A Preliminary Study of Automatic Playlist Continutation and Add-to-playlist Prediction

Li-An Yang A53242336 and Mingxiang Cai A53270893
CSE 258 FA18

***Abstract*— Nowadays, music streaming platforms account for 75% of the music industry revenue, with Spotify retaining 83 million premium account users worldwide. The market is still expanding as more personalized services come into play. In this manuscript, we explore preliminary results from two tasks that can be closely integrated into a music streaming service: *Automatic Playlist Continuation* and *Add-to-playlist Prediction*.**

I. INTRODUCTION

Automatic playlist continuation (APC) is a feature on Spotify platform which plays additional tracks after reaching the end of a playlist. We could view it as solving another recommendation task based on playlist information. Next, add-to-playlist prediction (ATP) is a binary classification problem and ask questions such as whether or not this track truly belongs to the playlist in reality given information at both track and playlist level.

The rest of this manuscript is ordered as follows. In section II, we introduce and analyze our data source and the data collection process. In section III, we present some common feature set representation methods. We then move on to illustrate our approach on APC in section IV, and on ATP to model track belongingness in section V. Next, experimental results are shown in section VI, discussion about potential future research directions regarding music recommendation in section VII. Lastly, we briefly investigated on related work and conclude in section VIII and IX respectively.

II. DATASETS AND ANALYSIS

A. Million Playlist Dataset

As its name suggests, Milliion Playlist Dataset [1] contains information of 1,000,000 playlists created by users on the Spotify platform. Each playlist in the MPD contains a playlist title, a list of tracks with track metadata, editing information (last edit time, number of playlist edits) and other miscellaneous information. The key statistics of the dataset is shown in table I.

It is also noticeable that the distribution of the number of tracks and followers both follow power law. According to Figure 1, 98.01% of the 1,000,000 playlists have fewer than 5 followers, and 78.35% of the playlists have fewer than 50 songs.

B. TrackInfo Dataset

The TrackInfo Dataset contains features of all the unique songs appeared in the Million Playlist Dataset. The feature vector of a track can be separated in two parts, audio features and track popularity. More specifically, the audio features are crawled using Spotify API with Python and contains information of danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and duration for each track. Detailed descriptions of these features can be found at the Spotify API documentation page [2].

t

Statistics	Number
Number of unique tracks	2,261,593
Number of unique artists	295,860
Number of unique albums	734,684
Max number of tracks in one playlist	376
Average number of tracks of in one playlist	66
Max number of followers in one playlist	71, 643
Average number of followers in one playlist	2.6

TABLE I
Statstics of the Million Playlist Dataset

C. Test Set Design

To evaluate our models' performance on APC and ATP, a test set is built in the following way. First, we collect 216,482 playlists that contain equal or more than 100 tracks to form the test set. The rest of the 978,352 playlists become the training set. Next, we build a positive and a negative set for every playlist in the test set as gold standard.

1) Positive (held-out) Set Design: For every playlist in the test set, 50 tracks are randomly chosen and held out. These tracks are guaranteed to have been observed at least once in the training set. The positive set will then be used to examine the performance of APC and as positive label in ATP scenario.

2) Negative Set Design: To generate negative set for ATP settings, we randomly draw $negSize$ tracks from database, yet excluding those that originally exists in that test data. These tracks form a "negative track set" used in the predictive task of ATP as entries with a negative label. In our

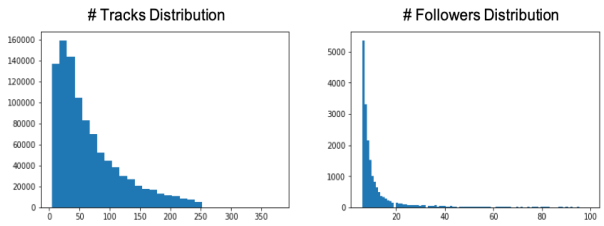

Fig. 1. Distribution of the number of tracks and followers in MPD.

experiment, we choose $negSize = 50, 100, 200$ to examine the impact on model performance. We increase $negSize$ under the assumption that in reality if we randomly pick a track and a playlist, it is very unlikely that the track is in the playlist. Therefore, to model such situation and to maintain similiar distribution between observed and unobserved data, it is necessary to experiment on different values of $begSize$.

