while building the tree we look for the features that can do the same, i.e. maximize our information gain.

Information Gain

$$\text{information gain} = \text{entropy}(\text{parent}) - \begin{bmatrix}\text{weighted}\\ \text{average}\end{bmatrix} \text{entropy}(\text{children})$$

Now, we can see simple implementation of Decision Tree Classifier in sklearn



```
>>> from sklearn.datasets import load_iris
>>> from sklearn.cross_validation import cross_val_score
>>> from sklearn.tree import DecisionTreeClassifier
>>> clf = DecisionTreeClassifier(random_state=0)
>>> iris = load_iris()
>>> cross_val_score(clf, iris.data, iris.target, cv=10)
...
...
array([ 1.     ,  0.93...,  0.86...,  0.93...,  0.93...,
        0.93...,  0.93...,  1.     ,  0.93...,  1.      ])
```

In the above code,

Line Number

1. Import the dataset module from sklearn library.
2. Imports cross validation score module from sklearn.cross_validation.
3. Imports the Decision Tree classifier from sklearn.tree.
4. Defines the DecisionTreeClassifier with random state=0, which is a seed for random number generator.
5. Loads the iris dataset.

6. Using training and test data set divided into 10 equal folds cross validation score is being calculated and can be seen in the next line.

Optimizing and tuning parameters of decision Tree

First of all, we define a dictionary (pair of key and its value) of features and their possible values and this passed as an argument to the K-fold cross validation scheme, itself computes the result for all possible combination of values of features within their respective given ranges of value in constraints specified.

Exhaustive search using K-fold cross validation and GridSearchCV, what it does is split the original dataset into k equal and separate folds and then randomly selects one fold and splits than into training and test data than generate cross validation score that is the accuracy of prediction in that fold the same process is repeated over k-1 times and all the cross validated score are taken and we take their mean. This process is repeated over the all the ordered pairs of parameters and best values are reported back by the cross validation procedure.

But as we can see that this is very tedious task both computationally and memory wise, so to overcome this problem we used RandomSearchCV.

RandomSearchCV reduces the complexity by avoid exhaustive search and focusing certain chosen values within the parameter grid.

Actual code Implementation

```
# importing neccessary libraries
import numpy as np
import math
from sklearn.cross_validation import train_test_split
from sklearn import tree
from sklearn.metrics import classification_report
import pickle as pkl
import matplotlib.pyplot as plt

#Loading labels and features from the pickle file
f=open('features','r')
labels=pkl.load(f)
features=pkl.load(f)
f.close()
print type(labels),type(features)

#spliting data into training and testing data
print "spliting the data into training and testing set"

feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)

print "Defining the classifier"
################################################
#Decision Tree
################################################

x=[]
y=[]
for min_split in range(1,400,5):
    clf = tree.DecisionTreeClassifier(min_samples_split=min_split)
    clf.fit(feature_train,label_train)
    pred=clf.predict(feature_test)
    accuracy=accuracy_score(label_test,pred)
    x.append(min_split)
    y.append(accuracy)
plt.plot(x,y)
plt.title('Min_Samples_Split  Parameter Optimization')
plt.xlabel('Value of min_samples_split for DecisionTreeClassifier ')
plt.ylabel('Cross Validated Accuracy')
plt.show()
```

Alternate method of applying Decision Tree Algorithm

```
clf = tree.DecisionTreeClassifier()
k_range=range(1,5)
param_grid=dict(min_samples_split=k_range)
grid=GridSearchCV(clf,param_grid,cv=10,scoring='accuracy')
grid.fit(features,labels)
print grid.grid_scores_

grid_mean_scores=[result.mean_validation_score for result in grid.grid_scores_]
print grid_mean_scores
plt.plot(k_range,grid_mean_scores)
plt.title('Min_Samples_Split Optimization')
plt.xlabel('value of k for KNN')
plt.ylabel('cross validated accuracy')
plt.show()
print grid.best_score_
print grid.best_params_
print grid.best_estimator_
```

Executing the Algorithm

So, for our purposes we tuned minimum sample split parameter of decision tree and obtained the following result and is shown below using graph generated by python library matplotlib. The Algorithm took significant time to execute over the dataset as the size of the dataset grew. As you can see in code rather than using GridSearchCV we have simply looped over the values for minimum sample split parameter by simple human intuition that its value can vary between 2 to 500. Minimum sample split Values above this range have underperformed drastically there for we took such constraint.

