Spotify Playlist Continuation
CSE 258: Web Mining and Recommender Systems
Assignment 2

Drishti Ramesh Megalmani
A59001657
dmegalma@ucsd.edu

Satvik Gupta
A59012252
sag005@ucsd.edu

Piyush Yadav
A59003976
p1yadav@ucsd.edu

I. INTRODUCTION

A list or sequence of audio tracks that are listened together makes up a playlist. The task of automated creation of these sequences of tracks is called automatic playlist generation. In this context, the ordering of songs in a playlist is often emphasized as a key feature of automatic playlist generation, which makes it a highly complex task. Considered a variation of automatic playlist generation, the task of automatic playlist continuation (APC) consists of adding one or more tracks to a playlist in a way that fits the same target characteristics of the original playlist. This has benefits in both the listening and creation of playlists: users can enjoy listening to continuous sessions beyond the end of a finite-length playlist, while also finding it easier to create longer, more compelling playlists without a need to have extensive musical familiarity.

In this project, we aim to devise a recommender system to effectively provide music recommendation and music playlist continuation using playlist and track-level metadata. Here, we describe the dataset used, pre-processing steps and experiments we performed with various heuristic based, learning based and hybrid models. We perform a comparative analysis of the various proposed models based on a list of defined evaluation metrics.

A. Literature Review

The Spotify Million Playlist Continuation Challenge consisted of two tasks- main and creative. Participants in the main track were only allowed to use the dataset provided by the challenge for training their models. In contrast, participants in the creative track were required to use external resources, such as public datasets, for solving the same task. The top scorer [3] in this challenge used a two-stage architecture. In the first stage, they used a Weighted Regularized Matrix Factorization (WRMF), and in the second stage they implemented a gradient boosting learning algorithm XGBoost to rank model. In addition to the output of the WRMF model, few models were used to produce features for the XGBoost model. These models include a convolutional neural network for playlist embedding, user-user and item-item neighborhood-based collaborative filtering models, and a set of hand-crafted features. The cold-start instances (those that only consists of a title with no track) were handled separately. For such cases, the team used a matrix factorization on top of the playlist titles.

An overall comparison of top performers in the challenge shows that several teams utilizied a multistage architecture for playlist continuation task. They also used matrix factorization, as a dominant collaborative filtering approach. The matrix factorization algorithms used by the top teams include weighted regularized matrix factorization (WRMF), LightFM with a weighted approximate-rank pairwise (WARP) loss, and Bayesian personalized ranking. Some teams used neural networks to do the task effectively. simple feed-forward networks for predicting tracks given each playlist, convolutional models for playlist embedding or extracting useful information from playlist titles, recurrent neural networks and in particular long short-term memory networks for modeling the sequence of tracks in the playlists, autoencoders for learning playlist representations etc,. Some top performing teams used information retrieval techniques mainly developed for the ad-hoc retrieval task. For instance, inverse document frequency (IDF) weighting, TF-IDF weighting, BM25 weighting, and relevance model.

B. Similar Datasets

1) Playlist Dataset: This dataset was collected by Shuo Chen (shuochen@cs.cornell.edu) from Dept. of Computer Science, Cornell University. The playlists and tag data are respectively crawled from Yes.com and Last.fm. Yes.com is a website that provides radio playlists from hundreds of radio stations in the United States. The collection lasted from December 2010 to May 2011. This lead to a dataset of 75,262 songs and 2,840,553 transitions.

2) Melon Playlist Dataset: The Melon Playlist Dataset [8] is a public MIR dataset provided by Kakao Corp. Using the data from Melon, the most popular music streaming platform in Korea, and used in research by Music Technology Group. The Melon Playlist Dataset consists of 148,826 playlists in the training set and 649,091 songs, it also contains genre information for the songs and tag information for the playlists. The number of unique tags is 30,652, the number of unique genres is 30 and sub-genres is 219. For all the songs, the mel-spectrogram representation of a segment (20-50s) of the audio is provided which enables the possibility of applying content-based approaches.

