Final Project report for IT350

Data Analytics

Submitted by

Saumya Karri (181ME273)

Sathvika B. Mahesh (181CV141)

Anshuman Sinha (181EE209)

VII SEM B.Tech

Under the guidance of

Dr. Nagamma Patil & Ms. Shruthi J R

Dept of IT, NITK Surathkal

in partial fulfillment for the award of the degree of

Bachelor of Technology

at

Department of Information Technology

National Institute of Technology Karnataka, Surathkal

November 2021

Table of Contents

Abstract

The idea of this project is to develop a music recommender system for the songs that a user has been listening to. The main goal of this recommender system is to suggest similar songs to the user based on some standard features, thus continuing the playlist with songs that the user might like. We have used the "Million Playlist Dataset" that was released by Spotify. Better the recommender system, the more the user is likely to keep using the application. Our task was to find the most accurate classification algorithm among the ones that exist in order to provide the best user-experience.

1. Introduction

In its most generic definition, a playlist is simply a sequence of tracks intended to be listened to together. The task of automatic playlist generation then refers to the automated creation of these sequences of tracks. Automatic music playlist continuation is a form of the more general task of sequential recommendation. Given a playlist of arbitrary length with some additional meta-data (for eg: album name, date of release, artists, etc.), the task was to recommend up to 500 tracks that fit the target characteristics of the original playlist.

While recommender systems have been around for a long time and are well researched, music recommender systems differ from their more common siblings in several important respects: the items are shorter (3-5 minutes for a song vs 90 minutes for a movie or months/years for a book or shopping item), the catalogue of items is larger (10s of millions of songs), the items are consumed in sequence with multiple items in a session.

The goal of this project is to provide automated playlist continuation, allowing any music platform (in this case, Spotify) to smoothly help its users in generating and expanding playlists by providing suggestions based on their choices and preferences. Furthermore, the recommender system requires just minimal information as input, such as the title of the playlist, the music currently in the playlist, and the artists associated with those tracks, rather than a vast and varied supply of user data.

2. Literature Survey

- "*Current Challenges and Visions in Music Recommender Systems Research* " M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, M. Elahi.
    - Paper link: https://arxiv.org/pdf/1710.03208.pdf
    - Gives a broad definition of the problem and provides many new scopes and potential innovations for further research in Music Recommender Systems.
- "*An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation*" H. Zamani, M. Schedl, P. Lamere, C.-W. Chen.
    - Base Paper link: https://arxiv.org/pdf/1810.01520.pdf
    - Lays out a complete summary of important metrics and approaches to solve the problem of Automatic Music Playlist Continuation

3. Methodology

1.1 Dataset

Process the dataset and find the correlation between the various tracks provided. Generates the next song in the playlist based on user preference using different classification techniques.

1.2 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 files, where each file contains 1000 playlists comprising 1,000,000 playlists in total. Authentication token required to get one of these objects, temporary token via Spotify's API can also be requested.

1.3 Data Understanding:

As mentioned, all the data is divided into 1000 files, each containing 1000 playlists. These are all JSON files and are in the form of dictionaries. These files follow a naming convention.

- Slice: The range of playlists contained in the file. Example: 0-999
- Version: Current version of MPD (currently v1)
- Created_on: Timestamp where the file or slice was generated

The command above extracts the data from all of the tracks. 'tracks' is a dictionary with fields like pos, artist name, track URI, artist URI, track name, album URI, duration (ms), and album name, as previously mentioned. It always includes a few examples of this. We've only looked at the json file formats and data at a basic level thus far. We'll then format the data and analyse some of the features to get it into the dataframe.

1.4 Data Pre-Processing:

Data before modelling, the dataset has to be pre-processed. In Microsoft Excel, some cleaning has to be done as follows:

- **Remove duplicates based on track ID** - there were obviously many duplicates because playlists don't all have separate songs because they were all combined.
- **Remove rows with headers -** (because they were all merged, every file that was merged had its own header row) After this, a combined CSV file will be created and processed.

1.5 Data Modelling:

The last step is data modelling which will be done using 6 classification models and the accuracy will be compared. Our ultimate goal is to return a list of any number of top-rated songs from our mode to them.

Classification models

1. Linear regression
2. Logistic regression
3. Random forest regression/ classification
4. Random forest classification
5. Gradient Boosting regression
6. K nearest neighbour classification

Linear regression

Linear Regression is a supervised Machine Learning approach for predicting continuous variables that is widely used. Linear regression assumes that the dependent and independent variables have a linear relationship, it seeks for the best-fitting line/plane for two or more variables.

$$Y = m_1 * x_1 + m_2 * x_2 + ... + m_n * x_n + c$$

Logistic regression

Logistic Regression is a supervised Machine Learning algorithm that helps in binary classification (separating discrete values).

