PROJECT REPORT

INFORMATION RETRIEVAL AND WEB SEARCH

COL 764(2021-2022)

Project by – Viseurs

Team members

Nikki Tiwari-2021SIY7560

Harsh Pandey-2021CSY7545

Problem statement-

The aim of the project is to develop a system 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. The task is given incomplete playlists ,we need to recommend a list of 100 tracks, ordered by relevance in decreasing order.

Dataset Description -

We are working on a subset of the Million Playlist Dataset which consists of 10000 playlists divided into 1000 slices each. The training dataset is splitted into test/validation set having the initial 200 playlists.

Metric-

In the following, we denote the ground truth set of tracks by|G|and the ordered list of recommended tracks by R. The size of a set or list is denoted by|·|, and we use from:to-subscripts to index a list.

R-precision -

R-precision is the number of retrieved relevant tracks divided by the number of known relevant tracks(the number of withheld tracks)

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

The metric is averaged across all playlists in the challenge set. The metric rewards total number of retrieved relevant tracks(regardless of the order)

Clicks –

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 recommendation system

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

nDCG-

Discounted Cumulative Gain (DCG) 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 perfectly ranked:

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

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

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

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

Implementation Details-

The vector Space model with each track represents a dimension and playlist as a vector in the track space.

$$p_i = [t_1, t_2, t_3, ...t_{|V|}]$$

ti > 0 accounts for the presence of track in the playlist and ti=0 for its absence

where

tf = term frequency (always 1)

idf = 1/ (p + s)

p is the number of playlist having that track
s is a parameter to avoid divide by zero

We will use the **cosine similarity** function for computing the similarity between the **playlists vector**

- So, for a given test playlist (Pq) we will find some top K (Hyper parameter) similar playlists.
- We will fetch the tracks from them which will give us the initial set of relevant tracks for the test playlist.

b) Re-ranking Part –

Here we have tried to include the similarity between the tracks

If two tracks are in same playlist, it increases the chances of the tracks being similar to each other.

This above is the basis of this way of ranking.

- We have created a co-occurrence matrix of tracks, i.e a matrix of size |V| X |V| {V = number of unique tracks}

	t1	t2	…	t|V|
t1				
t2				
.				
t|V|				

Co – occurrence Matrix of Terms (M)

- The values of this matrix are filled as –

  M[ti,tj] = Number of Playlists in which both tracks ti and tj are present together.

- Now from previous section we have set of similar songs so we compute the similarity of each track with the tracks in the query

playlist.

  - Similarity – sim(tq, ti) > sim(tq, tj) if M(tq, ti) > M(tq, tj) {tq = tracks of query playlist }

- We select the top 500 songs (as required in the challenge) and submit them.

We have computed the similarity between playlists using VSM which considers the independence between the terms(tracks in our case).

In order to incorporate the similarity between the tracks , we have used a reranking technique which suffices our requirement.

Now since the vectorized playlists are very sparse, which increases its computation time for comparing the similarities we have switched to compact vector representation implemented by PCA.

Modified playlist vector-

The average size of playlist is 66.3428 so a good estimate to represent all the playlist is 70.Therefore the n_components=70

Now the transformed playlist vector in a compact dimensional space is a 70-dimensional vector (pi is compact dimension vector).

$$p_i = [c_1, c_2, c_3, ...c_{70}]$$

Now computing the similarity between these transformed playlist vectors and the top playlist vectors are again selected based on higher similarity scores and their respective tracks are sent for reranking as above .

Result and Comparison with the baseline-

On comparing with the leader board of competition going on

PARTICIPANTS	R-precision	NDCG	clicks
Md_sadakat-hussain	0.220	0.386	1.932
Sroy2	0.196	0.334	2.292
Shwetanshu	0.196	0.334	2.292

Our model's result on slice

Train size – 800 playlists

Test size – 200 playlists

TECHNIQUE USED	R-precision	NDCG	clicks
With PCA	0.18312745	0.06236819853381026	12.90
Without PCA	0.1327807	0.130198	11.67

