ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Transitional playlists for new musical discoveries

Project of:
A Network Tour of Data Science
EE-558, Autumn semester 2018

Authors:

Martial Bernard-Michel
Ludovic Coullery
Victor Faraut
Christophe Muller

Teaching assistants :
Michaël Defferrard, Effrosyni Simou,
Hermina Petric Maretić, Rodrigo Pena,
Eda Bayram, Benjamin Ricaud

Professors :
Pascal Frossard and Pierre Vandergheynst

1 Introduction

In this project, we use graph processing techniques to study a specific dataset to discover or implement something new with it.

1.1 Dataset

The dataset we use is a subset of 9300 out of more than 100'000 music tracks taken from the Free Music Archive (FMA) library. It consists in metadata and several features of these music tracks. They are divided into 161 genres such as Jazz, Folk, Pop or Classical. The features of the songs have been previously extracted with the librosa library, the music can, therefore, be classified according to, for example, their tempo, their "danceability", their "acousticness" or about twenty other features.

1.2 Goal of the Project

A feature often presents in music apps and services is the possibility to find similar tracks according to how they sound like and create playlists based on the tastes of the user. These kinds of features are useful and work well most of the time, but they won't make you discover new styles or genres of music you may like. The primary goal of this project is to give anyone a new way to discover music pieces. The idea is to create a playlist starting from a known music track and linking it to an unknown one. This playlist of typically a dozen songs should consist of a smooth transition between the two chosen tracks. The developed program should find a path made of subsequent songs quite similar to each other but still looking more and more like the ending song.

2 Graph construction

To reduce the computational cost overall for the next steps, the graph should not be too connected. This is explained by the fact that to find a good and smooth way to link two tracks (the nodes) we will need to compute many different paths and the time it takes depends a lot on the number of edges in the graph. A simple problem we might get into is the impossibility to make a playlist if there is no existing path between the two selected nodes. It is, therefore, necessary to have a fully connected graph.

The original dataset is composed of around 9300 tracks. The computational cost to compute paths is very high if we consider all of them. As mentioned before the graph needs to be fully connected, and so it will create a lot of paths even for a relatively short playlist. To reduce this computational time and cost, a smaller subset of tracks is created. The popularity (or how many times the song was listened to) of the tracks is used to determine which tracks are discarded, and only the top 10% are kept. From here we keep only selected music genres, only genres with enough remaining tracks (>10) and representing a wide enough range of musical diversity are kept. The selected genres are rock, hip-hop, electronic, pop, classical, jazz and international. It can be easily changed to other genres as well.

When building the adjacency matrix, we tune the weight threshold in order have the biggest connected component still representing the vast majority of the data (*i.e.* avoid having several big separated components) while keeping the number of edges to a reasonable amount. This is done in order to reduce the cost of subsequent computations and ensure the resulting graph diameter has a maximum of 10. The final graph has 795 nodes and 16'786 edges (the selection of the most significant component removed only 2 separate nodes).

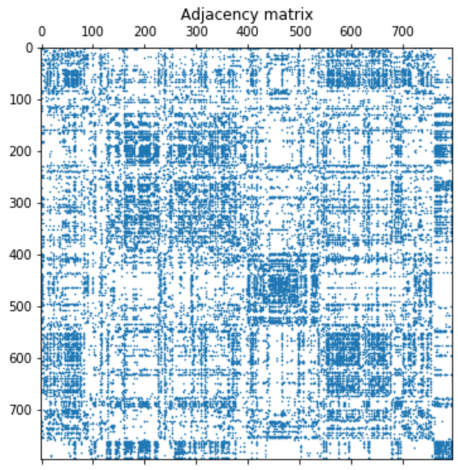


Figure 1 – Adjacency matrix

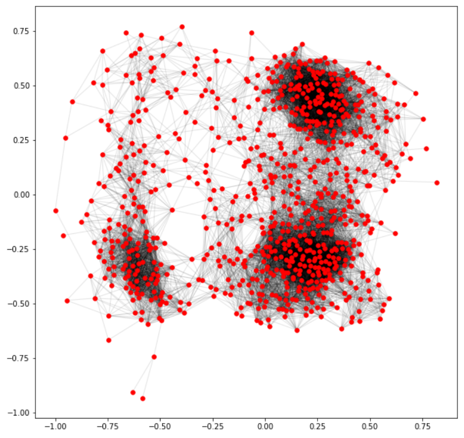


Figure 2 – NetworkX graph

3 Computation of all possible paths

To obtain the smoothest playlist linking two tracks ("start" and "target") we need to compute the smoothness of all possible paths of the desired length.