Sigmoid function: $\mathbf{S(x)} = 1/(1 + e^{-x})$

Random forest regressor/ Random forest Classification

A Random Forest is an ensemble approach that uses several decision trees and a technique called Bootstrap and Aggregation, also known as bagging, to solve both regression and classification problems. Instead of depending on individual decision trees, the main idea is to aggregate numerous decision trees to determine the final outcome. Classification differs from regression in that it transfers the input data item to a set of discrete labels. Regression, on the other hand, converts the input data item into continuous real values.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x'.

$$\hat{f} = 1/B \sum_{b=1}^{B} f_b(x')$$

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x'.

$$\sigma = \sqrt{\sum_{b=1}^{B} (f_b(x') - \hat{f})^2 / (B - 1)}$$

Typically, for a classification problem with p features, √p (rounded down) features are used in each split. For regression problems the inventors recommend p/3 (rounded down) with a minimum node size of 5 as the default. In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

Gradient Boosting regression

Gradient Boosting for regression. GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

Algorithm 1: Gradient Boost

- Initialize $F_0(x) = \arg\min_\rho \sum_{i-1}^{N} L(y_i, \rho)$
- For $m = 1$ to M do:
  - Step 1. Compute the negative gradient

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F_{x_i}}\right]$$

  - Step 2. Fit a model

$$\alpha_m = \arg\min_{\alpha,\beta} \sum_{i=1}^{N} [\tilde{y} - \beta h(x_i; \alpha_m)]^2$$

  - Step 3. Choose a gradient descent step size as

$$\rho_m = \arg\min_{\rho} \sum_{i-1}^{N} L(y_i, Fm - 1(x_i) + \rho h(x_i; \alpha))$$

  - Step 4. Update the estimation of $F(x)$

$$F_m(x) = F_{m-1}(x) + \rho_m h(x; \alpha_m)$$

- end for
- Output the final regression function $F_m(x)$

K-Neighbors Classification

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

4. Implementation

1.6 Data Extraction

In this implementation, we have used one randomly chosen slice from playlist 20000 to 20999 (which consists of a total of 35247 tracks). We have the index, track name and track ID for each track in each slice of the dataset. The next step was to manually choose attributes like album name, track ID and duration and audio features like danceability, energy, acousticness etc from the Spotify API.

The above screenshot shows the list of attributes that we chose. Then we extracted the value of each of these attributes for each track and created a dataframe. This dataframe was named after the range of playlists that we initially used and

```
# from Spotify track object
album_id = []
album_name = []
album_release_date = []

track_artist_ids = []

track_id = []
track_duration_ms = []
track_explicit = []
track_name = []
track_popularity = []

# from Spotify audio features object
danceability = []
energy = []
key = []
loudness = []
mode = []
speechiness = []
acousticness = []
instrumentalness = []
liveness = []
valence = []
tempo = []
audio_features_id = []

# from Spotify artist object
artist1_id = []
followers = []
artist_popularity = []
artist_genre1 = []
artist_genre2 = []
artist_genre3 = []

# from Spotify album object
album_label = []
album_popularity = []
```

contained data of each track versus various attribute values. It was later exported into a CSV file.

	album_id	album_name	album_release_date	track_artist_ids	track_id	track_duration_ms	track_explicit	track_name	track_p
0	1gIuyEXICtt24CEKrYSOw8	The Piano Guys	2012	[0jW6R8CVyVohuUJVcuweDI]	4eYaDRhiL5iesFp2EuoODr	275973	False	A Thousand Years	
1	7o2VLivg95UduHjTMTIEIf	Be OK	2008	[2vm8GdHyrJh2O2MfbQFYG0]	5GBWQszdw6PtAN0Negzut6	193946	False	Can't Help Falling In Love - Recorded Live at ...	
2	5h7fx8ILwOZ3I5yQ4eGBl7	The Lumineers	2012-01-01	[16oZKvXb6WkQlVAjwo2Wbg]	1jdNcAD8Ir58RlsdGjJdx	161226	False	Ho Hey	
3	3h4pyWRJIB9ZyRKXChbX22	Call Me Irresponsible (Standard Edition)	2007-04-27	[1GxkXlMwML1oSg5eLPiAz3]	4T6HLdP6OcAtqC6tGnQelG	212373	False	Everything	
4	6QJw1UxLAFYQfy8XirsXiW	Losing Sleep	2009-01-01	[2PCUhxD40qlMqsKHjTZD2e]	1q74TesvRCIo5RvJM5B84F	146066	False	She Is Love	

1.7 Data cleaning

The album release dates were in a variety of different formats like MM/DD/YY or YYYY-MM. The first step was to standardize this and bring it to a standard format. We chose the YYYY format. After that, we dropped all the non-quant and garbage columns along with all the "NaN" values in cells. Since our attributes were distributed across a wide range of values, before proceeding further, we scaled them using z-score normalization.

