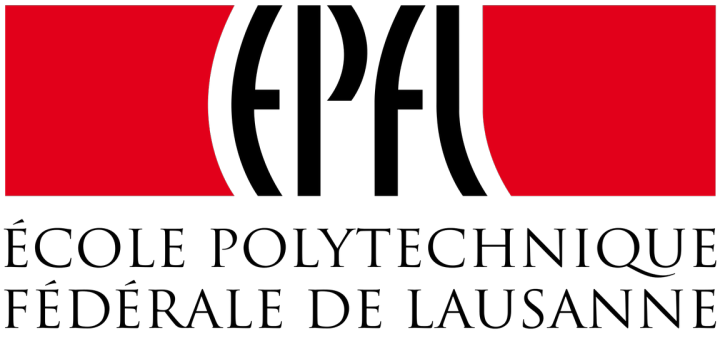


A Network Tour of Data Science

Smooth Radio

Automatic playlist generation using signal graph processing

Group 14

Pierre-Antoine Desplaces

Guillaume Gavillet

Lennard Ludwig

Simon Maksay

Prof. Frossard Pascal, Prof Vandergheynst Pierre

January 18, 2019

1 Introduction

Let's say you are with a friend. You both like different kinds of music but you want to listen to some music together. Wouldn't it be great to have a playlist that will transition from one genre to a different one in the smoothest way possible ? Or let's say you host a party at your house and you want to start with something quiet music and finish with something dancier without bothering to choose all the songs in the playlist.

The aim of this project is to automatically create a playlist that links two songs seamlessly by chaining songs that are similar.

Such automated "intelligent" playlists are usually based on recommendation systems. In this project, we propose a new approach, by constructing a graph based on the audio features of the songs and finding a smooth path on it.

2 Methodology

2.1 Dataset

The data used comes from the Free Music Archive (FMA). It consists mostly of 4 files :

- one has all the metadata on the artist, track, album, etc.
- one contains a list of 518 MFCC-derived audio features extracted with LibROSA.
- one forms a hierarchy of 163 genres.
- the last one contains audio features provided by Echonest (now Spotify) .

2.2 Graph construction

First, all the features are normalized accross all songs using the following formula :

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}, \text{ where X is a feature vector} \tag{1}$$

This way, all the features have the same importance when computing the distances. Our data is separated in 3 groups of features : Echonest's audio features, temporal features and the LibROSA audio features. For each group, the pairwise distances between all songs are calculated using the cosine or the euclidean metric.

The 3 distances are then combined linearly such that the sum of their coefficient is equal to 1. The edge weights are computed as simply the opposite of the distances and the adjacency matrix is created by keeping only the strongest weights by using a threshold.

The graph is then simply constructed from the resulting adjacency matrix and finally its giant component is extracted in order to obtain a connected graph.

2.3 Data Exploration

To observe the quality of a graph, the only qualitative feature available is the top genre of the songs.

In order to tune the groups coefficients, three different methods are used. The first one is to directly look at the spring layout of the graph, and measures its quality by considering how easy it is to distinguish the genre clusters. It is quite subjective and only used as an indicator.

The second metric is the clustering coefficient of the graph. It basically measures the presence of clusters in the graph. Alas, this coefficient does not take into account the real genre.

Therefore, we created a third metric, baptised the Genre Distinction Coefficient (GDC). It takes into the connectedness of each genre and their "anti"-connectedness with the other genres. By using the ratio between those two, we aim to maximize the strength of the connections inside a genre and minimize the ones outside, in a similar fashion as the K-Means algorithms. It is calculated as follows :

$$c_{int}(g) = \frac{1}{|E(G_{in})|} \sum_i d_i \quad i \in E(G_{in}) \tag{2}$$

$$c_{ext}(g) = \frac{1}{|E(G_{ext})|} \sum_i d_i \quad i \in E(G_{ext}) \tag{3}$$

$$GDC = \sum_g \frac{c_{ext}(g)}{c_{int}(g)} \tag{4}$$

Where d_i is the distance of the edge i, $E(G_{in})$ is the set of edges in the subset of our graph containing only tracks of genre g (for example, all edges going from classical music to classical music), and $E(G_{in})$ is a subset containing all the edges that bounds nodes from the latter subset to nodes outside of itself (for example, all edges going from classical music to any other genre).

