A MUSIC RECOMMENDATION SYSTEM

MILLION SONGS DATASET

DMT CS6220

Hello Everyone,

The topic we have chosen for our final project is Music Recommendation System. Our team consists of Adithya Chandrashekar and Anush Mandya Nagesh.

A recommender system, or a recommendation system is a system that usually uses large datasets to determine or predict most relevant products for a customer based on things such as user preferences, duration of use, rating provided by the customer and so on.

The large datasets usually do not make sense, therefore it goes through various steps to be made useable. This transformed data is then used to find patterns in consumer behavior. These patterns are exploited to customize results to a customer.

1. ABSTRACT

Let us start with the first question what is the jargon “Recommendation Systems”. Recommendation systems are nothing but engines that rearrange the information on data available based on various parameters such as reviews, likes, dislikes, popularity trends etc. Once the analysis is performed the job of the recommendation engines are to provide the users with items that they would have likely preferred. To introduce our CS6220 project for the first time “Music Recommendation System” we have built a recommendation engine that takes in a dataset comprising of music and suggests the users the most unbiased and specific music recommendation. In this project our main aim was to explore the two main types of filtering popularly that is Content Based Filtering and Collaborative Based Filtering. In Content Based we make use of the description or the keywords as the means of providing recommendations. For example, if a user likes a song the engine is said to have determined that if the user is suggested songs similar to the ones the user has liked then the user will tend to like them as well. In the Collaborative Filtering approach, we end up predicting what the users might like based on their similarities with other users who are using the same engine for recommendation. To better explain this approach let us consider two users A and B and if suppose somewhere at some point user A and B have liked the same songs, then the approach tends to say both the users have the same preferences. Therefore, in the future if user A likes a particular song, then user B is also suggested this song thinking that user B will also end up liking the song. In our project we have extended Collaborative Based Filtering to Item Based Filtering and Popularity Based Filtering. In Item Based Filtering we come up with recommendation in three different approaches the first being based on the similarity of a user’s preference is suggested to the user. In the second approach based on the top picks by users of similar preferences the songs are suggested. The third approach is nothing but songs similar to particular songs are recommended, for example if you listen to song A another song B which is in some sense similar to song A is recommended to the user. Finally, to finish up this section I would like to say we were successfully able to touch upon all the above-mentioned approaches and to make things more interesting we have used three different datasets which totally sum up to Fifty Million songs and this yielded satisfactory and acceptable results in all the experiments performed and for all the given approaches.

2. INTRODUCTION

Recommender Systems have become an integral part of most of the websites and mobile applications. Some of the most famous websites that make use of a recommendation system are, Netflix, Hulu, Amazon, YouTube and so on. Some famous mobile applications that use this are, Tinder, Instagram, Spotify and so on. In this project we will be focusing on a music recommendation system. As the name implies, it is a system that recommends songs to a user based on various criteria. In the current world, billions of people listen to music on a daily basis. And the interest and taste in music differs from a set of people to another set of people. Although there are soo many varied interests and liking towards songs among various groups of people, there is some sort of similitude among these groups. Due to this an opportunity arises. There are millions of songs and millions of listeners. When such large numbers are involved, we can always find a patterns and exploit them. Since there are soo many options it sometimes becomes confusing and overwhelming to choose something particular. In similar context it becomes difficult for listeners to pick a specific song. So, what our system is trying to accomplish is to find some patterns and similarities with both the song data and user data to come up with songs that are the best and most relevant songs a user might like. The problem we are trying to solve is to personalize songs for our listeners.

Earlier people used to purchase radio, cassettes, CDs to listen to songs. Nowadays with the advancement in technology and the internet becoming the go to platform for anything and everything, online music streaming has made listening to music very convenient and easy. This way people can access millions of songs that are digitally present on the internet. Searching through such an abundant resource to find songs a particular user likes can suddenly become a tedious process. Another major reason why developing a music recommendation system is important is, these online businesses such as Spotify, Apple music and so on do not have any physical interaction with their users. In such a case it is impossible to know user preferences and adhere to those requirements. In such a case, developing a Recommendation system will ensure that user's likes, dislikes, behaviors, number of times listened, ratings are all considered. Therefore, developing a music recommendation system that provides suggestions based on user behavior can be very useful.