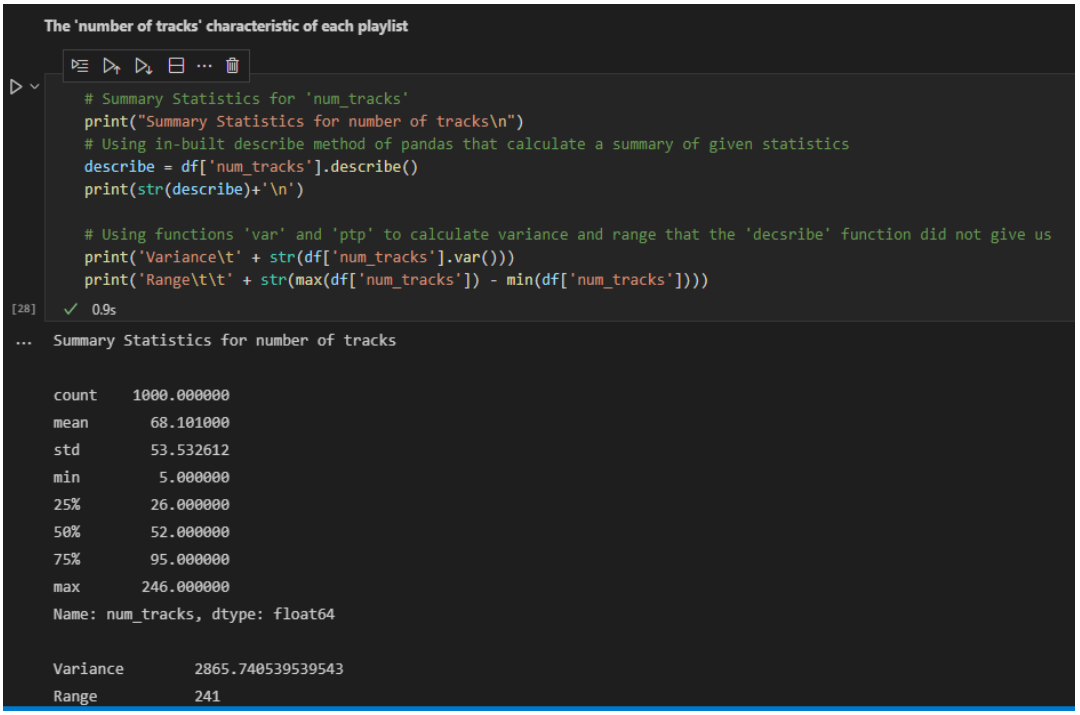
	track_duration_ms	track_popularity	danceability	energy	key	loudness	speechiness	acousticness
0	-2.381157	-2.358229	0.712777	-2.801331	0.508743	-1.908517	2.055703	2.673083
1	1.133484	-1.909557	-1.781186	-3.000556	1.065624	-6.857963	-0.155880	2.673083
2	-1.585774	1.031731	-0.283571	-2.953848	0.787183	-6.325566	-0.447248	2.673083
3	0.315446	-2.009262	-0.110293	-2.405265	1.065624	-3.916419	-0.212925	2.673083
4	-2.128138	-1.062068	-0.345456	-3.011518	-0.605017	-6.835130	-0.537642	2.673083

The above figure shows the normalized attribute values. For checking the correlation between these values, we plotted a heat map. It was found that the properties Loudness and Energy, and Album Popularity and Track Popularity were heavily correlated (greater than 0.75 in magnitude). Since all the parameters need to be independent, we decided to drop two of these. Finally, energy and album popularity were dropped since they were also highly correlated with a different predictor.

