MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Hit Song Prediction

Minor Project Report
Semester VI

Submitted by:

Vivek Jain	171112305
Siddarth More	171112218
Arsude Akash Nagnath	171112290
Pranshu Kumar	171112297

Under the Guidance of
Dr. Praveen Kaushik
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Session: 2019-20

MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project report carried out on "**Hit Song Prediction**"
by the 3rd year students:

Vivek Jain	171112305
Siddharth More	171112218
Arsude Akash Nagnath	171112290
Pranshu Kumar	171112297

Have successfully completed their project in partial fulfilment of their Degree in Bachelor of Technology in Computer Science and Engineering.

Dr. Praveen Kaushik
(Minor Project Mentor)

DECLARATION

We, hereby declare that the following report which is being presented in the Minor Project Documentation Entitled as "**Hit Song Prediction**" is an authentic documentation of our own original work and to best of our knowledge. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization. Any contribution made to the research by others, with whom we have worked at Maulana Azad National Institute of Technology, Bhopal or elsewhere, is explicitly acknowledged in the report.

Vivek Jain	171112305
Siddharth More	171112218
Arsude Akash Nagnath	171112290
Pranshu Kumar	171112297

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide and coordinator Dr. Praveen Kaushik, for her valuable help and guidance. We are thankful for the encouragement that she has given us in completing this project successfully.

It is imperative for us to mention the fact that the report of minor project could not have been accomplished without the periodic suggestions and advice of our project guide Dr. Praveen Kaushik and project coordinators Dr. Dhirendra Pratap Singh and Dr. Jaytrilok Choudhary.

We are also grateful to our respected director Dr. N. S. Raghuwanshi for permitting us to utilize all the necessary facilities of the college.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind cooperation and help. Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

ABSTRACT

Exploring the possibility of predicting hit songs is both interesting from a scientific point of view and something that could be beneficial to the music industry.

In this project, we have approached the Hit Song Science problem, which aims to predict which songs will become Billboard Hot 100 hits. The Billboard Hot 100 scale is a definitive measure to define the popularity of the song. We have prepared a dataset of approximately 8000 hits and non-hits songs and extract their audio features using Spotify Web API.

We have used different Machine Learning Models such as Logistic Regression, Naïve Bayes, K Nearest Neighbour, Support Vector Machine, Random Forest Classifier, Decision Tree and Artificial Neural Networks to make predictions. We have compared the accuracies of different classifiers.

Keywords: **Machine Learning, Billboard Hot 100, Spotify, Classification.**

TABLE OF CONTENTS

1. Introduction

The Billboard Hot 100 Chart remains one of the definitive ways to measure the success of a popular song. The Billboard Hot 100 is the music industry standard record chart in the United States for songs, published weekly by Billboard magazine.

The increasing amount of digital music available online today and the progression of technology have changed the way we listen to and consume music.

We have use machine learning techniques to predict whether or not a song will become a Billboard Hot 100 hit, based on its audio features. The input to each algorithm is a series of audio features of a track. We use the algorithm to output a binary prediction of whether or not the song will feature on the Billboard Hot 100. Not only will it help determine how best to produce songs to maximize their potential for becoming a hit, it could also help decide which songs could give the greatest return for investment on advertising and publicity. Furthermore, it would help artists and music labels determine which songs are unlikely to become Billboard Hot 100 hits.

The theory behind this science is that hit songs are related in the sense that they have a set of features that make them appealing to a majority of people. These features can be processed using machine learning where the aim is to find patterns in the data. If a pattern can be found, this could be used to predict if a song will become a hit.

There are certain characteristics for hit songs, what are the largest influencers on a song's success, and even old songs can predict the popularity of new songs. To make these predictions, we have prepared the Song Dataset using Spotify's API, and used various machine learning prediction models. We have predicted and compared the accuracies of different classifiers such as Logistic Regression, Neural Networks, SVM, Random Forest, Decision Tree.

For determining the accuracies of Machine Learning Models, K fold cross validation technique is used.

2. Literature review and Survey

Prediction tasks are common in machine learning. Thus, there is already a lot of literature on predicting the popularity of songs, or predicting hit songs. These projects ask questions like: are there certain characteristics for hit songs? What has the largest influence on a song's success? Can old songs predict the popularity of new songs? Music is likely a common area of interest, as it adds another layer of complexity beyond text and natural language alone.

In the article, "Predicting Hit Songs with Machine Learning," by Minna Reiman and Philippa Ornell, they described that the models with only text features give more accurate results that with the both audio and text features. They also utilized Spotify's API. The other models used for this project were K-Nearest Neighbours and Gaussian Naive Bayes. Their most accurate model was their Gaussian Naive Bayes model, with 60.17% accuracy.

In 2008, Pachet and Roy wanted to validate the hypothesis that popularity of songs can be predicted from acoustic or human features. Their research was based on a dataset of 32000 titles and 632 features, which can be seen as a massive number of features to consider. They were not able to develop a good classification model and claimed that the popularity of a song cannot be learnt by using state-of-the-art machine learning (Pachet & Roy 2008)

However, in 2011, a study made by Ni et al. provided a more optimistic result on the problem of predicting music popularity. They had the goal of distinguishing the top 5 from the bottom 10 in a top 40 hit list. Their dataset was based on UK charts during a time period of 50 years. 5947 unique songs were collected from the Official Charts Company (OCC), and the audio features were extracted from The Echo Nest. Their results indicated that it is possible to identify hits.

In 2011, Borg and Hokkanen investigated if they could predict the popularity of a song based on its audio features and Youtube view counts. They draw the conclusion that audio features alone do not seem to be good predictors of what makes a song popular.

Another related research carried out by Herremans et al. in 2014, focuses on classification of dance hit songs. This research also extracted audio features from The Echo Nest. Some of the machine learning algorithms they used where: Decision tree, Naive Bayes, Logistic Regression and Support Vector Machine. This study showed that the popularity of dance hit songs can indeed be predicted from analysing audio features. They concluded that the overall best algorithm was Logistic Regression, with an accuracy of 0.65 and a precision of 83% (Herremans et al. 2014).

3. Proposed Work

3.1. Dataset:

To properly implement Machine Learning Algorithms, we have prepared dataset consisting of both Billboards hits and non-hits songs.

The billboard top100 hits songs are extracted from Wikipedia (Hit songs are avaible in Wikipedia in the form of HTML tables) using pandas library along with the track name, artist name and year of release. We ended up collecting a list of about 3000 hit songs which are then converted into pandas dataframe.

The second dataset is from Kaggle, an online community which allows user to find and publish variety of datasets. This dataset consisted of large amount of songs majority of which did not make it into Billboard Hot 100. The dataset included the song name, artist name, album name and the playlist.

	song_name	artist_name	album_names	playlist
0	Boulevard of Broken Dreams	Green Day	Greatest Hits: God's Favorite Band	00s Rock Anthems
1	In The End	Linkin Park	Hybrid Theory	00s Rock Anthems
2	Seven Nation Army	The White Stripes	Elephant	00s Rock Anthems
3	By The Way	Red Hot Chili Peppers	By The Way (Deluxe Version)	00s Rock Anthems
4	How You Remind Me	Nickelback	Silver Side Up	00s Rock Anthems

Then the tracks obtained from Wikipedia (Billboard Hot 100 songs) were labelled 1 and the tracks obtained from the Kaggle dataset are labelled 0 indicating that they were not able to make it in the Billboard Hot 100.

