Final Project Report [Group #9]

Chelsea Chen, Mengyuan Li, Xinran Tang, Jing Xu

Abstract

Music plays an important role in the real life, and recommending suitable and popular playlists to users becomes a struggle for some music apps. Some previous articles had already come up with several models but the results were all not satisfying, so how to evaluate the popularity of a playlist remains unsolved. This paper aims to find out what makes a playlist popular and predicts the next hit focusing on data collected by Spotify.

keywords: Spotify | playlists | popular | XGBoost | Random Forest | LightGBM

1 Introduction

Recommending playlists for people is important to both apps and their users. On the one hand, providing songs that users prefer will win the favor of them, and also makes life much easier for someone struggling in choosing or cannot find any favorite. On the other hand, bad recommendation makes people feel frustrated or irritable.

Therefore, finding out the features a popular playlist may have becomes the centerpiece of this article. The aim of this paper is to work with the public data set on AIcrowd develop a predictive model for the popularity for a Spotify playlist. To make the problem easier, the popularity of a playlist is defined by the number of followers.

The following structure of the paper is section 2, description of original data; section 3, results from EDA; section 4, data cleaning section 5, feature engineering; section 6, models for predicting the popularity of playlists; section 7, conclusion.

2 Original data

The data is stored in multiple .json files. The basic structure is such that each observation is a playlist, and the attributes of a playlist are:

1. *name* [string]: The name of the user who created this playlist.
2. *collaborative* [boolean]: Whether or not the author of this playlist collaborate with others.
3. *pid* [integer]: The playlist ID for this user.
4. *modified_at* [integer]: The Unix Epoch Time when this playlist was lastly modified.
5. *num_tracks* [integer]: The number of tracks in this playlist.
6. *num_albums* [integer]: The numebr of albums in this playlist.
7. **num_followers** [integer]: The number of users that follow this playlist. This is the key attribute that measures the popularity of the playlist, and this project will focus on predicting on it to find out the next hit playlist.
8. *tracks*: list of track objects, containing

- *pos* [integer]: The position of the track in this playlist.
- *artist_name* [string]: The name of the artist.
- *track_uri* [string]: The unique track ID.
- *artist_uri* [string]: The unique artist ID.
- *track_name* [string]: The name of this track.
- *album_uri* [string]: The unique album ID.
- *duration_ms* [integer]: The duration of this track in milliseconds.
- *album_name* [string]: The name of the album.

3 EDA

In this section we will display several plots in order to provide readers a more intuitive impression of the data.

We first plot the distribution of the response variable.