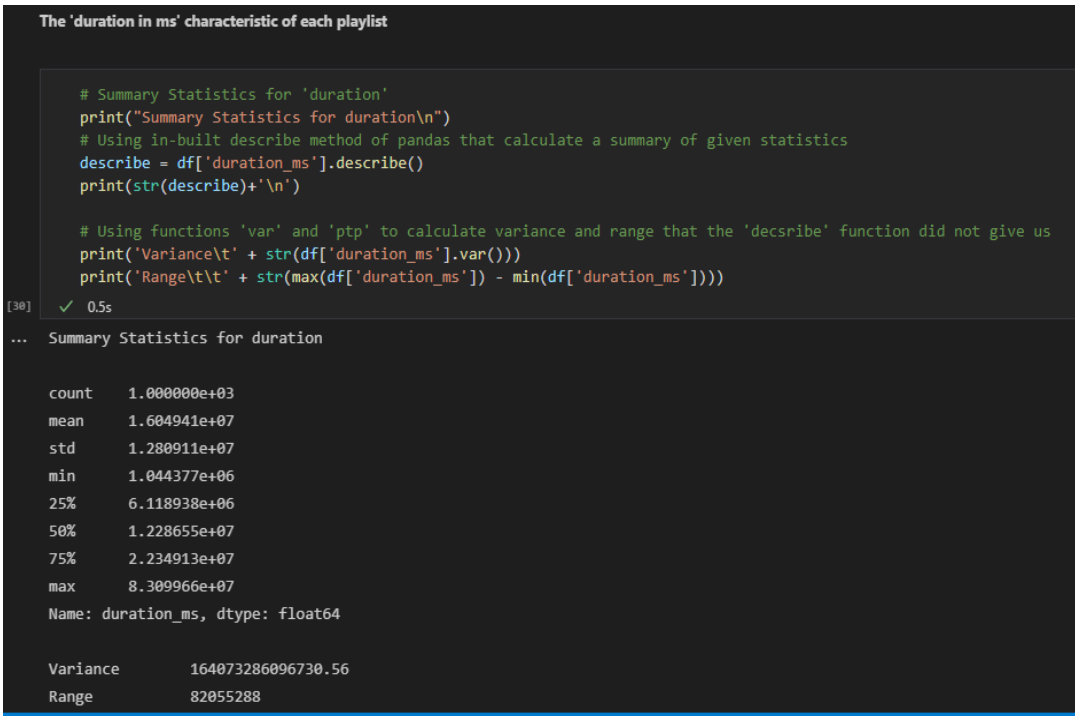
1.8 Data modelling

To find the best classification model for our recommender system, we decided to try out six different methods namely Linear Regression, Logistic regression, random forest classification,
Random forest regression, gradient boosting regression and k-neighbors classification. We train the model using each of these methods. After comparing the predicted ratings with the actual values, an accuracy score is allotted to each model. The model with the highest accuracy score will be chosen.

5. Snapshots with clear description



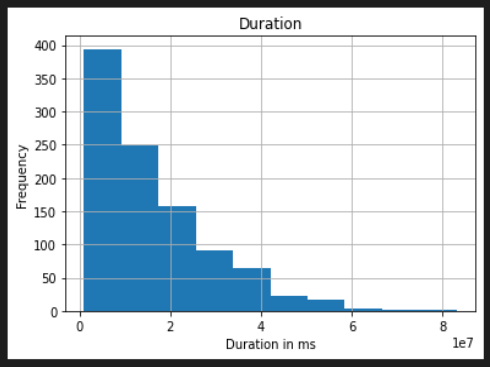
Summary Statistics of Tracks



Summary Statistics of Duration of each Playlist

```
Duration = df['duration_ms'].hist()
xlabel = Duration.set_xlabel("Duration in ms")
ylabel = Duration.set_ylabel("Frequency")
title = Duration.set_title("Duration")
```