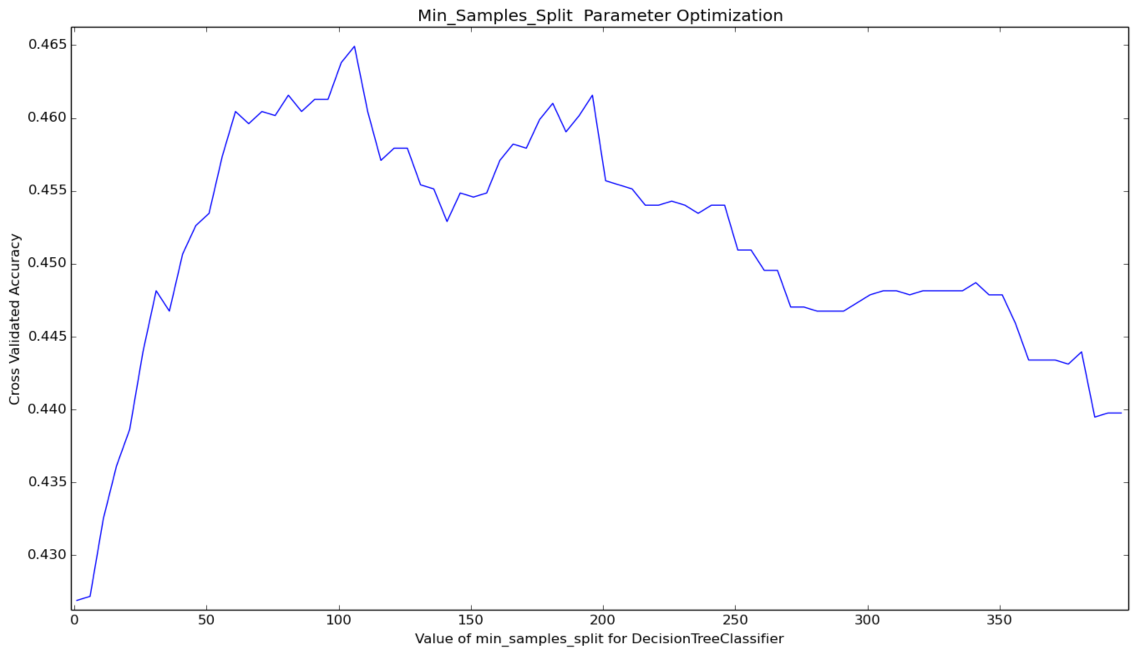


Figure 8. Decision Tree Classifier

As, it is clear from the graph the maximum accuracy obtained for classifying the genre was 0.465. It should be note that here we not actually talking about accuracy specifically but describing it in terms of F1-score. As in the case of multiclass classification it can be the case that the dataset be unbalanced, means to say that there can be very few songs of certain genre and a lot of songs of any other category this gives an unseen advantage to these classes with higher numbers and the algorithms favours those classes and works better predicting these classes.

This problem of biasing classes can be solved by two ways either changing class weights according to class frequency in the dataset or by taking only equal number of samples from all the available classes.

For our case we took the liberty of using the second scenario as it would reduce both computational time as we were running the code over our local system with strict system constraints as well as the algorithms performed better on balanced dataset.

Observation

It is clear from the graph that accuracy score (F1-score) initially with the increase of minimum_sample_split parameter value up to 100, then after it starts to degrade the accuracy score of the algorithm .Optimal value obtained somewhere near minimum_sample_split =105.

Further Analysis

For Detailed recall and precision score of various genre categories we can see the confusion matrix for the above algorithm at the optimal value of minimum_sample_split.

Genre name	precision	recall	F1-score	support
classical pop and rock	0.28	0.26	0.27	389
classical	0.68	0.72	0.70	355
dance and electronica	0.47	0.41	0.44	427
folk	0.40	0.43	0.41	426
jazz and blues	0.39	0.37	0.38	386
Metal	0.72	0.73	0.73	404
pop	0.33	0.39	0.36	395
punk	0.55	0.48	0.51	414
soul and reggae	0.37	0.40	0.38	381
avg / total	0.47	0.46	0.46	3577

Accuracy: 0.464355605256

Results

The highest obtained F1-score for the Decision Tree Classifier is 0.465 or 46% after tuning hyper parameters. Then, it starts to decrease with increase with value of min_sample_split.

4.2.3. K-nearest Neighbors Classifier

It is one of the coolest algorithm for classification and regression problems in the sense that it can accommodate to variety of problems of either type as compared to any other algorithm. Here, the value of k indicates the number of nearest k neighbors to consider to classify the given data point in a category. To every perfect thing there is always something bad in background same is the case with KNN algorithm though it is versatile in application but it is computationally very expensive.

In this classification the outcome is class label. An object is decided to categorize to a category by a majority vote of neighboring data points. If the value of K=1, then the given data point is categorized to single nearest neighbor.

This is a type of instance based learning algorithm a.k.a. lazy learning where the prediction is calculated according to local distribution of data points and till then the pre-processing and calculation is delayed until it classifies them.

Both for classification and regression problem the algorithm can be useful for assigning weights to the neighbors, so the nearer neighbors contribute more to the expectation than the distant ones.

The one of the major drawback that the algorithm sensitive to local structure of the data.

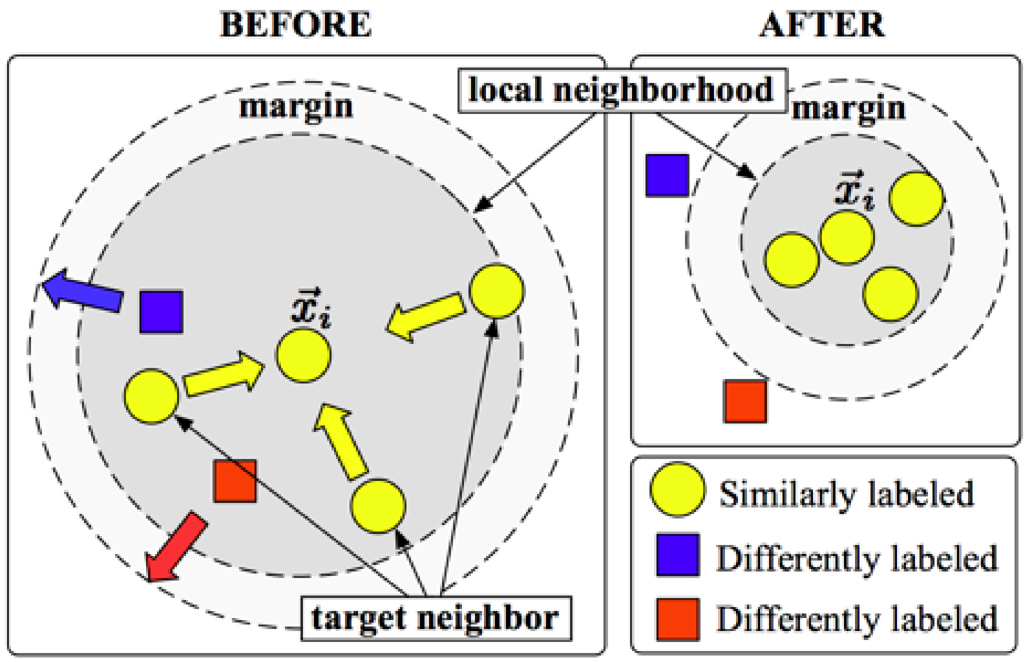


Figure 9. K Nearest Neighbor Example

Now, here example of K-NN implementation of K-nearest Neighbor Algorithm in Sklearn

