Music Playlist Recommendation System

CS 6220: Data Mining Techniques

Summer 2020

Prof. Sara Arunagiri

Team Members

Abhinav Agrawal

Chandrashekar Gubbi M.

Dishari Bhattacharya

Satwik Sabharwal

Abstract:

The idea behind the recommendation system is to develop a system which generates automatic playlist continuation. Given a set of playlist features, the recommender systems should generate a list of recommended tracks which can be added to the playlist, thereby continuing the playlist.

The main problem here is to identify the various way on how to recommend next song to the user or the playlist. Whether it should be based on type of music or language or genre or artists or popularity is a very big question to be discussed before working on the recommendation system for music platforms. Better the recommended songs, more likely the user to keep using that music service. Since user cannot search for all songs manually, if the music service provider is able to provide the recommendations of the songs that user would highly like, then there are higher chances of the music service staying relevant and becoming successful.
Since the dataset is very large and has lots of properties, there cannot be a single method which would work as a recommendation system for such large dataset and giving satisfactory results. Therefore, we tested many methods to achieve the goal. Two main methods are user ratings system where the user rated various songs and then the recommended songs were provided to them resulting in highly favorable outputs from the model. But since, you cannot always ask a user to rate the songs but rather songs are liked or disliked in any music service. Therefore, the second method to model the system was based on this idea. The songs were then recommended on the basis of various properties of the song liked by the user.
The generated results of the model were highly favorable, although the models could be trained
better with respect to time and better system architectures. The generated results were in according with the user's choices as well as on the statistical viewpoints. More information about the project and the methods used are detailed in the given report.

Introduction:

As online music streaming becomes the dominant medium for people to listen to their favorite songs, music streaming services are now able to collect large amounts of data on the listening habits of their customers. These streaming services, like Spotify, Apple Music or Pandora, are using this data to provide recommendations to their listeners. These music recommendation systems are part of a broader class of recommender systems, which filter information to predict a user's preferences when it comes to a certain item. By some accounts, almost half of all current music consumption is by the way of these services. While recommender systems have been around for quite some time and are very well researched, music recommender systems differ from their more common siblings in some characteristically important ways: the duration of the items is less (3-5 min for a song vs 120 minutes for a movie or months/years for a book or shopping item), the size of the catalog of items is larger (10s of millions of songs), the items are consumed in sequence with multiple items consumed in a session, repeated recommendations have a different significance.

Problem Description:

One of the major problems in Music Recommender Systems is the station/ playlist generation problem. At its heart, the playlist generation is about finding the set of songs to recommend to best extend the experience of a listener in the midst of a playlist. By suggesting appropriate songs to add to a playlist, a Recommender System can increase user engagement by making playlist creation easier, as well as extending listening beyond the end of existing playlists. One of Spotify's primary products is Playlists, collections of tracks that individual users (or Spotify) can build for every mood or event. Spotify users can make or follow as many playlists as they like. With over 40 million songs available, the company attempts to direct the most relevant songs to users based on their preferences, and Playlists often comprise the most convenient and effective way to convey these recommended songs to a user.

Spotify participates in the creation and curation of Playlists that are followed, or listened to, by millions of Spotify users. These Playlists are compiled in a complex manner, involving both human-led and computer-led processes. What stands is those algorithmically-curated discovery playlists, and their effectiveness remains an important business interest for the company. The goal is to better understand how these algorithms can be evaluated and improved with machine learning techniques learned in the class.

Our business objectives are-

i) Create a recommender system for automatic playlist continuation.

ii) Given a set of playlist features, recommender systems should be able to generate a list of recommended tracks that can be added to that playlist, thereby 'continuing' the playlist.

Methodology:

This section deals with the methodology followed to accomplish and achieve the goals of the project. Code snippets are shown here for the reference purposes, but the entire code is available in the zip files. The methodology we followed is divided into main four parts: Data Collection, Data Understanding, Data Preprocessing and Data Modelling.

Data Collection:

The data provided by Spotify is a Million Playlist Dataset (MPD) which contains 1 Million playlists created by the Spotify users. This dataset is quite huge (about 5.4 GB) and thus it has been divided into 1000 slices, where each slice is a file and contains 1000 playlists comprising 1,000,000 playlists in total. Due to hardware limitations, we worked with 6 json files and extracted important features using Spotify Api calls. Spotify structures its data pertaining to each song in four objects: a track object, an artist object, an album object, and an audio features object. They are all relatively self-explanatory in terms of the information stored except audio features - this object stores a wide variety of quantitative data that Spotify assigns songs, like energy, acousticness, etc. So, the general method for data collection was to request all four objects that corresponded with each song in the playlist, referenced via the track URI (a unique identifier for each track). This method of data collection presented several problems in terms of scalability.