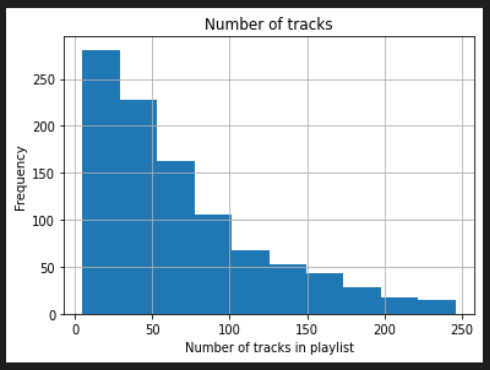
[31] ✓ 0.5s



Playlist Duration Histogram

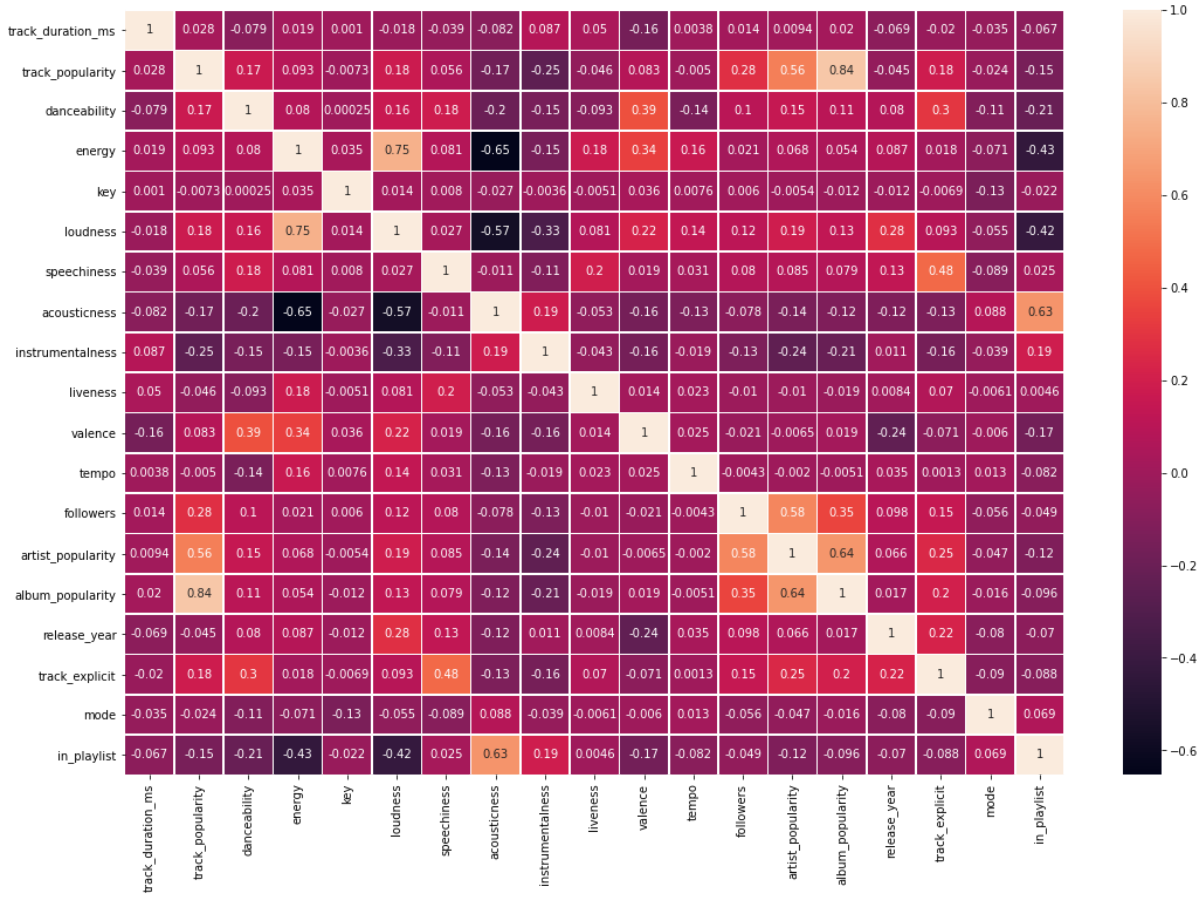
```
NTracksHist = df['num_tracks'].hist()
xlabel = NTracksHist.set_xlabel("Number of tracks in playlist")
ylabel = NTracksHist.set_ylabel("Frequency")
title = NTracksHist.set_title("Number of tracks")
```