3) The Million Song Dataset Challenge: The Million Song Dataset Challenge [9] is a joint effort between the Computer Audition Lab at UC San Diego and LabROSA at Columbia University. The user data for the challenge, like much of the data in the Million Song Dataset, was generously donated by The Echo Nest, with additional data contributed by SecondHandSongs, musiXmatch, and Last.fm. The task was to predict the listening history for 110k users having given the first half of the 110k users(10k validation, 100k test set) and a training set of 1M users with their full listening history.

II. Dataset Exploration

A. Dataset Description

The Spotify Million Playlist Dataset Challenge consists of a dataset and evaluation to enable research in music recommendations. It is a continuation of the RecSys Challenge 2018, which ran from January to July 2018. The dataset contains 1,000,000 playlists, including playlist titles and track titles, created by users on the Spotify platform between January 2010 and October 2017. The evaluation task is automatic playlist continuation: given a seed playlist title and/or initial set of tracks in a playlist, to predict the subsequent tracks in that playlist.

B. Dataset Pre-processing

Even though the MPD dataset is well structured and clean, it proved to be too big compared to the compute power the team had. We randomly sampled our training, validation and test data(table 1). We constrained our datasets by selecting playlists with number of tracks range between 20 and 60.

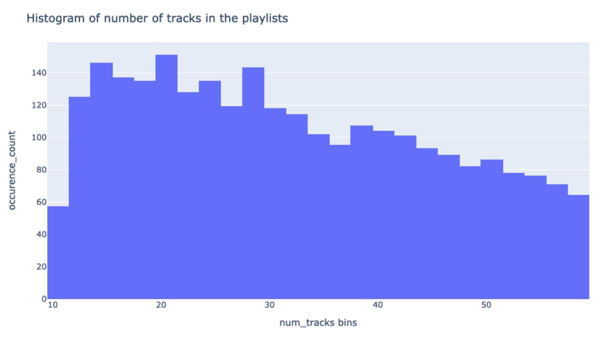
Dataset	Playlists	Tracks
Training	2656	40399
Validation	280	5917
Test	120	2463

TABLE I
DataSets

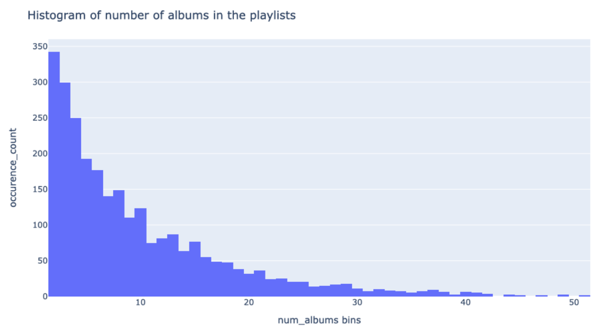
Leveraging our sampled datasets we created different exploratory pickle files such as artist-track-count, track-frequency which proved to be the core of our heuristic based models.

C. Exploratory Analysis

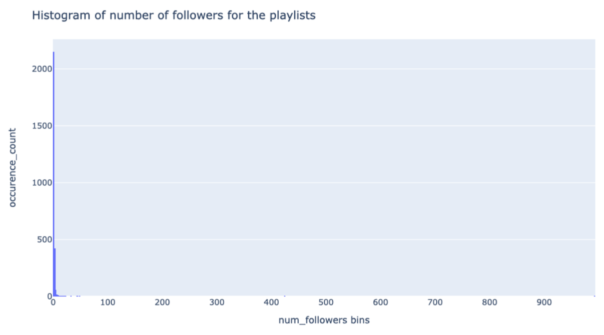
We analyse and visualise the data using plotly, to get a better feel of the data and what all columns and features are populated well enough or can be a good enough feature. After several plots and figures, here we share the top few important charts showing the essence of dataset we have chosen. As mentioned, we chose 2600 playlists based on the number of tracks present in the playlist. Following is a histogram showing the healthy distribution of tracks across playlists.