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[ 0.66666667  0.33333333]]
```

In the above code,

Line Number

1. Defines X as features numpy array.
2. Defines Y as a numpy array of labels.
3. Imports the KNeighborsClassifier from sklearn.neighbors.

4. Defines classifier neigh with nearest neighbor parameter set to 3.
5. Fits the classifier over the data.
6. Prints the prediction value of neighbor with X value 1.1.
7. Prints the probability estimates for test data 0.9.

Optimizing or tuning the parameter of K-Nearest Neighbor Classifier

The parameter under consideration of tuning this time is n_neighbors again we can apply exhaustive GridSearchCV or simple iterative approach over the values of n_neighbors can do the job for us. But since the K-nearest Neighbor is computationally expensive we would prefer to have RandomizedSearchCV for tuning this parameter.

Actual Code Implementation

```
from sklearn.cross_validation import train_test_split
import time
from sklearn.grid_search import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
import pickle as pkl
import matplotlib.pyplot as plt


#Loading labels and features from the pickle file
f=open('features','r')
labels=pkl.load(f)
features=pkl.load(f)
f.close()
print type(labels),type(features)
```

```
#spliting data into training and testing data
print "spliting the data into training and testing set"
feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)

print "Defining the classifier"

###########################################
# K Nearest Neighbor Classifier
###########################################