1. To get any of these objects requires an authentication token, but a temporary one can be easily requested from Spotify's API. This does, however, require us to manually get new tokens periodically, which significantly limits our timeframe for scrapin.

2. Spotify will block tokens that are hitting API endpoints too frequently, so on quick Wifi and computers, the scraping was ironically significantly more difficult, with new tokens required for as little as every 50 songs. We had to work around this by adding some sleep mechanism, which also reduced the size of our final data.

3. There were often random stops in the scraping due to request errors; these did not occur with any pattern and we are unsure as to why they did occur. Because of the above-mentioned factors, we were able to scrape over 32,000 songs from playlists, which amounts to 19,000 songs after removing duplicates. This is a significant amount of data, but without data for all songs that exist on Spotify, which would be a nearly impossible task, we had to design our models with certain workarounds.

Data Understanding:

As stated in the earlier, all the data have been divided into 1000 files, each of which contains 1000 playlists. These are all json files and in the form of a dictionary. These files follow the naming convention:

mpd.slice.STARTING_PLAYLIST_ID_-_ENDING_PLAYLIST_ID.json.

The ids start from 0 and go upto 999999. The json dictionaries have two fields: 'info' and 'playlists' .

The code for this section is in "*UnderstandingTheDataSet.ipynb*"

```
%%bash
head 'mpd.v1/data/mpd.slice.4000-4999.json'
```

```
{
    "info": {
        "generated_on": "2017-12-03 08:41:42.057563",
        "slice": "4000-4999",
        "version": "v1"
    },
    "playlists": [
        {
            "name": "skate",
            "collaborative": "true",
```

The above image shows the 'head' command run on of the json files after importing it in a notebook file. As stated already it is in a dictionary format and contains two fields, or say, keys: 'info' and 'playlists'. The 'info' is a dictionary which contains general information about the data

in that particular file and contains three keys:

- slice: The range of playlists that are contained within the files. E.g. 0 to 999
- version: The current version of MPD (It is v1 at present)
- generated_on: The time stamp on which the file, or slice was generated

The above image shows the 'grep' command executed on the json file. It shows the lines with 4 to 16 leading spaces, which would help us identify the dictionary structure. The 'playlists' is an array of 1000 playlists. Each playlist is in the form of a dictionary which contains the following keys. It contains a field 'track', which in itself is an array of dictionaries.

- pid: It represents the id of the playlist. It is an integer between 0 and 999999.
- name: Name of the playlist
- description: Description of the playlist, as and if any, provided by the user. It is an optional field.
- modified_at: Timestamp, in seconds since the epoch when the playlist was last modified. The times are rounded to midnight GMT of the date when the playlist was last updated.
- num_artists: The total number of unique artists in the playlist
- num_albums: The total number of unique albums in the playlist
- num_tracks: Number of tracks in the playlist
- num_followers: Number of the followers of the playlist, excluding the creator.
- num_edits: The number of edits. All the changes within a two hour window are considered to be a single edit.
- duration_ms: The total duration, in milliseconds, of all the tracks combined.

- collaborative: It is a boolean field. True means that the playlist was created by just a single user.
- tracks: It is an array of dictionaries and it contains the data about the tracks contained in that particular playlist. It contains the following fields:
    - track_name: Name of the track
    - track_uri: Spotify URI of the track. It can be used to make API calls, or for other such developer functions.
    - album_name: Name of the track's album
    - album_uri: Spotify URI of the album of the track
    - artist_name: The name of track's artist
    - artist_uri: The URI of the artist of the track
    - duration_ms: The duration, in milliseconds, of the track
    - pos: The position, starting from 0, of the track in playlist

```
column_names = [col["tracks"] for col in columns]
column_names
```

```
[[{'pos': 0,
   'artist_name': 'Ty Segall',
   'track_uri': 'spotify:track:53yXbISmPFewWiFP2kgal8',
   'artist_uri': 'spotify:artist:58XGUNsRNu3cVOIOYk5chx',
   'track_name': 'Circles',
   'album_uri': 'spotify:album:5bXUYhd6yvmry49a8WpEqS',
   'duration_ms': 186176,
   'album_name': 'Mr. Face'},
  {'pos': 1,
   'artist_name': 'Creedence Clearwater Revival',
   'track_uri': 'spotify:track:47atuTch0AN7NvP9vp42a5',
   'artist_uri': 'spotify:artist:3IYUhFvPQItj6xySrBmZkd',
   'track_name': 'Ramble Tamble',
   'album_uri': 'spotify:album:4GLxEXWI3JiRKp6H7bfTIK',
   'duration_ms': 431226,
   'album_name': "Cosmo's Factory"},
  {'pos': 2,
   'artist_name': 'Fugazi',
   'track_uri': 'spotify:track:7uzeJ1HasqTqqB2anNRFCs',
   'artist_uri': 'spotify:artist:62sC6lUEWRibFqXpMmOk4G',
```