There are multiple approaches to build a Recommender System. They can all be grouped under 3 main sub-classes namely,

1. Content based
2. Collaborative Filtering
3. Hybrid Filtering

2.1 Content based

Content based recommendation systems work on items that have descriptions, location, name, etc, and user's preferences. It is better suited for datasets that have items with such

descriptions. How this approach works is, the items are suggested based on user's likes, dislikes, rating and so on. We can achieve content-based recommendation systems in various ways, the simplest way is to use the average values of the rated items. Using machine learning to achieve this is slightly more complex. Machine Learning techniques such as Bayesian Classifiers, cluster analysis, decision trees, and artificial neural networks can be used to predict whether a user is going to like the item or not.

A major shortcoming of content-based filtering is, we cannot use user's actions and behaviors from one content source to predict items across different content types.

2.2 Collaborative Filtering

Collaborative filtering approach is one of the most widely used filtering technique in recommendation systems. This method works under the assumption that user's will like a suggested item which is similar to the item liked by the user in the past. Basically assumes that, users who agreed in the past will agree in the future.

Collaborative Filtering is divided into 2 different types:

a. Memory based filtering
b. Model based filtering

In this project I will be dealing with Memory based filtering techniques. Especially the Item Based Filtering technique. We will see how we use this approach to suggest most relevant songs for a specific user. We will also see how suggest songs based on a particular song.

2.3 Hybrid Filtering

As the name suggests, it is a technique that falls between content based filtering and collaborative filtering techniques. This algorithm surpasses the shortcoming of the individual filtering models and also know to improve performance. Although it has not been implemented in this project it is something we wish to do in the future.

2.4 Popularity Based Filtering

Popularity based filtering is a naïve approach to suggest songs that I have implemented in this project. It is based on the popularity of the songs. The most listened to song is placed at the top of the list. Since it is a naïve approach the suggestion for every user will be the same.

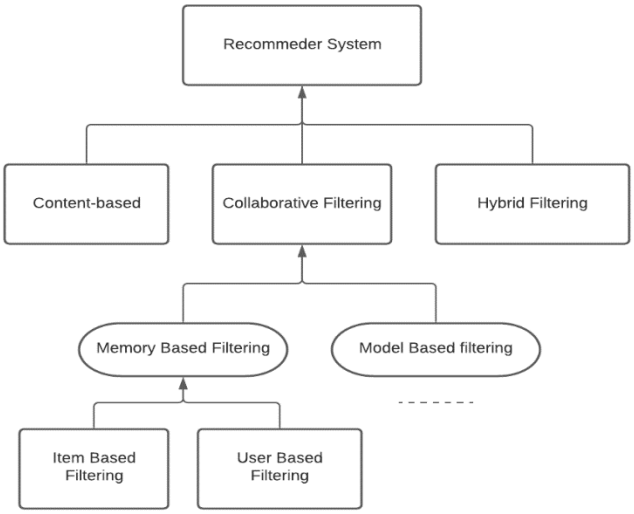

Fig 3.1 Types of Recommendation Systems

Figure 3.1 shows various ways to achieve a recommendation system. In this project we will be using Content-based and Collaborative Filtering (Item Based Filtering) and Popularity based filtering.

3. METHODOLOGY

EDA (Exploratory Data Analysis):

The first step in our project after we finalized the datasets was to perform the Exploratory Data Analysis on all the three datasets to investigate for any patterns, anomalies, missing data. We have summarized the statistics of our datasets. This is a very good practice in any data mining project where we need to gain as much as insights on the data and make sense of it before getting our hands dirty with it.

Data Preprocessing:

The second step in the process of building our very own recommendation engine was to perform data preprocessing. Data preprocessing is the process of cleaning up of data which is nothing but detection and removal of outliers. The actual dataset involved had more than 20 million songs we had to downscale the data due to limited processing power and time. For downscaling we just picked 10,000 song and decided to perform our experiments on this subset of data.

Content Based Filtering:

Which as discussed earlier uses descriptions and keywords as the means of providing recommendations to the users.

Popularity Based Filtering:

In this type of filtering our song dataset we first take the unique songs and user id's and get the count of how many times this song occurs in the dataset by counting how many user id's have the same song in the given dataset and a score is computed. Based on the score for each song then arrange the songs in descending order to get the most popular songs in our dataset.

