# Data science final project
電機 07 0310781 黃國祐

## A.專題介紹與貢獻

My final project is Recsys 2018 challenge. In this conference challenge, Recsys and Spotify provide Million Playlist Dataset. For each playlist in the datasets, it contains

- pid - the playlist ID
- name - (optional) - the name of the playlist.
- num_holdouts - the number of tracks that have been omitted from the playlist
- tracks - a (possibly empty) array of tracks that are in the playlist. Each element of this array contains the following fields:
    - pos - the position of the track in the playlist (zero offset)
    - track_name - the name of the track
    - track_uri - the Spotify URI of the track
    - artist_name - the name of the primary artist of the track
    - artist_uri - the Spotify URI of the primary artist of the track
    - album_name - the name of the album that the track is on
    - album_uri -- the Spotify URI of the album that the track is on
    - duration_ms - the duration of the track in milliseconds
- num_samples the number of tracks included in the playlist
- num_tracks - the total number of tracks in the playlist.

The goal of this challenge is to recommend 500 songs based on different information. There are ten tasks in this challenge,

1.Predict tracks for a playlist given its title only
2.Predict tracks for a playlist given its title and the first track
3.Predict tracks for a playlist given its title and the first 5 tracks
4.Predict tracks for a playlist given its first 5 tracks (no title)
5.Predict tracks for a playlist given its title and the first 10 tracks
6.Predict tracks for a playlist given its first ten tracks (no title)
7.Predict tracks for a playlist given its title and the first 25 tracks
8.Predict tracks for a playlist given its title and 25 random tracks
9.Predict tracks for a playlist given its title and the first 100 tracks
10.Predict tracks for a playlist given its title and 100 random tracks

For evaluation, Recsys uses R-precision, NDCG and recommended song clicks. However, they didn't provide click information. For the result, we can only evaluate with R-precision and NDCG for each separate task.

The equation for R-precision, NDCG and recommended song clicks:

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

$$\text{DCG} = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2(i+1)} \qquad \text{IDCG} = rel_1 + \sum_{i=2}^{|G|} \frac{rel_i}{\log_2(i+1)} \qquad \text{NDCG} = \frac{DCG}{IDCG}$$

During these semester, I utilized the information from the playlist title and song uri and provide two methods. One is for less information and the other is for large information. In the following two parts, I will first introduce my method and the algorithm and then I will provide my experiment result on it.

B.方法

1.less information

For number of seed songs less than 10 tracks, I use embedding method. For the embedding method, I tried word2vec, line (large-scale information network embedding) and deep walk. Word2vec is the embedding method from the natural language processing, line and deep walk are the method from graph. Among these method, word2vec with a corpus build by a special rule out perform the other two by 3-4%. I will first introduce word2vec and the rule I used to build my corpus.

Embedding method is the technic to reduce the dimension and build relation between each item. Originally, computer save each word as one of n encoding. Each word is the same to a computer, it won't understand the meaning of each word and only consider it as a symbol. By using the embedding method, we convert one of n encoding into lower dimension and the value in each dimension can be used to compute the relation between each word.

Word2vec comes from the nature of the language, we can learn a word from its context. We can use the context to predict the word or use the word to predict the context, which called cbow and ski-gram. By the objective function of co-occurrence probability and gradient descend method, we are able to learn the representation from the data.

$\mathrm{f}(co-occurence|context, word, \theta) = \frac{1}{1+e^{-context*word}}$, where * represent inner product.

The rule for building the corpus is: song_uri song_uri title song_uri song_uri.

Ex:

if there is a playlist

title:"MY FAV", with five songs

1."track_uri": "spotify:track:10M2Ex445zw585Ducldzkw",

2."track_uri": "spotify:track:3n2A6l4UayVe3GGKCvfQWV",

3."track_uri": "spotify:track:4Az17HfqonKSWNsaLgTBeK",

4."track_uri": "spotify:track:4VFE6ZNqa8jHAmbYICoAFg",

5."track_uri": "spotify:track:6xHI9KjUjYT0FPtGO8Mxa1",

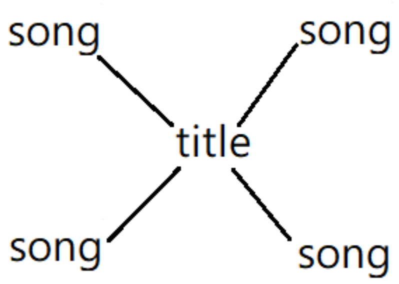
Then the corpus will be:

spotify:track:10M2Ex445zw585Ducldzkw spotify:track:3n2A6l4UayVe3GGKCvfQWV MY FAV spotify:track:4Az17HfqonKSWNsaLgTBeK spotify:track:4VFE6ZNqa8jHAmbYICoAFg MY FAV spotify:track:6xHI9KjUjYT0FPtGO8Mxa1

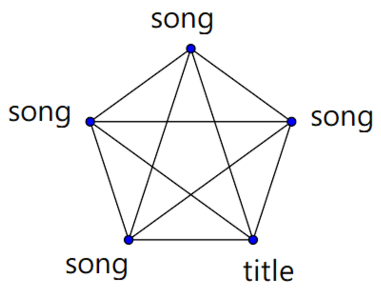
For network embedding method deep-walk and line, deep-walk uses a random walk method to generate the corpus and line use ski-gram to preserve 1-hot and 2-hot information. I had tried two kinds of rule to build my graph, first song only connects with title second song connects with song and title.

Ex:

First



second



However, both of these method didn't work better than word2vec. I chose word2vec as my word embedding to continue following experiment.