The dataframes obtained from the above two sources are then integrated into a single dataframe and then converted into a csv file.

```
combined_df.head()
```

	Artist	Title	Top100
1	Linkin Park	In The End	0
2	The White Stripes	Seven Nation Army	0
3	Red Hot Chili Peppers	By The Way	0
5	Evanescence	Bring Me To Life	0
6	Papa Roach	Last Resort	0

```
combined_df.tail()
```

	Artist	Title	Top100
14565	Daddy Yankee	Dura	1
14566	XXXTentacion	Changes	1
14567	Luke Combs	One Number Away	1
14568	Rae Sremmurd	Powerglide	1
14569	Dua Lipa	IDGAF	1

We reduced some songs labelled 0 from our datasets to maintain the balance between both classes.

3.2. Extracting Audio Features:

This section describes the audio features of the song that we have used for making predictions.
The audio features of the songs were extracted from the Spotify web API with the help of **Spotipy** python library.
Spotipy is a lightweight Python library for the Spotify Web API. With *Spotipy* we can get full access to all of the music data provided by the Spotify platform.

3.2.1 Audio Features

The Spotify API provides users with 13 audio features, of which we chose nine for our analysis: **Danceability, Energy, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Loudness, and Tempo**. The first seven features are represented as values between 0 and 1 by Spotify. Loudness is measured in decibels and tempo refers to the speed of the song in beats per minute.

- **Danceability**

  Describes how suitable a track is for dancing. This value is based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is the least danceable and a value of 1.0 is the most danceable.

- **Energy**

  A value representing a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud and noisy. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy. Energy is described as a value between 0.0 and 1.0.

- **Speechiness**

  Detects the presence of spoken words in a track. The more exclusively speech like the recording is, the closer the value is to 1.0. If the speechiness ranges between 0.66 and 1.0, the track is probably made entirely of spoken words (such as audio books, poetry etc.). Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered. Values below 0.33 most likely represent music and other non-speech-like tracks

- **Acousticness**

  A confidence measure between 0.0 and 1.0 of how acoustic a track is.

- **Instrumentalness**

Predicts whether a track contains no vocals. Sounds like ”Ooh” and ”Aah” are treated as instrumental in this context, while rap or spoken words are clearly ”vocal”. The attribute ranges between 0.0 and 1.0 and the closer to 1.0 the value is; the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks.

- **Liveness**

Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track was performed live.

- **Valence**

Describes the musical positiveness conveyed by a track. The value ranges between 0.0 and 1.0. Tracks with high valence sound more positive (happy, cheerful etc.), while tracks with low valence sound more negative (sad, depressed etc.)

- **Loudness**

The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). This value usually ranges between -60 and 0 dB.

- **Tempo**

The overall estimated tempo of a track in beats per minute (BPM). Tempo is the speed or pace of a given piece and derives directly from the average beat duration.

4. Methodology

After preparing the datasets and obtaining the audio features of the song using spotify web API, The following algorithms were used to predict the outcome.

4.1 Logistic Regression:

Logistic regression is a mathematical model which can be used to describe the relationship between one or more independent variables and one dependent variable. This model is therefore used for problems where the outcome can be classified in one of two categories. When applying the estimated logistic model to new cases of a test dataset, it provides a prediction of success probability, which is a number between 0 and 1. The logistic regression provides a rule for classifying the test data with a cut-off on the predicted success probability.

4.2 Support Vector Machines

A Support Vector Machine is a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In a 2-dimensional space this hyperplane is a line dividing a plane in two parts with one class on each side of the line. In a 3-dimensional space, this separator is instead a plane separating the different classes. The Support Vector Machine attempts to find a separating hyperplane with a margin that is as large as possible from the nearest data points.

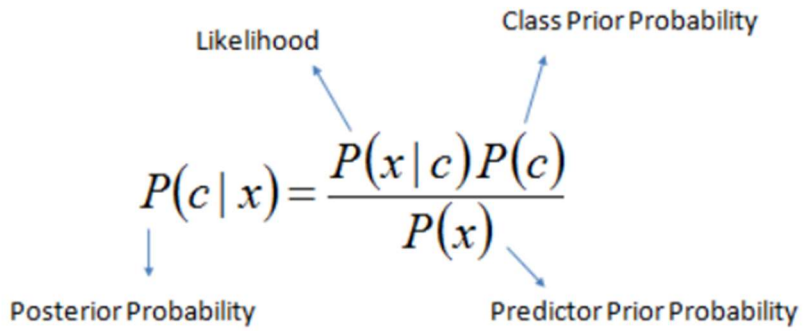
4.3 K Nearest Neighbours

K-Nearest Neighbours classification implements learning based on the K nearest neighbours of each query point. This method is a type of non-generalizing learning since it only stores instances of the training data, it does not create an internal model for generalizing the data. The classification is evaluated from a majority vote of the nearest neighbours of each query point, and each point is assigned to the class which is the most common among its neighbours. The choice of the value of k is dependent on the dataset: if k is set to a low value, the point is assigned to a class with only a few neighbours, and if k is a high value, then the effect of noise is suppressed and the classification is of a more general form.

4.4 Naïve Bayes

This is a classification technique based on an assumption of independence between predictors or what's known as *Bayes' theorem*. In simple terms, a Naïve Bayes assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Bayes theorem provides a way of calculating posterior probability **P(c|x)** from **P(c)**, **P(x)** and **P(x|c)**. The expression for Posterior Probability is as follows.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Here,

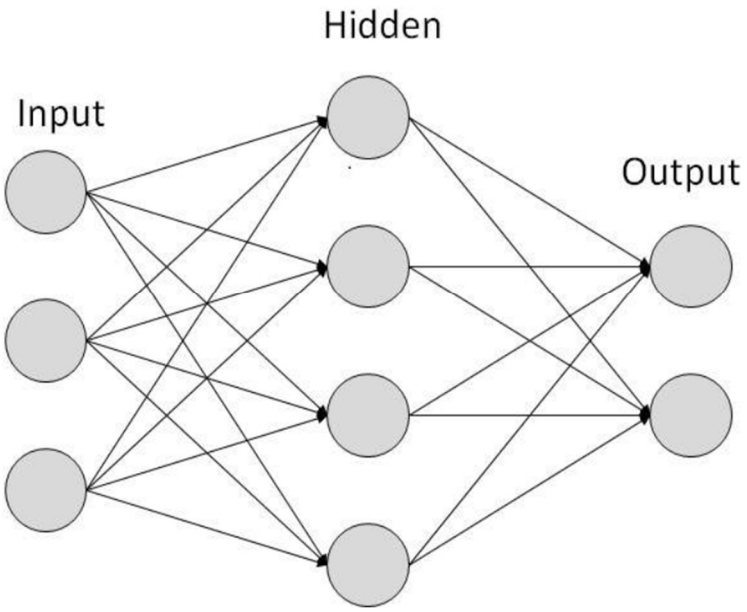
- ***P*(*c*|*x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).**
- ***P*(*c*) is the prior probability of *class*.**
- ***P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.**
- ***P*(*x*) is the prior probability of *predictor*.**