Collaborative Based Filtering:

We have implemented the item-based similarity technique in this part of the experiment. We have two types of approaches User Based and Song Based. In user-based technique user id is taken as the input and based on the user's past song choices and recent history songs are recommended. In song-based recommendation we give a particular song as an input and songs are recommended mostly if it is of the same genre, same artists and from the same timeline.

4. CODE

Import all the necessary libraries such as pandas, NumPy, matplotlib, seaborn, etc.

4.1 Content Based

- Read the data into a data frame using pandas.

```
songs = pd.read_csv('C:/Users/anush/OneDrive/Desktop/songdata.csv')
songs.head()
```

- Performed pre-processing on the data. Dropped unnecessary columns, replaced unnecessary symbols with an empty string.

- We assign value to the songs based on importance using TfidfVectorizer and measure similarity between items using cosine_similaities.

```
tfidf = TfidfVectorizer(analyzer='word', stop_words='english')
lyrics_matrix = tfidf.fit_transform(songs['text'])
```

```
cosine_similarities = cosine_similarity(lyrics_matrix)
```

- I have made use of a predefined class for content-based recommender.

- Next, we create an object for ContentBasedRecommender class. Then set up the inputs by providing song name and number of similar songs we need. Finally, call the recommend() to display all the songs suggested.

```
recommedations = ContentBasedRecommender(similarities)
```

```
recommendation = {
    "song": songs['song'].iloc[10],
    "number_songs": 4
}
```

```
recommendation
```

```
{'song': 'The Narrows', 'number_songs': 4}
```

```
recommedations.recommend(recommendation)
```

4.2 Collaborative Filtering

- Read the data into a dataframe. For this I am using different datasets. I am using Million Song Dataset which has 2 files. Triplets_file.csv and song_data.csv

- Perform some Exploratory Data Analysis on the imported data.

- Move on to data pre-processing
  - The imported data has too many entries. Reduce them to 10,000 entries.
  - After reducing the dataset size, merge the 2 dataframes into a single dataframe.

- Find top ten artists from our list. GroupBy artist_name by using the listen_count. Sort the artist_name by the value of listen_count in descending order.
- Plot a graph for artist_name versus listen_count.

- Find top ten songs. First merge title and artist name columns to a single column called song. GroupBy song by using the listen_count. Sort the song by the value of listen_count in descending order.
- Plot a graph for song versus listen_count.

- Next let us split our data into training data and test data. Set test data to 20% of the total data. Before we train out model it is always necessary and important split the data.

```
train_data, test_data = train_test_split(song_df,test_size=0.2, random_state=0)
```

4.2.1 Popularity Based Recommendation

- I have made use of a pre-existing class to check for popularity. This is a naïve approach therefore the song suggestions remain the same for all the users. We just display the top songs by based on score which is calculated.

```
Recommenders = popularity_recommender_py()
Recommenders.create(train_data, 'user_id', 'song')

#user the popularity model to make some prediction
user_id = users[5]
Recommenders.recommend(user_id)
```

4.2.2 Item Similarity Based Collaborative Filtering

- We have made use of a pre-exiting class for this.
- First let us look at Top picks for a user.
    - We call the method get_user_items() by passing the user Id we want. This retrieves the top picks for that user. How it does this is, we get all the unique songs corresponding to the given user. These songs are then displayed.
    - Next call the recommend() method by passing the user_id. This will suggest us the top picks based on the user's unique song list that was generated.

```
#Print the songs for the user in training data
user_id = users[6]
user_items = is_model.get_user_items(user_id)
#
print("----------------------------------------------------------------------------------------")
print("Training data songs for the user userid: %s:" % user_id)
print("----------------------------------------------------------------------------------------")

for user_item in user_items:
    print(user_item)

print("----------------------------------------------------------------------")
print("Recommendation process going on:")
print("----------------------------------------------------------------------")

#Recommend songs for the user using personalized model
is_model.recommend(user_id)
```

- Next, let us do Top picks based on a song as input.
    - All we need to do this call the get_similar_items() and pass the song same. It displays the top 10 most relevant songs for the input song.

```
is_model.get_similar_items(['U Smile - Justin Bieber'])
```

5. RESULTS

5.1 Content Based Filtering

Content Based filtering is an approach that depends on item descriptions to make suggestions to the user. It also considers the user preferences while making suggestions. The preferences may include likes, dislikes, ratings and so on.

