Spotify Million Playlist Dataset Challenge

COL764: A Course Project

Karan Tanwar(2017CS50411) **Gohil Dwijesh**(2017CS50407)
Pratyush Pandey(2017EE10938)

January 15, 2021

1 Introduction

Spotify Million Playlist Dataset Challenge is based on the ACM RecSys Challenge 2018. In this challenge, we are given training playlists and challenge playlists. Our task is to recommend 500 tracks for each of the challenge set playlists. In this project, we try to replicate the results of Kaenen Team(3^{rd} in creative track and 7^{th} in main track) by implementing subset of thier algorithms: Session-based Nearest Neighbor and String Matching. We briefly describe the theoretical aspects of above two algorithms in section-4. In section-5, we describe implementation aspects of above two algorithms including validation set creation for hyper-parameter tuning. We also submit the recommendations recommended by our system to the challenge website[3](with team name "zeus") and get placed at 1^{st} rank with a tie on the leaderboard[4]. We also compare our scores with SOTA[2] SKNN(session-based nearest neighbor) in section-7 and successfully replicate similar metric-scores. Finally, we conclude challenges faced and findings in section-8.

2 Dataset Description

The Dataset consists of 1 Million playlists divided in 1000 slice files. These files have the naming convention of: **mpd.slice.STARTINGPLAYLISTID-ENDINGPLAYLISTID.json**. For example, first 1000 playlists' metadata is stored in mpd.slice.0-999.json. Each playlist has attributes such as pid, name, num_tracks, num_artists, list of tracks, etc. The dataset also includes the challenge set queries in *challenge_set.json* file. The meta-data information is mostly similar to that of the training dataset. There are in-total 10000 query playlists. These 10000 queries are divided into 10 different classes. Each class either have title name available or not. Each of the classes have one of the following tracks available: [0, 1, 5, 10, 25, 100]. For example, one class may have title name and 10 tracks visible.

3 Metric

Borda count is taken over the rankings of the below three accuracy measures.

3.1 Precision@N

In the challenge dataset, for each playlist, N number of tracks are kept hidden. Precision is calculated as the ratio of those hidden tracks that also appear in first N tracks of the submitted 500 tracks.

3.2 NDCG@500

Normalized Discounted Cumulative Gain, which takes into account the position of the correctly predicted tracks.

3.3 Clicks@500

This metric is specifically designed in the spotify perspective. Spotify groups 10 tracks per page as a recommendation. click metric reflects how often a user have to change page until a relevant track is found. Unlike other two metrics, lower the value of click metric, better is the recommender system.

4 Methods

There are various ways of building a good recommendation system. There are mainly two types of approaches: Collaborative Filtering and Content based filtering. Collaborative Filtering builds model from users' past behaviour as well as similar decisions made by other users. The most commonly known algorithm is matrix factorization method where we predict the latent vector representations of item and user (in our case, item will be track and user will be the playlist). In our project we would like to perform two types of models inspired by the following research papers:

- Neural Collaborative Filtering using Implicit feedback[5]
- Effective Nearest-Neighbour Music Recommendations[2]

Our aim is to evaluate and compare the mentioned algorithms and draw inference from the same.

4.1 Neural Collab Filtering

For the sake of correctness, we are using only one of the slice ie. $mpd.slice.0 - 999.json$

- **Dataset:** Since we have only positive interactions, we can assume that all other entries in playlist-track matrix are negative samples. This is a strong assumption but easier to implement.
- **Evaluation:** For test set generation we incorporate a leave one out strategy. For each playlist we use the first track in test set. This evaluation methodology is called leave-one-out strategy and is used in the refereed paper.
- All the playlists in one slice that have same names are merged into one big playlist. Another big assumption, that tracks under playlists with same must resemble associativity. As observed, this increases the model performance.
- **Preliminary results:** We are considering 869 playlists (after merging same names such as Workout), 30049 unique tracks.

- Fig. 1 shows the latent space of different tracks among the first slice of data, using Principle Component Analysis.

Table 1: After 15 epochs, best **NDCG score:** 0.2155, best **HR:** 0.3751. Below are the first 10 epochs.