The above command extracts all the tracks' data. As discussed previously, 'tracks' is a dictionary which has fields like pos, artist_name, track_uri, artist_uri, track_name, album_uri, duration_ms and album_name.

```
# Reading csv data into the pandas dataframe
import pandas as pd
df = pd.DataFrame(columns)
```

```
# A sneekpeak into the 'playlists' data
df.head()
```

	collaborative	description	duration_ms	modified_at	name	num_albums	num_artists	num_edits	num_followers	num_tracks	pid	tracks
0	true	NaN	16539160	1432252800	skate	62	53	55	1	70	4000	[{'pos': 0, 'artist_name': 'Ty Segall', 'track...
1	false	NaN	3053039	1493510400	Work out	11	10	9	1	12	4001	[{'pos': 0, 'artist_name': 'Eminem', 'track_ur...
2	false	NaN	4419474	1462665600	Study	14	13	9	2	19	4002	[{'pos': 0, 'artist_name': 'Old Dominion', 'tr...
3	false	NaN	23594821	1493424000	HOLA	89	68	22	2	104	4003	[{'pos': 0, 'artist_name': 'Reggaeton Latino',...
4	false	NaN	30240074	1508889600	House	70	47	30	2	79	4004	[{'pos': 0, 'artist_name': 'Ben Watt', 'track_...

The above picture shows the head of the data frame generated from the dataset. As expected, the data frame contains the features such as collaborative, description, duration_ms, etc. as those were the fields of ‘playlists’ dictionary in the json file.

We would now try to analyse and understand some of these features by generating some statistics, histograms and pie charts, as required and appropriate.

1. ***num_albums*** feature

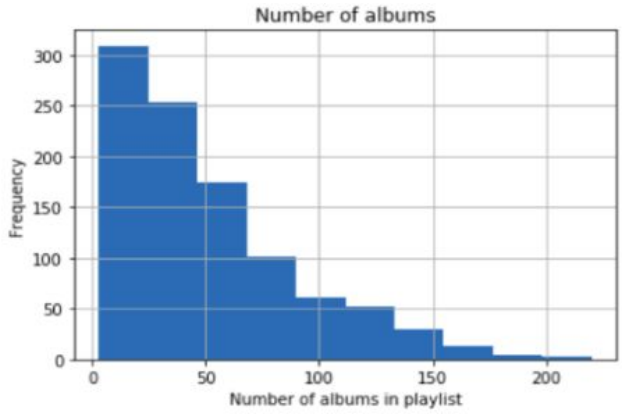
The ‘playlists’ contains number of albums as one of the fields which represents the total number of unique albums in the playlist.

```
Summary Statistics for number of albums

count    1000.000000
mean       51.270000
std        39.539632
min         3.000000
25%        20.000000
50%        41.000000
75%        70.000000
max       220.000000
Name: num_albums, dtype: float64

Variance        1563.3824824824826
Range           217
```

The count is 1000, as expected since there are 1000 playlists. The mean is 51.2, which means that each playlist, on average, contains about 51 albums. The standard deviation is 39.5 here. Also, the minimum is 3, which means that all the playlists contain minimum 3 albums and maximum 220. This makes sense as we already saw in section 2 that all the playlists contain a minimum 2 unique albums. The 25th percentile, 50th percentile, and 75th percentile values are 20, 41 and 70 respectively. The variance and range values are 1563 and 217 respectively.



