AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Bachelor of technology
In
Artificial Intelligence

21AIE205 – Python for Machine Learning

Project Phase 2

Topic:

Music Recommendation System

Done by:

V. Neeraj Chowdary – AIE21067
Thati Ayyappa Swamy – AIE21084
K. Sumitha – AIE21040
H. Sai Manasa Sowmya – AIE21057
P. Charishma Akshaya – AIE21051

Professor/Guide:

Dr. Sumi Suresh & Dr. Remya S

1 Acknowledgement:

We would like to express our deep gratitude to our project guide, Dr. Sumi Suresh & Dr. Remya, Assistant Professors, Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, for his/her guidance with unsurpassed knowledge and immense encouragement. We are grateful to Dr. Gopa Kumar, Head of the Department, Computer Science and Engineering, for providing us with the required facilities for the completion of the project work.

2 Abstract:

In this paper, we present a personalized music recommendation system based on the Matrix factorization and vectorization and machine learning algorithms. In personalized music recommendation system, we propose a collaborative filtering and content filtering recommendation algorithm to combine the output of the network with the log files to recommend music to the user. The proposed system contains the log files which stores the previous history of playlist of music by the user. The proposed music recommendation system extracts the user's history from the log file and recommends music under each recommendation. Content based methods gives recommendations based on the similarity of two song contents or attributes while collaborative methods make a prediction on possible preferences using a matrix with ratings on different songs. The plagiarism system extracts the music from input and finds music that are close to the query music which the query has plagiarized. We use the million-song dataset to evaluate the personalized music recommendation system. The data cleaning is done by the data science algorithms. The plagiarism detection is done by finding the similar music genre which minimizes the issue of copyrights.

3 Table of content:

4 List of abbreviations:

- Matrix factorization
- Vectorization
- NLTK

5 Introduction:

With the explosion of network in the past decades, internet has become the major source of retrieving multimedia information such as video, books, and music etc. People have considered that music is an important aspect of their lives and they listen to music, an activity they engaged in frequently. However, the problem now is to organize and manage the millions of music titles produced by society. A good music recommender system should be able to automatically detect preferences and generate playlists accordingly. The proposed system is to detect music plagiarism based on music similarity. The plagiarism system extracts the music from input and finds music that are close to the query music which the query has plagiarized. Meanwhile, the development of recommender systems provides a great opportunity for industry to aggregate the users who are interested in music. We need to generate the best music recommendation system which is need to predict based on customization, by using Vectorization, matrix factorization and Machine Learning algorithms.

The music recommender systems are double edged swords. The area of valuable use both to the user as well as the provider. They keep the user engaged by finding interesting music in the form of recommendations, lessening the burden on the user by reducing the set of choices to choose from. They give the scope for exploration and discovery of music that the user may not know exists. Because it is a music recommender there is never less entertainment.

5.1 Problem statement:

The basic task in music recommendation system with plagiarism detection is to generate the best music recommendation system by predicting based on customization and detecting the similar music genre to avoid copyrights issue, by using Collaborative filtering, Content based, Machine Learning, Data Analysis.

6 Literature survey:

An ideal music recommender system should be able to automatically recommend personalised music to human listeners. So far, many music discovery websites such as Last.fm, All music, Pandora, Audio baba Mog, Spotify, Apple Genius, have aggregated millions of users, and the development is explosive . In this section, we present the most popular approaches, metadata information retrieval, collaborative filtering, content-based information retrieval, emotion-based model, context-based information retrieval and hybrid models.

The proposed system contains the plagiarism in addition to the recommendation system which acts as a great advantage to resolve the copyright problems, the plagiarism module deals the check of similar music genre and detects the songs with similar musical notes.

6.1 Advantages:

The experimental results exhibited that the average scores, which are objectively collected by means of user evaluations, increases by degrees as the generation grows.

6.2 Disadvantages:

It is hard for people to come up with a good heuristic which actually reflects what we want the algorithm to do. It might not find the most optimal solution to the defined problem in all cases.

7 Methodologies and architecture:

7.1 Machine Learning:

A machine learning model is the output of the training process and is defined as the mathematical representation of the real-world process. The machine learning algorithms find the patterns in the training dataset, which is used to approximate the target function and is responsible for mapping the inputs to the outputs from the available dataset. These machine learning methods depend upon the type of task and are classified as Classification models, Regression models, Clustering, Dimensionality Reductions, Principal Component Analysis, etc.

7.1.1 Supervised Learning:

Supervised learning algorithms are used when the output is classified or labeled. These algorithms learn from the past data that is inputted, called training data, runs its analysis and uses this analysis to predict future events of any new data within the known classifications. The accurate prediction of test data requires large data to have a sufficient understanding of the patterns. The algorithm can be trained further by comparing the training outputs to actual ones and using the errors for modification of the algorithms.

7.1.2 Unsupervised Learning:

Unsupervised learning algorithms are used when we are unaware of the final outputs, and the classification or labelled outputs are not at our disposal. These algorithms study and generate a function to describe completely hidden and unlabelled patterns. Hence, there is no correct output, but it studies the data to give out unknown structures in unlabelled data.

7.2 Collaborative Filtering:

Collaborative filtering (CF) is a technique used by recommender systems. ... In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users.

Collaborative-based methods work with an interaction matrix, also called rating matrix. The aim of this algorithm is to learn a function that can predict if a user will benefit from an item-meaning the user will likely buy, listen to, watch this item. Among collaborative-based systems, we can encounter two types: user item filtering and item-item filtering.

