Playlist Generation using Community Detection

Harjyot Singh - s1463345

Abstract

This report provides a brief overview on how to use community detection on a music network and pagerank to generate a playlist based on a set of input songs

1. Introduction

In today's internet age we have access to everything on our fingertips. Music is one such thing - gone are the days of running to your friend because they had a new album you really wanted to hear, rip the CD, use various conversion softwares to get the songs in the right format, copy it to your mp3 player and then finally listen to the songs! No, everything is available in an instant and this leads to an interesting issue, how can systems be developed to recommend songs from this vast selection keeping in mind issues like varieties in taste of music, the 'right' song for the 'right' moment.

Most algorithms that exist today use a seed song to find a list of similar songs, but I believe that we can try a different approach - why should it be an issue of exploring just similar songs and not a discovery experience ie, incorporating variety to some extent? To tackle this I created a network of similar songs, and ran community detection algorithm to study the nature of this network and from that point on used a PageRank system to create a playlist.

2. Prior Work

There are 4 broad categories that have been explored for playlist generation. Namely,

- *Local search based approach (Steffen Pauws & Vossen, 2006)* - This method aims to search the neighboring songs to optimize a cost function given by the total weighted penalty of the playlist.
- *Constraint based approach (Pachet F & D, 1999)* - This is based on the local search approach but adds user-specific constraints like song categories and user preferences, which makes it a constraint satisfaction programming.
- *Content based approach (Martin Gasser & Tomitsch, 2008)* - In this approach, each song is represented as a vector of attributes like artist, composer, genre, mood etc. The constraints provided by the user or derived by the system will depend on the values of the attributes.
- *Similarity based approach (Crampes M & S, 2007)* - An anonymous list of similar songs is generated according to the user input query. This is a suggestion of what other songs are similar to the one user provide.

In the latest work by a lot of the big companies, for example Spotify, a playlist generator takes in a lot more parameters to justify it's similarity to another song - be it acoustic analysis, artist history, plays by different users, year of release etc - and then they are trained with highly sophisticated convolution neural networks, leading to a highly accurate but too big a system to be reproduced here.
Instead I decided to go with the similarity based approach as the obvious choice, however rather than just treating it as a problem of finding the next pieces for maximum similarity, the decision was made (Due to some past work having been done in the field by Stanford students (Bangzheng He & Nguy, 2015)) to use a Topic Sensitive Page Rank system(Haveliwala, 2002), which essentially would break down the problem to being more of a web search. This would also incorporate ideas of being able to add variety into the playlist using Community Detection on the network.

3. Data Collection Preprocessing

There are quite a few 'sources' when it comes to big commercial music data, however most of them have different use cases. The most appropriate one we could find was Last.fm data prepared by Million Song Dataset project (msd) which is largest research collection of song-level tags and precomputed song-level similarity. Here are some stats about the dataset:

- **943,347** matched tracks MSD <-> Last.fm
- **584,897** tracks with at least one similar track
- **56,506,688** (track - similar track) pairs

The dataset was created using Last.fm's *getSimilar* API, and behind the scenes the similarity scores are calculated using Listening Habits, Acoustic Analysis and Tag related information, however the validty of the similarity can always be debated, I am not going to ponder to much on it now.
One interesting thing about the dataset was that it's a directed graph with often different values in the opposite directions. The dataset is available in two formats *JSON files* and as a *SQLITE Database*

The latter was picked due to efficiency in usage through python. There are two tables in the database *similar-src* and *similar-dest* which are essentially the same but one captures the outgoing edges, whilst the other captures ingoing. Picking any one would have sufficed, I went with the first one.

4. Community Detection

They are 3 major algorithms used in community detection.

- *Girvan-Newman Algorithm* detects communities by progressively removing edges from the original network. The connected components of the remaining network are the communities. Instead of trying to construct a measure that tells us which edges are the most central to communities, it focuses on edges that are most likely "between" communities.
- *Clauset-Newman-Moore algorithm* is a heirarchical agglomeration algorithm for very large network which is made to greedily optimise modularity with quite high efficiency O(md log n).
- *The Louvain algorithm* Similar to the Clauset-Newman-Moore algorithm, the main aim of this algorithm is to greedily optimize modularity on very large networks.The optimization is performed in two steps. The first step is a "greedy" assignment of nodes to communities, favoring local optimizations of modularity. The second step is the definition of a new coarse-grained network in terms of the communities found in the first step. These two steps are repeated until no further modularity-increasing reassignments of communities are possible.The apparent efficiency of this algorithm is O(n log n). It also generally reveals a hierarchy of communities at different scales, and this hierarchical perspective can be useful for understanding the global functioning of a network.

Due to the very large size of our network(0.66 Million Nodes and 56.5 Million Edges), Louvian was the obvious choice.
I created a threshold to simplify the network, which picks similarity above 0.6 (After experimentation seemed to be the sweet spot balancing computation time and result accuracy) which gave me the following stats:
Modularity: *0.946*
Number of Communities: *21156*

5. PageRank and Playlist Generation

The Majority of the following algorithm was adapted from (Bangzheng He & Nguy, 2015). The idea of PageRank is that we want to assign some measure of importance to every node. We will view links from one node to another as endorsements. So, once we have decided that one node is important, we will believe that each of the pages to which it points are important as well. To limit the influence of any one page, we divide its votes for importance over its out-links.