Above is the histogram generated for the number of albums feature. The graph is a kind of decreasing step function, which means that the number of playlists decrease as the corresponding frequency increases. Majority of the playlists contain 1 to 25 albums, while a very few of them contain more than 150 albums.

2. ***num_tracks*** feature

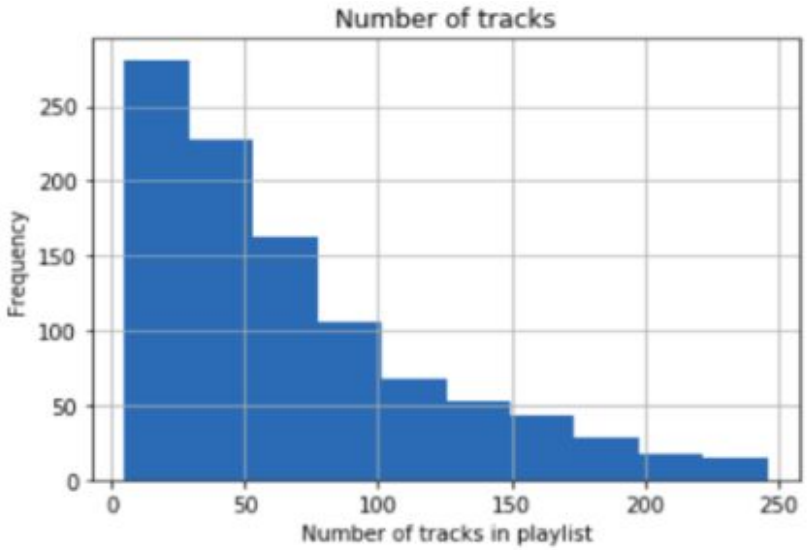
This is the total number of unique tracks in playlist.

```
Summary Statistics for number of tracks

count    1000.000000
mean       68.101000
std        53.532612
min         5.000000
25%        26.000000
50%        52.000000
75%        95.000000
max       246.000000
Name: num_tracks, dtype: float64

Variance        2865.740539539543
Range           241
```

The count is 1000, as expected since there are 1000 playlists. The mean is 51.2, which means that each playlist, on average, contains about 51 albums. The standard deviation is 39.5 here. Also, the minimum is 3, which means that all the playlists contain minimum 3 albums and maximum 220. This makes sense as we already saw in section 2 that all the playlists contain a minimum 2 unique albums. The 25th percentile, 50th percentile, and 75th percentile values are 20, 41 and 70 respectively. The variance and range values are 1563 and 217 respectively.



It is again a decreasing step function and most of the playlists have about 1 to 25 unique tracks in them. Just as the number of albums feature, here too the frequency of playlists decrease as the number of tracks increase. The major difference among the number of tracks and number of albums features is their range, which is 217 for number of albums, while 241 for number of tracks.

3. ***duration_ms*** feature

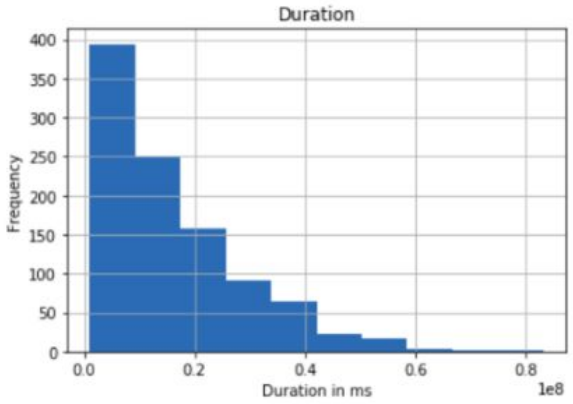
It represents the total duration, in milliseconds of all the tracks combined in that playlist.

```
Summary Statistics for duration

count    1.000000e+03
mean     1.604941e+07
std      1.280911e+07
min      1.044377e+06
25%      6.118938e+06
50%      1.228655e+07
75%      2.234913e+07
max      8.309966e+07
Name: duration_ms, dtype: float64

Variance        164073286096730.56
Range           82055288
```

The mean is 1.6*10^7 , which is about 266 minutes and means that each playlist contains about 68 tracks on average. The standard deviation is about 200 minutes here. Also, the minimum is 166 minutes, while the maximum is about 1383 minutes. The variance and range values are 164073286096730 milliseconds and 82055288 milliseconds respectively.



it is a decreasing step function which means that as the duration increases, the associated playlist frequency decreases. Majority of the playlists have about the total duration of 166 to 255 minutes of all the tracks combined.