Epoch	HR	NDCG	Loss
1	0.09	0.1845	1.000
2	0.32	0.048	0.4914
3	0.370	0.184	0.4479
4	0.373	0.2156	0.4256
5	0.375	0.216	0.4076
6	0.372	0.2153	0.3835
7	0.353	0.2159	0.3570
8	0.344	0.2029	0.3322
9	0.336	0.1957	0.3088
10	0.314	0.1898	0.2815

4.2 Effective Nearest-Neighbour Music Recommendations

4.2.1 Paper[2] implements the following algorithms:

- **Case 1: Only the query playlist name is given** String Matching, Title Matrix Factorization
- **Case 2: At least one track is given in the query playlist** Item based Collaborative Filtering, Session-based Nearest Neighbors, and Matrix Factorization

4.2.2 Our Focus

In their results and observation section, they claim that for the case when at least one track is given, among three implemented algorithms, **Session-based Nearest Neighbors** algorithm is identified as most accurate method. First, we will implement Session-based Nearest method for the case where at least one track is given. For the case when only the playlist name has been provided, from the two implemented algorithms, we will first implement **String Matching** method.

String Matching: For a query playlist, we find most similar playlists from the training dataset. We then rank the tracks based on their number of occurrences in the retrieved playlists. To make the string matching algorithm robust, playlist names from both training and query dataset are converted to lower case letters, special characters are removed, and each of the words are stemmed using the NLTK library.

Session-based Nearest Neighbor: Consider representing a playlist in track space. Each playlist is represented as a vector. Index of a specific track T for a given playlist P is set to

$$tfidf(t,p) = tf(t,p) * idf(t)$$

$$tf(t,p) = \frac{1}{|p| + s}$$

Here s is a smoothing parameter. For a query playlist, sample playlists from the training dataset such that there is at least one track common between the training playlist and the query playlist. Now, sub-sample M playlists based on the latest edit made since the last epoch. This sub-sampling step helps reduce the search space and allows more sophisticated calculations to be performed in order to