x=[]
y=[]
for k in range(1,400,5):
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(feature_train,label_train)
    pred=clf.predict(feature_test)
    accuracy=accuracy_score(label_test,pred)
    x.append(k)
    y.append(accuracy)
    print "k :",k
plt.plot(x,y)
plt.title('KNeighnors Classifier Optimization')
plt.xlabel('Value of k for KNN')
plt.ylabel('Accuracy Score')
plt.show()
```

Executing the Algorithm

So, for our purposes we tuned n_neighbors parameter of K nearest Neighbors and obtained the following result and is shown below using graph generated by python library matplotlib. The Algorithm took significant time to execute over the dataset as the size of the dataset grew. As you can see in code rather than using GridSearchCV we have simply looped over the values for n_neighbor parameter by simple human intuition that its value can vary 1 to 400. n_neighbor values above this range have underperformed drastically there for we took such constraint.

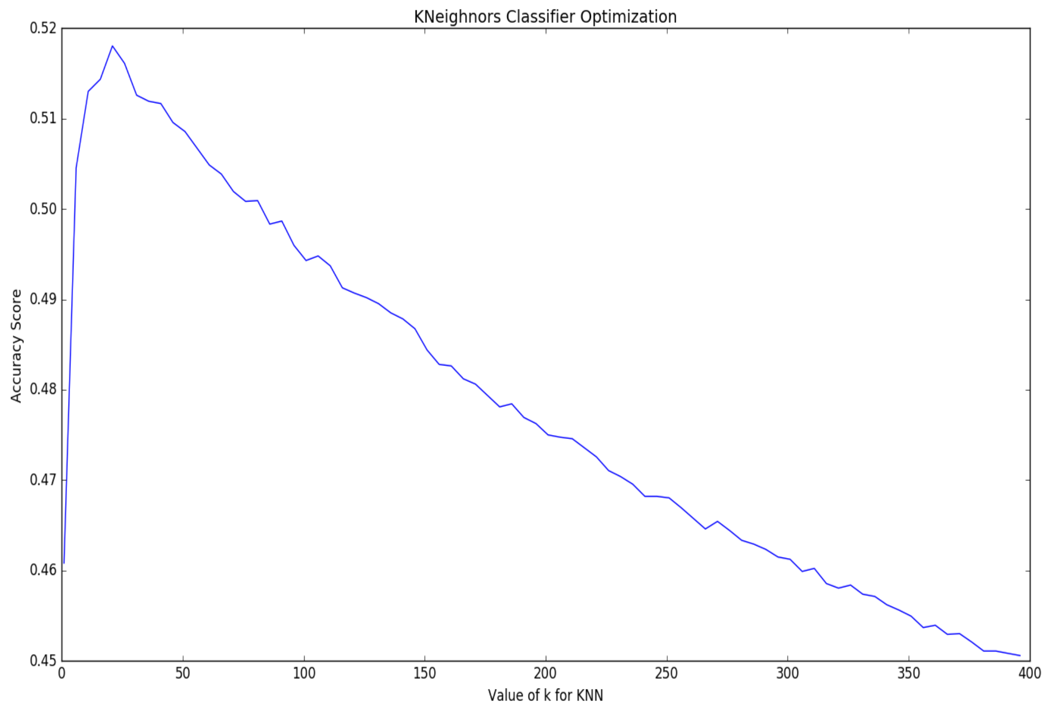


Figure 10. K Nearest Neighbor Classifier

As, it is clear from the graph the maximum accuracy obtained for classifying the genre was 0.519.

It should be note that here we not actually talking about accuracy specifically but describing it in terms of F1-score

Observation

It is clear from the graph that accuracy score (F1-score) initially with the increase of n_neighbor parameter value up to 15, then after it starts to degrade the accuracy score of the algorithm .Maximum value obtained near n_neighbors =15.

Further Analysis

For Detailed recall and precision score of various genre categories we can see the confusion matrix for the above algorithm at the optimal value of n_neighbors.

Genre name	precision	recall	F1-score	support
classical pop and rock	0.50	0.80	0.62	4742
classical	0.64	0.62	0.63	392
dance and electronica	0.59	0.19	0.29	991
folk	0.52	0.48	0.50	2666
jazz and blues	0.55	0.17	0.26	856
metal	0.66	0.47	0.55	409
pop	0.30	0.02	0.04	422
punk	0.55	0.22	0.31	644
soul and reggae	0.53	0.17	0.26	798
avg / total	0.52	0.52	0.52	11920

Accuracy: 0.515939597315

Results

The highest obtained F1-score for the Decision Tree Classifier is 0.519 or 52% after tuning hyper parameters. Then, it starts to decrease with increase with value of n_neighbors.

4.2.4. Random Forest Classifier (RFC)

It is a meta-estimator that formulates and calculates a large number of decision trees classifiers on various sub parts of the same data set and then averaging all those computed results to improve the prediction accuracy and controlling over fitting the given data set.

This classifier is an ensemble learning procedure for classification that operates on multiple decision trees. They act as saviours for decision trees preventing them from over fitting the dataset. The algorithm was developed by Leo Breiman and A. Cutler along with some features later on added independently by Ho and Amit and German to build them with control variance.

In this methodology, all the generated small trees gives some group of ill conditioned (biased) classifier and each one of them weighs certain features more than others and then we draw out final decision tree based on all those biased trees so that over all accuracy goes up for all the classes and at the same time the over fitting problem is solved.

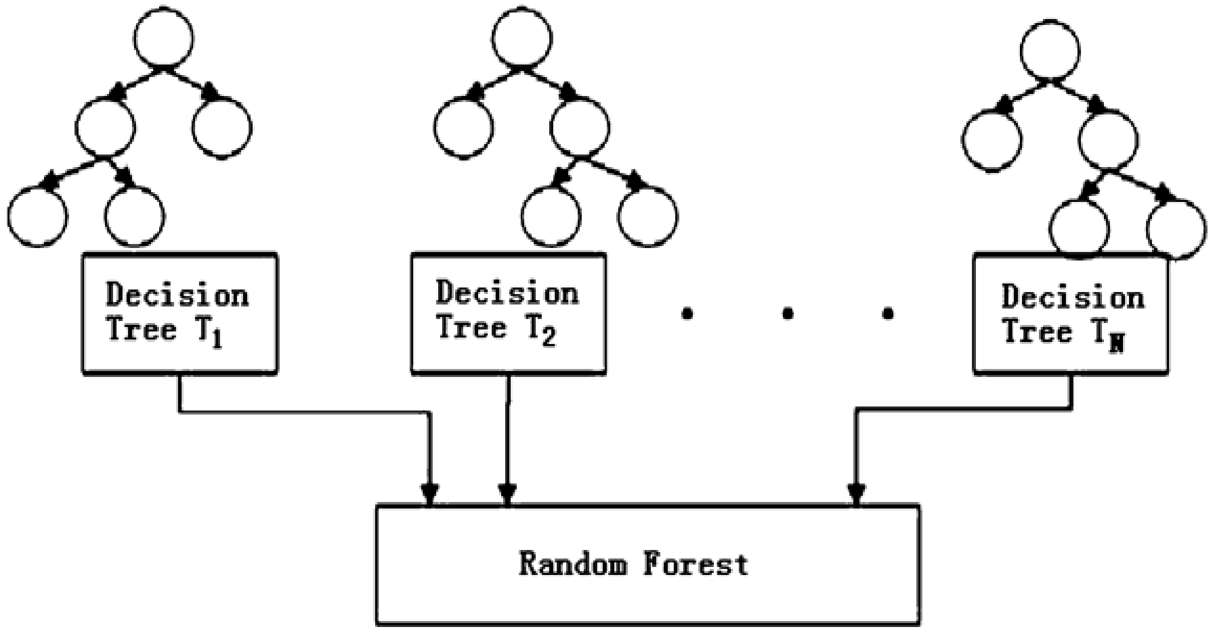


Figure 11.Random Forest Example

Some important parameter of RFC that need to be considered while fitting and making predicts are:

- n_estimators: This the number of biased decision trees to be formed while fitting the data set.
- criterion: This function determines the quality of split. It has two values “gini” for impurity measure and “entropy” for measure of information gain .By default its value is set to gini.
- max_features: This is the number of features to be considered while fitting the data. For our purpose we have used sqrt (square root (n_features, which is the number of features)) which is same as using auto.
- min_sample_split: This determines the minimum number of data points for splitting criterion.
- n_jobs: To added parallel processing if supported by the system.

Optimizing or tuning the parameter of Random Forest Classifier

Since in the case of Random forest the number of parameter which we would like to tune or optimize are many and since it is a Gaussian process what we can do is apply Bayesian Optimization to tune the parameters. If we had used RandomisedSearchCV It would have taken more than a day to complete its execution that too is not guaranteed to complete in that interval.

Actual Code Implementation

```
import numpy as np
import math
from sklearn.multiclass import OneVsRestClassifier
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pickle as pkl
import matplotlib.pyplot as plt
from sklearn.grid_search import RandomizedSearchCV
```

```
#Loading labels and features from the pickle file
f=open('features','r')
labels=pkl.load(f)
features=pkl.load(f)
f.close()
print type(labels),type(features)