Data Pre-Processing:

Dataset needed to be preprocessed before modelling was done. The following are some examples where data cleaning was required:

- Then the Album release date was in different formats and all those formats were converted to YYYY format.
- Then non quant and garbage columns were dropped. There were many columns with N/A values which were also dropped. This was done due to the fact that those columns would not help regarding the modelling and would only hinder the overall process.
- All the quantitative columns were standardized for consistent range. There were columns which were having values in decimals and there were some which were having values in multiples of 10 and 100. Thus for proper analysis, these values would have created problems. So they were standardized so that their comparison and collaboration could be done properly.
- We generated a Correlation Matrix for the data. Using this matrix, some data columns were dropped to reduce redundancy. Dropped features 'energy' and 'album_popularity' since they are highly related to 'loudness' and 'track_popularity' respectively and would just increase redundancy in the model. We decide which one to drop based on the one that has the next highest correlation with a different predictor, i.e. we drop energy over loudness because energy has a stronger correlation with acousticness, and we drop album over track popularity because album pop has a stronger correlation with artist popularity.

Data Modelling:

Music is a very subjective domain, and given the size of the dataset, we experimented with a few different methods.

Model 1: The first approach is based on a user rating system. We chose 3 different users, and gave them a few randomly generated songs (20 in our case) from our database. They heard the songs and gave ratings on a scale of 1-10 for the songs (with 10 being the highest). We then trained a model to learn their preferences and predict what ratings they would give for any other song, based on the characteristics of those given songs and how they rated them. Our ultimate goal is to give them back a list of any number of top rated songs from our mode (we chose to give them their 5 top favourites).

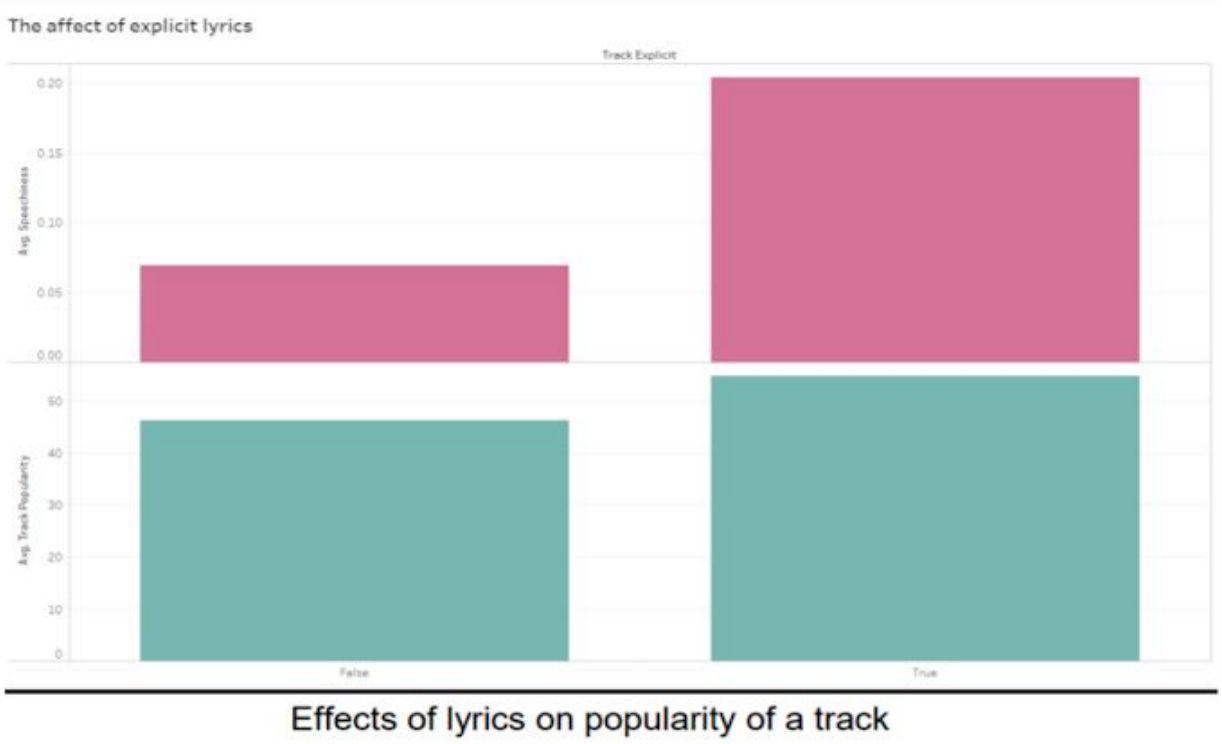
With so many predictors in comparison to observations we begin falling victim to some degree of the curse of dimensionality, but there was really no way to avoid this due to real life limitations. Each model was fit on those 20 songs, with quantitative attributes as predictors, and the user-provided ratings as the response variable. Treating the response variable as continuous, we were aware we would get predictions that went under 1 and over 10, but that would not impact our ability to take, say, the highest 5 predictions, no matter how high they were. We tried various model types for Model 1 which are simple linear regression, random forest and neural networks for which we got the best results for simple regression models.