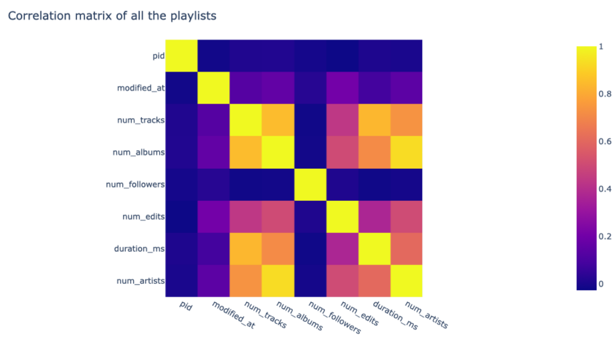


To gauge the album distribution in playlists, we can look at the histogram showing how many albums are present in all the playlists,



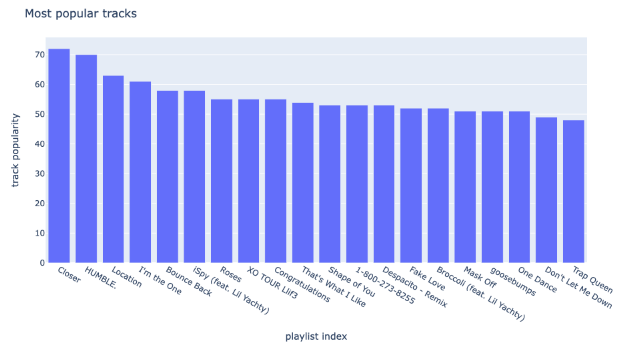
Next, we were hoping to use num_followers as a strong feature to depict popularity of the playlists, thereby impacting popularity of the tracks. Unfortunately, that data is sparsely filled and couldnt be used as a feature. Most playlists have 0-3 followers, while 1 playlist has 900 followers. Correlation is apparent from the correlation matrix as shown below the histogram of num_followers.

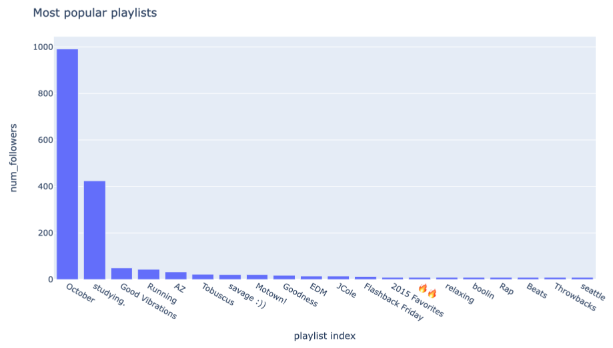


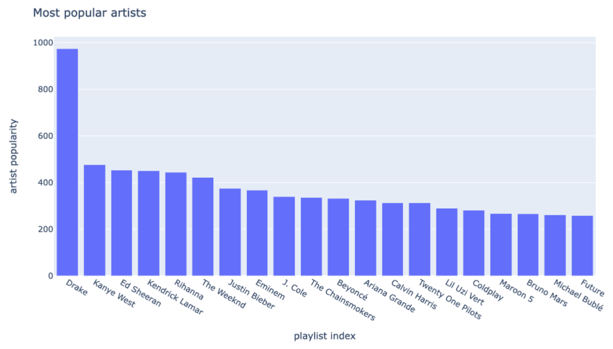


Even when we removed the outlier to work with the cohesive set based on this feature, the num_followers didnt

turn out be a strong feature as most values lied between 0-3 followers.
Finally, following show the popularity of tracks, playlists and artists in our dataset.







III. Methodology

A. Predictive Task Description

For every playlist in the test set, we receive the playlist title and a number of audio files in that playlist. Our defined predictive task is to predict 10 new songs that capture the mood of the playlist and are best continuation audio tracks for the playlist. If we have n audio tracks in a test playlist, we set aside, 10 tracks to evaluate our predictions upon, and use the rest (n-10) tracks as seed, the playlist title and all of their metadata to perform the predictive task.

B. Feature Extraction

We extract and utilize various features like popularity based heuristics, similarity between tracks, artists, normalized playlist and track name based latent factors and many more. We explain them in further sections and how we experimented with combinations of these types of features to train our recommender systems.

C. Evaluation Metrics

In the following, we denote the ground truth set of tracks by G and the ordered list of recommended tracks by R. We have defined 4 metrics to evaluate our models and their performance:

1) R-precision: R-precision is the number of retrieved relevant tracks divided by the number of known relevant tracks (i.e., the number of withheld tracks). In other words, this metric gives the ratio of the number of audio tracks recommended by our model are actually relevant in the considered test playlist.

$$R - Precision = \frac{|G \cap R_{1:|G|}|}{|G|}$$

2) R-precision with Artist: Here, the quality of the recommended continuation playlist is quantified based on the criterion that the recommended songs are by the same artists as in the test playlist. Due to the sheer number of tracks, a 1-1 mapping of the tracks is tough to get. Hence, we incorporate artist match to widen the sample space. The intuition behind this is that if a playlist contains a song from a particular artist, it's likely to contain other songs from the same artist.

3) Normalized Discounted Cumulative Gain (NDCG): NDCG measures the ranking quality of the recommended tracks, increasing when relevant tracks are placed higher in the list. Normalized DCG (NDCG) is determined by calculating the DCG and dividing it by the ideal DCG in which the recommended tracks are the actual ground-truth tracks, perfectly ranked.

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2 i}$$

$$IDCG = 1 + \sum_{i=2}^{|G \cap R|} \frac{1}{\log_2 i}$$

$$NDCG = \frac{DCG}{IDCG}$$

4) NDCG with Artist: We calculate the NDCG as explained above in the same manner, but we compare the Artists of the recommended tracks and the test playlist set.

D. Baseline 1: Most Similar Playlist

This data seems like something which should have a strong correlation with similarity and popularity as a feature. Hence, in this model, we get the most similar playlist to the current one during testing using Jaccard similarity. Then, we find the top K songs from the most similar playlists. We then find the R-Precision and NDCG for these tracks with respect to the withheld tracks(test data). This model is our lowest baseline and we work upwards from here. The evaluation metrics for this model can be seen from Table II.

topK	RP(A)	RP(T)	NDCG(T)	NDCG(A)
10	0.002	0.001	0.004	0.008
20	0.006	0.002	0.006	0.015
30	0.008	0.003	0.007	0.018
40	0.013	0.003	0.008	0.025
50	0.013	0.004	0.007	0.024
60	0.015	0.004	0.008	0.026
70	0.016	0.004	0.008	0.025
100	0.02	0.009	0.013	0.029
200	0.059	0.016	0.018	0.060
300	0.094	0.022	0.022	0.080
400	0.126	0.031	0.030	0.094
500	0.143	0.037	0.034	0.099

TABLE II
BASELINE 1 PERFORMANCE, A=ARTIST, T=TRACK

E. Baseline 2: Most Popular Artist

One intuitive baseline solution for the problem at hand is choosing from the tracks of the most popular artist. During preprocessing we store the most popular playlists, tracks and artists in different dictionaries. Hence, for this baseline model, we predict from these tracks randomly, of the most popular artist for each playlist in the test and validation data. We, then go on to find the values for our evaluation metrics given this heuristic and get the values as shown in Table III:

topK	RP(A)	RP(T)	NDCG(T)	NDCG(A)
10	0.039	0.043	0.003	0.110
20	0.067	0.071	0.005	0.130
30	0.085	0.092	0.006	0.140
40	0.103	0.110	0.007	0.147
50	0.121	0.129	0.008	0.152
60	0.137	0.146	0.009	0.157
70	0.152	0.163	0.009	0.162
100	0.194	0.204	0.011	0.175
200	0.283	0.293	0.014	0.192
300	0.342	0.352	0.017	0.197
400	0.386	0.396	0.020	0.200
500	0.418	0.430	0.023	0.202

TABLE III
BASELINE 2 PERFORMANCE, A=ARTIST, T=TRACK

The performance for the previous 2 baselines can be attributed to the data that we are working with. The exhaustive dataset contains 1 million playlists. However, we only choose around 2600 playlists to work with, given the computation power limitations and the time constraints. This implies the most similar playlist for our randomly chosen 2600 playlists may most likely not be the actual most similar playlist/track to the current data during testing. We see this issue being predominant for our baselines.