[29] ✓ 1.3s



The 'duration in ms' characteristic of each playlist

Playlist Length Histogram

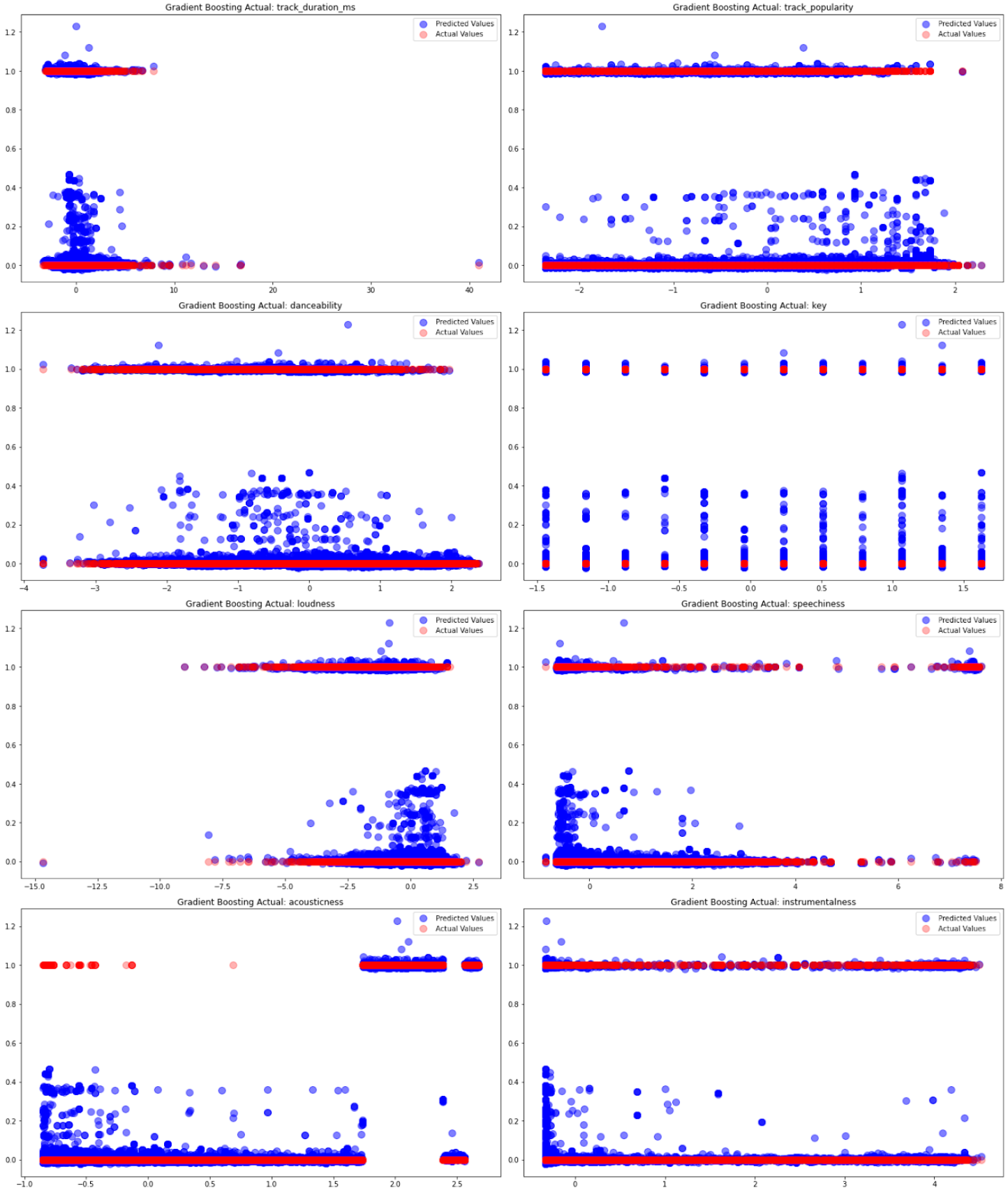


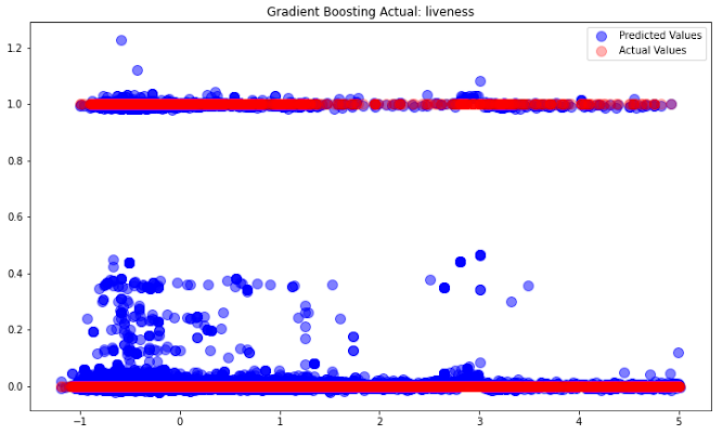
Track Feature Correlation HeatMap

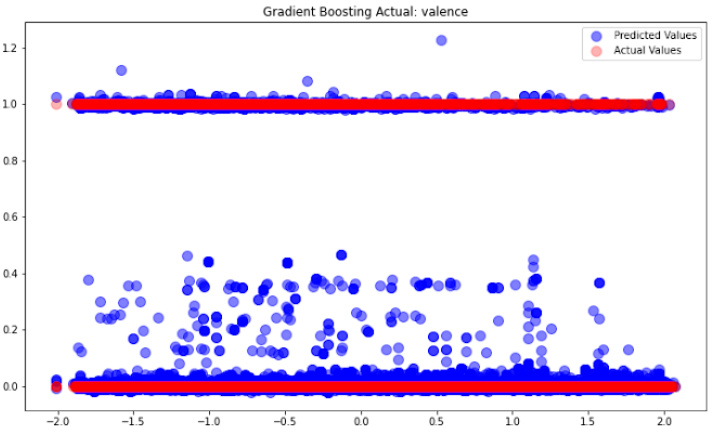
This graph shows the correlation among the 16 features per song of the chosen set of 1000 playlists.