The aim of this algorithm is to learn a function that can predict if a user will benefit from an item meaning the user will likely listen to a song. This can be done by using rating. There are two ways to collect user ratings: Explicit Rating and Implicit Rating.

7.2.1 Explicit Rating:

This means we explicitly ask the user to give a rating. This represents the most direct feedback from users to show how much they like a song.

7.2.2 Implicit Rating:

We examine whether or not a user listened to a song, for how long or how many times, which may suggest that he/she liked that particular song.

7.3 Content -Based filtering:

Content-based methods gives recommendations based on the similarity of two song contents or attributes while collaborative methods make a prediction on possible preferences using a matrix with ratings on different songs. content-based recommendation algorithm has to perform the following two steps. First, extract features out of the content of the song descriptions to create an object representation. Second, define a similarity function among these object representations which mimics what human understands as an item-item similarity. Because we are working with text and words, Term Frequency-Inverse Document Frequency (TF-IDF) can be used for this matching process.

Content-based methods are computationally fast and interpretable. Moreover, they can be efficiently adapted to new items or users. However, one of the biggest limitations of content-based recommendation systems is that the model only learns to recommend items of the same type that the user is already using or, in our case, listening to. Even though this could be helpful, the value of that recommendation is significantly less because it lacks the surprise component of discovering something completely new.

In the mood prediction, we do data pre-processing and feed to feature engineering models like count vectorizer model, TfidfVectorizer Model, Tfidf-NGram Model. Now the modified dataset is used in prediction models like Random Forest, Logistic regression, SVM, and multinomial naive baise, to predict the test data set and plot the confusion matrix.

7.4 Lyrics based recommendation:

Lyric-based song recommendation can be done using NLTK in several steps:

Data collection: Collect a dataset of lyrics from various songs.

1. Text preprocessing: Clean and preprocess the lyrics by removing stopwords, punctuation, and special characters.

2. Vectorization: Convert the preprocessed lyrics into numerical vectors using techniques such as bag-of-words or TF-IDF.
3. Similarity computation: Compute the similarity between the vectors of different songs using techniques such as cosine similarity.
4. Recommendation: Use the computed similarities to recommend songs to users. For example, you can recommend songs that have a high similarity to the lyrics of a song that a user is currently listening to.

Additionally, you can also use the techniques like Latent Semantic Analysis (LSA) or Latent Dirichlet Allocation (LDA) to extract the topics from the lyrics and use them to recommend similar songs.

It's worth mentioning that this is a basic method and there are many other factors that can be considered for recommendation like artist, album, genres, etc. Also, you may need to fine-tune the model with more data and by trying different techniques.

8 Datasets Description:

8.1 Collaborative filter dataset:

The One Million Songs Dataset (also known as the Million Song Dataset or MSD) is a dataset of one million audio tracks, along with metadata such as artist name, song title, release year, and other information. The dataset was created by the Echo Nest (now part of Spotify) and LabROSA at Columbia University. It is intended for use in research on music information retrieval, and has been used in a number of studies and projects in this field. The dataset includes a wide variety of music genres and is freely available for download and use under a Creative Commons license. The data is mainly in HD5 format and a few in Text format. The key features in this data set is listen count that is merging of triplet.csv dataset.

	user_id	song_id	listen_count	title	release	artist_name	year
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cove	Thicker Than Water	Jack Johnson	0
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Dos Aguas	Flamenco Para Niños	Paco De Lucia	1976
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1	Stronger	Graduation	Kanye West	2007
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1	Constellations	In Between Dreams	Jack Johnson	2005
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1	Learn To Fly	There Is Nothing Left To Lose	Foo Fighters	1999

Here the datasets are merged with the help of song id.

8.2 Content based filtering dataset

songdata.csv is a file that contains a dataset of song information. The exact content of the file will depend on the source, but it typically includes fields such as song title, artist name, release year, and other metadata. The file is usually in CSV (Comma Separated Values) format, which means that the data is separated by commas and can be easily imported into a spreadsheet or other program for analysis. The songdata.csv is not a standardized dataset and can be used by different authors with different structure, but it is commonly used as sample data for educational purposes and small-scale projects.

It is important to know the source of the dataset, as it can give you more information about the data and how it was collected.

The key feature here is artist and the lyrics of the song which use the TF-IDF vectorizer to find the similar kind lyrics song.

	artist	song	text
0	Hank Williams	THE FUNERAL	I was walking in Savannah past a church...
1	Glen Campbell	Words	Smile an everlasting smile a smile coul...
2	Hank Williams Jr.	Almost Nearly But Not Quite Plumb	Well, I ain't been home for so doggone ...
3	Lana Del Rey	Burning Desire	Every Saturday night I get dressed up t...
4	Ace Of Base	Experience Pearls	Give me all your tears Let me turn the...
5	Boney M.	I Feel Good	What a beautiful morning Just to wake ...
6	Britney Spears	I'm So Curious	So curious So curious I'm so curiou...
7	Metallica	Mercyful Fate	They're walking by the night The moon ...
8	Journey	Natural Thing	Winter and moonlight It's an old-fashi...
9	Westlife	Until The End Of Time	Until the end of time I'm longing fo...