Model 2: For the second approach, we replicated how actual music recommendation systems base their recommendations off songs in playlists. When users add particular songs to their playlist, they are assumed to consider those songs "good" and to their taste. The issue with this, however, is that user-provided playlists could very easily have songs that were not in our smaller database. Thus we would be unable to actually test this model on real users, however, we still tried the model.

Rather than training on a dataset with only 20 observations, this model is trained on the entire database. The response variable is a binary classification, with 0 indicating that the song is not in the user-provided playlist, and 1 indicating that it is present. The issue with this model is, that depending on the size of the playlist, the number of classification 1's can be very low in comparison to classification 0's. It was an unbalanced dataset due to which we did not really get desirable results. From there we tested a multitude of classification models. Unfortunately, because users could not provide playlists with songs that were not in our database, we could not test this on user data. We did, however, randomly assign songs to be "in the playlist" to check our mechanisms. It is to note that as a result of the extremely low number of observations in Model 1, and low number of classifications in Model 2, we chose not to split our data into a test set for model 1. Rather, (for Model 1) we literally tested the model on the same human users, asking them how they liked to output songs. It is to also note that we chose not to reduce the large number of dimensions relative to observations with PCA. We did eliminate as many predictors as we could through other means, including highly correlated ones, but we wanted to maintain interpretability. Using PCA would prevent us from saying, for example, that a positive coefficient for a year means that the user prefers (correlation) more recent songs.

Data Visualization:

A. Effect of lyrics on track popularity



Effects of lyrics on popularity of a track

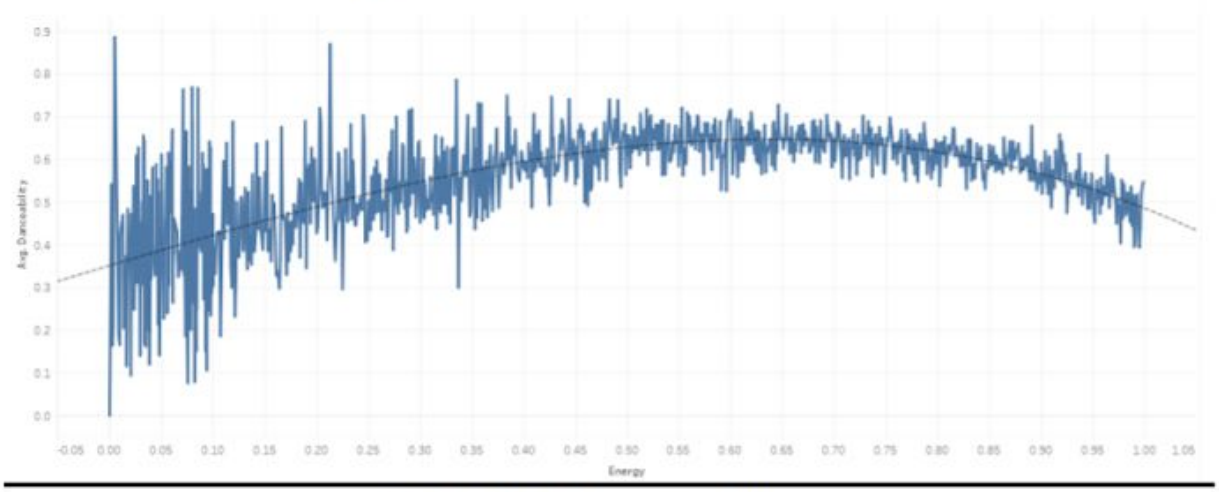
Interpretation:

• Here, we see above that we are plotting the 2 bar graphs comparing track explicit with Average Speechiness and Average Track Popularity.