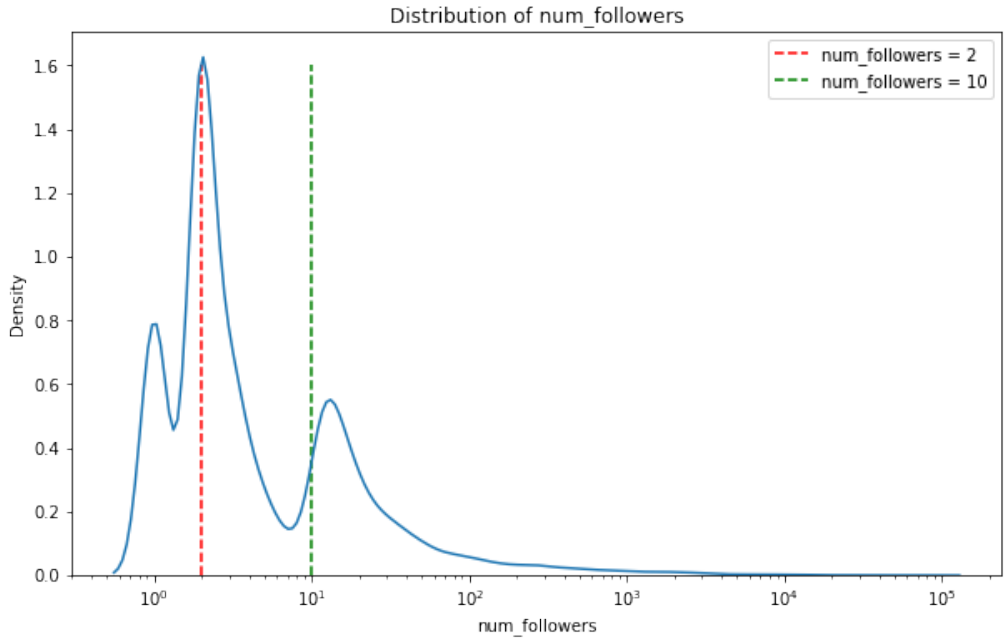
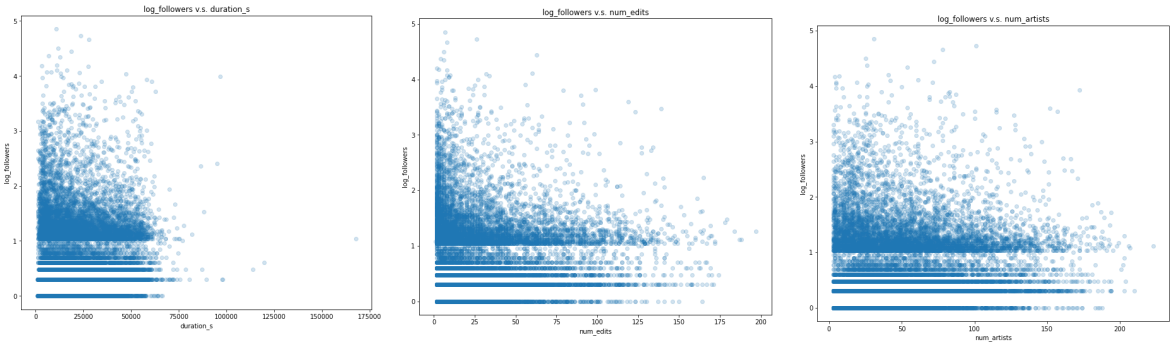


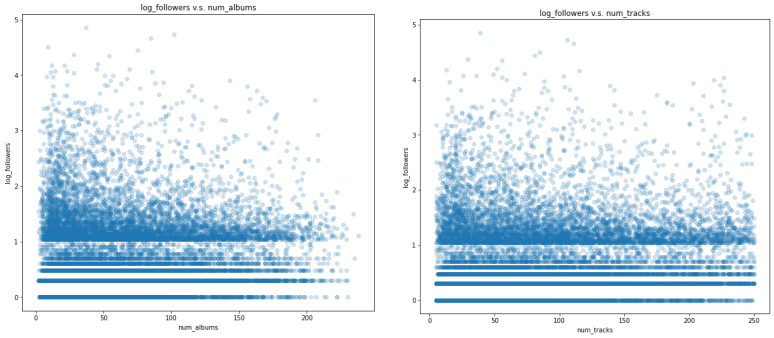
Figure 1. The density of *num_of_followers*.

From the Fig.1we can see that most of the playlists only have a few number of followers, in terms of numbers, lower or around one hundred. Also, from the position of peak density, single-digit followers forms the largest part in the data. This is reasonable since we can image that for most of the playlists, they are just "private" for users, not able to be a hit. However, for the popular playlists, we want to find their common features such as a specific theme, or contains a lot of famous artists.

Then want to have a look at the relationship between predictors and response. Due to the large scale of the response, we take log of it.



(a) log_followers v.s. duration_s (b) log_followers v.s. num_edits (c) log_followers v.s. num_artists



(d) log_followers v.s. num_albums (e) log_followers v.s. num_tracks

Figure 2. Relationship between numerical predictors and response.

For predictor *duration_s*, we can see a slight trend that longer playlist may correspond to fewer followers. And for predictors *num_edits, num_artists, num_albums*, the relationship related to the response is more obvious that more edits, artists or albums in the playlist, the fewer followers is it likely to receive. However, we point out that here larger number of duration, edits, artists and albums in a playlist has a smaller data set, thus exploiting these trends is not conclusive.

Predictor *num_tracks* shows the least relationship between number of followers since we cannot find a clear trend in the data. The data is almost evenly distributed, which means this variable might contain little information that we want.