Since this type of filtering is specific to the user, it can be very useful. It can also be used on large number of users.

In our project implementation, the song is an item. Based on the description of the input song we determine similar songs.

The number of similar songs can also be passed as an input.

Below are the results we determined for Content based filtering.

```
recommendation2 = {
    "song": songs['song'].iloc[120],
    "number_songs": 6
}
```

```
recommedations.recommend(recommendation2)
```

```
The 6 recommended songs for Best Thing In Town are:
Number 1:
The Best Thing by Savage Garden with 0.308 similarity score
--------------------
Number 2:
Every Little Thing You Do by Westlife with 0.289 similarity score
--------------------
Number 3:
One Dumb Thing by Devo with 0.271 similarity score
--------------------
Number 4:
What Do They Know by Westlife with 0.268 similarity score
--------------------
Number 5:
Livin' Thing by Electric Light Orchestra with 0.264 similarity score
--------------------
Number 6:
Best Friend by J Cole with 0.261 similarity score
--------------------
```

Fig 5.1: Song name and number of similar songs needed.

In fig 5.1 we can see that the inputs are a song at location 120 in our dataframe and the number of similar songs we want are 6. These inputs are passed to the recommend function of content-based filtering class.

As we can see, the songs we selected was “Best Thing In Town”. The 6 songs suggested are listed in the decreasing order of similarity score.

```
recommendation
```

```
{'song': 'The Narrows', 'number_songs': 4}
```

```
recommedations.recommend(recommendation)
```

```
The 4 recommended songs for The Narrows are:
Number 1:
The Reason by Westlife with 0.121 similarity score
--------------------
Number 2:
One Hundred Punks by Offspring with 0.117 similarity score
--------------------
Number 3:
Bold Soul Sister by Tina Turner with 0.109 similarity score
--------------------
Number 4:
Change by Dolly Parton with 0.089 similarity score
--------------------
```

Fig 5.2: Song name and number of similar songs needed.

Fig 5.2 shows the input song as “The Narrows” and we want 4 most relevant songs to this input. Our recommendation system generated top 4 songs based on similarity score. But if we observe closely, we can see that the top song suggested in Fig 5.2 has a very low similarity score of just 0.121. The top song suggested in Fig 5.1 has a decent similarity score of 0.308

What we can take away from this is that not always do we get recommendations that match perfectly with our interests. It depends on various factors such as, size of the dataset, efficiency of the algorithm and so on.

5.2 Popularity Based Filtering

To get started with Popularity Based filtering we first merge the song and artist name into a single column and perform aggregation of number of times a particular song is listened to.

In the given song_df we group by the number of listens in ascending order. Then in the very next line in the code snippet shown we perform the group_by performing the summation of all the listen_counts of each of the given song.

In the next step we rather had a very naïve approach to solve this that was get the count of unique users and songs in the subset of data that we have. We then split this data into test and train sets. When we do this, we famously split it in 80:20 train test ratio. We then have introduced an instance of the popularity-based recommender class and input our training data to it. We then create a recommender that accepts user_id as the input and outputs a list of recommended song to that user.

The reason why we decided to include Popularity Based Filtering in our recommendation engine experiment is due to the phenomenon of “Cold Start”. To explain this in detail consider the recommendation engine encounters a user for the very first time and has no search, preference history, then what the popularity-based filtering approach does is it recommends the most

listened/popular songs already present from past user data. Therefore, allowing even the newest user to be recommended some songs.

```
Recommenders = popularity_recommender_py()
Recommenders.create(train_data, 'user_id', 'song')

#user the popularity model to make some prediction
user_id = users[5]
Recommenders.recommend(user_id)
```

]:

	user_id	song	score	Rank
3201	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Sehr kosmisch - Harmonia	40	1.0
4083	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Undo - Björk	30	2.0
4451	4bd88bfb25263a75bbdd467e74018f4ae570e5df	You're The One - Dwight Yoakam	27	3.0
938	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Dog Days Are Over (Radio Edit) - Florence + Th...	25	4.0
3819	4bd88bfb25263a75bbdd467e74018f4ae570e5df	The Scientist - Coldplay	23	5.0
4114	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Use Somebody - Kings Of Leon	22	6.0
1569	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Hey_ Soul Sister - Train	21	7.0
3197	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Secrets - OneRepublic	21	8.0
3040	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Revelry - Kings Of Leon	20	9.0
1207	4bd88bfb25263a75bbdd467e74018f4ae570e5df	Fireflies - Charttraxx Karaoke	18	10.0