• We observe that for the Speechiness Vs Explicit, if the track has high speechiness then there are more number of collaborations in the playlist i.e people use these songs and create collaborative playlists.

• Observing the Track Popularity Vs Explicit graph, we conclude that there is not much difference between the tracks which are popular or unpopular to be found in collaborative playlists.

B. Effect of energy of the track on danceability factor of humans



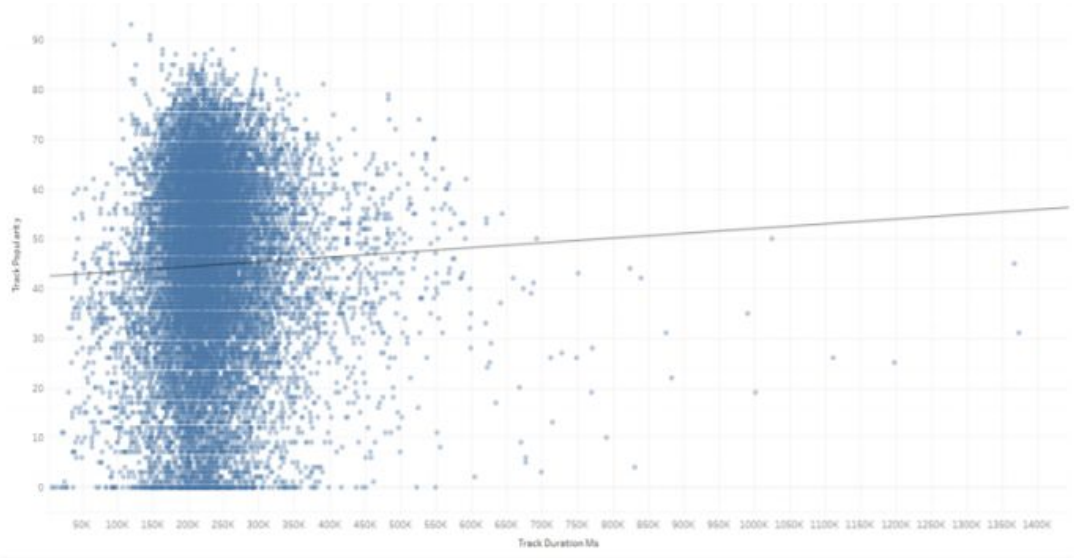
Dance Vs Energy Graph

Interpretation for the above graph:

• The above plot shows how the energy between 0.45 to 0.75 of a track correlates to the danceability of the track.

• We conclude that when the energy is too high, there is a slight decrease in the danceability factor. It means the people don't prefer to dance a lot if the energy of the track is way too high.