Those observations tell us that users may more likely to follow a playlist that focus on several artists or albums, and does not have too many songs. This still make sense since we can imagine that many people want to choose songs that meet their taste, and similar types of artists and albums is a good choice. Also, an overly lengthy playlist will use up listeners' patience if they find the first few songs are not good.

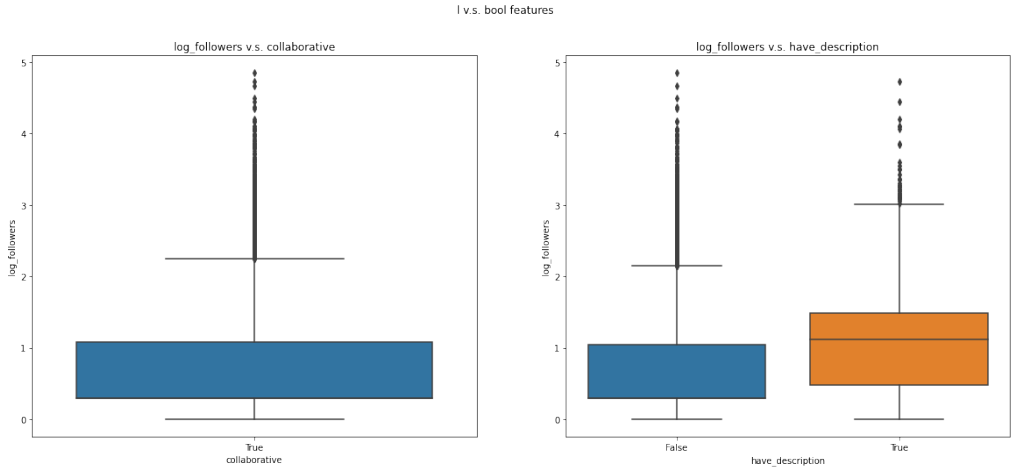
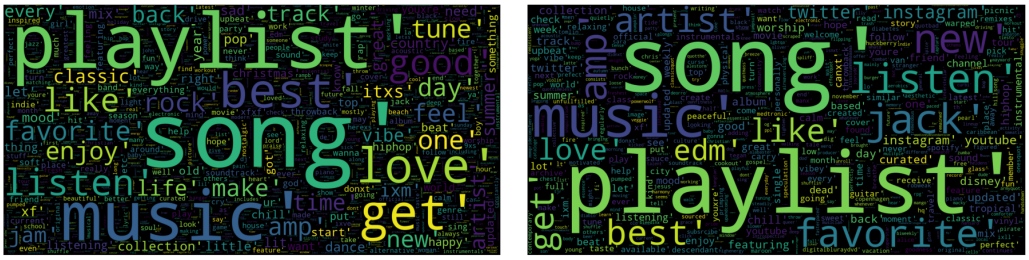


Figure 3. Relationship between numerical predictors and response.

Since all the playlists in the data set are collaborative, this variable is not indicative and will not be considered in our model.

For the playlists having description, they tend to have a greater number of followers since the mean of it is clearly higher. This shows that users are more willing to take look at playlists that have a description of the songs or theme they have. This is reasonable since strong descriptions make playlists more attractive and help people judge if they will love them. We can extract useful information, such as types or artists from briefs and it will drives us to listen.



(a) Word Cloud for all descriptions

(b) WordCloud for more popular playlists' description

(c) Word cloud for all artists

(d) Word cloud for all genres

Figure 4. Word Clouds for text features

At the first glance of the Fig.4(c), it's easy to see artists like Rihanna, ED Sheeran, BTS, Steely Dan,

Justin Bieber and so on. Fig.4(d) also has stand-out words such as pop, dance, rock and trap. Those names and genres are familiar to us and they are all well-known around the world. Looking at the word cloud, we are inspired that, it's possible that having songs from famous singers or tracks related to popular genres is related to playlist popularity. To further examine this issue, we later extract more data for artists and genres and do some feature engineering to create new features for our prediction model.

4 Data Acquiring & Cleaning

Considering the data from the data set is limited, we also retrieve other data that might be useful through Spotify API.

1. Through EDA We found that the playlist data set is highly imbalanced in terms of number of followers. We sub-sampled the playlists from the data set on AICrowd and down-sampled the majority playlists, which has only one follower to construct a less imbalanced data set with 33% playlists having one follower.