Our model's result on subset

Train size – 4000 playlists

Test size – 1000 playlists

TECHNIQUE USED	R-precision	NDCG	clicks
With PCA	0.187235358	0.09486413	9.0660

The calculation for **1000** playlists test size took **3hrs 40 mins**

So the average time to recommend 500 tracks for a playlist = 13.2 sec

Some outputs on validation data-

```
Total similar playlists 100
reccommended len (finally 500) 3396
relevent tracks size: 44
sampled relevent tracks 20
pid        : 102000
click      : 1.5
r_precision: 0.22727272727272727
ndcg       : 0.06112084280298722

Total similar playlists 100
reccommended len (finally 500) 3404
relevent tracks size: 86
sampled relevent tracks 20
pid        : 102001
click      : 4.0
r_precision: 0.09302325581395349
ndcg       : 0.11673490829789557

Total similar playlists 100
reccommended len (finally 500) 2703
relevent tracks size: 131
sampled relevent tracks 20
pid        : 102003
click      : 0.8
r_precision: 0.12213740458015267
ndcg       : 0.06364260907286357

Total similar playlists 100
reccommended len (finally 500) 2375
relevent tracks size: 52
sampled relevent tracks 20
pid        : 102019
click      : 0.0
r_precision: 0.34615384615384615
ndcg       : 0.13188287530091308
```

Observations-

Using VSM Similarity

We have observed that the first part of algorithm is a good solution as it does not require lots of storage or preprocessing and we could directly use this even if we have changing training set, as it only require 1 pass and gives the set of similar playlists.

Using Reranking

The intution of the second part came from the fact that VSM assumes independence of trems (tracks on our case) so we also wanted to incorporate the similarity between the tracks.

Although we can precompute the similarities in a file and for every new query playlist we can use this similarity to rank the tracks (which we did) but because of the huge size of the database (31 GB) the estimated size of this similarity file turned out to be 8TB, so we only took a subset of this dataset to test this intution.

Even on 10,000 playlist we got the similarity file's size of 43.5GB, which is huge.

On comparision with the leaderboard scores we can clearly see that although the presision is comparable but the clicks and nDCG takes a big hit, this is proabbly because we are only using a subset of dastabase for reranking so while reranking we are almost ignoring the songs which are not in the current train set and only giving preference to songs which are mostly in the current train set. This generates a bias for train set songs and therefore not giving a proper result.

Using PCA-

Finally when we used PCA we saw a bit more drop in the precision, and although the speed of computing similarity did increased a lot, but still we needed to perform an iteration over the playlist to generate the original vector and then pass it to generate reduced playlist vector.

Conclusion -

We attempted to perform similarity based search for generating relevent tracks using a subset of the dataset.

First we used Colaborative Filtering memory based (Playlist- track) and found playlists similar to the query playlists

Later we tried to inforporate the track-track similarity using the concept of reeranking, which initially looks very space hungry but we have provided some intution of reducing this massive file.

Finally we would point out that there are other IR algorithms which requires very less storage space, computation and can perform equally or even better.

Future Work-

Reducing the size of the track-track similarity file.

We noticed that the 43.5GB file of track-track has lots of zeros as most of the songs are not present even in the single playlist, these null values play a significant role in increasing the file size (for eg- on just zipping this 43.5GB file we got file of 64MB)

Other estimation-

Also for estimation this file was created for 10,000 playlist, and on average the playlist length is 67 (no. Of tracks).

So for 1 playlist we need to view all pairs of 67 tracks i.e 67*(67-1)/2 = 2211 pair of tracks for 1 playlist.

And if this similarity is stored in a float of 4 Byte

so total relevent information = 10,000 * 2211 * 4 Bytes

= 88440000 B

= 88.44 MB

So there is definitely scope to store this relevent information and then perform the same process on the entire dataset.

Github Link of implementation -

https://github.com/abstruse020/COL764_Project

References-

1)research paper on nearest neighbors https://www.researchgate.net/publication/328088745_Effective_Nearest-Neighbor_Music_Recommendations

2)Spotify Million Playlist Dataset Challenge-https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge/leaderboards

3)https://github.com/mrthlinh/Spotify-Playlist-Recommender

4)https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge/leaderboards

5)https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60