Hence, the resulting test set consists of 216,482 playlists, each of which has 50 held-out tracks for ground truth and $negSize$ tracks to test on ATP.

III. Feature Representation

Generally, there are two common ways to represent features in recommendation task, constructing either user/item features separately or to involve interplay information in within. In this specific study, the former refers to track and playlist audio features as well as popularity. The latter refers to the inclusion relationship between tracks and playlists, namely, whether a track is added to a playlist. As shown in section II, the scales of the TrackInfo Dataset and the Million Dataset both resides at a million level. The resulting playlist-track matrix will not only be huge but also extremely sparse. Hence, with limited computational power and a purpose of conducting a preliminary study, our work primarily use audio and popularity features as model inputs to avoid cold-start issues.

To elaborate, a track-level feature vector simply contains ten audio features plus the popularity index, such as

$$(danceability, acousticness, tempo, ..., popularity)$$

As for the design of a playlist-level feature vector, we adopt two variations, $feature_{median}$ and $feature_{percentile}$. The first one look for the median of all the features to form a vector of 11 dimensions.

$$[m(danceability), m(acoustiness), ..., m(popularity)]$$

where *m* calculates the median of the specified audio feature.

Another way to represent the distribution of track information in a playlist is to first sort the list of values of a specific feature, and compute the percentiles of 0%, 25%, ..., 75%, 100%.

$$[dist(danceability), dist(acoustiness), ..., dist(popularity)]$$

where each *dist* returns a list of values of audio features at each specified percentile.

In summary, $feature_{median}$ represents a playlist by the medians of each fields, while $feature_{percentile}$ represents a playlist by the value distribution of each of its feature.

IV. Automatic Playlist Continuation

To cope with playlist continuation problems, we outline our approach as follow.

- Construct playlist-level or track-level feature vectors from training and test set.
- Adopt k-means clustering on the training set to construct K clusters of playlists or tracks.
- Assign each test playlist its closest cluster based on distance in the feature space.
- For each test playlist, rank the tracks in its corresponding cluster based on similarity and recommend top tracks accordingly.

Again, the task of APC is to recommend a relatively small number of tracks to complete the playlist, where 50 tracks are held out. Because there is a library of 1,532,406 tracks to choose from, we decide to narrow down the range of candidate tracks through clustering. While there are a handful of clustering algorithms to choose from, we use K-means algorithms because it requires less computation and is easy to interpret. Based on K-means, we propose two levels of clustering strategies, playlist-level clustering (soft clustering) and track-level clustering (hard clustering).

A. Playlist-level clustering

Playlists serves as the minimal unit for playlist-level clustering. We adopt $feature_{percentile}$ strategies to build playlist-level vectors. After clustering, we transform each cluster of playlist to a cluster of tracks by flattening the list of tracks in each playlist assigned to the cluster. Note that one track might show in multiple playlists residing in different clusters. Hence, We also call it a soft clustering method.

B. Track-level clustering

Tracks serves as the minimal unit for track-level clustering. The feature representation strategy used to compute distance is $feature_{median}$. Note that one track is assigned to only one cluster; hence, we also call it a hard clustering method.

In our experiment, we have tried different number of clusters (K = 10, 20, 40) to construct various numbers of clusters. We then first try to recommend all the tracks in the assigned cluster. The next step is to only recommend the top *N* tracks in a cluster as playlist continuation and intersect with the corresponding held-out set of tracks. The evaluation results are presented in section **??**.

C. Efficiency index

To evaluate the performance of APC, we define the following metrics:

$$overlapping\ rate = \frac{|\{recommended\ tracks\} \cap \{heldout\ tracks\}|}{|\{heldout\ tracks\}|}$$

$$selectivity = \frac{|\{recommended\ tracks\}|}{|\{candidate\ tracks\}|}$$

$$efficient\ index = \frac{overlapping\ rate}{selectivity}$$

Particularly, a higher overlapping rate and a lower selectivity is desirable, leading to a higher efficient index. In other words, a higher efficient index means we are able to match more held-out tracks with fewer recommendations.

V. Add-to-playlist Prediction

In ATP(Add-to-Playlist Prediction), our task is to predict whether the track is added to the playlist given a pair of (track, playlist) and its metadata. Note that this is a typical binary classification problem. We use two commonly used models to perform the task.

A. Logistic Regression

Logistic model is one of most widely used classification algorithms. Its simplicity and efficiency make it a good fit for this task, which scales up to millions of entries of data. In practice, we use L2 norm for regularization parameter lambda. And there exists two variations of the probability functions:

$$P_1(track \in playlist) = \sigma(\vec{w_1} * feature[track] + \vec{w_2} * feature_{median}[playlist] + \alpha * dis(track, playlist) + \beta)$$

$$P_2(track \in playlist) = \sigma(w_1 * feature[track][popularity] + w_2 * feature_{median}[playlist][popularity] + \alpha * dis(track, playlist) + \beta)$$

where

$$dis(track, playlist) = ||feature[track] - feature_{median}[playlist])||_2$$

Notice, P_1 includes all the features of tracks and those of playlists, stacking up a 23-dimension feature vector. P_2 includes only the popularity for track features and playlist features, constructing a 3-dimension feature vector.

B. XGBoost Classifier

XGBoost[3] is an optimized distributed gradient boosting library that has brought success in many previous Kaggle competitions. One evident advantage of using XGBoost is capable of discovering useful features from interpretable results and assign correspondent weights to them along the way of boosting trees enumeration. In practice, a few parameters we use are learning rate (0.3), max depth (5), min childweight (1), and gamma (1). Similar to the logistic regression classifiers, we also have two variations of the model inputs with different length of feature vector.

$$featureVector_1 = (feature[track], feature_{median}[playlist], dis(track, playlist)$$

$$featureVector_2 = (feature[track][popularity], feature_{median}[playlist][popularity], dis(track, playlist))$$

As mentioned, $featureVector_1$ has 23 dimensions and $featureVector_2$ has 3 dimensions.

Type	Avg. cluster size	Overlapping rate	Selectivity	Efficiency index
Track, K=10	153,240	45.8%	10%	4.58
Track, K=20	76,620	23.4%	5%	4.68
Track, K=40	38,310	9.3%	2.5%	3.7
Playlist, K=10	326,208	92.9%	21.3%	4.36
Playlist, K=20	217,388	89.7%	14.2%	6.32
Playlist, K=40	138,648	86%	9%	9.51

TABLE II

APC statistics between different types of clustering. Note that an overlapping rate of 86% means hitting 46 tracks out of the 50 heldout tracks.

VI. Result

A. Automatic Playlist Continuation

The results of mapping our heldout set with all the tracks in its closest cluster is shown in table VI-A. We compare several statistics between different values of K as well as between track-level and playlist-level clustering. From the table, we can see that playlist-level clustering outperforms track-level by a wide margin. When K equals to 40, soft clustering method can cover 86% of the heldout set on average with less selectivity, as opposed to 9.3% of tracks using hard clustering. Figure 2 and Figure 3 shows the cluster size distribution on both training data and test data for the best performance in terms of efficiency index on track-level and playlist-level clustering. It is already obvious from the shape of the plot that playlist-level clustering results in higher efficiency index.