F. Model 1: Word Embeddings

Intuitively we should be able to get some thematic idea from the playlist title. For instance playlist title 'Wedding Songs' gives a pretty good idea about the theme of the songs it may contain. Building upon this idea we define title embedding vectors for each playlist title in training set. For this task, (a) we tokenize playlist title, (b)leverage Stanford's GloVe[10] to calculate cumulative title embedding vector, (c) for each test playlist we calculate the cosine similarity matrix with all training playlist titles and choose topN playlist matches, (d) the model returns weighted number(cosine similarity) of random songs from each playlist (listing 1) summing to topK predictions(table IV). A possible tweak in this model is to use artist information just like Baseline 1 model. However, the performance of the tweaked model was not at par with original idea. Which allows us to think that same artist do not have multiple popular songs for same theme or there are good number of artist choices within same theme for the user.

```
deno = sum([p[0] for p in simMat)
  result = []
  for p in simMat:
    wt = ceil(topK*p[0]/deno)
```

Listing 1. Weighted prediction count

topK	RP(A)	RP(T)	NDCG(Track)	NDCG(A)
10	0.039	0.008	0.028	0.112
20	0.073	0.017	0.049	0.154
30	0.081	0.023	0.052	0.156
40	0.114	0.035	0.069	0.180
50	0.118	0.039	0.064	0.165
60	0.137	0.046	0.072	0.177
70	0.151	0.048	0.074	0.187
100	0.181	0.071	0.089	0.192
200	0.236	0.096	0.097	0.204
300	0.266	0.109	0.102	0.206
400	0.275	0.116	0.104	0.198
500	0.281	0.119	0.106	0.200

TABLE IV
MODEL 1 PERFORMANCE, TOPN=10, A=ARTIST, T=TRACK

Scalability is one of the major challenges for this model. Our cosine similarity matrix grows linearly with number of playlists and this leads to bad response time. But compared to our baselines this model performs better and that is because we have constrained our suggestion pool and reduced randomness a lot.

G. Model 2: Alternating Least Square Matrix Factorization

Certain problems that we face with recommendation using either a knn based approach or heuristic based approach are popularity biases, item cold-start problem and scalability issues. Matrix factorization is the factorization of a matrix into a product of matrices. In the case of collaborative filtering, matrix factorization algorithms work by decomposing the user-item interaction(here, we consider user as the playlist and item as the song) matrix into the product of two lower dimensionality rectangular matrices. Model learns to factorize the rating matrix into playlist and song representations, which allows model to predict better personalized songs for playlists. With matrix factorization, less-known songs can have rich latent representations as much as popular songs have, which improves recommender's ability to recommend less-known songs. Alternating Least Square (ALS) is a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is built for a larges-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

The basic intuition behind this model was a learning based approach that would incorporate the titles of both the playlists as well as the seed songs in the playlist. We derived a list of normalized playlist and song title names by converting the text to lowercase, stripping, splitting and stemming the words. We created a corpus of 1371 playlist normalized names and 16861 normalized track names. We utilized these normalized track names as features to form the latent factors for our model. For each playlist in the set, we normalized the title and assigned a value of 1 for each of the words. To convert this task into a classification task, due to the missing ratings for prediction, we performed negative sampling for each playlist. We created an equal number of playlist-song negative pairs as the already present playlist-song positive pairs. For each playlist name pin the test set, we created a p-trackname latent feature for every track in our pool our tracks, and tested for whether the model predicts for the presence or absence of that song in the playlist p. This does become computationally intensive. As the number of songs increase the size of the vectors increases. The predictions of the trained model on the test set were sorted and the topK songs were considered as recommended songs. This was evaluated on two metrics i.e R-Precision(Artist) and NDCG(Artist). A possible tweak we can do for this model is a combination of popularity and latent factor based model. For every playlist, we can shortlist certain similar playlists or even utilize the seed songs to get similar songs and use those songs as a smaller sample space to run our predictive model on.

topK	RP(Artist)	NDCG(Artist)
100	0.06074	0.09248
200	0.13171	0.13397
300	0.19005	0.15656
400	0.25731	0.17795
500	0.31287	0.18881