Embedding method works well on less input information, however, for large input information it didn't perform well. Maybe it's due to long playlist contain too many different information, such as in a long playlist a person might like to listen to classic music at first and ruck music in the end. In this situation, embedding might not work well. Thus, I had tried some tricks to deal with this question.

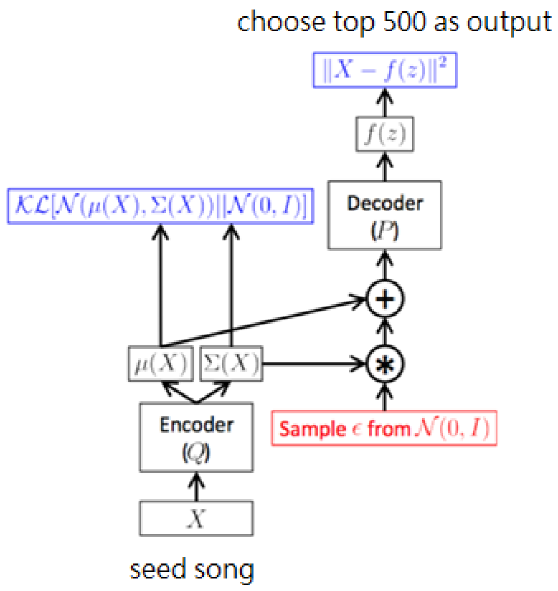
K means: I cluster input seed songs into 3 groups and treat them as three different playlists. With these three playlists, I get the final answer by find top500 out of 1500.

Top k: Similar to K means, I assume playlist is combine by k major singer. Thus I treat it as a filter, for each playlist I compute the major k singer and consider song from other singer to be noise term. Finally, I recommend only base on the top k singers' song.

Both these two trick; unfortunately, didn't work better than the origin method. Thus, I choose to take average as my final method to aggregate different seed songs.

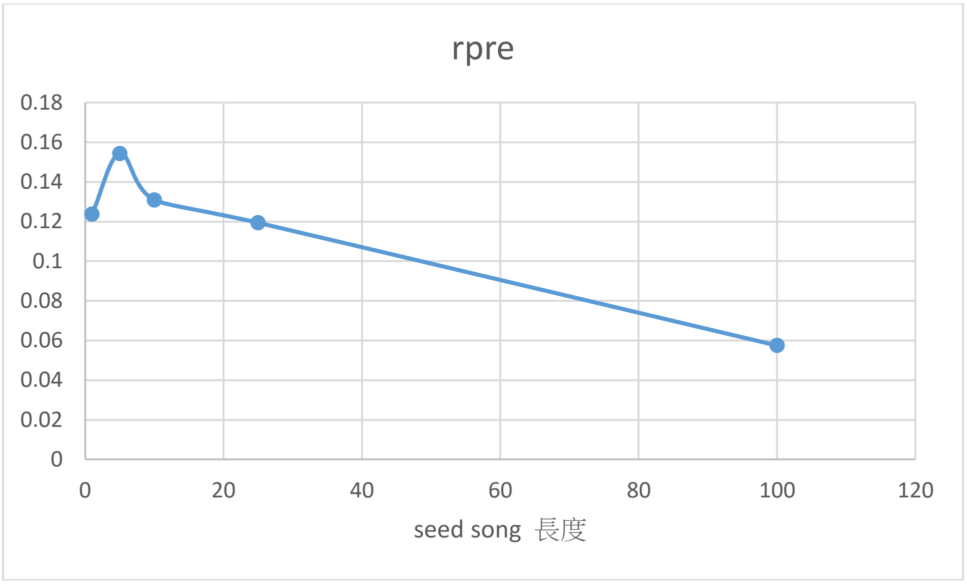
2.large information

Embedding didn't work well on large information; therefore, I turn to use generative model. I used variational autoencoder as my generative model and the result is 4% better in random 25 case and 6% in random 100.


choose top 500 as output
$\|X - f(z)\|^2$
$f(z)$
Decoder ($P$)
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X)) \| \mathcal{N}(0, I)]$
$\mu(X)$
$\Sigma(X)$
Encoder ($Q$)
Sample $\epsilon$ from $\mathcal{N}(0, I)$
$X$
seed song

the size of encoder is [num_data 800 300] and the activate function is tanh.

## C.實驗結果



This is embedding result for title with first n seed song.

| | embedding | Variational autoencoder | |
|---|---|---|---|
| random 25 | 0.181034 | 0.23 | r-precision |
| | 0.5306445 | 0.44 | NDCG |
| random 100 | 0.1467 | 0.26 | r-precision |
| | 0.4903 | 0.512 | NDCG |
| first 25 | 0.1194 | 0.15 | r-precision |
| | 0.436 | 0.34 | NDCG |
| first 100 | 0.0741 | 0.16 | r-precision |
| | 0.349 | 0.37 | NDCG |

We can find out that vae out perform embedding on large information case.
(there are some problem on NDCG equation in my code, so the value of NCDG might be strange. QAQ)

## D.參考資料

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. CoRR, abs/1301.3781, 2013.
[2] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: Large-scale Information Network Embedding. In WWW, 2015.
[3] B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: Online learning of social representations. In KDD, 2014
[4] C. Doersch. Tutorial on variational autoencoders. arXiv:1606.05908, 2016.
[5] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara .Variational autoencoders for collaborative filtering, in The Web Conference (aka WWW) 2018.

## E.團隊分工

我是自己一個人一組拉，不過有朋友也有在做這個，我們有一起討論。