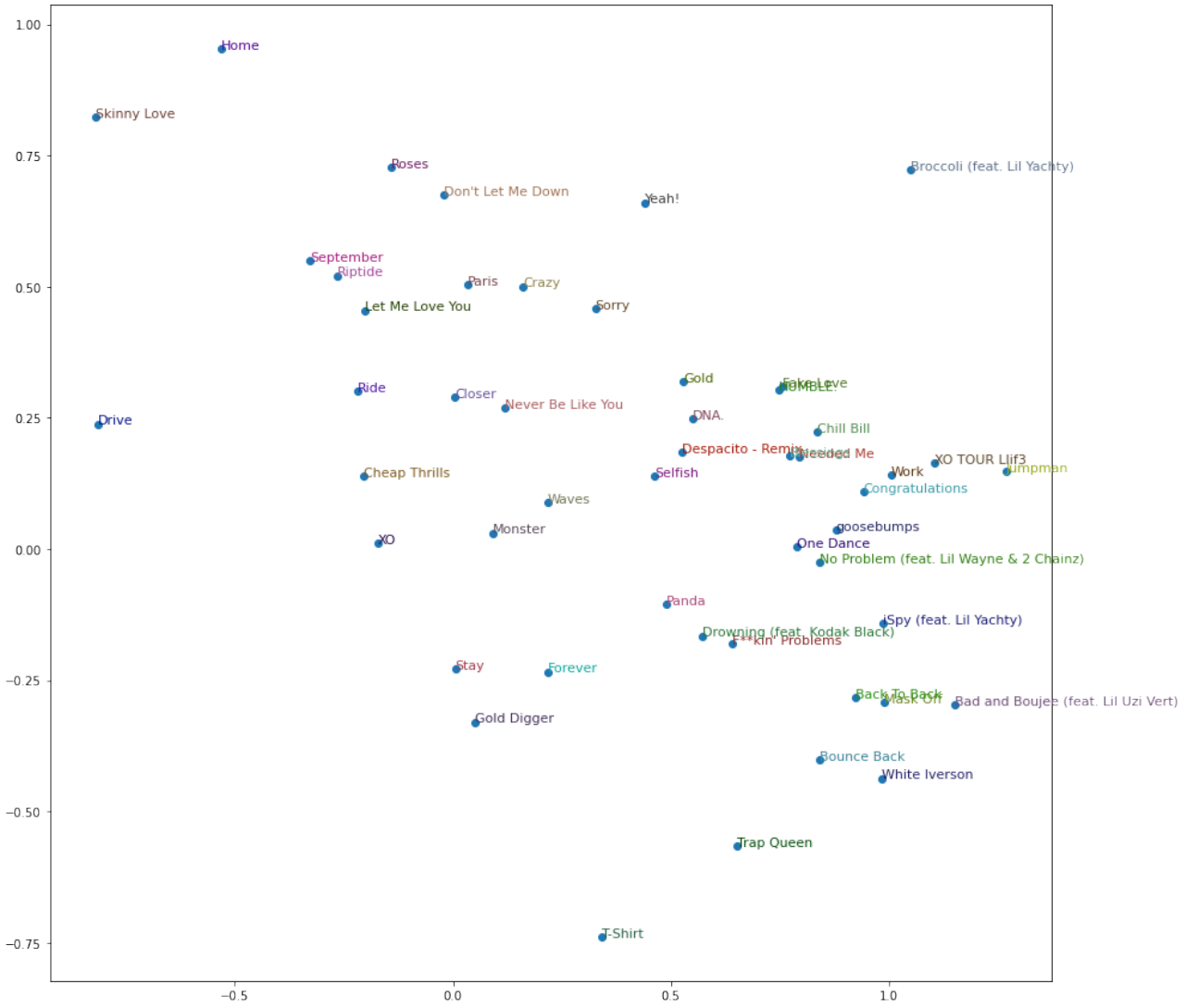

Figure 1: Visualizing latent space after PCA in Neural Collab. Filtering

rank the playlists. From this sub-sampled set of playlists, choose K playlists which are most similar to the query playlist based on the cosine similarity measure:

$$sim(p1, p2) = \frac{p1.p2}{|p1||p2|}$$

Tracks belonging to the k most similar playlists are the candidate tracks to rank. To **rank** a track T for a given playlist P:

$$rank(T, P) = \sum_{n \in N_p} sim(P, n).1_n(T)$$

Where N_p is the set of K most similar playlists and $1_n(T)$ is an indicator function that returns 1 if track $T \in n$ otherwise 0. Once all candidate tracks are ranked, we recommend top-500 tracks. In case there are not enough tracks to recommend, we pick remaining number of tracks from the most-popular tracks.

5 Implementation Details

5.1 Effective Nearest-Neighbor Music Recommendation

Fig-2 represents an end-to-end IR system for music recommendation. Node in the graph represents a python file. Incoming edge represents the input file and outgoing edge represents the generated file. Our IR system is divided into three parts as shown by the red dotted line in the fig-2. First part implements the *String Matching* algorithm to recommend music tracks when only the query playlist name is given. Second part implements the *Session-based Nearest Neighbor* algorithm to recommend music tracks when at least one track is given for query playlists. Third part merges the recommendations of the first two parts in a format that is accepted by the challenge website[3]. There are three hyper-parameters in the entire system: M(sub-sample size), S(smoothing parameter in the equation to calculate term freq), and NEIGHBORHOOD_SIZE(for a query playlist, how many similar playlists to consider for the ranking purpose). These hyper parameters are shown in the fig-2 with pink color background.