#spliting data into training and testing data
print "spliting the data into training and testing set"
feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)
print "Defining the classifier"
clf=OneVsRestClassifier(RandomForestClassifier(min_samples_split=2,min_samples_leaf=1,n_estimators=300))
clf.fit(feature_train,label_train)
pred=clf.predict(feature_test)
print "confusion_matrix:"
print(classification_report(label_test,pred))
```

Executing the Algorithm

First of all, we calculated the optimal parameter values that were obtained by running Bayesian Optimization over the algorithm and after few hours and roughly 30 iterations we got the optimized values for our algorithm. As can see we have used meta-estimator that is, OneVsRestClassifier that is used for multiclass classification problem but we already know that Random forest already supports multiclass classification but it is the fact that the every time we run this algorithm all those biased trees are different every time. Therefore, we followed the OneVsRest strategy that runs this algorithm n times and predicts or fits one class versus rest of other classes.

Observation

When we ran the above code and generated the confusion matrix for the same. The result that we got was much better than what was calculated by any other algorithm. It reported a F1-Score of 0.62 or 62% Accuracy.

Further Analysis

For Detailed recall and precision score of various genre categories we can see the confusion matrix for the above algorithm at the optimal value of various parameters (min_samples_split, n_estimators, and max_features).

After Bayesian Optimization

Genre name	precision	recall	F1-score	support
classical pop and rock	0.46	0.41	0.43	387
classical	0.77	0.84	0.80	362
dance and electronica	0.63	0.62	0.62	422
folk	0.54	0.60	0.57	403
jazz and blues	0.61	0.57	0.59	390
metal	0.82	0.81	0.81	431
pop	0.47	0.54	0.50	390
punk	0.70	0.59	0.64	414
soul and reggae	0.54	0.57	0.56	384
avg / total	0.62	0.62	0.62	3583

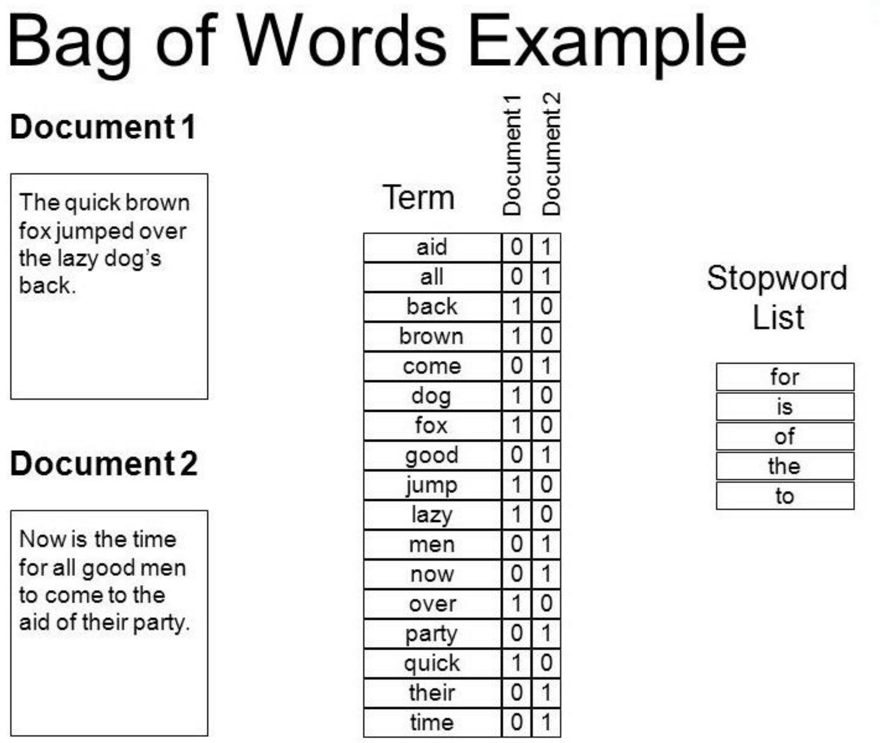
Results

The highest obtained F1-score for the Random Forest Classifier is 0.62 or 62% after optimizing hyperparameters.

4.3.**Bag of words Classification Scheme (Tf-Idf Vectorizer)**

The bag -of-words (BOW) method is simplified way of storing information of words in documents in terms of word count document wise. This scheme is used in NLP (Natural Language Processing), in this model a text (any word or sentence or document) is stored in the form of bag (multisets) of its words, irrespective of the grammar and even the word sequence but just keeping their frequency.

In recent days, BOW (Bag-of-word) have been started to be deployed for solving computer vision problems. It is commonly used for document classification, where the frequency of each word in the document is important and it is used as a feature in prediction making.



Bag of Words Example

Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Stopword List
for
is
of
the
to

Figure 12. Bag of Word Example

In this project we have implement this scheme over the lyrics of the songs that are there in the dataset and analyzed them for classification. For that we created a python script that ran over the dataset and downloaded the available lyrics from lyrics.wikia.com .Now , as the lyrics of any song are copyrighted material many of them are not available online for downloading them. Out of 60000 songs in the dataset we got lyrics of 24000 but again to balance the dataset we worked over 7500 song lyrics.

The script required used of beautifulsoup and requests library for downloading and parsing out the lyrics out of the html document. After downloading those lyrics they were preprocessed all the punctuations, UTF-8 symbols were removed.

After this step, we used NLTK (National Language Toolkit) Library to remove all the stopwords from the lyrics as they provide no information about the genres. After that we stemmed each word to its root using SnowBallStemmer so that words that almost mean the same but differ in prefixes get bagged in same bag and hence improving accuracy for predicting genre. Following this step, we used Tf-Idf (Term frequency –Inverse Document Frequency) Vectorizer to form bag-of- words.

After that we used Decision Tree classifier for to work on the generated bag of words.

Actual Code Implementation

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pickle
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
stem_data=open("stemmed_text","r")
```

```
data=pickle.load(stem_data)

feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)
vectorizer=TfidfVectorizer(sublinear_tf=True,max_df=0.5)
feature_train=vectorizer.fit_transform(feature_train)
feature_test=vectorizer.transform(feature_test).toarray()

from sklearn import tree
clf = tree.DecisionTreeClassifier(min_samples_split=23)
clf.fit(feature_train,label_train)
pred=clf.predict(feature_test)
importances = clf.feature_importances_
import numpy as np
indices = np.argsort(importances)[::-1]
print 'Feature Ranking: '
for i in range(10):
    print "{} feature no.{} ({})".format(i+1,indices[i],importances[indices[i]])
accuracy=accuracy_score(label_test,pred)
print "accuracy=",accuracy
print "confusion_matrix:"
print(classification_report(label_test,pred))
```

Observation

When we ran the above code and generated the confusion matrix for the same. The result that we got was much better than what was calculated by bag-of-word scheme used in the base paper. It reported a F1-Score of 0.51 or 51% accuracy.