Fig 5.3: Top 10 recommendations for User Id = 5

```
Recommenders = popularity_recommender_py()
Recommenders.create(train_data, 'user_id', 'song')

#user the popularity model to make some prediction
user_id = users[10]
Recommenders.recommend(user_id)
```

	user_id	song	score	Rank
3201	17aa9f6dbdf753831da8f38c71b66b64373de613	Sehr kosmisch - Harmonia	40	1.0
4083	17aa9f6dbdf753831da8f38c71b66b64373de613	Undo - Björk	30	2.0
4451	17aa9f6dbdf753831da8f38c71b66b64373de613	You're The One - Dwight Yoakam	27	3.0
938	17aa9f6dbdf753831da8f38c71b66b64373de613	Dog Days Are Over (Radio Edit) - Florence + Th...	25	4.0
3819	17aa9f6dbdf753831da8f38c71b66b64373de613	The Scientist - Coldplay	23	5.0
4114	17aa9f6dbdf753831da8f38c71b66b64373de613	Use Somebody - Kings Of Leon	22	6.0
1569	17aa9f6dbdf753831da8f38c71b66b64373de613	Hey_ Soul Sister - Train	21	7.0
3197	17aa9f6dbdf753831da8f38c71b66b64373de613	Secrets - OneRepublic	21	8.0
3040	17aa9f6dbdf753831da8f38c71b66b64373de613	Revelry - Kings Of Leon	20	9.0
1207	17aa9f6dbdf753831da8f38c71b66b64373de613	Fireflies - Charttraxx Karaoke	18	10.0

Fig 5.4: Top 10 recommendations for User Id = 10

Figures 5.3 and 5.4 tell us a lot about popularity-based recommendation system. In figure 5.3 the user 5 is given as input. We get a list of top 10 songs ranked based on score. In figure 5.4 the user 10 was given as input. But the song suggestions did not change. This is because the top songs have been determined using scores. And as discussed before, this type of song suggestion is suited best for new users.