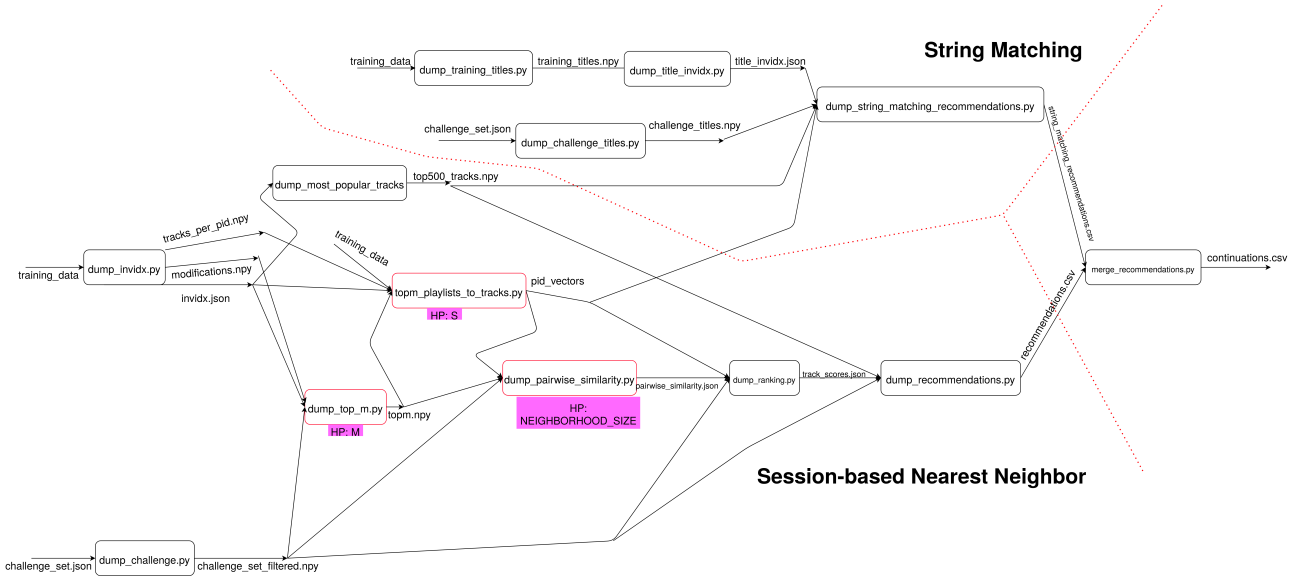

Figure 2: End-to-end IR system for music recommendations. Every node is a python file. Incoming edge represents the command line arguments and outgoing edge represents the generated output file.

5.1.1 String Matching

There are 1000 query playlists in the challenge set such that they only provide the name of the playlist. We have to recommend music tracks based on the playlist name. To handle such scenario, we implement a simple string matching based algorithm. We first pre-process the training playlist names and query playlist names by converting every letter to small case, special character removal, tokenization followed by the stemming. We then create the inverted index for training playlist names. Now for a query playlist name, we fetch all playlists from the training data such that there is at least one common token between the training playlist name and the query playlist name. These fetched playlists are the candidate playlists for further processing. Tracks are ranked based on their number of occurrences in the candidate playlists in the decreasing order. Since we have to recommend exactly 500 tracks, it is possible that by following above procedure, we get less than 500 tracks. In such case, we recommend the remaining tracks from the most popular set of tracks.

5.1.2 Session-based Nearest Neighbor

Training data and challenge set have many meta-data information which is not used by our recommender system. We first filter both datasets to only store the relevant information on disk, e.g. inverted index, 500 most popular tracks, playlist lengths, time when playlists were last modified, etc. For a query playlist, we sample all playlists from the training data such that there is at least one track common between the query playlist and the training playlist. From such sampled playlists, we sub-sample M playlists which were most recently modified. Here M is a hyper-parameter. We then calculate tfidf representation for training playlists. Here to calculate term freq of a track in a playlist, there is a smoothing parameter used: S which is also an hyper-parameter. For a query playlist and corresponding sub-sampled playlists we choose K most similar playlists from the sub-sampled playlists based on the cosine similarity measure. Here K(e.g. in the fig-2 it is termed as NEIGHBORHOOD_SIZE) is a hyper-parameter. By now, for a query playlist, we have K most similar set of playlists. We apply the ranking formula as described in section-4 to rank the tracks. If there are not enough tracks to recommend then we recommend the remaining tracks from the 500 most popular set of tracks.

5.1.3 Hyper-parameter Tuning

In the entire recommender system there are three hyper-parameters to tune. Unfortunately, the challenge organizers[3] have not provided any validation set or answers to the challenge set. So we created our own validation set from the training data. We first sample playlists from the training dataset such that each of the playlists have at least 200 tracks in it. From these sampled playlists we sub-sample 9000 playlists randomly. For these 9000 playlists, we keep certain number of tracks as visible(e.g. for recommending the next set of tracks) and remaining number of tracks as hidden as shown in table-2. We then run our recommender system with different hyper-parameter values on

	0-999	**1000-2999**	**3000-4999**	**5000-6999**	**7000-8999**
#visible tracks	1	5	10	25	100

Table 2: number of visible tracks in validation playlists

these 9000 validation playlists and predict 500 tracks based on the visible tracks. We compare the recommended tracks against the hidden tracks using precision and click-metric to tune the hyper-parameters. Table-3 represents the experiments we ran to tune the hyper-parameters on validation

K	S	M	**Precision**	**Click-metric**
100	50	2000	0.221	0.486
100	50	4000	0.229	0.509
100	30	4000	0.23	0.495
100	70	4000	0.23	0.513
100	10	4000	0.229	0.476
150	10	4000	0.226	0.492
50	10	4000	0.233	0.563
300	50	5000	0.219	0.508

Table 3: K, S, and M are hyper-parameters and precision, click-metric are corresponding scores on validation data

set. First three columns being the hyper-parameters and last two columns being the metric scores. It can be seen that even though on varying different hyper-parameter, there is not much difference on

the metric scores. Ideally, the recommender system should have high precision and low click-metric score which seems to be achieved at (K, S, M) = (100, 10, 4000) values. In section-6, we will show the performance of our recommender system with varying hyper parameters on the original challenge set.