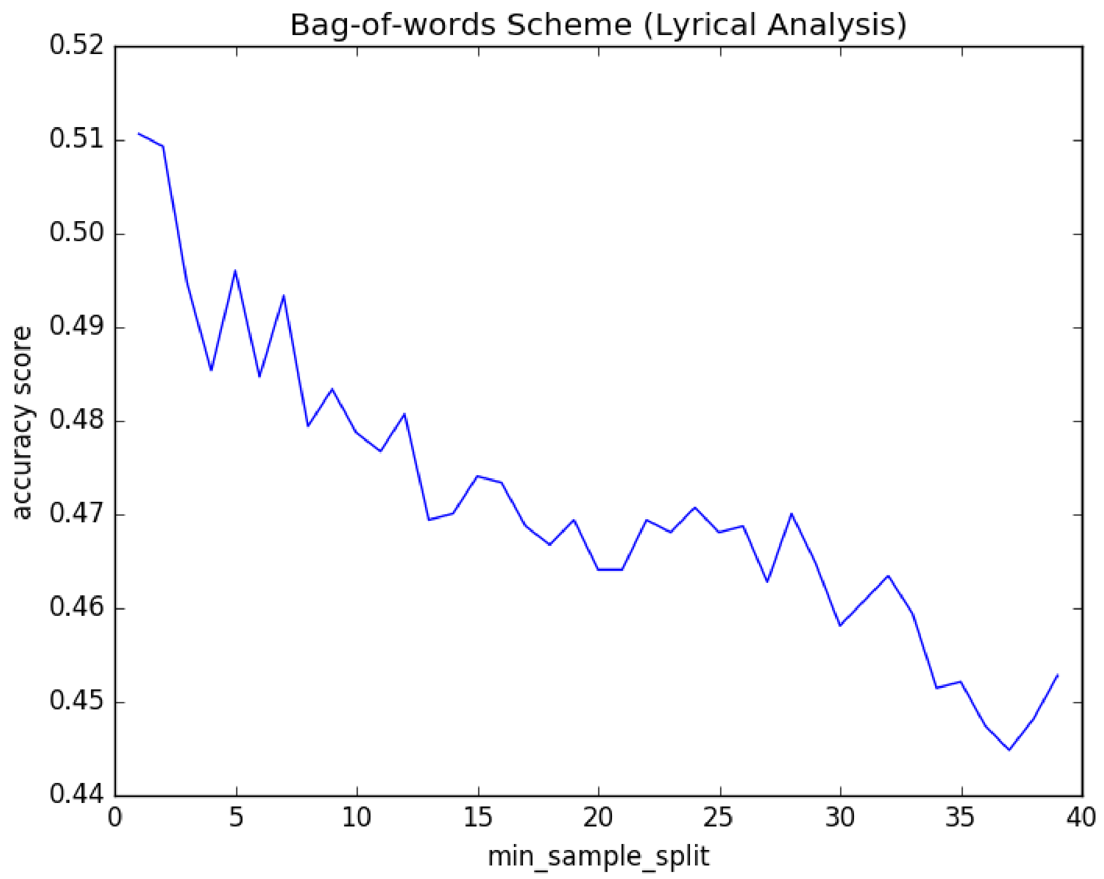


Figure 13. Bag of Word Scheme Accuracy plot

Results

The highest obtained F1-score for the Bag of words scheme with decision tree Classifier is 0.51 or 51% after optimizing hyperparameters.

5. Conclusion of Experimentation

After thorough analysis with various implementation of the problem following results were concluded.

1. Naïve Bayes classifier was very easy to implement and had very less execution time ,but due to strong co-relation between the various features the algorithm went south and unexpected results were obtained
2. Talking about accuracies that we obtained from the remaining three classifiers

Classifier Name	F1-Score
Naïve Bayes	0.43
Decision Trees	0.46
K- Nearest Neighbor	0.52
Random Forest	0.62

3. Bag of words approach went very well for the smaller dataset, its accuracy is sure to go up if larger number of songs were used.
4. Unbalanced Datasets produce underperforming results.
5. Appropriate preprocessing of the dataset is required before it can be used in machine learning Algorithms.

6. Important Codes In various phases of project

FeatureExtraction.py

This script helps in extracting out the various features and labels in the million song genre dataset and storing them in suitable format so that they are available to apply machine learning algorithms over them

```
"""
This file is extracting out the features from the million song genre dataset as per our requirement.
Making even, balanced dataset to work upon.

'''

"""
 No of songs:  59600
*************************************************************************
Dataset composition:
jazz and blues 4334
classic pop and rock 23895
classical 1874
punk 3200
metal 2103
pop 1617
dance and electronica 4935
hip-hop 434
soul and reggae 4016
folk 13192

"