2.4 Shortest path algorithm

The standard algorithm to compute a shortest path is Dijkstra's. We cannot use A^* as we do not really have a heuristic. Using Dijkstra, we minimize the sum of the distances (not the weights !) in the paths. It usually results in a quite direct path from the start song to the end song.

However, the smoothness of this path heavily depends on the layout of the graph. Indeed the transitions are not necessarily smooth as it can jump from genre to genre because this path is shorter even if there exists a slightly longer path with less abrupt transitions.

2.5 Smoothest path algorithm

Our goal is to find a smoother path between two songs. In order to to this, this notion of "smoothness" has to be defined. This can be seen as trying to minimize the distance between each step and not overall. However, the playlist still needs to connect the two chosen songs in a reasonable amount of jumps.

This problem is well defined in various papers such as the study of J. Andrew and H. Elaarag [3] or the work of L. Liu and R. Wong [5]. In both studies, the notion of smoothness is defined with respect to a signal on the graph. In those cases, the signal is the topology of the ground. In the case of the music, a similar measure needs to be found in order to have a "topology map" of the dataset.

By placing two dirac impulses on the start and end track respectively and then applying a heat kernel to this signal, an heat signal representing the relevance/similarity of each song with respect to the start and end tracks is obtained. With this "elevation map", both methods described by that papers reference beforehand could be applied. In the work of L. Lui and R. Wong [5], a vertex decimation methodology is used. With this method, each node of which the angle to the next one is too steep, is removed from the graph with all its edges. In the paper of J. Andrew and H. Elaarag [3], a methodology based on the A^* algorithm is used. They modified the Heuristic of the algorithm to take into account the topology of the graph.

The method used in this project is inspired from those papers : a new measure called "cost" is computed for each edge and it is simply the sum of its distance and the difference of the two connected nodes in the signal :

$$cost_{ij} = d_{ij} + (h_i - h_j) \tag{5}$$

where h_i and h_j are the altitude/heat associated with two adjacent nodes i and j.

The Dijkstra algorithm can now be applied to find the shortest path in terms of this cost.

3 Results

3.1 Clustering

The following tables contain the results for all different configuration of graphs.

audio coeff.	temporal coeff.	librosa coeff	clustering coeff.	gdc
1	0	0	0.40768	8.32282
0	1	0	0.35337	8.30906
0	0	1	0.42140	8.36297
0.4	0.2	0.4	0.39672	8.40572
0.2	0.2	0.6	0.39614	8.41026
0.3	0.1	0.6	0.40566	8.40892
0.2	0.1	0.7	0.39988	8.40819

Table 1: Table of coefficient using the cosine metric

audio coeff.	temporal coeff.	librosa coeff.	clustering coeff.	gdc
1	0	0	0.40943	8.20578
0	1	0	0.34563	8.20542
0	0	1	0.41509	8.22717
0.4	0.2	0.4	0.41190	8.23486
0.2	0.2	0.6	0.38778	8.23939

Table 2: Table of coefficient using the euclidean metric

The first observation is that when using only a single group, independently of the metric, the LibROSA audio features give the better clustering coefficient and GDC. This is also the case by looking qualitatively at the graphs (see table 4 in the appendices). Therefore, a bigger number is asses to the LibROSA coefficient for the linear combinations.

The second observation is that the cosine metric works better than the euclidean metric (greater clustering coeff. and GDC) when using only a single group. This aim is also supported by the discussion of similarity metrics from Philip Shanley [2] and the post from Chris Emmery [4]. The cosine metric treats the genres of music more as a direction than a distance in the graph.

Finally, it can be seen that when doing the the linear combination, the clustering coeff. and the GDC does not agree. The credit is given to the GDC as it consider the real genre. Therefore, the best linear combination is the [0.2, 0.2, 0.6] and is shown in the following figure.