   We define playlists with more than 10 followers as the minority and less than or equal to 10 followers as the majority. For the majority, we downsampled them, making a final data set with 25% playlists as the minority.

2. For each playlist of the 335,860 down-sampled playlists set, we retrieve the audio features for the tracks in this playlist:

   - Danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature

   We collected 1,788,171 pieces of the tracks audio feature information in total.

3. For each track, we retrieve the artists information associated with the track, collecting 62,997 pieces of artists information. From the artists information, we retrieve the name and popularity of the artists included in the playlist, as well as the genres of each artist.

4. For each playlist, we retrieve the genres words related to the artists included in the playlist from artistsǵenres values:

   - The top 50 words that are most frequent are: album, alternative, art, atl, australian, blues, canadian, ccm, christian, classic, coast, contemporary, country, dance, edm, electro, electropop, folk, funk, gangster, gold, hip, holler, hop, house, indie, jazz, latin, mellow, metal, modern, neo, new, pop, post-teen, punk, rb, rap, road, rock, soft, soul, southern, stomp, trap, tropical, uk, urban, wave, worship.
   - We create Genres Scores based on artists' genres value counts and frequency of top genres words.

5. We thoroughly cleaned the constructed data sets and major impacts included removing duplicates and irrelevant observations, converting date string, handling missing data etc.

5 Feature Engineering

Based on intuition and domain knowledge, we do some feature engineering to transform some raw data into probably more informative forms.

1. For each playlist, we take the average across the tracks for each audio feature, so that each playlist will have features such as danceability being the average of danceabilities of its tracks.

    - Danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature

2. For each playlist, we create and compute homogeneity-related values, because we guess that if a playlist is more focused with a theme/topic, it might be more popular or the other way around.

    - How many tracks in a playlist belong to the same artist?
        * *track_artist_homo*: This is the ratio of num_tracks to num_artists; if the value is large, then a lot of tracks in the playlist belong to the same artist, meaning that the playlist has a focus on the artist.
    - How many tracks in a playlist come from the same album?
        * *track_album_homo*: This is the ratio of num_tracks to num_albums; if the value is large, then a lot of tracks in the playlist come from the same album, meaning that the playlist has a focus on the album.

3. Each artist is binded with a list of different genres. We pick the 50 genres words that have the highest frequencies showing up in the data set and use them as new variables.

    - For instance, a feature pop will have a score of the total frequency of word "pop" times the number of word "pop" from the artists' genres associated with all tracks within the playlist, and a value of 0 if the playlist does not have a pop track.

4. The popularity of playlists might be related to how long the playlists have existed or how often the playlists are modified. Though there are no exact match features related to this issue, we build a feature called modified_at_FROMNOW to measure the last modification time from now.

    - This feature is the difference between current time and the last modification time modified_at divided by some constant (so that the value is not too large).

5. The popularity of the artists of the tracks in the playlist might be related to the playlist's popularity. We created a new feature artist_popularity and its value is the average of the popularities of the artists included in the track.