```

```
"""
No of songs:  59600
****************************************************************************
Dataset composition:
jazz and blues 4334
classic pop and rock 23895
classical 1874
punk 3200
metal 2103
pop 2051
dance and electronica 4935
soul and reggae 4016
folk 13192

'''
"""
No of songs:  59600
****************************************************************************
Dataset composition:
jazz and blues:  2001
classic pop and rock:  2001
classical:  1874
punk:  2001
metal:  2001
pop:  2001
dance and electronica:  2001
soul and reggae:  2001
folk:  2001
'''

############################################################################
```

```
"""
Helper functions for pre-processing the data before creating pickle object
'''
def processdata(lines):
    features=[]
    labels=[]
    for line in lines:
        temp=line.split(',')
        labels.append(temp[0])
        l=[]
        for feature in temp[4:]:
            l.append(float(feature))
        features.append(l)
    return labels,features

def targetFeatureSplit( data ):
    """
        given a numpy array like the one returned from
        featureFormat, separate out the first feature
        and put it into its own list (this should be the
        quantity you want to predict)
        return targets and features as separate lists
        (sklearn can generally handle both lists and numpy arrays as
        input formats when training/predicting)
    """
    target = []
    features = []
    for item in data:
        target.append( item[0] )
        features.append( item[1:] )

    return target, features
##########################################################################
import numpy as np
import os
import pickle
from sklearn.preprocessing import MinMaxScaler
b=open("balanced_msd.txt",'w')
line_number=0
genres={}
with open('msd_genre_dataset.txt','r') as f:
    lines=f.readlines()
    for line in lines:
        line_number+=1
        if line_number>10:
            temp=line.split(',')
            if(temp[0]=='hip-hop'):
                temp[0]='pop'
            b.write(",".join(temp))
            try:
                genres[temp[0]]+=1
            except:
                genres[temp[0]]=1
"""
print "File composition"
for g in genres.keys():
    print g+": ",genres[g]
'''
b.close()
##############################################################################
# phase 2: balancing the dataset
##############################################################################
```

```
genres={}
b=open("balanced_msd_final.txt",'w')
with open('balanced_msd.txt','r') as f:
    lines=f.readlines()
    for line in lines:
        temp=line.split(',')
        try:
            if(genres[temp[0]]<=2000):
                genres[temp[0]]+=1
                b.write(",".join(temp))
        except:
            genres[temp[0]]=1
            b.write(",".join(temp))
try:
    os.remove('balanced_msd.txt')
except:
    print "balanced_msd.txt is missing "
print "########################################################################"
print "                         Dataset composition"
print "########################################################################"
for g in genres.keys():
    print g+": ",genres[g]
b.close()
################################################################################
"""
Formatting the data into numpy arrays ,suitable using them on the go.

Using pickle to save the serialised object for future analysis of dataset
by various algorithms
'''
################################################################################
```

```
with open('balanced_msd_final.txt','r') as f:
    lines=f.readlines()
    labels,features=processdata(lines)
np_features=np.array(features)
labels=np.array(labels)

###########################
#feature Scaling on dataset
###########################
scaler=MinMaxScaler()
np_features=scaler.fit_transform(np_features)

########################
#pickling the data
########################
#print len(features)
#print len(np_features)

f=open("features","w")
pickle.dump(labels,f)
pickle.dump(np_features,f)
f.close()

print "Successfully did preprocessing without any error"
```

BayesianOptimization.py

This code helps in Tuning the hyper parameters of random forest using Bayesian Optimization library

```
#from __future__ import print_function
from __future__ import division
```

```
rom sklearn.datasets import make_classification
from sklearn.cross_validation import cross_val_score
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.cross_validation import train_test_split
from sklearn.svm import SVC
import pickle as pkl
from bayes_opt import BayesianOptimization

f=open('features','r')
labels=pkl.load(f)
features=pkl.load(f)
f.close()
print "spliting the data into training and testing set"
feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)
print "Defining the classifier"

def rfccv(n_estimators, min_samples_split, max_features):
    return cross_val_score(RFC(n_estimators=int(n_estimators),
                               min_samples_split=int(min_samples_split),
                               max_features=min(max_features, 0.999),
                               random_state=2),
                           feature_train, label_train, 'f1_weighted', cv=5).mean()

if __name__ == "__main__":
    rfcBO = BayesianOptimization(rfccv, {'n_estimators': (250, 400),
                                         'min_samples_split': (2,15),
                                         'max_features': (0.1, 0.999)})


    print('-'*53)
    rfcBO.maximize()

   print('-'*53)
    print('Final Results')
    print('RFC: %f' % rfcBO.res['max']['max_val'])
```

Lyrics.py

The Script helps in downloading and parsing the lyrics through lyrics.wikia.com and saving them into a text file for further processing.

```
import requests
from bs4 import BeautifulSoup, Comment, NavigableString
import sys, codecs, json


def getLyrics(singer, song):
    #Replace spaces with _
    singer = singer.replace(' ', '_')
    song = song.replace(' ', '_')
    r = requests.get('http://lyrics.wikia.com/{0}:{1}'.format(singer,song))
    s = BeautifulSoup(r.text,'lxml')
    #Get main lyrics holder
    lyrics = s.find("div",{'class':'lyricbox'})
    if lyrics is None:
        #raise ValueError("Song or Singer does not exist or the API does not have Lyrics")
        return None
    #Remove Scripts
    [s.extract() for s in lyrics('script')]

    #Remove Comments
```

```
        comments = lyrics.findAll(text=lambda text:isinstance(text, Comment))
        [comment.extract() for comment in comments]

        #Remove unecessary tags
        for tag in ['div','i','b','a']:
            for match in lyrics.findAll(tag):
                match.replaceWithChildren()
        #Get output as a string and remove non unicode characters and replace <br> with newlines
        lyrics= str(lyrics).replace('\n','').replace('<br/>',' ')
        output=lyrics[22:-6:]
        try:
            return output
        except:
            return output.encode('utf-8')


genres={}
lyrics_found=0
import codecs
y=codecs.open("songLyrics1.txt","w")
line_num=0
with codecs.open('msd_genre_dataset.txt','r') as f:
    for line in f:
        if line_num<=27432:
            pass
        else:
            temp=line.split(',')
            #print "artist name=",temp[2],"Title=",temp[3]
            lyrics=getLyrics(temp[2],temp[3])
            if(lyrics!=None):
                if(lyrics.split(' ')[0]!='<span'):
```

```
          try:
            genres[temp[0]]+=1
          except:
            genres[temp[0]]=1
          lyrics_found+=1
          y.write(temp[0]+'**/**'+lyrics)
          y.write('\n')
      if(line_num%100==0):
        print "At present progress\n"
        print "Songs processed: "+ str(line_num)
        print "Lyrics saved upto now:" +str(lyrics_found)
        for gen in genres.keys():
          print gen,":",genres[gen]
    line_num+=1
y.close()
```