We then adopt the playlist-level clustering with K=40 for the next phase. We rank the tracks in each cluster based on similarity with $features_{median}$ of the heldout set and select top results out for recommendation. The relation between the number of hits and recommendations is shown in Figure VI-A. As observed from the plot, our approach only locates around 15 tracks out of 50 tracks on average among the set of 100,000 recommended tracks. The results indicates that

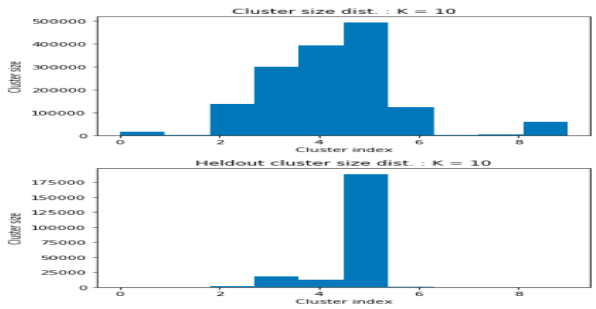

Fig. 2. Track-level clustering with K = 40. Recommend on average 5% of total tracks in training data to achieve an overlapping rate of 23.4% heldout tracks, effenciency index: 4.68

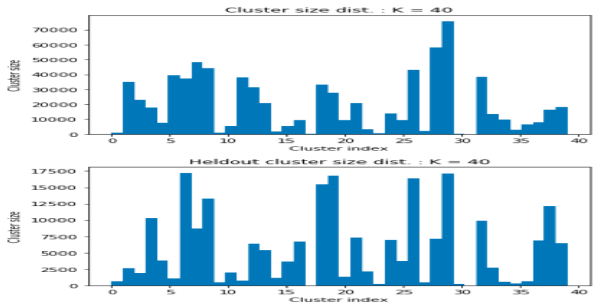

Fig. 3. Playlist-level clustering with K = 40. Recommend on average 9% of total tracks in training data to achieve an overlapping rate of 86% heldout tracks, effenciency index: 9.506

#Features	negSize	Model	Accuracy	Precision	Recall
3	50	XGBoost	0.81	0.85	0.76
23	50	XGBoost	0.84	0.88	0.79
3	100	XGBoost	0.83	0.74	0.75
23	100	XGBoost	0.85	0.78	0.79
3	50	Logist. Reg.	0.8	0.79	0.81
23	50	Logist. Reg.	0.81	0.78	0.83
3	100	Logist. Reg.	0.83	0.69	0.77
23	100	Logist. Reg.	0.84	0.69	0.81

TABLE III

Performance of ATP under difference parameter sets

the granular task of hitting the exact heldout track is still challenging and far-fetched.

B. Add-to-playlist Prediction

The results are shown in Figure III. We can discover that using XGBoost with more features is able to improve the accuracy because the framework is famous for automatic feature selection, whereas there is little difference when we apply logistic regression under same condition. Additionally, it is also obvious that precision and recall drops gradually as we increase $negSize$. Our models tend to give negative results as the dataset becomes more imbalanced. The precision further drops to 64% with XGBoost when we set $negSize = 200$ for sampling, despite having a 87% accuracy.

VII. Discussion

According to our preliminary results, it is generally easy to locate the right cluster of tracks for automatic playlist continuation, yet rather difficult to match the right heldout track. In terms of the Add-to-playlist challenge, XGBoost and logistic regression model deliver an accuracy of at least 80% with the risk of imbalanced classification as we try to approximate reality by scaling $negSize$. Inspired by these discoveries, we discover potential research topics and interesting applications to work on in the future.

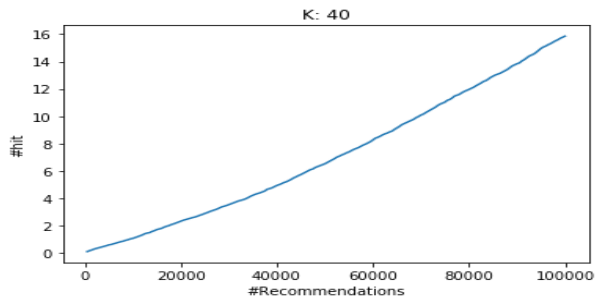

Fig. 4. #Hit v.s. #Recommendations, playlist-level clustering.