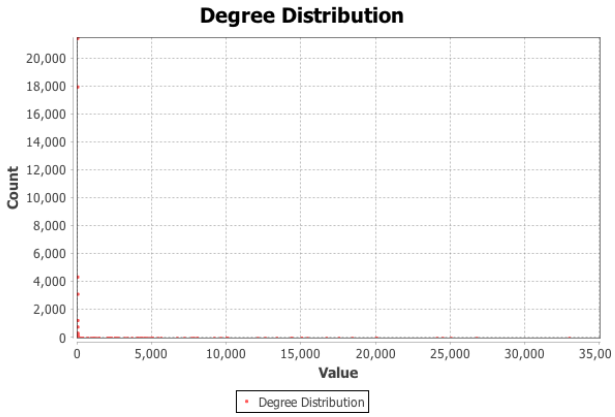

Figure 1. Weighted Degree Distribution of network

In the standard random walk construction for PageRank on an unweighted graph, the probability of transitioning from node i to any of its neighbours j is the same. Weighted alters this assumption such that the walk preferentially visits high-degree nodes. Thus, the probability of transitioning from node i to node j depends on the degree of j relative to the total sum of degrees of all i neighbours.
If A is the Adjency matrix for a directed graph, D is the Diagonal Matrix consisting of weighted out-degree of nodes i ie, $d_i = \Sigma_j A_{ij}$.If there are no sinking(nodes with zero outgoing weight), the transition matrix for a random walk on this weighted graph is $P = D^{-1}A$, which is a row-stochastic matrix. The random walker at each step moving to a new node by transitioning along the edge with probability α or by teleporting to position independent of the previous location with probability $1 - \alpha$. Then we can write the Markov process as

$$x^{T(t+1)} = \alpha x^{T'} P + (1 - \alpha)\upsilon^T$$

where P represents the edge transition probabilities and the personalization vector υ represents teleportation probabilities. The PageRank vector x is the stationary vector for the above process.
On picking different seed songs, the personalisation vector is biased with the use of the community structure. The calculation of the pagerank vector x is done using power iteration method. Although the size of the transition matrix was massive 663234×663234, due to its sparse nature it was computationally possible. The implementational details of the above algorithm are left out, but can easily be found in the code.
Now on to combining the pagerank and community detection. The Idea is simple, the personalisation vector υ is modified using the community partition data at seed time and then that is fed into the PageRank System. A control was added to change the range of 'variety' in the playlist and is called the 'variety rate' denoted by β.

So if β is closer to 1, the songs in the playlist will be very diversified in terms of communities whereas if it is closer to 0, the algorithm will pick only the most similar songs.
Assumptions:

- *S - Query Set*
- *N - Total Nodes*
- C_S *- Communities that Query Songs belong in*
- β *- Variety Rate*

$$v = (v_1, v_2..., v_N)$$

We set

$$v_i = \frac{\lambda \times \beta}{N}, v_i \notin C_S$$

$$v_i = \frac{1}{N}, v_i \in C_S - S$$

$$v_i = \frac{C_{vi}}{N}, v_i \in S$$

6. Results

The success of this project can be very subjective, as there aren't really metrics we can use to quantify a good playlist from a bad playlist, what I tried however was to compare my results to one of the online services.
Let me first begin by explaining that what didn't work as well as I had hoped for it to work - the variety rate, although you see a variation in the playlists, it is minuscle, and largely depended on the communities the seed query song(s) were in. Also what was noted was that a lot of songs that came up were from the same artist, now to a certain extent that makes sense, but at the same time if the playlist is flooded with songs from just one artist, it's more of a radio generator for that artist rather than a diversified playlist.
I think a lot of this can be blamed down to the dataset we have as well - there's no correlation between songs except a deadpan similarity score, had there been other extrinsic data like user's history, skip-rate, genres, artist influences, time of release - it might have been a more powerful albeit a highly complicated system.

So as a conclusion, even though this system cannot compete with the state of the art technologies present at the moment, it does seem to be quite functional and does come up with a few interesting results as well as providing an alternative take on music analysis rather than the norm these days of neural nets.

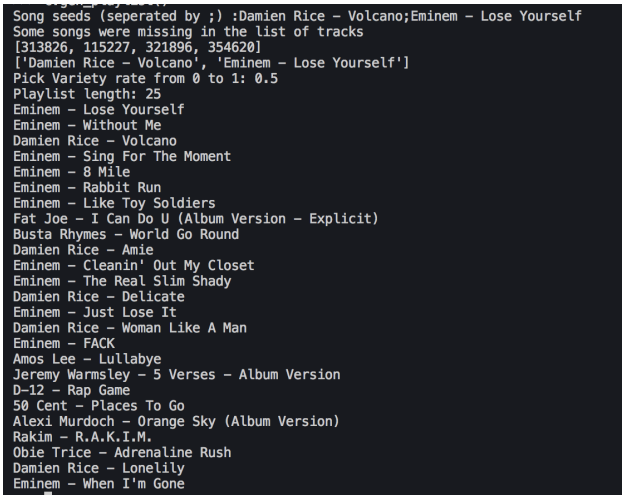

Figure 2. At Variety Rate - 0.1

References

Last.fm dataset, the official song tags and song similarity collection for the million song dataset. URL http://labrosa.ee.columbia.edu/millionsong/lastfm.