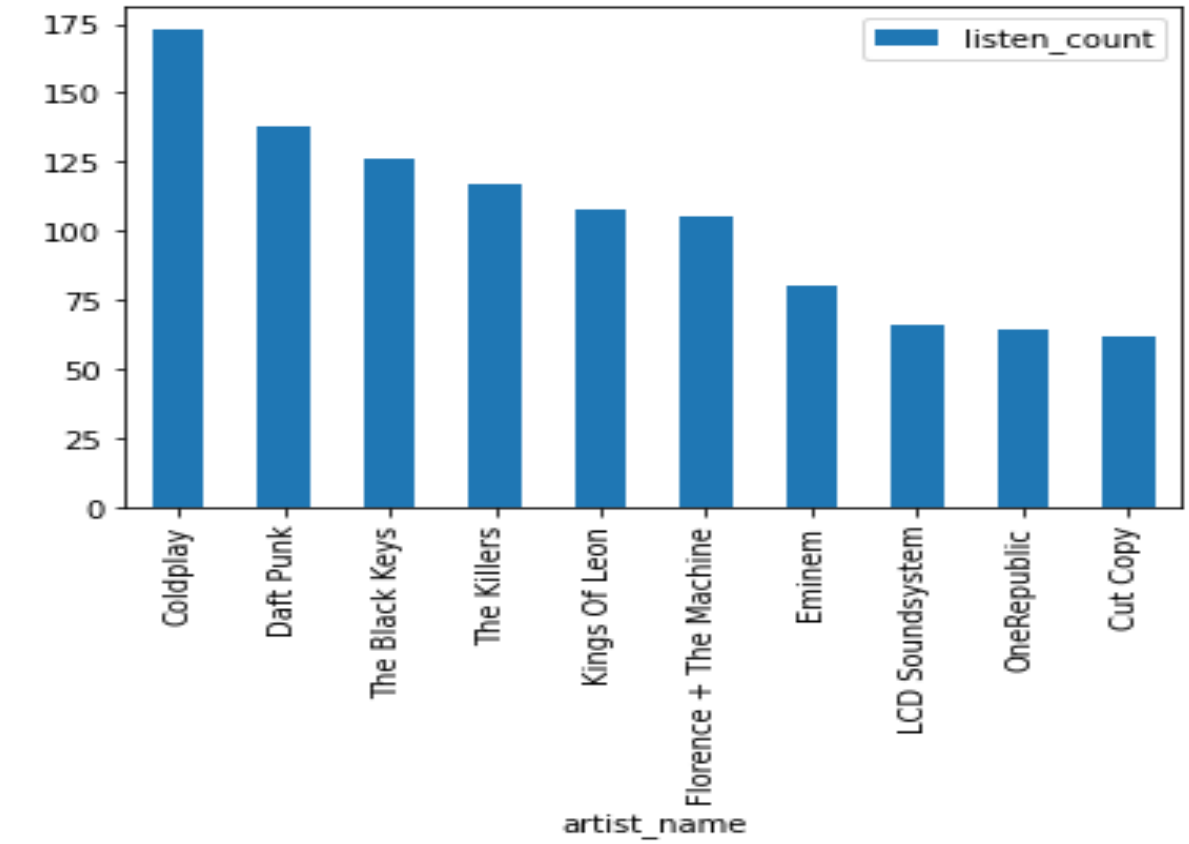

Fig 5.5: Top Ten Artists

Figure 5.5 gives us the most listened artists. To achieve this we aggregated the listen_count for each artist and displayed the entries in descending order of listen_count. This type of suggestion is best suited for fresh users.