The main problem with computing all the paths between two nodes of a certain length (say 10 for an 11-tracks playlist) in a well-connected graph is that it is very time-consuming. In order to compute all the paths of length 6 or more is very long. Finding using this method an 11 tracks playlist seems unrealistic.

To overcome this limitation and still generate a decent playlist, several tricks are used. They utilise some properties of the graph. For instance, computing the length of the shortest path between start and target nodes can give an idea of the size of the playlist linking them. If they are particularly close, we might not generate a long playlist that will include tracks more distant to the start and target nodes than they are between themselves.

Even then, the computation time to find all paths is very long and would guarantee an inferior user experience since no one wants to wait hours for a playlist.

The concept developed to overcome this problem is to divide the path into two halves at an intermediary node to compute paths with a length of 5 at most. The method to determine this node is as follow: we list all points at a distance of x from the source node and all the points at a distance of y from the target (with x and y both ≤ 5). We then find the nodes that are common in the two lists, since x and y

are chosen such that $x + y$ is smaller or equal to the graph diameter 10 there will always be some common nodes.

We can now pick a node from this list of potential intermediary nodes, compute all paths of length x linking it to the start and all of the length y to the target, assess the smoothness of each path and keep the best one.

4 Find the smoothest path

In order to compute the smoothness of the paths created, it is needed to base the computation on some metrics. The FMA dataset already consists of a lot of processed metrics (acousticness, danceability, instrumentalness, energy, liveness, tempo, speechiness). So the built-in metrics are directly used as they are. Other metrics are computed using the temporal data of the dataset and performing an average and the standard deviation over all of this temporal data.

We can select the metric or metrics we wish to use in order to compute this smoothness. In order to take them into account to create a playlist, the smoothness is computed with:

$$f^\intercal L f = \|\nabla_{\mathcal{G}} f\|_2^2 = \sum_{i \sim j} W_{ij}(f_j - f_i)^2 \tag{1}$$

This uses the difference between metrics of two connected nodes in the computed path and multiply it by their corresponding weight from the adjacency matrix. In our case, the weights are close to 1 (between 0.885 and 1) for connected nodes. The sum is done over all the nodes in this path for this metric, and this is done over all the different paths and metrics chosen. Regarding the metrics used, some of them have a big variance and others don't. This is cancelled out with by normalizing the values. The final smoothness over each path is the sum of the smoothness of each metrics over each path.

In the end, the playlist is done using the path with the lowest smoothness value. The nodes used for this path are plotted on a graph with all the active nodes highlighted and the music composing the playlist are displayed.

5 Results

We show in this section one possibility of result. In this example, we selected the nodes shown in Figure 3 and used the average and standard deviation of the temporal features as metrics.

Equation 1 was taken from the tutorial 7 of the course "Network tour of data science" at EPFL. See https://github.com/mdeff/ntds_2018