Bangzheng He, Yandi Li and Nguy, Bobby. Music playlist generation based on community detection. Technical report, Stanford University, 2015.

Crampes M, Villerd J, Emery A and S, Ranwez. Automatic playlist composition in a dynamic music landscape. In *Proceedings of the international workshop on semantically aware document processing and indexing*, 2007.

Haveliwala, T. H. Topic-sensitive pagerank. In *11th International World Wide Web Conference*, 2002.

Martin Gasser, Elias Pampalk and Tomitsch, Martin. A content-based user-feedback driven playlist generator and its evaluation in a real-world scenario. Technical report, The Australian Research for Artificial Intelligence, 2008.

Pachet F, Roy P and D, Cazaly. A combinatorial approach to content-based music selection. In *Proceedings of IEEE multimedia computing and systems international conference*, 1999.

Steffen Pauws, Wim Verhaegh and Vossen, Mark. Fast generation of optimal music playlists using local search. In *Proceedings of the 6th International Conference on Music Information Retrieval(ISMIR)*, 2006.

```
Song seeds (seperated by ;) :Eminem – Lose Yourself
Some songs were missing in the list of tracks
Pick discovery rate from 0 to 1: 0.9
Playlist length: 25
Eminem – Lose Yourself
Eminem – Without Me
Eminem – Sing For The Moment
Eminem – 8 Mile
Eminem – Rabbit Run
Fat Joe – I Can Do U (Album Version – Explicit)
Busta Rhymes – World Go Round
Eminem – Like Toy Soldiers
Eminem – Cleanin' Out My Closet
Eminem – Just Lose It
Eminem – The Real Slim Shady
Rakim – R.A.K.I.M.
Nas – U Wanna Be Me
50 Cent – Places To Go
Obie Trice – Adrenaline Rush
Eminem – FACK
D–12 – Rap Game
D–12 – My Band
Eminem – When I'm Gone
Eminem – The Way I Am
D–12 – Git Up
Eminem / Dido – Stan
Eminem – My Name Is
Eminem – Mockingbird
50 Cent – Wanksta
```

Figure 3. At Variety Rate - 0.9

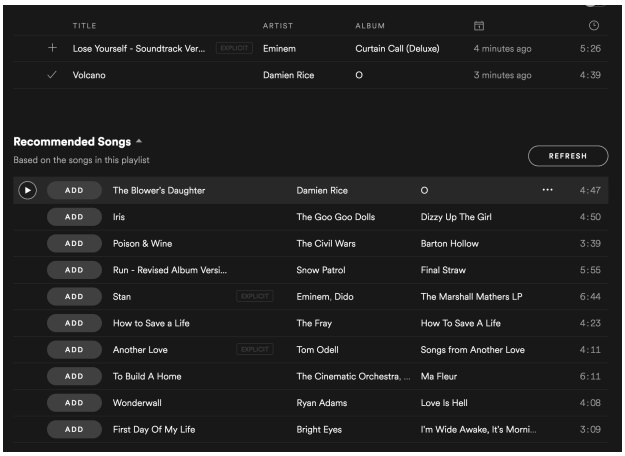

Figure 5. 2 Very different songs With Spotify's Playlist Recommender

```
Song seeds (seperated by ;) :Damien Rice – Volcano;Eminem – Lose Yourself
Some songs were missing in the list of tracks
[313826, 115227, 321896, 354620]
['Damien Rice – Volcano', 'Eminem – Lose Yourself']
Pick Variety rate from 0 to 1: 0.5
Playlist length: 25
Eminem – Lose Yourself
Eminem – Without Me
Damien Rice – Volcano
Eminem – Sing For The Moment
Eminem – 8 Mile
Eminem – Rabbit Run
Eminem – Like Toy Soldiers
Fat Joe – I Can Do U (Album Version – Explicit)
Busta Rhymes – World Go Round
Damien Rice – Amie
Eminem – Cleanin' Out My Closet
Eminem – The Real Slim Shady
Damien Rice – Delicate
Eminem – Just Lose It
Damien Rice – Woman Like A Man
Eminem – FACK
Amos Lee – Lullabye
Jeremy Warmsley – 5 Verses – Album Version
D–12 – Rap Game
50 Cent – Places To Go
Alexi Murdoch – Orange Sky (Album Version)
Rakim – R.A.K.I.M.
Obie Trice – Adrenaline Rush
Damien Rice – Lonelily
Eminem – When I'm Gone
```

Figure 4. 2 Very different songs at Variety Rate - 0.5