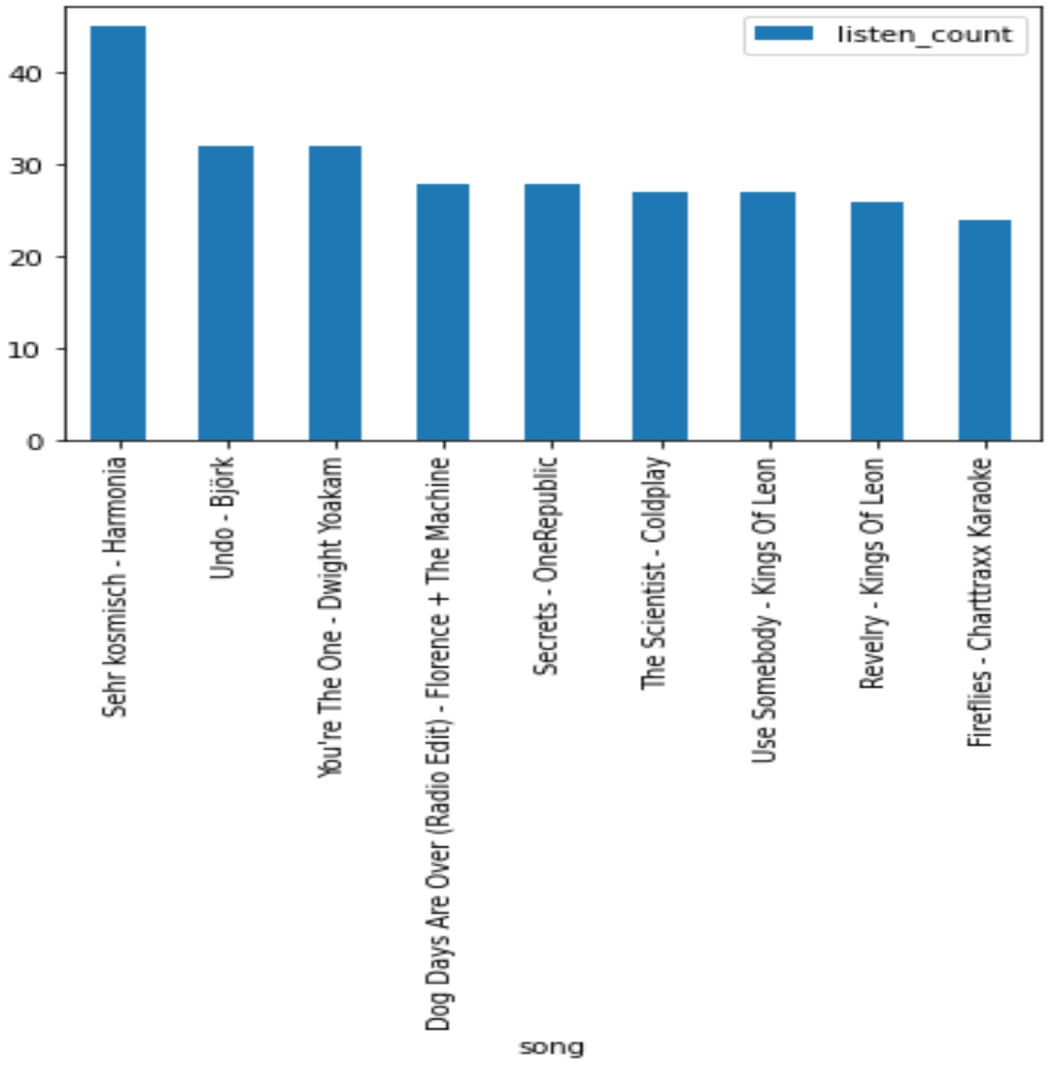

Fig 5.6: Top Ten Songs

Figure 5.6 shows the top 10 songs that are displayed using the score value. Application of this approach is best suited to new users.

5.3 Collaborative Based Filtering

Item Similarity Based Recommender System

To extend the discussion in further detail let us consider the item-based filtering to do this we must first define a co-occurrence matrix which is defined based on what song a user likes. The reason behind creating this matrix is to find out for each song what is the number of times a user has listened to the song, and if user will also listen to another set of songs. Additionally, this also takes into account what a user likes in the past and a similar song that you may like based on what other similar users have liked.

To create an Item Based Filtering we create an instance of the class and feed it with our training data. Inside our class we have a method called generate_top_recommendations the purpose of this function is to compute the weighted average of all the scores in the co-occurrence matrix for all songs. The co-occurrence matrix is a sparse matrix as it is impossible to predict if one user likes a particular song will like a millionth song in the dataset. We can predict the list of songs a user will like using the class we just defined. We can also use the same model to find similar songs to any songs in our dataset.

User Id as Input:

```
------------------------------------------------------------------------------------
Training data songs for the user userid: e006b1a48f466bf59feefed32bec6494495a4436:
------------------------------------------------------------------------------------
You're The One - Dwight Yoakam
Dog Days Are Over (Radio Edit) - Florence + The Machine
Sehr kosmisch - Harmonia
Blow Me Away - Breaking Benjamin
Secrets - OneRepublic
Rhyme & Reason - DAVE MATTHEWS BAND
Revelry - Kings Of Leon
Horn Concerto No. 4 in E flat K495: II. Romance (Andante cantabile) - Barry Tuckwell/Academy of St Martin-in-the-Fields/S
ir Neville Marriner
Undo - Björk
Ain't Misbehavin - Sam Cooke
Come As You Are - Nirvana
Someone Else's Arms - Mae
Marry Me - Train
Drop The World - Lil Wayne / Eminem
Use Somebody - Kings Of Leon
```

Fig 5.7a: Unique songs user has listened to

```
Drop The World - Lil Wayne / Eminem
Use Somebody - Kings Of Leon
Corn Bread - DAVE MATTHEWS BAND
OMG - Usher featuring will.i.am
Fireflies - Charttraxx Karaoke
Where Did You Sleep Last Night - Nirvana
High Life - Daft Punk
Lady In Black - Ensiferum
For You (Amended/Radio Edit LP) - Staind
Hey_ Soul Sister - Train
----------------------------------------------------------------------
Recommendation process going on:
----------------------------------------------------------------------
No. of unique songs for the user: 23
no. of unique songs in the training set: 4495
Non zero values in cooccurence_matrix :7914
```

Fig 5.7b: Unique songs user has listened to

	user_id	song	score	rank
0	e006b1a48f466bf59feefed32bec6494495a4436	Lucky (Album Version) - Jason Mraz & Colbie Ca...	0.100652	1
1	e006b1a48f466bf59feefed32bec6494495a4436	Just Dance - Lady GaGa / Colby O'Donis	0.098547	2
2	e006b1a48f466bf59feefed32bec6494495a4436	Heartbreak Warfare - John Mayer	0.096370	3
3	e006b1a48f466bf59feefed32bec6494495a4436	The Scientist - Coldplay	0.094576	4
4	e006b1a48f466bf59feefed32bec6494495a4436	Somebody To Love - Justin Bieber	0.090138	5
5	e006b1a48f466bf59feefed32bec6494495a4436	Bulletproof - La Roux	0.088839	6
6	e006b1a48f466bf59feefed32bec6494495a4436	Bleed It Out [Live At Milton Keynes] - Linkin ...	0.087470	7
7	e006b1a48f466bf59feefed32bec6494495a4436	Alejandro - Lady GaGa	0.082268	8
8	e006b1a48f466bf59feefed32bec6494495a4436	The Only Exception (Album Version) - Paramore	0.080941	9
9	e006b1a48f466bf59feefed32bec6494495a4436	Creep (Explicit) - Radiohead	0.079770	10