Boxplot

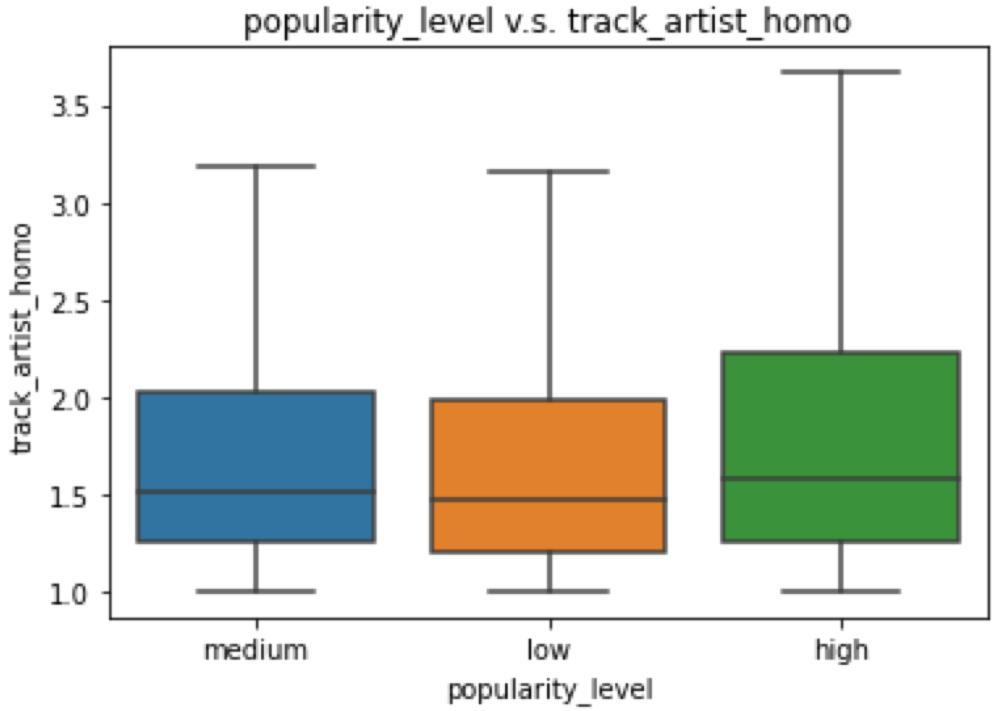


Figure 5. Boxplot between popularity_level and track_artist_homo, outliers removed.

Above we created a scatter plot to see whether or not homogeneity is related to the log number of followers. We also would like to see the difference in homogeneity among different playlist popularity groups. Note that at the beginning, we label playlists with only 1 follower low, playlists with 2 to 10 followers medium, and playlist with more than 10 followers high. Below are the boxplots for homogeneity-related features across different playlist popularity groups.

Tukey's Test

	group1	group2	Diff	Lower	Upper	q-value	p-value
0	medium	low	0.243755	-0.340731	0.828240	1.384321	0.583750
1	medium	high	0.391818	0.120063	0.663572	4.785904	0.002138
2	low	high	0.635572	0.069626	1.201519	3.727752	0.023152

Figure 6. Table for Tukey's Test for track_artist_homo among groups.

To further examine whether or not there is a homogeneity difference among playlist popularity groups, we perform Tukey HSD tests. We choose a 95% percent confidence and if a p-value is smaller than 0.05, we mark the respective difference between the groups significant.

According to the table, for track_artist_homo, there are significant differences of homogeneity between low popularity playlists and high popularity playlists, and between medium popularity playlists and high popularity playlists.

6 Model

1. MinMaxScaler is used to scale the train and test data for all the models.
2. Multiple Linear Regression
   - It's always good to first try with simpler models to see how the data performs in the prediction task.
   - Result: About 8% on test.
3. Random Forest
   - As our multiple linear regression model is not performing very well, we would like to try with more complex models, and RF is also a commonly used algorithm. Random Forest is a bagging method that can efficiently decrease the variance of the model, which can minimize the problem of overfitting. Random Forest can achieve great performance but it also has some disadvantages. Since RF will ensemble multiple trees together, it will take a huge amount of time if we want to do grid search to fine tune the model.
   - We do Cv grid search on hyper parameters.
   - Result: About 12% on test.
4. XGBoost
   Why choose XGBoost:
   Compared to Gradient Boost, XGBoost has advantages in two fields:
   - Better performance: XGBoost uses a more regularized model formalization to control over-fitting. To be more specific, XGBoost stands for eXtreme Gradient Boosting, which is a scalable system based Tree on Boosting Method. Similar to GBDT, the basis decision maker (CART) of xGBoost is serial; the difference is that the former only expands the loss function Taylor to order 1, while the latter expands Taylor to order 2. The objective function to be optimized in step t is as follows:

$$\mathcal{L}^{(1)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)\right) + \Omega(f_t), \qquad (1)$$

   where $\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$. Among them, (1) represents the form of the loss function, $y_i^{(t-1)}$ represents the prediction of x_i from the first $t-1$ tree samples, f_t represents the t-th tree (note that only CART here), and represents the regular term used for the t-th tree. After expanding (1) Taylor to the second order, the form is as follows [1]:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i)\right] + \Omega(f_t), \qquad (2),$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$. Then we have

$$\hat{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 = \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T, \qquad (3),$$

where (3) is the quad function of prediction w_j, therefore, the minimize of (3) is at