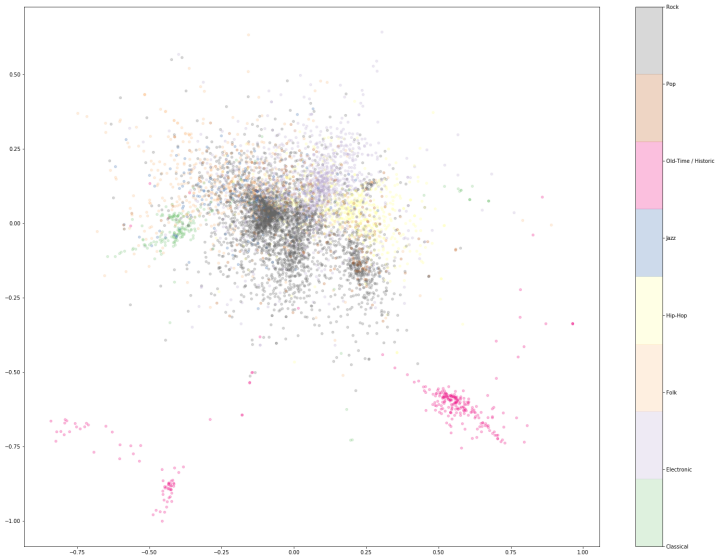


Figure 1: Graph using cosine linear combination Echonest a.f. = 0.2, Echonest t.f.= 0.2, LibROSA a.f.= 0.6

3.2 Shortest Path vs Smoothest Path Algorithm

As expected, the smoothest path yields better results in terms of transitions than the shortest path. To measure the smoothness, the average cost of each edge of the path is calculated.

shortest path	0.2812
smoothest path	0.1750

Table 3: Average cost of both methods

The results show that in the smoothest path, the average cost between each node is much smaller than in the shortest path case. This way, the generated playlist will have much smoother transitions between songs as it can be seen in Figure 2.

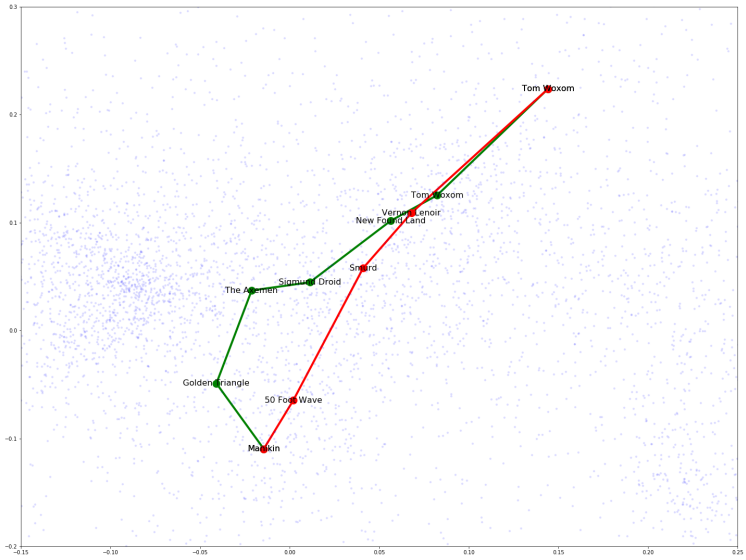


Figure 2: Generated playlists using both methods (red: shortest, green: smoothest)

In a more qualitative way, it can be observed that the green path (smoothest path) is longer, but the length of each jump is smaller in general than in the red path (shortest path). It can also be noted that the shortest path is almost a straight line whereas the green path follows the topology of the signal. On the upper part of the path, it can be seen that the smooth path get closer to the target on the penultimate node.

4 Conclusion

During the graph construction, the focus has been put on having a good genre separation, which was chosen to be the main criterion to evaluate the graph. We then study the shortest path between two nodes and notice that it lead to big variations between songs on that path. Hence minimizing the sum of distance leads to some brutal transitions.

Therefore we added an information to our graph that would allow us to quantify those variations by filtering an impulse signal starting from the start and end songs. The new cost function obtained is minimized as a linear combination of the distance and variation in signal.

The results obtained are convincing and complete the initial aim. Even though the generated path can be longer (which is not actually a problem for playlist generation), the transition between each songs are smoother.

References

[1] DEFFERRARD, Michaël, BENZI, Kirell and VANDERGHEYNST, Pierre:
Fma: a dataset for music analysis arXiv preprint arXiv:1612.01840, 2016.