ProcessLyrics.py

After the lyrics have been downloaded and saved into text file, it has to be preprocessed before it, bag of words is generated and this script helps in the same.

```
from __future__ import division
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
import unicodedata
import codecs
import string
import pickle
stemmer = SnowballStemmer('english')
```

```
cachedStopWords = stopwords.words("english")
data=[]
with codecs.open('finalLyricsList2.txt','r') as f:
    lines=f.readlines()
    count=0
    for line in lines:

        labels,features=line.split('**/**')
        features = unicode(features, "utf-8")
        features = unicodedata.normalize('NFKD',features).encode('ascii','ignore')
        features=features.translate(None, string.punctuation)
        features=features.lower()
        features=[word for word in features.split() if word not in cachedStopWords]
        for word in features:
            stemmer.stem(word)
        features=" ".join(features)
        data.append([labels,features])
        count+=1
        if(count%500==0):
            print "completed:",count/len(lines)*100


    genres={}
    for line in lines:
        labels,features=line.split('**/**')
        try:
            genres[labels]+=1
        except:
            genres[labels]=0
    print "File composition"
    for g in genres.keys():
        print g+": ",genres[g]

#saving into serialized object
stem_data=open("stemmed_text",'wb')
pickle.dump(data,stem_data)
stem_data.close()
```

load_stem_data.py

Script is working with processed data to use bag of words and make prediction

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pickle
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
stem_data=open("stemmed_text","r")
data=pickle.load(stem_data)
#print data[0]
"""
b=open("finalLyricsList.txt","w")
with open('songLyrics.txt','r') as f:
    lines=f.readlines()
    for line in lines:
        labels,features=line.split('**/**')
        if(labels=='hip-hop'):
            b.write("pop**/**"+features)
        if(labels!='classical' and labels!='hip-hop'):
            b.write(line)
b.close()
```

```
b=open("finalLyricsList.txt","r")
lines=b.readlines()
genres={}
for line in lines:
    labels,features=line.split('**/**')
    try:
        genres[labels]+=1
    except:
        genres[labels]=0
print "File composition"
for g in genres.keys():
    print g+": ",genres[g]
'''

"""
 genres={}
b=open("finalLyricsList2.txt","w")
with open('finalLyricsList.txt','r') as f:
    lines=f.readlines()
    for line in lines:
        labels,features=line.split('**/**')
        try:
            genres[labels]+=1
        except:
            genres[labels]=0
        if(genres[labels]<=1000):
            b.write(line)
b.close()
b=open("finalLyricsList2.txt","r")
lines=b.readlines()
genres={}
```

```
for line in lines:
    labels,features=line.split('**/**')
    try:
        genres[labels]+=1
    except:
        genres[labels]=0
print "File composition"
for g in genres.keys():
    print g+": ",genres[g]

'''
def targetFeatureSplit( data ):
    """
        given a numpy array like the one returned from
        featureFormat, separate out the first feature
        and put it into its own list (this should be the
        quantity you want to predict)

        return targets and features as separate lists

        (sklearn can generally handle both lists and numpy arrays as
        input formats when training/predicting)
    """
    target = []
    features = []
    for item in data:
        target.append( item[0] )
        features.append( item[1] )
    return target, features

labels,features=targetFeatureSplit(data)
```

```
#print data[0]
"""

feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)
vectorizer=CountVectorizer()
train_counts = vectorizer.fit_transform(feature_train)
print train_counts.shape
print train_counts
print vectorizer.vocabulary_.get(u'la')

'''
feature_train,feature_test,label_train,label_test = train_test_split(features, labels, test_size=0.20, random_state=42)
vectorizer=TfidfVectorizer(sublinear_tf=True,max_df=0.5)
feature_train=vectorizer.fit_transform(feature_train)
feature_test=vectorizer.transform(feature_test).toarray()
from sklearn import tree
clf = tree.DecisionTreeClassifier(min_samples_split=23)
clf.fit(feature_train,label_train)
pred=clf.predict(feature_test)
importances = clf.feature_importances_
import numpy as np
indices = np.argsort(importances)[::-1]
print 'Feature Ranking: '
for i in range(10):
    print "{} feature no.{} ({})".format(i+1,indices[i],importances[indices[i]])
print vectorizer.get_feature_names()[14343]
accuracy=accuracy_score(label_test,pred)
print "accuracy=",accuracy
print "confusion_matrix:"
print(classification_report(label_test,pred))
```

7. Future scope of the project

In future both bag of words and random forest strategy can be implemented together to further improve the accuracy of the resulting classifier and then same can be used in MIR systems (Music Identification and Recommender), that can be incorporated in a music app and can improve user experience.

8. References

[1] From Classical to Hip-hop: Can Machine Learn Genre? A student publication by Aaron kravitz, Eliza lupone, Ryan Diaz

[2] Music Genre
https://en.wikipedia.org/wiki/Music_genre

[3] Analysis of the organizational and informational value of links in psychology and geology popular science hyper articles,
http://www.scielo.cl/scielo.php?pid=S071809342008000300004&script=sci_arttext

[4] MSD genre dataset." [Online]. Available:
http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/msd genre dataset.zip

[5] Lyrics Retrieving
http://lyrics.wikia.com/

[6] Introduction to Machine Learning,
https://www.udacity.com/course/intro-to-machine-learning--ud120

[7] Preprocessing data
http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing

[8] Python Libraries
http://scikit-learn.org/stable/tutorial/basic/tutorial.html