8.3 Lyrics based recommendation:

It is a text based data set the variable are divided by the (;)(,)(.). the file will be divided with help of the nltk method library.

9 Python Packages:

9.1 Numpy:

NumPy is the fundamental package for scientific computing in Python. NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

9.2 Panda:

Pandas is an open source library in Python. It provides ready to use high- performance data structures and data analysis tools. Pandas module runs on top of NumPy and it is popularly used for data science and data analytics. It allows us to store and manipulate tabular data as a 2-D data structure.

Pandas is the most popular python library that is used for data analysis. It provides highly optimized performance with back-end source code is purely written in C or Python.

9.3 Python tuples & dictionaries:

9.3.1 Tuples:

A cone is a type of data type similar to a sequence of items. Cone is a set of values separated by commas. The pods, unlike the list, are surrounded by parentheses. Lists are placed in parentheses ([]), and the elements and sizes can be changed, but the lumps are wrapped in brackets (()) and cannot be sorted. Powders are the same.

9.3.2 Dictionaries:

Python dictionaries in Python are a way of a hash table. They are similar to Perl's combination schemes or hashes, and are made up of two key numbers. The dictionary key can be any type of Python, but numbers and strings are very common. Prices, on the other hand, can be anything you choose Python. Curly braces () surround dictionaries, and square braces ([) are used to assign and access values.

10 Algorithms:

10.1 Collaborative filtering algorithm (Matrix factorization):

Matrix factorization is a technique used to decompose a matrix into the product of two or more matrices. These matrices are typically lower-dimensional and have some specific structure, such as being triangular or diagonal. The goal of matrix factorization is to represent the original matrix in a more compact and interpretable form.

Matrix factorization has various forms, but the most widely used forms are:

1. Singular Value Decomposition (SVD): SVD is a factorization of a real or complex matrix that decomposes it into a unitary matrix, a diagonal matrix and another unitary matrix. It is used in a variety of applications including data compression, denoising, and collaborative filtering.
2. Non-negative Matrix Factorization (NMF): NMF is a factorization of a non-negative matrix into the product of non-negative matrices. It can be applied to data that has non-negativity constraints, such as images and text documents.
3. Principal Component Analysis (PCA) : PCA is a technique used to identify patterns in data, by finding the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions than the original one. PCA uses SVD internally to perform the decomposition of the matrix.
4. Low-rank matrix factorization: This is a general framework for matrix factorization, which can be applied to a wide range of problems. The algorithm can be implemented using a variety of optimization techniques such as gradient descent, alternating least squares, and convex optimization.

In collaborative filtering, matrix factorization techniques are used to approximate the user-item interaction matrix into a lower-dimensional representation by factorizing it into the product of two lower-dimensional matrices: user-factor matrix and item-factor matrix. These two matrices can be used to make predictions about missing entries or to recommend new items to users.

There are two main types of collaborative filtering algorithms: user-based and item-based.

1. User-based collaborative filtering: This algorithm makes recommendations based on the similarity between the preferences of a target user and other users. It finds other users who have similar preferences to the target user and recommends items that those similar users have liked.

2. Item-based collaborative filtering: This algorithm makes recommendations based on the similarity between the items that a target user has liked and all other items. It finds other items that are similar to the items the target user has liked and recommends those similar items to the user.

Both user-based and item-based collaborative filtering can be calculated using nearest neighbour algorithm or matrix factorization techniques.

Matrix factorization techniques such as SVD, NMF, and others are also used for collaborative filtering.

10.2Content based filter (cosine similarities):

Content-based recommendation algorithm has to perform the following two steps. First, extract features out of the content of the song descriptions to create an object representation. Second, define a similarity function among these object representations which mimics what human understands as an item-item similarity. Because we are working with text and words, Term Frequency-Inverse Document Frequency (TF-IDF) can be used for this matching process.

Recommendations done using content-based recommenders can be seen as a user-specific classification problem. This classifier learns the user's likes and dislikes from the features of the song.

The most straightforward approach is keyword matching. The idea behind is to extract meaningful keywords present in a song description a user likes, search for the keywords in other song descriptions to estimate similarities among them, and based on that, recommend those songs of the user.

TF-IDF is a information retrieval technique that weighs the terms frequency (TF) and inverse document frequency(IDF).The product of these scores of a term is called the TF*IDF weight of that term.

10.3 Lyrics based recommendation:

Explanation of code, assumptions and process flow in project:

Each Lyrics in training data has three features : Lyrics,Name,Sentiment.

Training data songs have been classified into three sentiments: positive, negative, not defined.

Training :

1. Created a function to identify lyrics and sentiments from training data.
2. Removed stop word from lyrics.
3. Created a bag of words of all lyrics.
4. Found frequency distribution of all words.
5. Applied features to filtered corpus and used Naive-Bayes algorithm to train the classifier on training data.

Testing :

1. Applied classifier on testing data to find sentiments of song based on lyrics.
2. Calculated accuracy of the model.

Used model to identify sentiment of a string in lyrics for verification purposes.