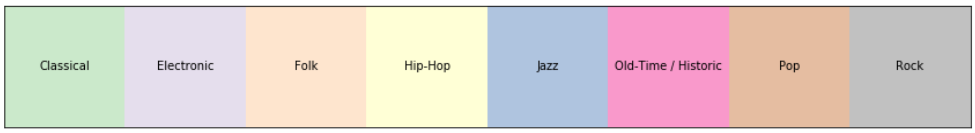
[2] SHANLEY, Philip: *Data Mining Portfolio*
`http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/sphilip/similarity.html`

[3] ROLES, J. Andrew and ELAARAG, Hala: *A smoothest path algorithm and its visualization tool*
In : Southeastcon, 2013 Proceedings of IEEE. IEEE, 2013. p. 1-6.

[4] EMMERY, Chris: *Euclidean vs. Cosine Distance*
Reply in the Data Mining course, the 25.03.2017. `https://cmry.github.io/notes/euclidean-v-cosine`

[5] L. Liu and R. Wong, *Finding Shortest Path on Land Surfaces.*
Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, 2011

Appendices



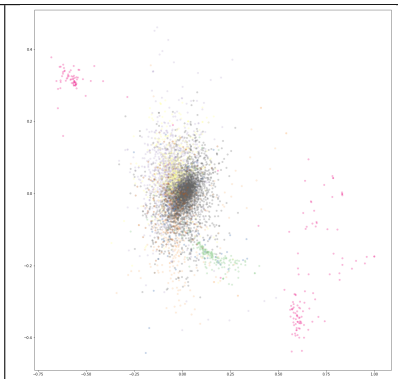
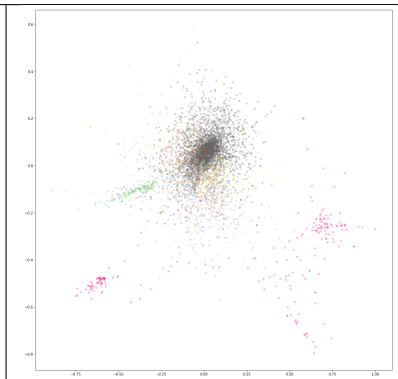
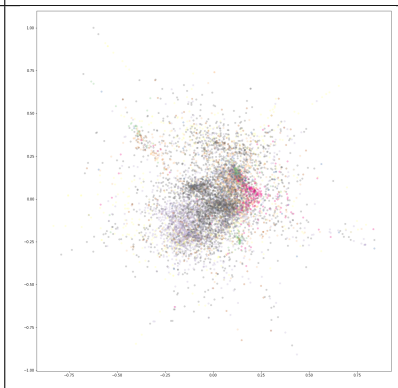
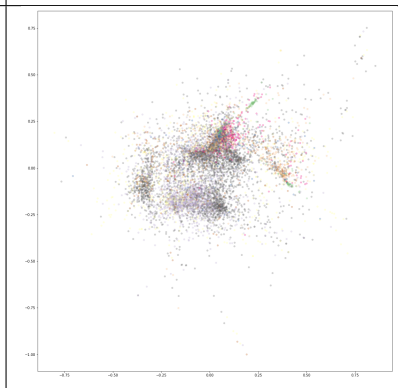
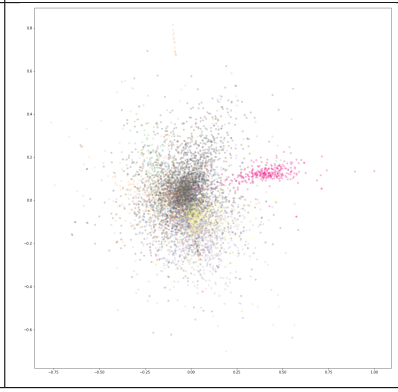
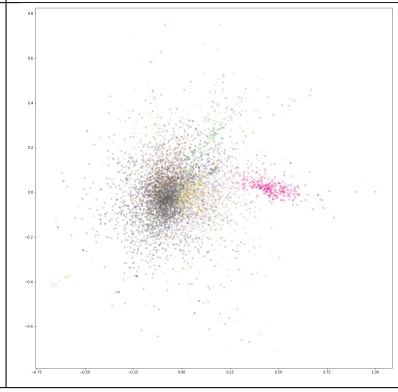
	Cosine norm	Euclidian norm
LibROSA only		
Audio features only		
Temporal features only		

Table 4: Graphs using the audio features separately