A. The application of Add-to-playlist Prediction

One meaningful application for ATP to improve user experience is when a user hits like for a song, the predictive model can automatically suggest the most appropriate self-curated playlists to add the song to (or gives a range of top playlists for user to choose from).

On the other hand, if we want to recommend a song to a friend, but are not quite sure about his/her preference. It might be a better idea to first estimate the effectiveness of the recommendation by modeling the add-to-playlist feasibility given any of his/her playlists. This is especially important if we want to impress an important person.

B. Interplay between APC and ATP

For the task of ATP, we can exploit our method in APC to make predictions. Specifically, we first apply track-level clustering on the training set. Next, we assign each playlist to its closest cluster c with $feature_{median}$ strategies. Lastly, for each (track, playlist) pair, we check whether this track is in c and make positive prediction if it does, negative otherwise.

APC and ATP can also be pipelined in a sequence of recommendation process. For instance, if a certain track recommended by APC is liked by the user, we can adopt ATP to suggest the top N user-curated playlists to which users can add the liked track. Namely, it is a full set of recommendation process from tracks to playlists.

C. Feature Importance Examination

After the training phase of XGBoost, the framework is capable of plotting feature importance to visualize the relative weights of each field taken into consideration for predictions. From Figure 5 we can observe that track information is attached with a lot more importance than that of playlist. However, the similarity between track and play is surprisingly unimportant in the decision process of prediction.

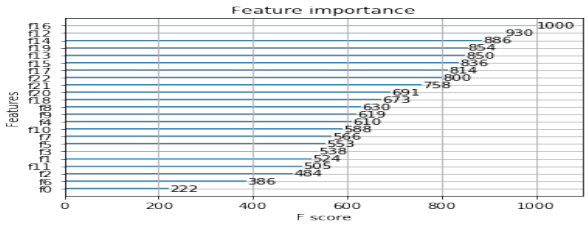

Fig. 5. Feature importance plot for the features used in XGBoost. F0 is the similarity, F1-11 and F12-22 represent playlist and track features, respectively.

D. Popularity Prediction

As the track popularity can be crawled from Spotify API to serve as ground truth. We can also model the popularity level to access the ”viralness” of any particular song. The problem can be formulated either as a linear regression or classification model with popularity intervals. For playlist popularity, we discover that playlist's number of followers have similar implication.

As far as feature is concerned, we can adopt similar strategies as described in our work to construct feature vectors. The problem seems easy to overfit and thus requires careful parameters tuning.

VIII. Related Work

The ACM Recsys challenge[4] in 2018 was held around the topic of automatic playlist continuation. The participants are required to provide a list of 500 ranked tracks for each playlist to match the varied number of heldout tracks. The competition differs from our ACP task in that the number of heldout track in each playlist varies, as opposed to having a fixed-sized heldout size in our experiment. The best team of the challenge is capable of matching an average 20% of the heldout tracks. Namely, for 100 heldout tracks, it is capable of matching 20 tracks among 500 recommended tracks.

IX. Conclusion

In this manuscript, we have explored preliminary results in music recommendation on a streaming platform. We first examine the feasibility of automatic playlist continuation and recommend a list of similar tracks to extend the playlist. Next, we model the belongingness of a track to a playlist to tackle the challenge of add-to-playlist predictions. While the results might not surpass state-of-the-art methodologies, we discover much potential in this area and are confident that these insights can help build a more powerful music recomender system to enhance user experience.

X. Appendix

References

[1] The Million Playlist Dataset
[2] Spotify API Dumentation
[3] Chen, Tianqi, and Carlos Guestrin. ”Xgboost: A scalable tree boosting system.” Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.
[4] https://goo.gl/ThT1UoRecSys Challenge 2018 By Spotify