6 Empirical Results

6.1 Effective Nearest Neighbor Music Recommendation

The MPD challenge organizers[3] also provide a world-wide leaderboard[4] for submitted recommended tracks. There are submissions from 9 teams(including our team). There are total of 42 submissions(including ours). Simple yet effective nearest neighbor algorithm stands 1^{st} (looks like there are two teams at 1^{st} position). Table-4 shows different metric scores along with the leaderboard[4] ranking for a given hyper-parameters. It can be seen that (K, S, M) = (100, 10, 4000) achieves the best performance with 1^{st} rank.

K	S	M	**Precision**	**NDCG**	**Click-metric**	**Leaderboard Rank**
100	10	4000	0.192	0.335	2.453	1^{st} with a tie
100	50	2000	0.189	0.3321	2.42	2^{nd}
50	10	4000	0.188	0.317	2.629	> 2

Table 4: K, S, and M are hyper-parameters and precision, click-metric, and NDCG are corresponding scores on challenge set

Depending upon the hyper-parameter values, execution time for an end-to-end recommendation takes on the order of few hours: typically 3-4 hours on a 2-core machine with 8GB RAM. Around 6GB of intermediate data gets generated in one complete execution.

7 Discussion

If we consider traditional IR algorithms for music recommendations then this paper[2] implements many of them as described in subsection-4.2. In our project we have implemented subset of those algorithms, mainly session-based nearest neighbor and string matching.

	K	S	M	**Precision**	**NDCG**	**Click-metric**
SKNN in SOTA[2]	1000	50	NA	0.199	0.361	2.34
SKNN + string matching (implemented by us)	100	10	4000	0.192	0.335	2.453

Table 5: K, S, and M are hyper-parameters and precision, click-metric, and NDCG are corresponding scores on challenge set. This table compares our implementation with the SOTA[2] implementation

Table-5 compares the SKNN(Session-based Nearest Neighbor) implementation as proposed in the original paper[2] with our implementation of the same. K, S, and M are hyper-parameters and scores on Precision, NDCG, and Click-metric are on challenge dataset. It is of note that the values of hyper-parameters we used are quite different from the proposed ones[2] because of the RAM issues but nevertheless we are able to achieve almost similar scores for different metrics. The large value of their hyper-parameters(e.g. $K = 1000$) results in large sized intermediate data whereas in our case with small hyper-parameter values(e.g. $K = 100$), we produce much less amount of intermediate data and yet achieve similar end results.

8 Conclusion

In this project, we tried to replicate the results by implementing the subset of algorithms as proposed by a team[2](3^{rd} in creative track and 7^{th} in main track in ACM RecSys Challenge 2018). More precisely, we implemented a session-based nearest neighbor and string matching algorithm. We showed that similar results can be replicated with less amount of intermediate data by tweaking the hyper-parameters. One of the challenge was to deal with 33GB of uncompressed training data. Inspired from the implementation level details[1] for session-based nearest neighbor, we lay out similar data-structures(e.g. inverted index) and operations to optimize the execution time. Our end-to-end recommender system takes 3-4 hours to recommend 10000 query playlists based on 1M training playlists with creating around 6GB of intermediate data. It is possible to achieve good music recommendation results using traditional IR algorithms which require far less computational resources when compared with neural retrieval.

References

[1] Jannach Dietmar and Ludewig Malte. "When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation". In: *RecSys '17: Proceedings of the Eleventh ACM Conference on Recommender Systems* (2017), pp. 306–310. URL: https://doi.org/10.1145/3109859.3109872.

[2] Ludewig Malte et al. "Effective Nearest-Neighbor Music Recommendations". In: *RecSys Challenge '18: Proceedings of the ACM Recommender Systems Challenge 2018* 3 (2018), pp. 1–6. URL: https://doi.org/10.1145/3267471.3267474.

[3] *Spotify Million Playlist Dataset Challenge*. URL: https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge.

[4] *Spotify Million Playlist Dataset Challenge: Leaderboard*. URL: https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge/leaderboards.

[5] He Xiangnan et al. "Neural Collaborative Filtering". In: (). URL: https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf.