Spotify Playlist Recommender

Table of Contents

The task

The goal of the challenge is to develop a system for the task of automatic playlist continuation. Given a set of playlist features, participants' systems shall generate a list of recommended tracks that can be added to that playlist, thereby 'continuing' the playlist. We define the task formally as follows

Input

Given N incomplete playlists

Output

- A list of 500 recommended candidate tracks, ordered by relevance in decreasing order.

The dataset

The Million Playlist Dataset (MPD) contains 1,000,000 playlists created by users on the Spotify platform. These playlists were created during the period of January 2010 through October 2017.

Test set

In order to replicate competition's test set, we remove some playlists from original playlists such that

- All tracks in the challenge set appear in the MPD
- All holdout tracks appear in the MPD
  Test set size: 1000 playlists

Train set

Train set will be the original set subtracts tracks in test sets

Metrics

G: the ground truth set of tracks

R: and the ordered list of recommended tracks by **R**.

The size of a set or list is denoted by **| â‹… |**

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):

$$\text{R-precision} = \frac{|G \cap R_{1:|G|}|}{|G|}.$$

Normalized discounted cumulative gain (NDCG)

Discounted cumulative gain (DCG) measures the ranking quality of the recommended tracks, increasing when relevant tracks are placed higher in the list.

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2(i+1)}.$$

The ideal DCG or IDCG is, on our case, equal to:

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

If the size of the set intersection of G and R, is empty, then the DCG is equal to 0. The NDCG metric is now calculated as:

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

Recommended Songs clicks

Recommended Songs is a Spotify feature that, given a set of tracks in a playlist, recommends 10 tracks to add to the playlist. The list can be refreshed to produce 10 more tracks. Recommended Songs clicks is the number of refreshes needed before a relevant track is encountered. It is calculated as follows:

$$\text{clicks} = \left\lfloor \frac{\arg\min_i \{R_i : R_i \in G|\} - 1}{10} \right\rfloor$$

If the metric does not exist (i.e. if there is no relevant track in R), a value of 51 is picked (which is 1 + the maximum number of clicks possible).

Proposed Solutions

For our problem, we treat "user" as "playlist" and "item" as "song". Because the dataset don't provide much information about each song so we won't use content-based filtering. Therefore we would only focus on

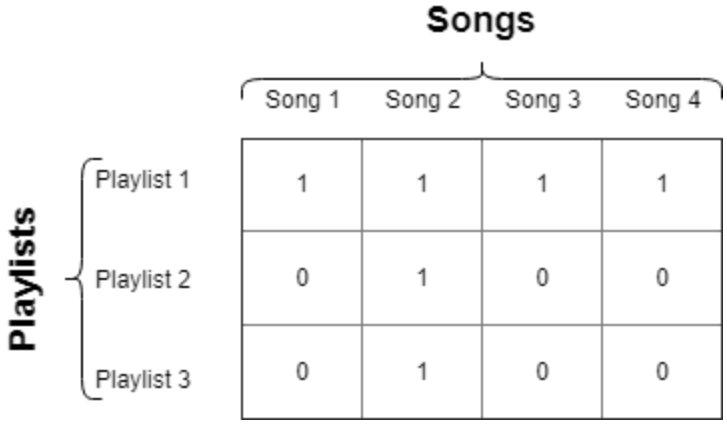
- KNN
- Collaborative Filtering
- Frequent Pattern Growth
- Matrix Factorization

Collaborative Filtering (CF)

- Playlist based CF: From similarity between each playlist and how other playlists "rate" (include or not) a track, I can infer the current "rate".
- Song based CF: From similarity between each songs and how current playlist "rate" other songs, I can infer the current "rate".

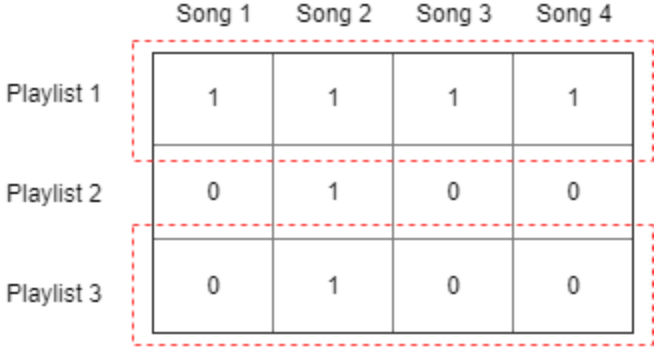
- Advantage:
  - easier to implementation
  - new data can be added easily and incrementally
  - don't need content of items
  - scales well with correlated items
- Disadvantage:
  - are dependent on human ratings
  - cold start problem for new user and item
  - sparsity problem of rating matrix
  - limited scalability for large datasets

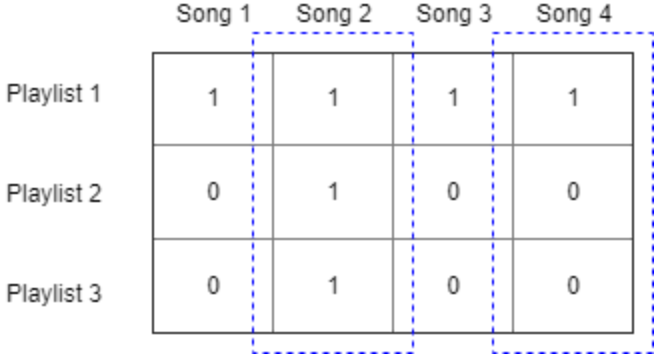
1. Construct playlist-song matrix



"1" means that song is included in the playlist and "0" otherwise. For example, playlist 1 contains song 2 and song 3, song 2 also includes in playlist 2.

2. Calculate the cosine similarity between song-song or playlist-playlist. In playlist-playlist similarity, we take each row as a vector while in song-song similarity we take column as a vector.





For playlist-playlist, we predict that a playlist **p** contains song **s** is given by the weighted sum of all other playlists' containing for song **s** where the weighting is the cosine similarity between the each playlist and the input playlist **p**. Then normalizing the result.

$$\hat{r}_{ps} = \frac{\sum_{p'} sim(p,p') r_{p's}}{\sum_{p'} |sim(p,p')|}$$

With song-song, we simply replace similarity matrix of playlists by that of songs.

$$\hat{r}_{ps} = \frac{\sum_{s'} sim(s,s') r_{ps'}}{\sum_{s'} |sim(s,s')|}$$

KNN

- Playlist-based: Find most similar playlist with current playlist and add all non-duplicate songs.
- Song-based: Find most similar songs with current songs in playlist and add them to the playlist.

- Advantage:
  - Same as Collaborative Filtering
- Disadvantage:
  - Need to mantain the large matrix of similarity
  - Cold start problem

Frequent Pattern Growth

I have a number of current tracks, I will look at other playlists to find common tracks associated with currents tracks and add them.

- Advantage:
  - Scalability
  - Don't have problem with sparsity
- Disadvantage:
  - Expensive model building

1. The first step of FP-growth is to calculate item frequencies and identify frequent items.
2. The second step of FP-growth uses a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly, which are usually expensive to generate.
3. After the second step, the frequent itemsets can be extracted from the FP-tree

Matrix Factorization

Decompose the playlist-songs matrix into dot product of many matrix and use these matrix to make inference.

Matrix factorization can be done with orthogonal factorization (SVD), probabilistic factorization (PMF) or Non-Negative factorization (NMF).

- Advantage:
  - Better addresses the sparsity and scalabity problem.
  - Improve prediction performance
- Disadvantage:
  - Expensive model building
  - Trade-off between prediction performance and scalability
  - Loss of information in dimension reduction technique

Due to the large memory use when compute smaller matrix, I will not implement Matrix Factorization.

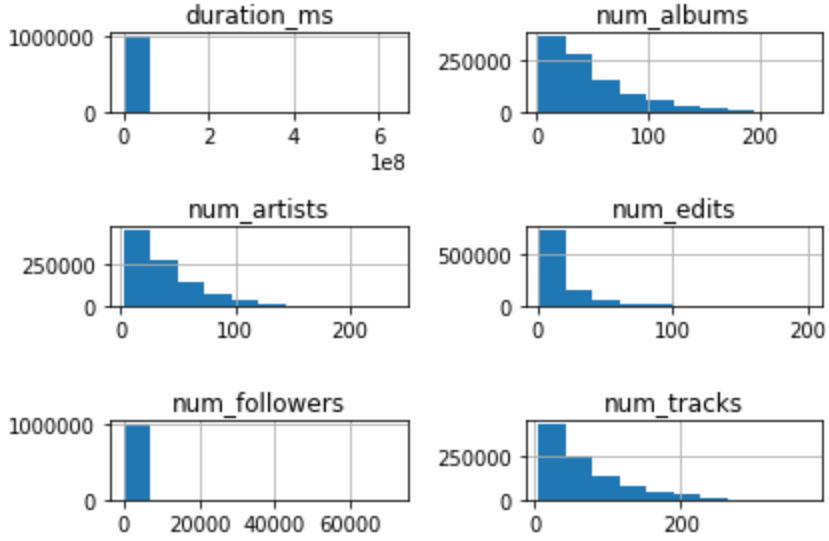
Word2vec

We can use Word2vec to reduce the dimension of matrix and perform KNN on the new dimension

Eploratory Data Analysis

- Number of:
  - playlists: 1,000,000
  - Tracks: 66,346,428
  - Unique tracks: 2,262,292
  - Unique albums: 734,684
  - Unique Titles: 92,944

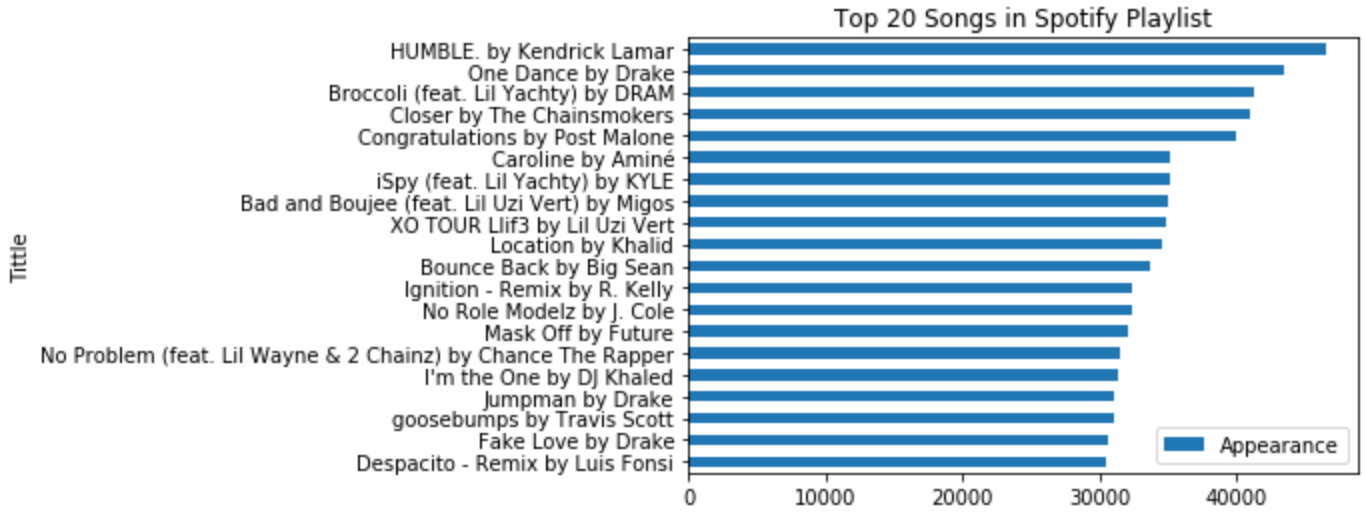
- Distribution of: Playlist length, Number of Albums / Playlist, Number of Artist / Playlist, Number of edits / Playlist, Number of Followers / Playlist, Number of Tracks / Playlist



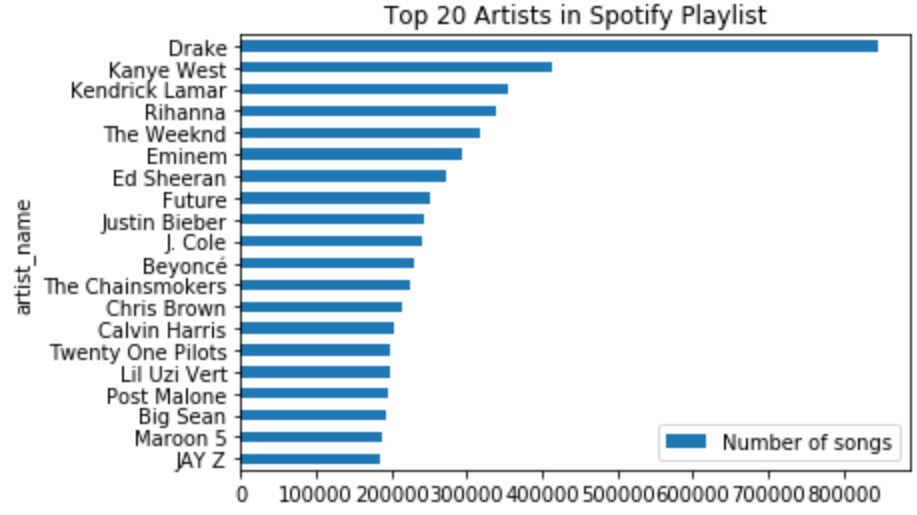
As we can see all distributions are left-skewed which means if we are looking for average value, we should go for "Median" not "Mean"

- Median of playlist length: 11422438.0
    - Median of number of albums in each playlist: 37.0
    - Median of number of artists in each playlist: 29.0
    - Median of number of edits in each playlist: 29.0
    - Median of number of followers in each playlist: 1.0
    - Median of number of tracks in each playlist: 49.0

- Top 20 Songs in Sporify playlists



- Top 20 Artist in Spotify Playlists



Result

Method	R-precision	NDCG	Song Click	Time Taken (s)
Playlist-based KNN	0.7766	1.6010	0.0	41.42
Song-based KNN	0.7847	0.7975	0.0	4183
Word2Vec + Song-based	0.0030	0.004	10.35	
Word2Vec + Playlist-based	0.0171	0.015	8.086	
Playlist-based CF (get top K rating songs)	0.7844	0.8011	0.0	approx 12000
FP Growth				

Conclusion

- Playlist-based and song-based KNN perform really well on this dataset. Thanks for multi-processing, the inteference is really fask. However similarity matrix and playlist-song matrix are built beforehand.
- Collaborative filtering also show comparable result with KNN but take much longer to infer result.
- Word2Vec dimension reduction failed to capture the similarity between songs / playlists. It means that despite that playlists are sequence of songs, its behaviour is difference from text sequence.

Further improvement

- Implement FP-growth in Spark and compare with current solutions
- Matrix Factorization