11 Code:

11.1 Collaborative filter code(Matrix factorization):

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
#%%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#%%
from scipy.sparse import csr_matrix
#%%
import Recommenders as Recommenders
#%% md
song_df_1 = pd.read_csv('triplets_file.csv')
song_df_1.head()
#%%
song_df_2 = pd.read_csv('song_data.csv')
song_df_2.head()
#%%
# combine both data
songs = pd.merge(song_df_1, song_df_2.drop_duplicates(['song_id']), on='song_id',
how='left')
songs.head()
#%%
songs.head()
#%% md
#%%
songs.to_csv('songs.csv', index=False)
#%% md
#%%
df_songs = pd.read_csv('songs.csv')
#%% md
## Exploring the data
#%% md
#%%
df_songs.head()
#%% md
#%%
#Get total observations
print(f"There are {df_songs.shape[0]} observations in the dataset")
#%% md
#%%
df_songs.isnull().sum()
#%% md
#%%
df_songs.dtypes
#%% md
#%%
#Unique songs
unique_songs = df_songs['title'].unique().shape[0]
print(f"There are {unique_songs} unique songs in the dataset")
#%% md
#%%
#Unique artists
unique_artists = df_songs['artist_name'].unique().shape[0]
print(f"There are {unique_artists} unique artists in the dataset")
```

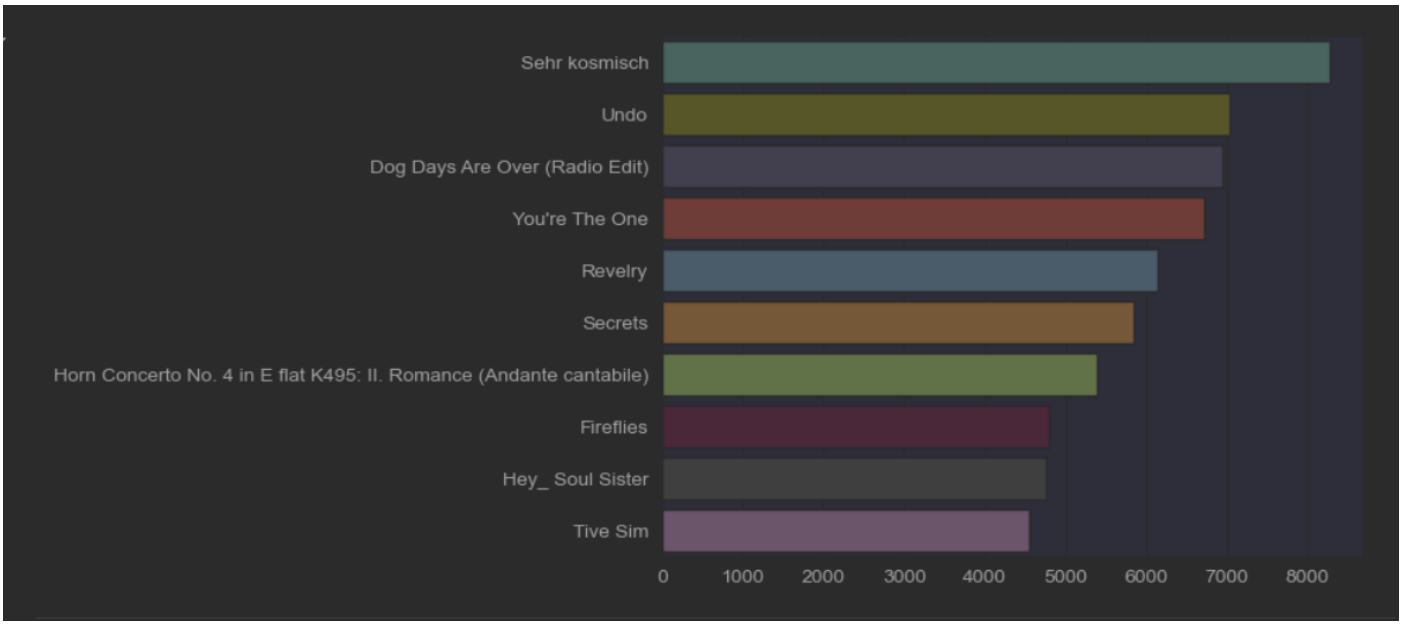
```
#%% md
#%%
#Unique users
unique_users = df_songs['user_id'].unique().shape[0]
print(f"There are {unique_users} unique users in the dataset")
#%% md
#%%
#count how many rows we have by song, we show only the ten more popular songs
ten_pop_songs =
df_songs.groupby('title')['listen_count'].count().reset_index().sort_values(['listen_co
unt', 'title'], ascending = [0,1])
ten_pop_songs['percentage']  =
round(ten_pop_songs['listen_count'].div(ten_pop_songs['listen_count'].sum())*100, 2)
#%%
ten_pop_songs = ten_pop_songs[:10]
ten_pop_songs
#%%
labels = ten_pop_songs['title'].tolist()
counts = ten_pop_songs['listen_count'].tolist()
#%%
plt.figure()
sns.barplot(x=counts, y=labels, palette='Set3')
sns.despine(left=True, bottom=True)
#%% md
#%%
#count how many rows we have by artist name, we show only the ten more popular artist
ten_pop_artists  =
df_songs.groupby(['artist_name'])['listen_count'].count().reset_index().sort_values(['l
isten_count', 'artist_name'],

ascending = [0,1])
#%%
ten_pop_artists = ten_pop_artists[:10]
ten_pop_artists
#%%
plt.figure()
labels = ten_pop_artists['artist_name'].tolist()
counts = ten_pop_artists['listen_count'].tolist()
sns.barplot(x=counts, y=labels, palette='Set2')
sns.despine(left=True, bottom=True)
#%% md#%%
listen_counts = pd.DataFrame(df_songs.groupby('listen_count').size(),
columns=['count'])
#%%
print(f"The maximum time the same user listened to the same songs was:
{listen_counts.reset_index(drop=False)['listen_count'].iloc[-1]}")
#%% md
#%%
print(f"On average, a user listen to the same song {df_songs['listen_count'].mean()}
times")
#%% md
#%%
plt.figure(figsize=(20, 5))
sns.boxplot(x='listen_count', data=df_songs)
sns.despine()
#%% md
#%%
listen_counts_temp = listen_counts[listen_counts['count'] > 50].reset_index(drop=False)
#%%
plt.figure(figsize=(16, 8))
sns.barplot(x='listen_count', y='count', palette='Set3', data=listen_counts_temp)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show();
#%% md
#%%
```

```
song_user = df_songs.groupby('user_id')['song_id'].count()
#%%
plt.figure(figsize=(16, 8))
sns.distplot(song_user.values, color='orange')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show();
#%%
print(f"A user listens to an average of {np.mean(song_user)} songs")
#%%
print(f"A user listens to an average of {np.median(song_user)} songs, with minimum
{np.min(song_user)} and maximum {np.max(song_user)} songs")
#%% md
#%%
# creating new feature combining title and artist name
df_songs['song'] = df_songs['title']+' - '+df_songs['artist_name']
df_songs.head()
#%% md
short samples
#%%
# taking top 10k samples for quick results
df_songs = df_songs.head(10000)
#%%
# cummulative sum of listen count of the songs
song_grouped = df_songs.groupby(['song']).agg({'listen_count':'count'}).reset_index()
song_grouped.head()
#%%
grouped_sum = song_grouped['listen_count'].sum()
song_grouped['percentage'] = (song_grouped['listen_count'] / grouped_sum ) * 100
song_grouped.sort_values(['listen_count', 'song'], ascending=[0,1])
#%% md
## Popularity Based Recommendation
#%%
pr = Recommenders.popularity_recommender_py()
#%%
pr.create(df_songs, 'user_id', 'song')
#%%
# display the top 10 popular songs
pr.recommend(df_songs['user_id'][5])
#%%
pr.recommend(df_songs['user_id'][100])
#%% md
## Item based popularity
#%%
ir = Recommenders.item_similarity_recommender_py()
ir.create(df_songs, 'user_id', 'song')
#%%
user_items = ir.get_user_items(df_songs['user_id'][5])
#%%
# display user songs history
for user_item in user_items:
    print(user_item)
#%%
# give song recommendation for that user
ir.recommend(df_songs['user_id'][5])
#%%
# give related songs based on the words
ir.get_similar_items(['Oliver James - Fleet Foxes', 'The End - Pearl Jam'])
```

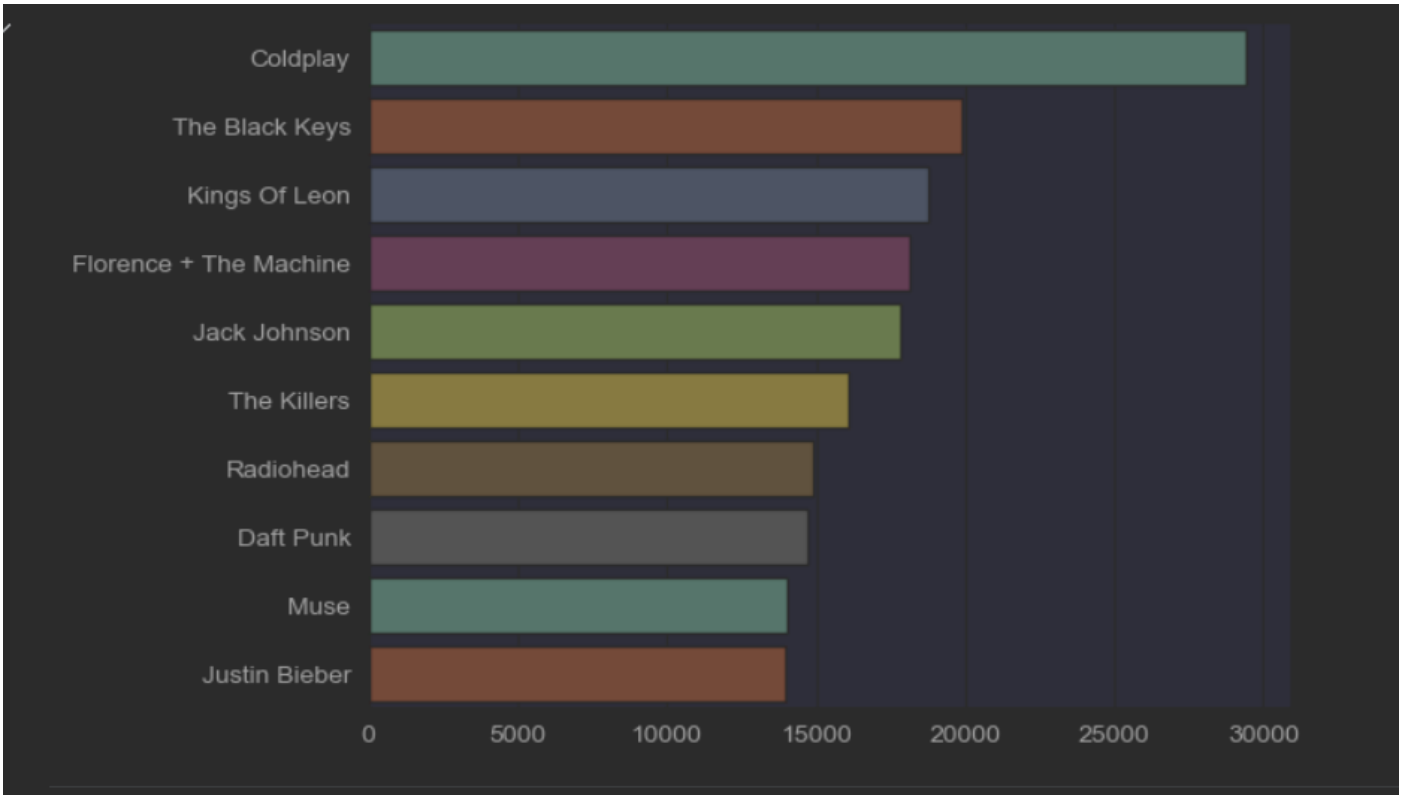
11.1.1 Graphs:

11.1.1.1 Most Popular Songs:



we'll count how many times each song appears. The listen_count represents how many times one user listen to the same song. The graph is for top ten popular songs which are sorted based on the listen_count.

11.1.1.2 Most Popular Artist:



we'll count how many times each artist appears. Again, we'll count how many times n the same artist appears.

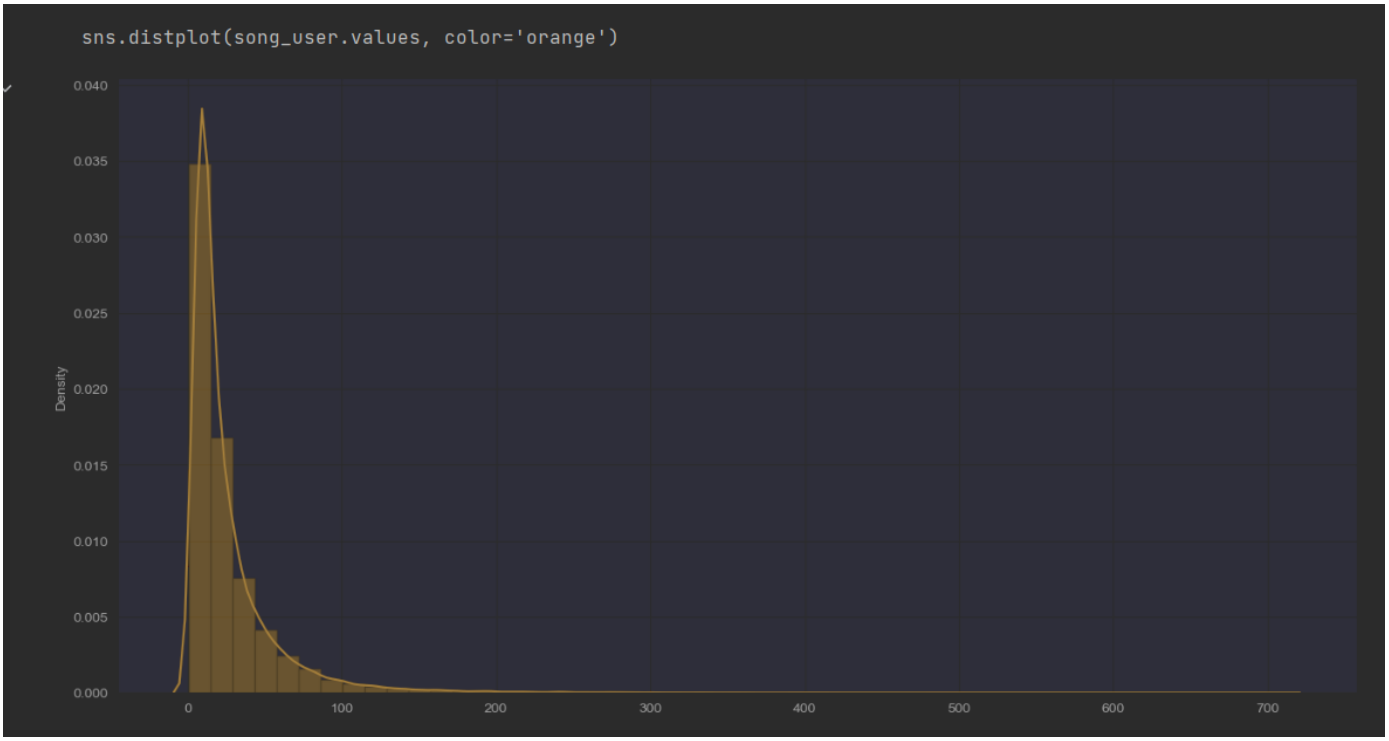
11.1.1.3 Listen Count by User:

A box plot or boxplot is a method for graphically depicting groups of numerical data through their quartiles.

It tells about other information based on the listen_count:

1. What was the maximum time the same user listen to a same song?
2. How many times on average the same user listen to a same song?

```
sns.distplot(song_user.values, color='orange')
```



11.1.2 Output for collaborative filter:

```
In 46
# give song recommendation for that user
ir.recommend(df_songs['user_id'][5])
```

```
No. of unique songs for the user: 45
no. of unique songs in the training set: 5151
Non zero values in cooccurence_matrix :6844
```

Out 46

	user_id	song	score	rank
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	Oliver James - Fleet Foxes	0.043076	1
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	Quiet Houses - Fleet Foxes	0.043076	2
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	Your Protector - Fleet Foxes	0.043076	3
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	Tiger Mountain Peasant Song - Fleet Foxes	0.043076	4
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	Sun It Rises - Fleet Foxes	0.043076	5
5	b80344d063b5ccb3212f76538f3d9e43d87dca9e	The End - Pearl Jam	0.037531	6
6	b80344d063b5ccb3212f76538f3d9e43d87dca9e	St. Elsewhere - Dave Grusin	0.037531	7
7	b80344d063b5ccb3212f76538f3d9e43d87dca9e	Misled - Céline Dion	0.037531	8

10 rows × 4 columns Open in new tab

```
In 47
# give related songs based on the words
ir.get_similar_items(['Oliver James - Fleet Foxes', 'The End - Pearl Jam'])
```

```
no. of unique songs in the training set: 5151
Non zero values in cooccurence_matrix :75
```

Out 47

	user_id	song	score	rank
0		Quiet Houses - Fleet Foxes	0.75	1
1		St. Elsewhere - Dave Grusin	0.75	2
2		Misled - Céline Dion	0.75	3
3		Your Protector - Fleet Foxes	0.75	4
4		Oil And Water - Incubus	0.75	5
5		Tiger Mountain Peasant Song - Fleet Foxes	0.75	6
6		Meadowlarks - Fleet Foxes	0.75	7
7		Sun It Rises - Fleet Foxes	0.75	8

10 rows × 4 columns Open in new tab

11.2 Content based filter(matrix factorization):

```
import numpy as np
import pandas as pd
#%%
from typing import List, Dict
#%% md
#%%
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
#%% md
### Dataset
#%% md
#%%
songs = pd.read_csv('songdata.csv')
#%%
songs.head(10)
#%% md
#%%
songs = songs.sample(n=5000).drop('link', axis=1).reset_index(drop=True)
#%% md
#%%
songs['text'] = songs['text'].str.replace(r'\n', '')
#%% md
#%%
tfidf = TfidfVectorizer(analyzer='word', stop_words='english')
#%%
lyrics_matrix = tfidf.fit_transform(songs['text'])
#%% md
#%%
cosine_similarities = cosine_similarity(lyrics_matrix)
#%% md
#%%
similarities = {}
#%%
for i in range(len(cosine_similarities)):
    # Now we'll sort each element in cosine_similarities and get the indexes of the
songs.
    similar_indices = cosine_similarities[i].argsort()[:-50:-1]
    # After that, we'll store in similarities each name of the 50 most similar songs.
    # Except the first one that is the same song.
    similarities[songs['song'].iloc[i]] = [(cosine_similarities[i][x],
songs['song'][x], songs['artist'][x]) for x in similar_indices][1:]
#%% md
#%%
class ContentBasedRecommender:
    def __init__(self, matrix):
        self.matrix_similar = matrix

    def _print_message(self, song, recom_song):
        rec_items = len(recom_song)

        print(f'The {rec_items} recommended songs for {song} are:')
        for i in range(rec_items):
            print(f"Number {i+1}:")
            print(f"{recom_song[i][1]} by {recom_song[i][2]} with
{round(recom_song[i][0], 3)} similarity score")
            print("--------------------")

    def recommend(self, recommendation):
        # Get song to find recommendations for
        song = recommendation['song']
        # Get number of songs to recommend
        number_songs = recommendation['number_songs']
        # Get the number of songs most similars from matrix similarities
        recom_song = self.matrix_similar[song][:number_songs]
```

```
        # print each item
        self._print_message(song=song, recom_song=recom_song)
#%% md
Now, instantiate class
#%%
recommedations = ContentBasedRecommender(similarities)
#%% md
#%%
recommendation = {
    "song": songs['song'].iloc[10],
    "number_songs": 4
}
#%%
recommedations.recommend(recommendation)
#%% md
#%%
recommendation2 = {
    "song": songs['song'].iloc[120],
    "number_songs": 4
}
#%%
recommedations.recommend(recommendation2)
```

11.2.1 Output:

```
In 17
recommendation2 = {
    "song": songs['song'].iloc[120],
    "number_songs": 4
}

In 18
recommedations.recommend(recommendation2)

The 4 recommended songs for It's The Most Wonderful Time Of The Year are:
Number 1:
It's The Most Wonderful Time Of The Year by Christmas Songs with 0.735 similarity score
-------------------
Number 2:
Wonderful Christmas Time by Demi Lovato with 0.327 similarity score
-------------------
Number 3:
Christmas Time Is Here by Christmas Songs with 0.236 similarity score
-------------------
Number 4:
Christmas Day by Beach Boys with 0.212 similarity score
-------------------
```

11.3 Lyrics Based recommendation:

```
import ast
import nltk
from nltk.corpus import stopwords
from collections import Counter
from random import randint
import nltk
nltk.download('stopwords')
#%% md
#%%
def create_tuples( fileName ):
   file_open = open(fileName)
   file_read = file_open.read()
   collection = ast.literal_eval(file_read)
   result = []
   for i in collection:
      record = (i["lyric"], i["sentiment"])
      result.append(record)
   return result
```

```
#%% md
#%%
def stop_words_removal(lyrics):
    array = []
    for i in lyrics:
        sentiment = i[1]
        stop = set(stopwords.words('english'))
        filtered_words = [word for word in i[0].split() if word not in stop]
        combine = (filtered_words , sentiment)
        array.append(combine)
    return array
#%% md
#%%
def get_words(lyrics):
    all_words = []
    for(words, sentiment) in lyrics:
        all_words.extend(words)
    return all_words

#%% md
#%%
def word_features(List_word):
    List_word = nltk.FreqDist(List_word)
    word_features = List_word.keys()
    return word_features
#%%
def extract_features(doc):
    doc_words = set(doc)
    features = {}
    for word in word_features:
        features['contains(%s)' %word] = (word in doc_words)
    return features

#%% md
#%%
def stop_word_removal1(lyrics):
    stop = set(stopwords.words('english'))
    filtered_words = [word for word in lyrics.split() if word not in stop]
    return filtered_words
#%%
# function to return list based on user-history
def user_sentiment(file):
    file_open = open(file)
    file_read = file_open.read()
    file_split=file_read.split("|")
    result_list=[]
    for lyrics in file_split:
        filtered_lyrics=stop_word_removal1(lyrics)
        output = classifier.classify(extract_features(filtered_lyrics))
        result_list.append(output)
    return result_list
#%% md
#%%
def recommendedSong(history):
    c= Counter(history)
    file_open = open("testing_original.txt")
    file_read = file_open.read()
    collection = ast.literal_eval(file_read)
    if c['P']>=c['N']:
        songs_positive=[i for i in collection if i["sentiment"]=='P']
        return songs_positive[randint(0,len(songs_positive)-1)]["name"]
    else:
        songs_negative=[i for i in collection if i["sentiment"]=='N']
        return songs_negative[randint(0,len(songs_negative)-1)]["name"]
#%% md
#%%
```

```
# creating tuples
# give path of file "training.txt"
lyrics = create_tuples("training_original.txt")
# removing stopwords
filtered_corpus = stop_words_removal(lyrics)
# Extracting Features
word_features = word_features(get_words(filtered_corpus))
# Applying Features
training_set = nltk.classify.apply_features(extract_features,filtered_corpus)
# Training Classifier
classifier = nltk.NaiveBayesClassifier.train(training_set)
#%% md
#%%
# creating tuples
# give path of file "testing.txt"
lyrics_test = create_tuples("testing_original.txt")
# removing stopwords
test_corpus = stop_words_removal(lyrics_test)
# applying classifier model on test dataset
test_set = nltk.classify.apply_features(extract_features,test_corpus)
#%%
########  Checking Accuracy
print("\t" + "Accuracy of the model is:" + str(nltk.classify.accuracy(classifier,
test_set)))
#%% md
#%%
def sentiment(output):
    if str(output) == "P":
        return "Positive"
    if str(output) == "N":
        return "Negative"
    if str(output) == "A":
        return "Sentiment not defined"
#%%
lyrics = "Gotta take my chance tonight"
output = classifier.classify(extract_features(lyrics.split()))
print(sentiment(output))
#%% md
#%%

#  Using model to recommend a song to user based on his/her listening habits
# give path of file "User_Last_5_Songs.txt"
file_open = open("User_Last_5_Songs.txt")
file_read = file_open.read()
file_split=file_read.split("|")
result_list=[]
#%%
sentiment_history= user_sentiment("User_Last_5_Songs.txt")
print("recommend song: " + recommendedSong(sentiment_history))
```

11.3.1 Output:

```
In 38  1  sentiment_history= user_sentiment("User_Last_5_Songs.txt")
       2  print("recommend song: " + recommendedSong(sentiment_history))

          recommend song: Back in the USSR
```

The output represents the song titles along with the similarity score which are based on the lyrics that are taken from sample set of the input.

12 GitHub links:

Music recommendation system whole project repo

Collaboratory filter code

Content based filter code

Lyrics based recommendation

Datasets included

13 Functionality requirement:

A Functional Requirement (FR) is a description of the service that the software must offer. It describes a software system or its component. ... It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

1. The system can add, read datasets.
2. The system performs the classifier training process and display the music recommended from the training data.
3. The system can display a set of musical records from the content, collaborative filtering.
4. The system displays the musical records based on emotion of the user and displays the personalized music.
5. The system can detect and displays the plagiarism in the musical records.

14 Non-Functionality requirement:

A non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions.

1. The system can run in various web browser which support the system environment.
2. The system gives a fast response.
3. The system has user friendly interface

14.1 Hardware-requirements:

Hardware Requirements:

1. RAM: 4GB
2. Storage: 500GB
3. CPU: 2 GHz or faster
4. Architecture: 32-bit or 64-bit

15 Conclusion:

The following are our conclusions based on experiment results. First, music recommender system should consider the music genre information to increase the quality of music recommendations. The music recommender can recommend the songs based on the song features. The music Recommender can check plagiarism in the dataset taken by generating the similarity score for each recommended song. The mood of the song is predicted by examining the lyrics of the given song with all the other songs in the dataset and predicting the mood and similarity scores and recommending the songs based on the mood. The complex nature of the machine learning systems like the Music Recommendation System can't have a standardized structure because different music recommender systems work in different way. Based on our analyses, we can suggest for future research to add other music features in order to improve the accuracy of the recommender system, such as using tempo gram for capturing local tempo at a certain time.