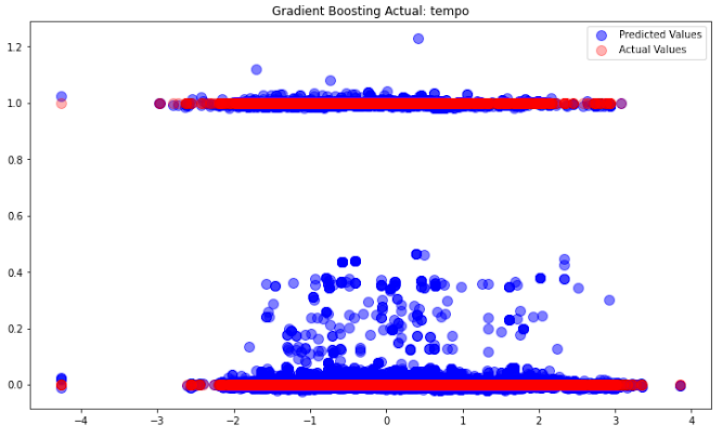
These are the scatter plots of actual vs predicted values for **Gradient Boosting Regressor**.

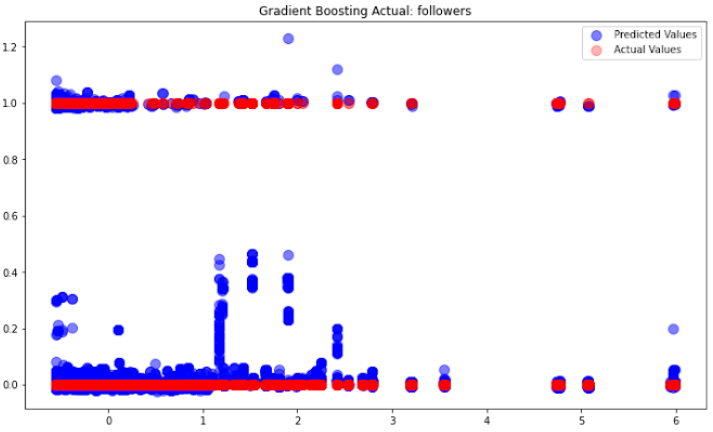
16 graphs for 16 features in the training data.

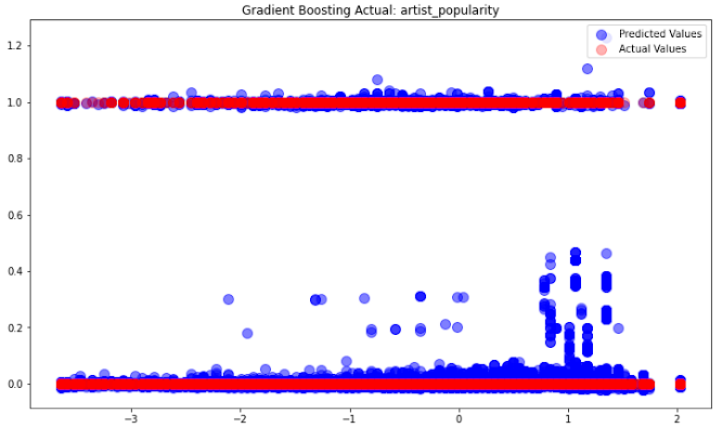


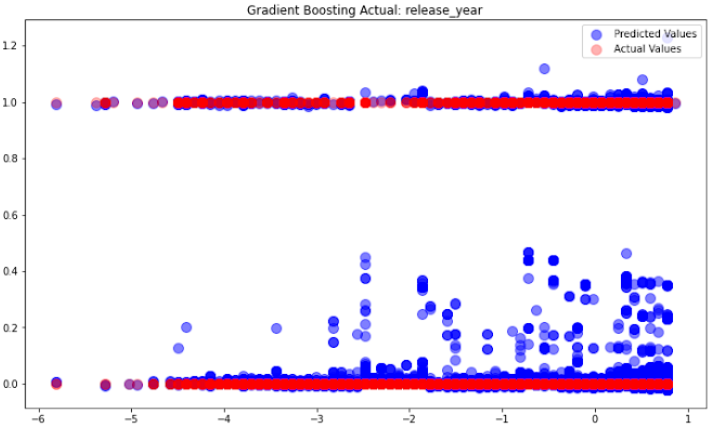

Gradient Boosting Actual: liveness
Predicted Values
Actual Values

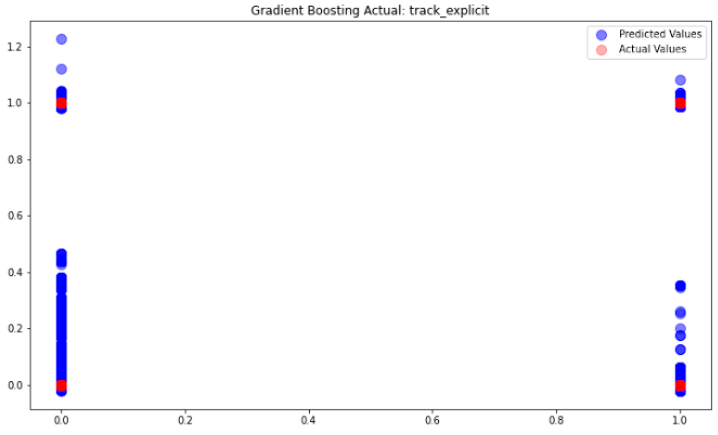

Gradient Boosting Actual: valence
Predicted Values
Actual Values

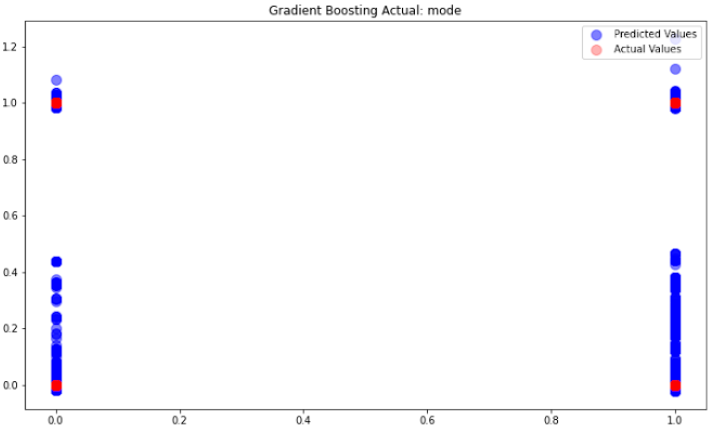

Gradient Boosting Actual: tempo
Predicted Values
Actual Values


Gradient Boosting Actual: followers
Predicted Values
Actual Values


Gradient Boosting Actual: artist_popularity
Predicted Values
Actual Values


Gradient Boosting Actual: release_year
Predicted Values
Actual Values


Gradient Boosting Actual: track_explicit
Predicted Values
Actual Values


Gradient Boosting Actual: mode
Predicted Values
Actual Values

6. Results and Analysis

Classification Method	Accuracy Scores	Confusion Matrix
Logistic Regression	0.95589	[[5749 120] [169 373]]
Random Forest Classifier	0.99988	[[5869 0] [8 534]]
Gradient Boosting Regressor	0.973254	[[5869 0] [8 534]]
K-neighbor classification	0.96064	[[5723 146] [154 388]]

We carried out 10-fold cross-validation and the CV score came out to be 0.95492. Further, it is clearly visible that Random Forest Classifier is the best method for this problem with a mean accuracy of 0.99988, highest among all the methods that were tried out. Hence, we finalize the Random Forest Classifier method for our recommender system.

7. Conclusion

Concept of recommendations boosts purchases and does not directly increase income for companies that provide music streaming services, it has similar indirect tangible impacts on user engagement and sustainability. Implementing and testing a better recommendation engine entails being able to communicate with each user individually. The more the accuracy, the greater the engagement, and the greater the money. Of course, there are plenty of other companies that offer similar services to Spotify, but the goal here is to make Spotify a little stand out from its competitors when it comes to song discovery. Making song discovery a really natural process for consumers of the services makes a lot of sense. Because of the massive data collection, there is no right or wrong approach to create a recommendation engine, lack of time spent iterating and training the models, our recommendation engine which uses random forest classification performed admirably in predicting the tracks under various scenarios.

8. References

[1] Spotify Challenge: https://recsys-challenge.spotify.com/readme

[2] Yiwen Ding,Chang Liu. Exploring drawbacks in music recommendation systems. Bachelor Thesis in Informatics, 2015.

[3] Yajie Hu. A model-based music recommendation system for individual users and implicit user groups. Dissertation, 2014.

[4] Spotify developer echo next API: https://developer.spotify.com/spotifyecho-nest-api/.

[5] Quartz Spotify: https://qz.com/571007/the-magic-thatmakes-spotifys-discover-weeklyplaylists-sodamn-good/

[6] Yifan Hu, Yehuda Koren, Chris Volinsky. Collaborative Filtering for implicit feedback datasets. 2009.

[7] http://benanne.github.io/2014/08/05/spotify-cnns.html