TABLE V
MODEL 2 PERFORMANCE:VALIDATION

topK	RP(Artist)	NDCG(Artist)
100	0.06518	0.08061
200	0.12780	0.12747
300	0.19588	0.15679
400	0.25120	0.16947
500	0.29686	0.17486

TABLE VI
MODEL 2 PERFORMANCE: TEST SET

H. Model 3: N Most Popular Tracks (and artist weighted)

As mentioned in section 2(b), we leverage track frequency to determine track popularity. For every test playlist we return topK most popular songs discarding the seed songs already present in playlist. As can be seen from the statistics in table VII, the R-Precision and NDCG for track match are 0 for all topK. This just goes to show that most of our popular songs lie in the seed tracks already present in the playlist or dont lie in the playlist altogether, since we drop predictions whichever already exist in the seed tracks. We further enhanced this model to factor artist information into track recommendation. Instead of returning topK popular songs we return weighted count of songs for each artist present in the seed playlist. Note that, here we do not discard seed tracks. This model improved R-precision with artist as well as tracks a bit(table VIII).
Challenge with this model is that it is too random for our prediction task. Most of the times, the popular songs end up being part of seeded track list and therefore the next topK songs, most of the times will not have any overlap with withheld(test) data. Results get better when we incorporate artist weighted popularity, thereby showing affinity of users to artists.

topK	RP(A)	RP(T)	NDCG(T)	NDCG(A)
10	0.039	0.0	0.0	0.119
20	0.081	0.0	0.0	0.159
30	0.115	0.0	0.0	0.186
40	0.134	0.0	0.0	0.189
50	0.146	0.0	0.0	0.188
60	0.157	0.0	0.0	0.189
70	0.174	0.0	0.0	0.194
100	0.222	0.0	0.0	0.207
200	0.326	0.0	0.0	0.223
300	0.384	0.0	0.0	0.223
400	0.433	0.0	0.0	0.227
500	0.459	0.0	0.0	0.226

TABLE VII
MODEL 3 PERFORMANCE, A=ARTIST, T=TRACK

topK	RP(A)	RP(T)	NDCG(Track)	NDCG(A)
10	0.114	0.034	0.094	0.256
20	0.197	0.069	0.153	0.285
30	0.233	0.106	0.181	0.293
40	0.306	0.134	0.206	0.286
50	0.377	0.175	0.217	0.280
60	0.382	0.206	0.210	0.267
70	0.390	0.224	0.218	0.356
100	0.410	0.238	0.225	0.335
200	0.415	0.291	0.235	0.392
300	0.430	0.308	0.266	0.372
400	0.455	0.322	0.269	0.400
500	0.468	0.355	0.301	0.412

TABLE VIII
MODEL 3 PERFORMANCE, ARTIST WEIGHTED, A=ARTIST, T=TRACK

TeamName	R-Precision	NDCG
vl6	0.2241	0.3946
hello world!	0.2233	0.3932
Avito	0.2153	0.3845
Creamy Fireflies	0.2201	0.3856
MIPT_MSU	0.2167	0.3823

TABLE IX
STATE OF THE ART PERFORMANCE, TOPK = 500

IV. RESULTS AND DISCUSSION

We start off with baseline model 1 with results as low as 0.143 and 0.099 for RP(a) and NDCG(A) respectively. We build on top of this model by using various features based on intuition and data exploration, and we see that popularity and artist name are strong features, but artist similarity seemed to be a better indicator since NDCG for the artist URIs had a better value than that of the tracks. Model 3 performs the best where we use track popularity as well as artist popularity in that playlist as heuristics to predict the tracks for continuation. This goes to show that, for our data, the users are more likely to listen to the same artists that they have been listening to in their playlists, given that we work with popular tracks. We can provide a well rounded and cohesive results by using more playlists that were part of the global data, so that our heuristics stand on the whole set. We can also use a more complex ensemble model incorporating all these models in a weighted manner to get a better overall precision and NDCG.

A. *Comparison with the state-of-the-art*

Here, we compare our recommender system algorithms and results with the top scoring teams of the Spotify Playlist Continuation Challenge (Table IX)

- Considering the fact the we worked with only 0.2% of the total challenge dataset, we lose the essence of the data due to subsampling. With use of the entire dataset and more computational power to handle the large scale data, we can provide better and more robust models and thereby results. To reduce the computational load, we randomly subsample a small chunk of the whole dataset, whereas the state-of-the-art top scorer performed dimensionality reduction by utilizing features extracted from the whole dataset.
- We proposed simple heuristic based and learning based models whereas, the state-of-the-art model was a two-stage architecture that utilized multiple complex models like convolutional neural networks for feature extraction and ensemble model training.
- Our best model is still at par with the state of the art results and sometimes performs better too, but this is only when we compare higher topK values. For eg, if we compare results for topK = 500 of our Model 3 and the corresponding values in table IX, we can see that our NDCG(Artist) is almost as good and our R-Precision(Artist) is better. This is attributed to the fact that we are considering top 500 popularity from a set of only 40K tracks, while the actual data for state of the art models was 2M tracks.

B. *Future Scope*

Reproducing the results for the entire dataset would be the first step. Tom improve the results, we can take combine the different parts of the proposed models that worked and devise an ensemble model. To perform some form of dimensionality reduction, we can combine the popularity heuristic metric with the latent factor learning model. For each test playlist, we can sample the most popular or similar artists/ playlists and predict the song recommendations over this smaller sample space.

We can also utilize the open access Spotify API that provides audio characteristics of each song like 'danceability', 'energy', 'loudness', 'acousticness' and 'liveness' which would provide track-specific audio features that can be used to find tracks of a similar vibe. It is also to be noted that, the creative track (can use the Spotify API) of the challenge had slightly worse results than the main track (without using Spotify API). This maybe because including a lot more of the information make the problem more complex and difficult to generalize over all the data.

V. CONCLUSION

We use a very small portion of the global dataset, owing to computation and time constraints. However, we pre-process data to ensure cohesiveness given these conditions, by choosing a good threshold of number of tracks in each playlist and splitting data into validation and test data appropriately to have a fair model in place. Using data visualisation, we look at the correlation and distribution of some features to judge which ones have a better essence of the data. Then, we try several models, and set 2 baselines to begin with. We then go on to improve the performance of our models starting from the baseline to our best model - which works with track popularity and artist popularity. Previous section compares our models with the state of the art models as published by the actual participants of the Spotify Playlist Continuation Challenge. Through our experiments and results we conclude that popularity of track and artist seem to be the dominant features for our model, on point with our intuition.

REFERENCES

[1] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. Recsys challenge 2018: automatic music playlist continuation. In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18). Association for Computing Machinery, New York, NY, USA, 527–528.

[2] Domokos M. Kelen, Dániel Berecz, Ferenc Béres, and András A. Benczúr. 2018. Efficient K-NN for Playlist Continuation. In Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–4.

[3] Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. 2018. Two-stage Model for Automatic Playlist Continuation at Scale. In Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18). Association for Computing Machinery, New York, NY, USA, Article 9, 1–6.

[4] Malte Ludewig, Iman Kamehkhosh, Nick Landia, and Dietmar Jannach. 2018. Effective Nearest-Neighbor Music Recommendations. In Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18). Association for Computing Machinery, New York, NY, USA, Article 3, 1–6.

[5] Shuo Chen, Joshua L. Moore, Douglas Turnbull, Thorsten Joachims, Playlist Prediction via Metric Embedding, ACM Conference on Knowledge Discovery and Data Mining (KDD), 2012.

[6] Joshua L. Moore, Shuo Chen, Thorsten Joachims, Douglas Turnbull, Learning to Embed Songs and Tags for Playlists Prediction, International Society for Music Information Retrieval (ISMIR), 2012.

[7] Shuo Chen, Jiexun Xu, Thorsten Joachims, Multi-space Probabilistic Sequence Modeling, ACM Conference on Knowledge Discovery and Data Mining (KDD), 2013.

[8] Ferraro A., Kim Y., Lee S., Kim. B., Jo N., Lim S., Lim S., Jan J., Kim S., Serra X. Bogdanov D. (2021). “Melon Playlist Dataset: a public dataset for audio-based playlist generation and music tagging”. International Conference on Acoustics, Speech and Signal Processing (ICASSP 2021).

[9] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.