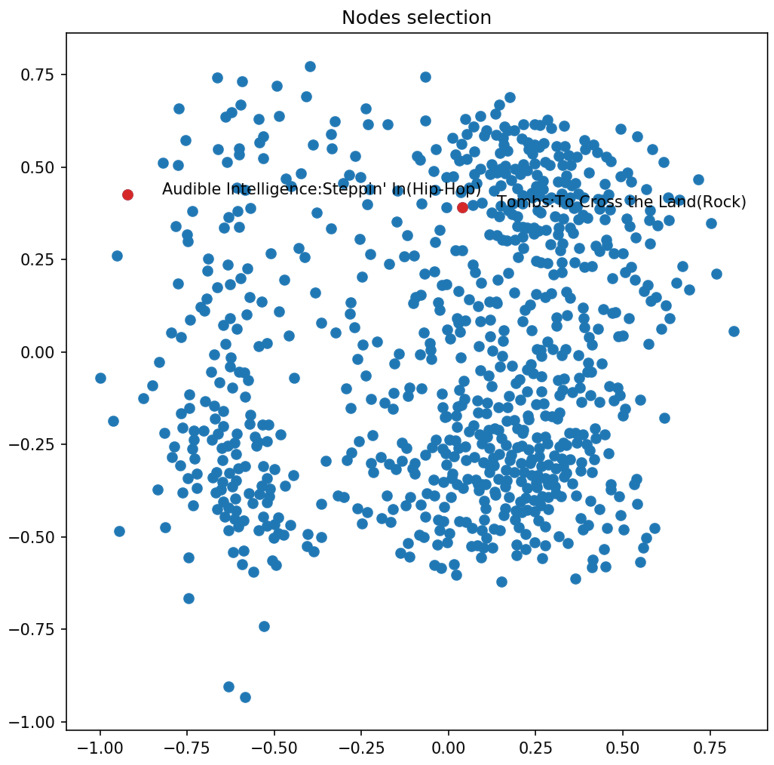


Figure 3 – Selected nodes shown on the graph with their corresponding information

The randomly chosen node between the possible middle nodes is the node 697. The smallest smoothness computed over the two parts of the path are respectively 0.637 and 2.644. As expected, those values are pretty low.

The final path that will be used to build the playlist is composed of the following nodes: 214, 328, 248, 364, 493, 697, 697, 494, 758, 37, 676 and 9, they are displayed in Figure 4.

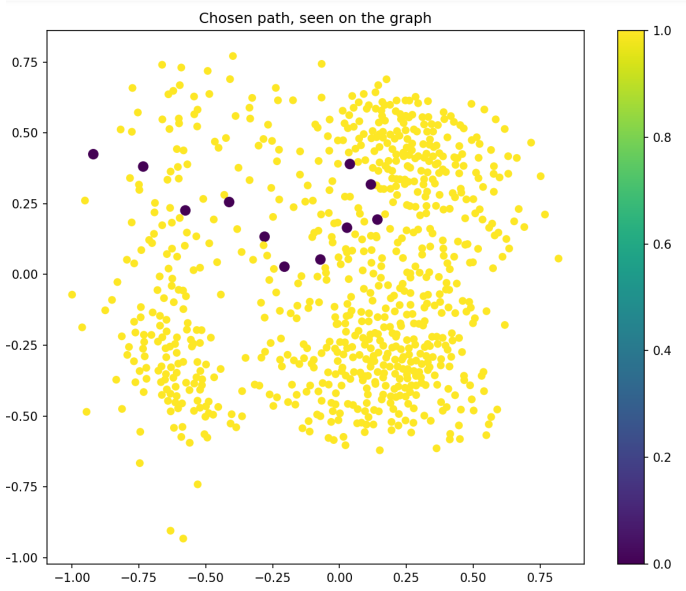
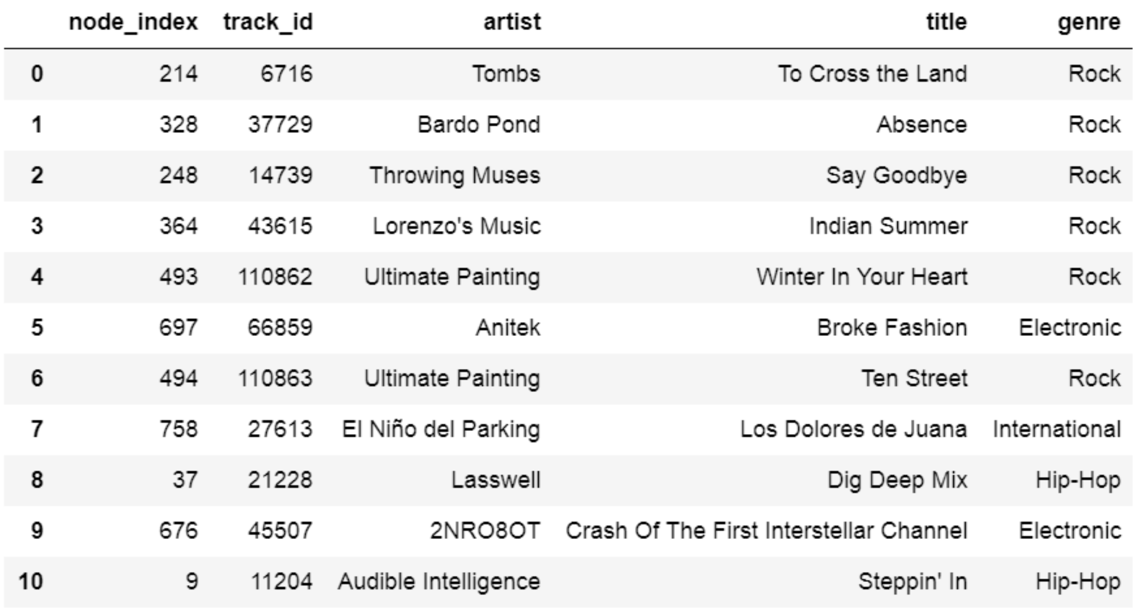


Figure 4 – Illustration of the track used in the path. The purple nodes are the ones selected for the playlist.



	node_index	track_id	artist	title	genre
0	214	6716	Tombs	To Cross the Land	Rock
1	328	37729	Bardo Pond	Absence	Rock
2	248	14739	Throwing Muses	Say Goodbye	Rock
3	364	43615	Lorenzo's Music	Indian Summer	Rock
4	493	110862	Ultimate Painting	Winter In Your Heart	Rock
5	697	66859	Anitek	Broke Fashion	Electronic
6	494	110863	Ultimate Painting	Ten Street	Rock
7	758	27613	El Niño del Parking	Los Dolores de Juana	International
8	37	21228	Lasswell	Dig Deep Mix	Hip-Hop
9	676	45507	2NRO8OT	Crash Of The First Interstellar Channel	Electronic
10	9	11204	Audible Intelligence	Steppin' In	Hip-Hop

Figure 5 – Ordered tracks of the created playlist.

If we listen to the tracks in the playlist it goes through very different music, but they should all contain the same temporal features as the playlist is based on that. This is confirmed by the value of their temporal features on all the track from the playlist.

We can see that our choice to pre-separate the path into two different paths is visible here. As the middle point was chosen to be an electronic style music, it allows the playlist to not go directly from the starting point to the target point (staying in the Rock genre) but rather goes through some different styles. This means that the result we get is maybe not the smoothest path between the selected nodes from a human point of view (maybe we would have had a smoother path overall the playlist if we didn't separate it in two path).

6 Conclusion

To conclude, we really enjoyed working on this project. One of its limit was the computation time resulting of the computation of all the paths between the selected nodes. To decrease the latter, we had to reduce the dataset that we used for our graph and also randomly choose a middle node in the path. We get satisfying results that are always fun to try to understand by listening to the different music of the playlist.

7 Appendix: README file

NTDS project: Transitional playlists for new musical discoveries

This project is done in the frame of the EPFL course *Network tour of data science*. The main goal is to take some of the most listened tracks found in the *Free Music Archive* and help someone wishing to discover a new style or a new artist to transition smoothly from a song he knows and likes to an other unknown music. Basically the idea is to let the user choose two points in a graph representing the music tracks and return him a 10(max)-tracks playlist gradually changing from musics similar to the first track to others more like the the second track.

Code and notebooks

- `Load_save_dataset.ipynb` loads the metadata from the dataset used in this project. The data with scripts to process it can be found on the fma github repository, it is contained in this archive: fma_metadata.zip. The two files used for this project are `tracks.csv` and `echonest.csv`, the relative path to these files can be modified at the beginning of the notebook. This first notebook loads the data with the Pandas library, keeps a subset to create and save an adjacency matrix as well as key features of the selected music tracks.
- `Project_ntds_2018_team36.ipynb` loads and uses the data processed by the first notebook. It uses the NetworkX library to create a graph out of the adjacency matrix and compute paths between selected nodes (music tracks) to create a new playlist. It is important to NOT run the whole notebook in one go! The user should click on the two desired nodes in the graph created in section **Node selection** before running subsequent cells.

Dependencies:

- numpy
- pandas
- matplotlib
- scipy
- math
- collections
- pyunlocbox
- networkx