4.5 Artificial Neural Networks

Neural Networks are the biomimicry of human brain. Neural networks are multi-layer networks of neurons that we use to classify things, make predictions, etc. They are inspired by biological neural networks and the current so-called deep neural networks have proven to work quite well. Neural Networks are themselves general function approximations, which is why they can be applied to almost any machine learning problem about learning a complex mapping from the input to the output space. The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**

Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN −



The algorithm used in implementing Neural Networks is back propagation. The algorithm is as follows:-

- There are two types of outputs, model output and desired output. The difference between these two outputs is calculated which is the error. Mean Square Error is also calculated.
- The weights assigned to each input is either increased or decreased with the motive of minimizing the error.
- The model output is recomputed, error rechecked until the error cannot be minimized further. Minimum error is found by finding minima using gradient descent.

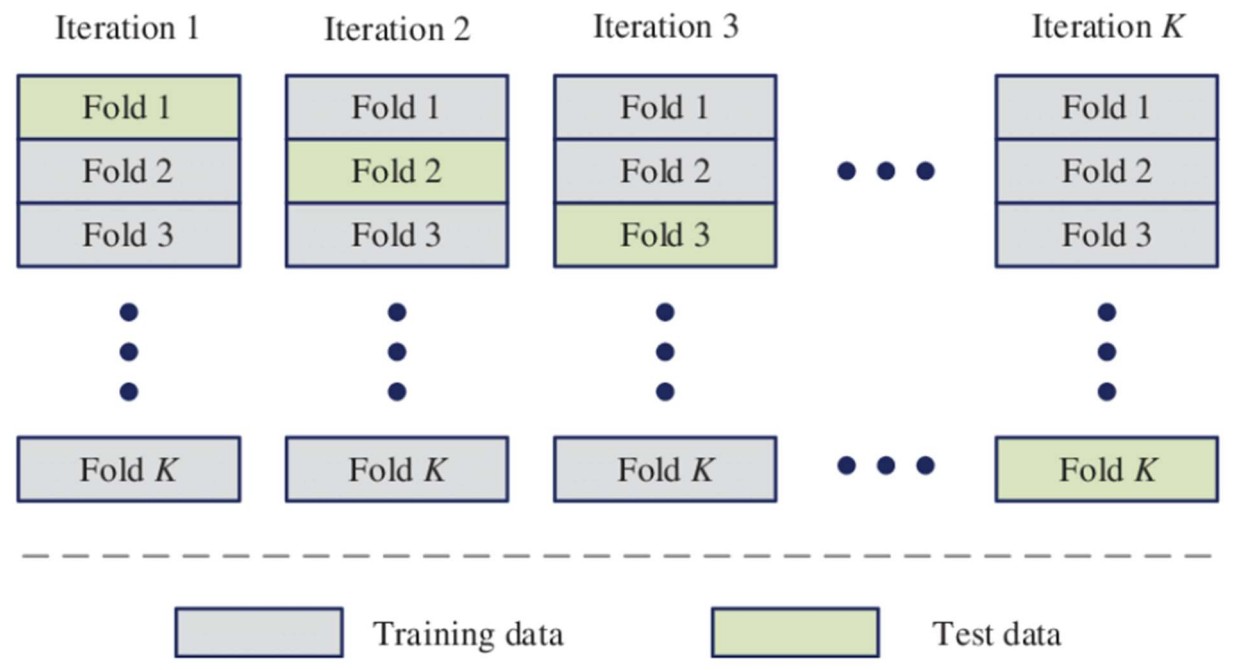
4.6K-Fold Cross Validation

After training the models, we need to evaluate the performance of the models. We split the dataset into train and test set and use the training set to train the model and the test set to test the model based on some evaluation metrics like accuracy.

This method is not reliable method for evaluating the models as the accuracy obtained for one test set might differ from another test set.

This problem is solved by using K-Fold Cross Validation. In K Fold Cross Validation, the training set is split into folds of size k and each fold is used as a test set at some point.

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



K-fold cross-validation method.

5. Tools and technologies used

5.1 Software requirement

The various tools and technologies to be used are as follows:

I) Python Libraries to implement Machine Learning Models -

• **Pandas** - pandas is a software library written for the Python programming language for data manipulation and analysis.

• **NumPy** - NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

• **Scikit learn** - Scikit is an open source Python library that implements a range of machine learning, pre-processing, cross-validation and visualization algorithms using a unified interface.

• **Spotipy -** *Spotipy* is a lightweight Python library for the Spotify Web API. With *Spotipy* you get full access to all of the music data provided by the Spotify platform.

• To build the machine learning models we used The **Jupyter Notebook**. **Jupyter Notebook** is an open-source web application that can be used to implement statistical modelling, data visualization, machine learning etc.

5.2 Hardware requirement

Working Computer system

Processor Requirement: Intel I3 core and above.

Memory Requirement: 4GB and above.

6. Implementation

The dataset was prepared as described in the Proposed Methodolgy section. Various Machine Learning Models were Applied and their accuracies were Compared.

6.1 Code Snippets

- **Billboard_data_extraction.ipynb**

  This notebook consists of codes for the preparation of dataset , extraction of audio features of songs.

In [2]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
billboard_data1= pd.DataFrame(columns=['Title', 'Artist', 'Year'])
```

In [5]:
```
years = [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011,
2014, 2015, 2016, 2017, 2018]
url_template = 'https://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles
_of_'
```

In [15]:
```
url = 'https://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_2000'
temp_df = pd.read_html(url, header=0)[0]
```

In [16]:
```
temp_df.head()
```

Out[16]:

	No.	Title	Artist(s)
0	1	"Breathe"	Faith Hill
1	2	"Smooth"	Santana featuring Rob Thomas
2	3	"Maria Maria"	Santana featuring The Product G&B
3	4	"I Wanna Know"	Joe
4	5	"Everything You Want"	Vertical Horizon

In [18]:
```
for year in years:
    url = url_template + str(year)
    temp_df = pd.read_html(url, header=0)[0]
    temp_df.columns.values[2] = "Artist"
    temp_df = temp_df.drop(temp_df.columns[0], axis=1)
    temp_df['Artist']=temp_df['Artist'].astype(str)
    temp_df.Artist = [artist.split(' featuring')[0] for artist in temp_df.Artis
t]
    temp_df.Title = [title.strip('\"') for title in temp_df.Title]
    temp_df['Year'] = year
    billboard_data1 = billboard_data1.append(temp_df)
```

In [24]:
```
billboard_data1.head()
```

Out[24]:

	Title	Artist	Year
0	Breathe	Faith Hill	2000
1	Smooth	Santana	2000
2	Maria Maria	Santana	2000
3	I Wanna Know	Joe	2000
4	Everything You Want	Vertical Horizon	2000

In [23]:
```
billboard_data1 = billboard_data1.reset_index(drop=True)
billboard_data1['Year'] = pd.to_numeric(billboard_data1['Year'])
billboard_data1['Title'] = billboard_data1['Title'].astype(str)
billboard_data1['Artist'] = billboard_data1['Artist'].astype(str)
```

In [26]:
```
billboard_data1.to_csv('C:\\Users\\DELL\\Desktop\\minor-project\\billboard_data1.csv')
```

In [64]:
```
billboard_data1=pd.read_csv('C:\\Users\\DELL\\Desktop\\minor-project\\data-files\\billboard_data1.csv',index_col=0)
billboard_data1=billboard_data1.drop('Year',axis=1)
billboard_data1 = billboard_data1.drop_duplicates(subset="Title", keep="first")
```

In [65]:
```
#Data integration
spotify_songs = pd.read_csv('C:\\Users\\DELL\\Desktop\\minor-project\\data-files\\song_info.csv')
```

In [66]:
```
spotify_songs.head()
```

Out[66]:

	song_name	artist_name	album_names	playlist
0	Boulevard of Broken Dreams	Green Day	Greatest Hits: God's Favorite Band	00s Rock Anthems
1	In The End	Linkin Park	Hybrid Theory	00s Rock Anthems
2	Seven Nation Army	The White Stripes	Elephant	00s Rock

In [67]:
```
spotify_songs = spotify_songs.drop(['album_names','playlist'], axis=1)
```

In [68]:
```
spotify_songs.head()
```

Out[68]:

	song_name	artist_name
0	Boulevard of Broken Dreams	Green Day
1	In The End	Linkin Park
2	Seven Nation Army	The White Stripes
3	By The Way	Red Hot Chili Peppers
4	How You Remind Me	Nickelback

In [69]:
```
spotify_songs.columns.values
```

Out[69]:
```
array(['song_name', 'artist_name'], dtype=object)
```

In [70]:
```
spotify_songs = spotify_songs.rename(columns={"song_name": "Title", "artist_nam
e": "Artist"})
```

In [71]:
```
spotify_songs.head()
```

Out[71]:

	Title	Artist
0	Boulevard of Broken Dreams	Green Day
1	In The End	Linkin Park
2	Seven Nation Army	The White Stripes
3	By The Way	Red Hot Chili Peppers
4	How You Remind Me	Nickelback

In [72]:
```
spotify_songs = spotify_songs.drop_duplicates(subset="Title", keep="first")
```

In [73]:
```
spotify_songs['Top100'] = 0
billboard_data1['Top100'] = 1
```

In [74]:
```
combined_df = pd.concat([spotify_songs, billboard_data1], ignore_index=True, sort=True)
```

In [75]:
```
combined_df = combined_df.drop_duplicates(subset="Title", keep="last")
```

In [76]:
```
combined_df.head()
```

Out[76]:

	Artist	Title	Top100
1	Linkin Park	In The End	0
2	The White Stripes	Seven Nation Army	0
3	Red Hot Chili Peppers	By The Way	0
5	Evanescence	Bring Me To Life	0
6	Papa Roach	Last Resort	0

In [77]:
```
combined_df.tail()
```

Out[77]:

	Artist	Title	Top100
14565	Daddy Yankee	Dura	1
14566	XXXTentacion	Changes	1
14567	Luke Combs	One Number Away	1
14568	Rae Sremmurd	Powerglide	1
14569	Dua Lipa	IDGAF	1

In [78]:
```
billboard_data1.head()
```

Out[78]:

	Title	Artist	Top100
0	Breathe	Faith Hill	1
1	Smooth	Santana	1
2	Maria Maria	Santana	1
3	I Wanna Know	Joe	1
4	Everything You Want	Vertical Horizon	1

In [80]:
```
combined_df.to_csv('C:\\Users\\DELL\\Desktop\\minor-project\\data-files\\songs-list.csv')
```

In [81]:
```
#Getting songs uris
combined_df.shape
```

Out[81]:
```
(13993, 3)
```

In [1]:
```
import spotipy
import spotipy.util as util
from tqdm import tqdm_notebook
```

In [105]:
```
def get_spotify_uri(title, artist):
    query = title + " " + artist
    search = sp.search(q=query, limit=50, offset=0, type='track', market='US')
    search_items = search['tracks']['items']
    for i in range(len(search_items)):
            uri = search_items[i]['id']
            return uri
    return 0
```

In [106]:
```
token = util.oauth2.SpotifyClientCredentials(client_id='b7fc404a069d4a0f83049834a7d09a25', client_secret='990a5a3792e9469fac00c288ec610d7d')
cache_token = token.get_access_token()
spotify = spotipy.Spotify(cache_token)
sp = spotipy.Spotify(auth=cache_token)
```

In [112]:
```
titles = list(combined_df.Title)
artists = list(combined_df.Artist)
top100 = list(combined_df.Top100)
spotify_uri = list()
errors = list()
```

In [114]:
```
temp1 = list()
```

In [115]:
```
for i in tqdm_notebook(range(0,13993)):
    uri = get_spotify_uri(titles[i], artists[i])

    if uri != 0:
        temp1.append(uri)
    else:
        temp1.append(uri)
```

```
        errors.append(1)
spotify_uri = spotify_uri + temp1
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: TqdmDeprecat
ionWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  if __name__ == '__main__':
```

In [123]:
```
len(spotify_uri)
```

Out[123]:
```
13993
```

In [124]:
```
songs_with_id = pd.DataFrame()
```

In [126]:
```
songs_with_id['Title'] = titles;
songs_with_id['Artist'] = artists;
songs_with_id['Top100'] = top100;
songs_with_id['URI'] = spotify_uri;
songs_with_id=songs_with_id[songs_with_id.URI!=0]
```

In [127]:
```
songs_with_id.head()
```

Out[127]:

	Title	Artist	Top100	URI
0	In The End	Linkin Park	0	7KSjlzDJvwAG0utAYE9Vvg
1	Seven Nation Army	The White Stripes	0	7i6r9KotUPQg3ozKKgEPIN
2	By The Way	Red Hot Chili Peppers	0	3ZOEytgrvLwQaqXreDs2Jx
3	Bring Me To Life	Evanescence	0	0COqiPhxzoWICwFCS4eZcp
4	Last Resort	Papa Roach	0	5W8YXBz9MTIDyrpYaCg2Ky

In [128]:
```
songs_with_id.tail()
```

Out[128]:

	Title	Artist	Top100	URI
13988	Dura	Daddy Yankee	1	6KuqAtoeVzxAYOaMveLNpH
13989	Changes	XXXTentacion	1	7AFASza1mXqntmGtbxXprO
13990	One Number Away	Luke Combs	1	4qPZUkYUhJVJ5REQivmaq4

In [131]:
```
songs_with_id.shape
```

Out[131]:
```
(13507, 4)
```

In [133]:
```
songs_with_id.reset_index(drop=True,inplace=True)
```

In [134]:
```
songs_with_id.tail()
```

Out[134]:

	Title	Artist	Top100	URI
13502	Dura	Daddy Yankee	1	6KuqAtoeVzxAYOaMveLNpH
13503	Changes	XXXTentacion	1	7AFASza1mXqntmGtbxXprO
13504	One Number Away	Luke Combs	1	4gB7HrYHbJVJ5RFOjxmoq4
13505	Powerglide	Rae Sremmurd	1	1BuZAIO8WZpavWVbbq3Lci
13506	IDGAF	Dua Lipa	1	1f7HfxcJ151yN485mZhhWn

In [135]:
```
songs_with_id.to_csv('C:\\Users\\DELL\\Desktop\\minor-project\\data-files\\songs-with-id.csv')
```

In [136]:
```
#Getting song features
uris = list(songs_with_id.URI)
danceability_list = list()
energy_list = list()
key_list = list()
loudness_list = list()
mode_list = list()
speechiness_list = list()
acousticness_list = list()
instrumentalness_list = list()
liveness_list = list()
valence_list = list()
tempo_list = list()
duration_list = list()
time_signature_list= list()
```

In [137]:
```
def get_audio_features(uri):
    search = sp.audio_features(uri)
```

```
time_signature_list= list()
```

[137]:
```
def get_audio_features(uri):
    search = sp.audio_features(uri)
    if search[0] == None:
        danceability_list.append(np.nan)
        energy_list.append(np.nan)
        key_list.append(np.nan)
        loudness_list.append(np.nan)
        mode_list.append(np.nan)
        speechiness_list.append(np.nan)
        acousticness_list.append(np.nan)
        instrumentalness_list.append(np.nan)
        liveness_list.append(np.nan)
        valence_list.append(np.nan)
        tempo_list.append(np.nan)
        duration_list.append(np.nan)
        time_signature_list.append(np.nan)
        return ('Error on: ' + str(uri))

    search_list = search[0]

    danceability_list.append(search_list['danceability'])
    energy_list.append(search_list['energy'])
    key_list.append(search_list['key'])
    loudness_list.append(search_list['loudness'])
    mode_list.append(search_list['mode'])
    speechiness_list.append(search_list['speechiness'])
    acousticness_list.append(search_list['acousticness'])
    instrumentalness_list.append(search_list['instrumentalness'])
    liveness_list.append(search_list['liveness'])
    valence_list.append(search_list['valence'])
    tempo_list.append(search_list['tempo'])
    duration_list.append(search_list['duration_ms'])
    time_signature_list.append(search_list['time_signature'])
```

[139]:
```
token = util.oauth2.SpotifyClientCredentials(client_id='b7fc404a069d4a0f830
49834a7d09a25', client_secret='990a5a3792e9469fac00c288ec610d7d')
cache_token = token.get_access_token()
spotify = spotipy.Spotify(cache_token)
sp = spotipy.Spotify(auth=cache_token)
```

In [140]:
```
for i in tqdm_notebook(range(len(uris))):
    get_audio_features(uris[i])
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: TqdmDepr
ecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  if __name__ == '__main__':
```

In [141]:
```
songs_with_id['Danceability'] = danceability_list
songs_with_id['Energy'] = energy_list
songs_with_id['Key'] = key_list
songs_with_id['Loudness'] = loudness_list
songs_with_id['Mode'] = mode_list
songs_with_id['Speechiness'] = speechiness_list
songs_with_id['Acousticness'] = acousticness_list
songs_with_id['Instrumentalness'] = instrumentalness_list
songs_with_id['Liveness'] = liveness_list
songs_with_id['Valence'] = valence_list
songs_with_id['Tempo'] = tempo_list
songs_with_id['Duration'] = duration_list
songs_with_id['Time_Signature'] = time_signature_list
```

In [142]:
```
songs_with_id.dropna()
```

Out[142]:

Title	Artist	Top100	URI	Danceability	Energy	Key	Lou
In The End	Linkin Park	0	7KSjlzDJvwAG0utAYE9Vvg	0.476	0.632	9	
Seven Nation Army	The White Stripes	0	7i6r9KotUPQg3ozKKgEPIN	0.737	0.463	0	
By The Way	Red Hot Chili Peppers	0	3ZOEytgrvLwQaqXreDs2Jx	0.618	0.938	9	
Bring Me To Life	Evanescence	0	0COqiPhxzoWICwFCS4eZcp	0.331	0.943	4	
Last Resort	Papa Roach	0	5W8YXBz9MTIDyrpYaCg2Ky	0.589	0.890	4	
Sex on Fire	Kings of Leon	0	0ntQJM78wzOLVeCUAW7Y45	0.542	0.905	9	
Smooth Criminal	Alien Ant Farm	0	5z6xHjCZr7a7AIcy8sPBKy	0.653	0.964	9	

In [143]:
```
songs_with_id.head()
```

Out[143]:

	Title	Artist	Top100	URI	Danceability	Energy	Key	Loudr
0	In The End	Linkin Park	0	7KSjlzDJvwAG0utAYE9Vvg	0.476	0.632	9	-6
1	Seven Nation Army	The White Stripes	0	7i6r9KotUPQg3ozKKgEPIN	0.737	0.463	0	-7
2	By The Way	Red Hot Chili Peppers	0	3ZOEytgrvLwQaqXreDs2Jx	0.618	0.938	9	-3
3	Bring Me To Life	Evanescence	0	0COqiPhxzoWICwFCS4eZcp	0.331	0.943	4	-3
4	Last Resort	Papa Roach	0	5W8YXBz9MTIDyrpYaCg2Ky	0.589	0.890	4	-3

In [144]:
```
songs_with_id.tail()
```

Out[144]:

	Title	Artist	Top100	URI	Danceability	Energy	Ke
13502	Dura	Daddy Yankee	1	6KuqAtoeVzxAYOaMveLNpH	0.783	0.840	
13503	Changes	XXXTentacion	1	7AFASza1mXqntmGtbxXprO	0.669	0.308	1
13504	One Number Away	Luke Combs	1	4gB7HrYHbJVJ5RFOjxmoq4	0.544	0.781	
13505	Powerglide	Rae Sremmurd	1	1BuZAIO8WZpavWVbbq3Lci	0.713	0.831	
13506	IDGAF	Dua Lipa	1	1f7HfxcJ151yN485mZhhWn	0.711	0.551	

In [145]:
```
songs_with_id.to_csv('C:\\Users\\DELL\\Desktop\\minor-project\\data-files
\\songs-with-features.csv')
```

- **Classification_models.ipynb**

  This notebook contains the implementation of various Classification algorithms on dataset.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
#Reading dataset
dataset = pd.read_csv('C:\\Users\\DELL\\Desktop\\ASSIGNMENTS\\minor-project\\dat
a-files\\appended_songs.csv')
dataset = dataset.drop(dataset.columns[0],axis = 1)
```

In [5]:
```
dataset.head()
```

Out[5]:

	Artist	Top100	URI	Danceability	Energy	Key	Loudness	Mode	S
	Myles Cameron	0	3MPcfaxPTQPR3w58qaODvX	0.575	0.352	1	-13.813	1	
	RealLiveAnimals	0	01pmPw3Tfm08YF1rRdTcv6	0.742	0.301	2	-10.652	0	
	Souly Had	0	3nPxnMPQG3J8ndJqAetZya	0.458	0.502	1	-7.483	1	
	Amilli	0	4DU9lFHzSh1M52pZEeBEwb	0.817	0.482	6	-6.715	1	
	Denyah	0	3hiaRDT2SmsR8aYl93MqgT	0.829	0.408	9	-10.172	1	

In [6]:
```
#Spilling dataset into features and target variable
X = dataset.iloc[:, 4:15].values
y = dataset.iloc[:, 2].values
```

In [7]:
```
X
```

Out[7]:
```
array([[5.75000e-01, 3.52000e-01, 1.00000e+00, ..., 9.74000e-02,
        4.47000e-01, 1.00023e+02],
       [7.42000e-01, 3.01000e-01, 2.00000e+00, ..., 1.08000e-01,
        2.68000e-01, 1.16914e+02],
       [4.58000e-01, 5.02000e-01, 1.00000e+00, ..., 1.02000e-01,
        6.12000e-01, 1.74098e+02],
```

In [9]:
```
#Splitting the dataset into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, rando
m_state = 0)
```

In [10]:
```
#Performing feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [12]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
classifierLR = LogisticRegression(random_state = 0)
classifierLR.fit(X_train, y_train)
```

Out[12]:
```
LogisticRegression(random_state=0)
```

In [14]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifierLR,X = X_train ,y = y_train,
cv = 10)
```

In [15]:
```
accuracies.mean()
```

Out[15]:
```
0.6583496487389151
```

In [16]:
```
accuracies.std()
```

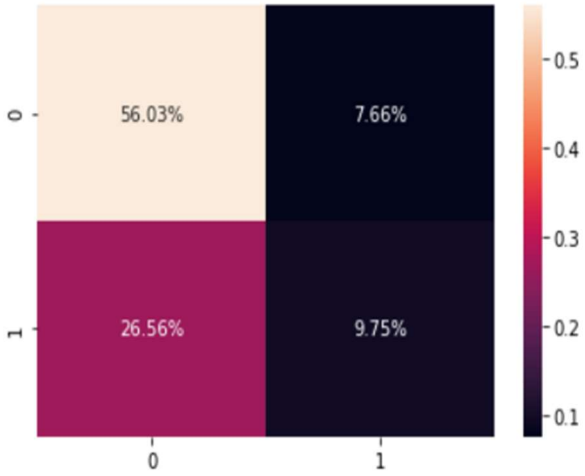
Out[16]:
```
0.01409667132414898
```

In [17]:
```
y_pred_LR = classifierLR.predict(X_test)
```

In [18]:
```
from sklearn.metrics import confusion_matrix
cm_LR = confusion_matrix(y_test, y_pred_LR)
```

In [20]:
```
import seaborn as sns
sns.heatmap(cm_LR/np.sum(cm_LR), annot=True, fmt='.2%')
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2399cb36c48>
```



In [21]:
```
#Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
classifierNB = GaussianNB()
classifierNB.fit(X_train, y_train)
```

Out[21]:
```
GaussianNB()
```

In [22]:
```
y_pred_NB = classifierNB.predict(X_test)
```

In [23]:
```
cm_NB = confusion_matrix(y_test, y_pred_NB)
```

In [29]:
```
accuraciesNB = cross_val_score(estimator = classifierNB,X = X_train ,y = y_trai
n, cv = 10)
```

In [30]:
```
accuraciesNB.mean()
```

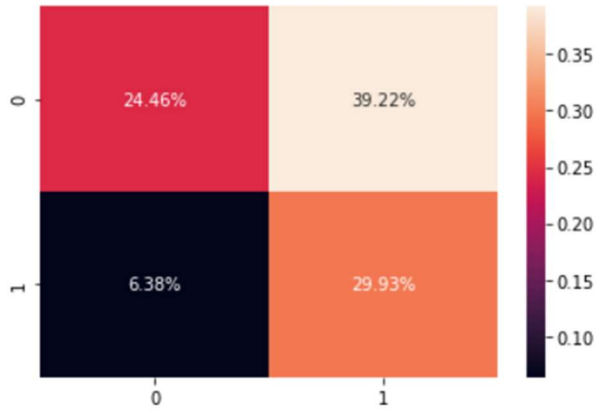
Out[30]:
```
0.5319685976429037
```

In [31]:
```
accuraciesNB.std()
```

Out[31]:
```
0.024186358712161675
```

In [32]:
```
sns.heatmap(cm_NB/np.sum(cm_NB), annot=True, fmt='.2%')
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x2399ce07048>



In [33]:
```
#KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
classifierKNN = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
2)
classifierKNN.fit(X_train, y_train)
```

Out[33]: KNeighborsClassifier()

In [34]:
```
y_pred_KNN = classifierKNN.predict(X_test)
```

In [35]:
```
accuraciesKNN = cross_val_score(estimator = classifierKNN,X = X_train ,y = y_tra
in, cv = 10)
```

In [36]:
```
accuraciesKNN.mean()
```

Out[36]: 0.6296609274828209

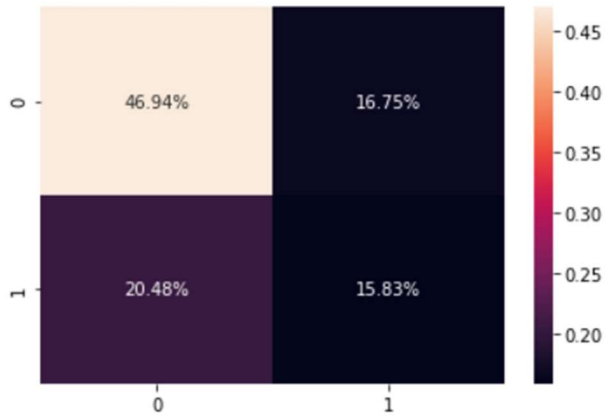
In [37]:
```
accuraciesKNN.std()
```

Out[37]: 0.01679633778135714

In [38]:
```
cm_KNN = confusion_matrix(y_test, y_pred_KNN)
```

In [39]:
```
sns.heatmap(cm_KNN/np.sum(cm_KNN), annot=True, fmt='.2%')
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2399cecc708>
```



In [40]:
```
#Decesion Tree Classification
from sklearn.tree import DecisionTreeClassifier
classifierDT = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifierDT.fit(X_train, y_train)
```

Out[40]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [41]:
```
y_pred_DT = classifierDT.predict(X_test)
```

In [42]:
```
accuraciesDT = cross_val_score(estimator = classifierDT,X = X_train ,y = y_trai
n, cv = 10)
```

In [43]:
```
accuraciesDT.mean()
```

Out[43]:
```
0.6270269492111022
```

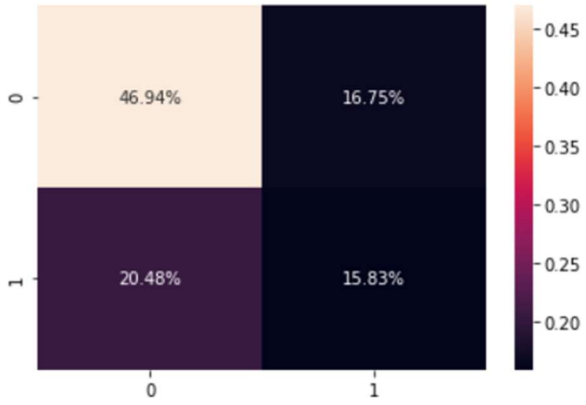
In [44]:
```
accuraciesDT .std()
```

Out[44]:
```
0.019938334557703608
```

In [46]:
```
cm DT = confusion matrix(y test, y pred DT)
```

In [47]:
```
sns.heatmap(cm_KNN/np.sum(cm_KNN), annot=True, fmt='.2%')
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2399cf8ce48>
```



In [48]:
```
from sklearn.ensemble import RandomForestClassifier
classifierRF = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
random_state = 0)
classifierRF.fit(X_train, y_train)
```

Out[48]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [49]:
```
y_pred_RF = classifierRF.predict(X_test)
```

In [50]:
```
accuraciesRF = cross_val_score(estimator = classifierRF,X = X_train ,y = y_trai
n, cv = 10)
```

In [51]:
```
accuraciesRF.mean()
```

Out[51]:
```
0.6728070175438596
```

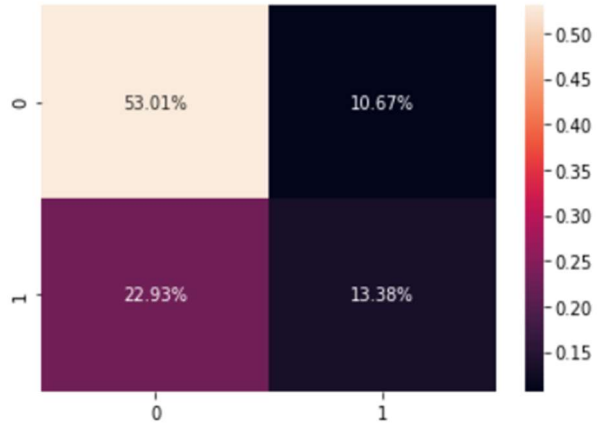
In [52]:
```
accuraciesRF.std()
```

Out[52]:
```
0.022186100466937583
```

In [53]:
```
cm_RF = confusion_matrix(y_test, y_pred_RF)
```

In [54]: sns.heatmap(cm_RF/np.sum(cm_RF), annot=True, fmt= '.2%')

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x2399d122888>



In [55]:
```
from sklearn.svm import SVC
classifierSVM = SVC(kernel = 'linear', random_state = 0)
classifierSVM.fit(X_train, y_train)
```

Out[55]: SVC(kernel='linear', random_state=0)

In [56]:
```
y_pred_SVM = classifierSVM.predict(X_test)
```

In [57]:
```
accuraciesSVM = cross_val_score(estimator = classifierSVM,X = X_train ,y = y_tra
in, cv = 10)
```

In [58]:
```
accuraciesSVM.mean()
```

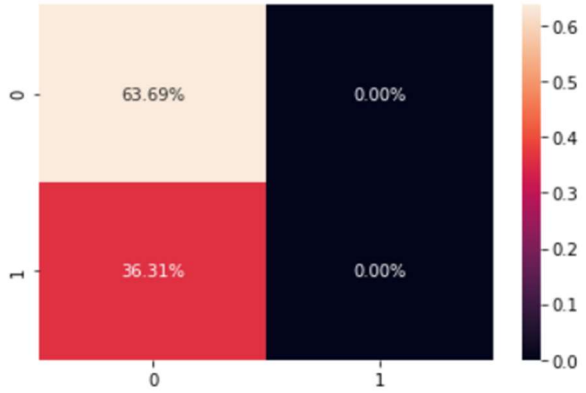
Out[58]: 0.6395101539406504

In [59]:
```
accuraciesSVM.std()
```

Out[59]: 0.0006864476610356235

In [60]:
```
cm_SVM = confusion_matrix(y_test, y_pred_SVM)
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x2399d1c6ac8>



In [62]:
```
#ANN
import keras
from keras.models import Sequential
from keras.layers import Dense
classifierANN = Sequential()

classifierANN.add(Dense(6,input_dim = 11,activation = 'relu' ))
classifierANN.add(Dense(6,activation = 'relu'))
classifierANN.add(Dense(1,activation = 'sigmoid'))
```

In [63]:
```
classifierANN.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [64]:
```
history = classifierANN.fit(X_train, y_train, epochs=100, batch_size=64)
```

```
Epoch 1/100
72/72 [==============================] - 0s 5ms/step - loss: 0.6534 - accuracy:
0.6196
Epoch 2/100
72/72 [==============================] - 0s 2ms/step - loss: 0.6337 - accuracy:
0.6380
Epoch 3/100
72/72 [==============================] - 0s 4ms/step - loss: 0.6241 - accuracy:
```

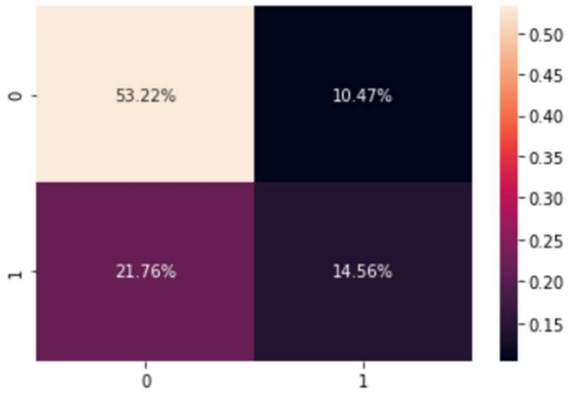
```
0.6975
Epoch 97/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5623 - accuracy:
0.6991
Epoch 98/100
72/72 [==============================] - 0s 4ms/step - loss: 0.5620 - accuracy:
0.6982
Epoch 99/100
72/72 [==============================] - 0s 4ms/step - loss: 0.5618 - accuracy:
0.6982
Epoch 100/100
72/72 [==============================] - 0s 5ms/step - loss: 0.5619 - accuracy:
0.7008
```

In [66]:
```
y_pred_ANN = classifierANN.predict(X_test)
y_pred_ANN = (y_pred_ANN > 0.5)
```

In [67]:
```
cm_ANN = confusion_matrix(y_test, y_pred_ANN)
```

In [68]:
```
sns.heatmap(cm_ANN/np.sum(cm_ANN), annot=True, fmt='.2%')
```

Out[68]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x239a4ba8808>
```



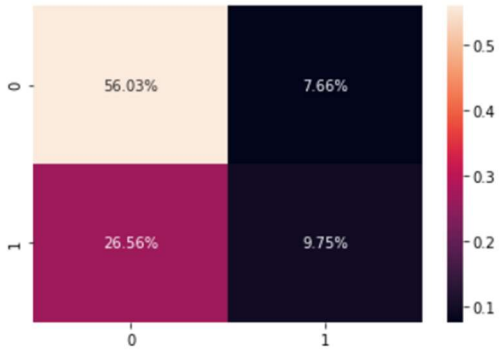
7. Results and Outcomes

We calculated accuracies and draw the confusion matrix for each of the algorithms and the following insights were obtained.

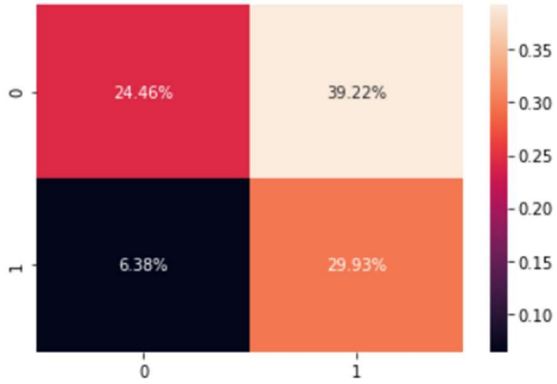
The mean of accuracies obtained after 10-fold cross validation is summarized in below table:

Algorithm	Accuracy mean
Logistic Regression	65.83%
Naïve Bayes	53.19%
K Nearest Neighbours	62.96%
Decision Tree	62.70%
Random forest Classifier	67.28%
Support Vector Machine	63.95%
Artificial Neural Network	70.08%

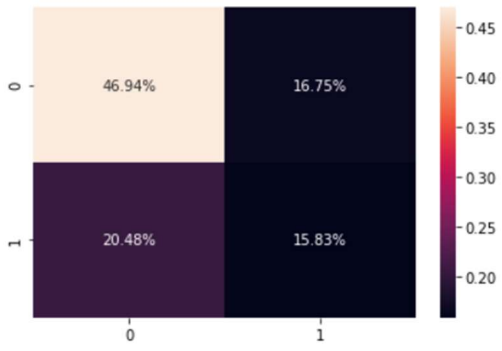
The Confusion Matrix Obtained for the prediction on testset are:



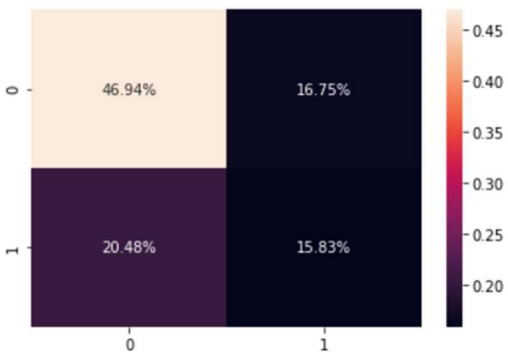
Logistic Regression



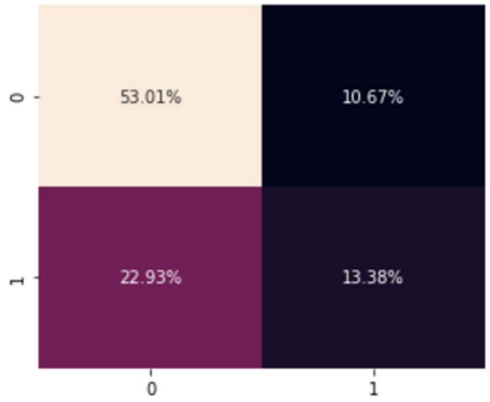
Naïve Bayes



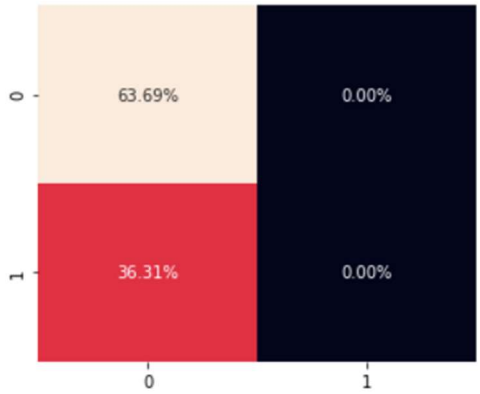
K Nearest Neighbors



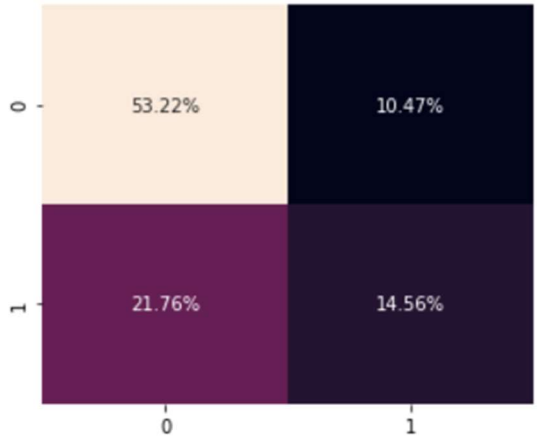
Decesion Tree



Random Forest



Support Vector Machine



Artificial Neural Networks

The following insights are gained from the above results:

- While the Logistic Regression Models gives many false negative, the Naïve Bayes gives many false positives.
- Artificial Neural Network gives the best accuracy of 70%. It has 2 hidden layers.
- SVM fails to identify the positive class completely and has not performed well.

8.Conclusion

This project is relevant to musicians and music labels. Not only will it help determine how best to produce songs to maximize their potential for becoming a hit, it could also help decide which songs could give the greatest return for investment on advertising and publicity. Furthermore, it would help artists and music labels determine which songs are unlikely to become Billboard Hot 100 hits. This will help musicians to review their work and improve the quality of song. In this synopsis, we have introduced a machine learning based model for predicting hit songs based on audio features, artist, genre. This project will also help the music labels to review songs.

9. Future Scope

This project can be further extended to give more accurate results by considering following facts.

- Popularity of Song also depends upon the popularity of artist. There can be a way to identify artist score and take it into account for making predictions.
- Music genre can also be taken into account along with audio features for more accurate results.
- Further , the lyrics of the music can be analysed to see whether it can give much accurate results or not along with audio features.

10. References

1. HITPREDICT: PREDICTING HIT SONGS USING SPOTIFY DATA STANFORD COMPUTER SCIENCE 229: MACHINE LEARNING by Elena Georgieva, Marcella Suta, and Nicholas Burton.
2. “Predicting Hit Songs with Machine Learning" by Minna Reiman and Philippa Ornell.
3. www.towardsdatascience.com
4. www.medium.com
5. Spotipy python documentation: https://spotipy.readthedocs.io/en/2.13.0/
6. Predicting Hit Songs with Machine Learning by MINNA REIMAN PHILIPPA ÖRNELL
7. https://scikit-learn.org/stable/ Scikit learn documentation Python