C. Effect of length of the track on the popularity of the track



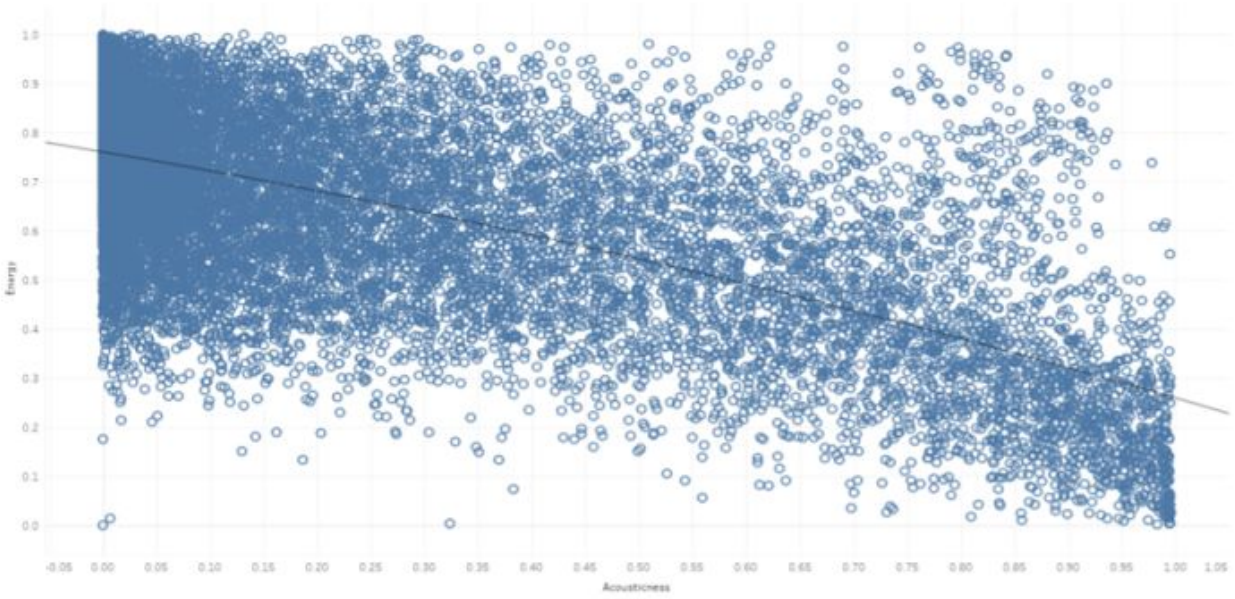
TrackTime vs TrackPopularity

Interpretation for the above graph:

• We observe that there is a positive slope with increase in playlist time. It means as the playlist time increases, the popularity of tracks also increases.

• Generally, most of the tracks are of a common length in the playlist as they are grouped together near the 150K -250K milliseconds mark.

D. Comparison between energy and acoustiveness



Interpretation for the above graph:

• We see that if the energy of the song is high then the acoustic-ness value is low. It is possible that it is maybe a slow or a romantic song.

• We also observe that most of the songs in the dataset are grouped together in low acoustic and high energy range.

Results

Model 1: The first model is validated based on human feedback. The fact that the entire model is based on human intervention made it a little easier for us to actually verify whether our recommender engine works or not. Our recommendation system worked pretty well with this approach. The idea of making a user to rate songs and then training a model based on those song choices generated a list of probable audio features that the user subconsciously ends up liking is practical, viable and implementable. The practical aspect of this approach made it a very suitable one for at least smaller datasets. Cold start problem has been taken care of with this approach. Whenever a user doesn't have a prior playlist with tracks added to it, this approach would work just fine. However, this recommender engine would definitely work better if the user has some initial playlist with 'rated' tracks added to it to make recommendations even more personal and better.

It shows from the notebook that 'User1' preferred the predicted playlist from the Simple Linear Regression(SLR) model over the other 2 models(Random Forest and Neural Network), and the other two people didn't want intensive analysis, so we skimmed through the dataset and chose which they might like at random and they both chose the Simple Linear Regression as well. It appears that the SLR works best however with such a low sample size and subjective opinions, the results cannot be taken as exactly conclusive but important enough.

Model 2: The second approach is validated based on CV scores. In this approach, we addressed the problem through a classical binary classification model. We appended an extra column in our dataset called "in_playlist" which is binary with 0 meaning the track is not in playlist and 1 means that track is present in the playlist. We ran Logistic Regression, RandomForest, Gradient Boosting, K-NN & created a neural network as well to classify the tracks. We divided the dataset in train/test and did upto 10-fold cross validations on each of the models. We generated the R 2 Error values and compared them. The cross validated scores were used to validate the errors generated. Also, we used cross validation to get the optimal number of neighbors for our K-NN model.

Overall after the comparisons, we found that the Random Forest have an accuracy of ~0.99 on the test set which makes it a stand out model. Since, we did the cross validations, the risk of overfitting the model is also reduced. All the scores and measures are documented well in the jupyter notebooks.

Conclusion and Future Work:

Designing a personalised music recommender is complicated and it is challenging to thoroughly understand the users needs and meet their requirements. The future research direction will be mainly focused on user-centric music recommender systems. One of the surveys among athletes showed practition-ers in sport and exercise environments tend to select music in a rather arbitrary manner without full consideration of its motivational characteristics. Therefore, future music recommenders should be able to lead the users to reasonably choose music.

From the development of music recommenders over the past years, the results tend to be more personalised and subjective. Only considering the music itself and human ratings are no longer sufficient. A great amount of work in recent years has been done in music perception, psychology, neuro-science and sport which study the relationship between music and the impact of human behaviour. Due to the subjective nature in music, considering active and social information, emotion-based model and context-based model largely improve the quality of recommendation. However, this research is still at an early stage.

With the existing model we can think of following

1. Using the whole dataset to get more closer recommendations. It would require increasing efficiency and appropriate system architecture.
2. Iterative training of the model over the time to give even more accurate results.