Fig 5.8: Songs recommended based on user's unique songs

In the results of the collaborative-based filtering approach where user id is taken as the input. The results are interpreted the following way.

1. Given a user_id as input, the engine goes through the available data and outputs all the unique songs a user has listened to. This is seen in the Fig 5.7a and 5.7b.

2. We next find the songs that a user might like based on the unique songs that were fetched as mentioned in the previous step. The songs customized for a user are shown in Fig 5.8

Song Name as Input: Example 1

	user_id	song	score	rank
0		Bad Company - Five Finger Death Punch	0.600000	1
1		Sexy Can I - Ray J / Yung Berg	0.400000	2
2		That Should Be Me - Justin Bieber	0.333333	3
3		All The Right Moves - OneRepublic	0.333333	4
4		Teach Me How To Dougie - California Swag District	0.333333	5
5		Te Amo - Rihanna	0.300000	6
6		My Name Is - Eminem	0.285714	7
7		One - Metallica	0.285714	8
8		What You Know - Two Door Cinema Club	0.285714	9
9		The Middle - Jimmy Eat World	0.285714	10

Fig 5.9a: Song suggestions based on a song as input

Song Name as Input: Example 2

```
is_model.get_similar_items(['Te Amo - Rihanna'])
```

```
no. of unique songs in the training set: 4495
Non zero values in cooccurence_matrix :314
```

	user_id	song	score	rank
0		Bad Company - Five Finger Death Punch	0.375000	1
1		The Middle - Jimmy Eat World	0.333333	2
2		U Smile - Justin Bieber	0.300000	3
3		Sayonara-Nostalgia - Base Ball Bear	0.300000	4
4		If I Had You - Adam Lambert	0.272727	5
5		Rabbit Heart (Raise It Up) - Florence + The Ma...	0.272727	6
6		Whatcha Say - Jason Derulo	0.266667	7
7		I'm Not Calling You A Liar - Florence + The Ma...	0.250000	8
8		One Less Lonely Girl - Justin Bieber	0.250000	9
9		My Beloved Monster - Eels	0.250000	10

Fig 5.9b: Song suggestions based on a song as input

The figures 5.9a shows different song suggestions based on the input song. As we can see the scores are reasonably high for the input song “U Smile - Justin Bieber”. This implies that the suggested songs are the most relevant songs.

In figure 5.9b, the input song is “Te Amo – Rihanna”. We get the top 10 songs that are similar to the input. The scores are not as high as we saw in 5.9a. But these are the most similar songs for “Te Amo – Rihanna”.

P.T.O

6. FUTURE WORK AND CONCLUSION

Going further we would love to implement and explore the hybrid recommendation system where we are able to combine two or more strategies in different ways to gain numerous benefits from their complementary advantages. Another thing we would love to work on is to build a very user-friendly GUI to this project that allows user a very organized and well-structured user-interface to access and navigate through the recommendation engine.

In conclusion of this report, we would like to state that each of the methods and approaches mentioned in this report are beneficial in their own way. By going through and developing this recommendation engine we leant a lot about different types of approaches and methodologies involved in building a recommendation system and the importance of why some of the very big companies such as Spotify, Netflix, Amazon etc. spend numerous resources to have the most efficient recommendation engine.

7. REFERENCES

https://www.analyticsvidhya.com/blog/2020/08/recommendation-system-k-nearest-neighbors/

https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15

https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85

https://github.com/ugis22/music_recommender/tree/master/collaborative_recommender_system

We have referred to various projects in this topic. We have used some pre-existing classes to perform the recommendation. We even watched some YouTube videos on music and movie recommendation systems. Through this process we learnt a lot about data mining and recommendation systems.