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i \lambda},$$

thus the loss prediction for the t-th tree is

$$\hat{\mathcal{L}}^{(t)}(q) = \frac{1}{2} \sum_{i=1}^{T} \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \qquad (5)$$

- Less training time and memory consumption: Better data structure and algorithm, better processor cache utilization and support for multicore processing which boost the speed, which is more suitable to train huge datasets and do grid search to fine-tune models.
- Result: about 11% on test.

5. LightGBM

- Why choose LightGBM:
  Similar to XGBoost, LightGBM is also an optimized version of Gradient Boosting, but it uses a different split algorithm: GOSS (Gradient-based One-Side Sampling) [2].
  LightGBM is much faster than XGBoost since XGBoost applies level-wise tree growth where LightGBM applies leaf-wise tree growth, and leaf-wise is mostly faster than the level-wise. So LightGBM can be used to do grid search among more values in less time. [2]
  But leaf-wise is also more likely to overfit, so the accuracy of LightGBM may be lower than the accuracy of XGBoost.
  We use it as we have very limited computation power but want to try more hyper-parameters.
- Result: about 11% on test.

7 Conclusion

For this project, we would like to explore what makes a Spotify playlist popular and predict on the number of followers for the playlists. We make use of data sets from AICrowd and also retrieve possibly useful data through Spotify API. Considering the imbalance of the data, with most of the playlists having very few number of followers and only a few playlists having many followers, we performed downsampling. We also do some feature engineering based on intuitive and domain knowledge to create new variables to be used to fit the models.

The variable have_description has the highest feature importance, meaning that whether or not the playlist has a description is important for whether or not the playlist is popular. The following important features are duration_s, the total time of the tracks in the playlist, and num_edits, the number of times the playlist has been edited. The feature track_artist_homo also shows up as a important feature in most of the models.

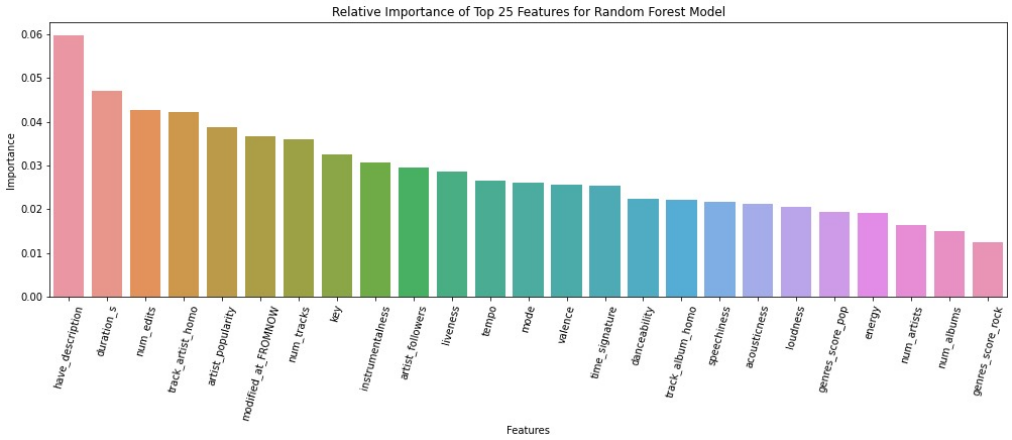


Figure 7. Feature importance.

However, even the best model has a really poor performance in predicting the log number of followers of the playlists. There are still much space for improvement, either by getting more informative data or building better models.

We speculate that playlist popularity can be related to playlist creators' popularity. If we are able to retrieve features related to users who created the playlists, we might be able to improve the performance of the model. However, if this strongly boosts model performance, models are less focused on exploring what playlist characteristics make them popular, and perhaps popularity of a playlist does not really depend on its own characteristics but more on the popularity and publicity of the creators, and this might mean that there is much space for Spotify to improve the recommendation system so that playlists created by non-popular users can also get noticed.

References

1. T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

2. G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.