Network Tour of Data Science:

Free Music Alternative Playlists

L. Biotto & A. Buchegger & J. Tapparel & T. Tuuva

1 Introduction

It can be tough to know what we would like to listen when faced with an archive of thousand of musics. For our project we wondered how we could automatically generate a list of songs that would correspond to one's taste. We will explore the features of the Free Music Archive (FMA) to generate a graph that will contain many songs of selected genres. These songs can then be sorted and a playlist be generated out of a region of this graph. We will explore different methods of generating playlists. At the end, we will show how this tool can be used to generate a free alternative playlists from non-free musics.

2 Generating the graph

Thanks to Free Music Archive, we have access to a dataset of music from which we have a list of extracted features. In order to construct a graph from this song list, we considered each song as a node and calculated distance between each node in the features space. We normalized all these features to a same weight. After trying different distances, the euclidean one seems to give us the best separation between genres which could indicate that similar songs would end up close to one another. We can then use these distances to define the weight of the edges between each nodes using the following formula :

$$\text{Edge}_{\text{weight}} = e^{-\text{dist}^2/\text{dist}^2_{\text{mean}}}$$

This would give us a strongly connected graph which would be long to process due to the high number of edges and having weakly weighted edges doesn't give very useful informations about the data structure. We can solve this problem by defining a minimal weight to keep an edge in the graph. One issue in doing that is the apparition of multiple components in the graph. For the future usage of

this graph, we need it to be connected and to ensure that, we will only take into account the principal connected component and drop the nodes that aren't in it. In our implementation, we set the threshold value in order ensure that around 90% of the nodes are in the main component.

Finally we place each node in a bi-dimensional space using a spring layout. This means that each node tries to get as far away from the others as it can, while being held back by the edges which are assimilated to springs, having a spring constant related to their corresponding weight.

Since processing the graph and placing the nodes in a 2D space requires an important amount of time, we limited the dataset to three balanced genres. The result of the created graph is presented on the figure 1. However, these parameters can be chosen by the user at the moment he creates his own graph depending of his time and resources. Moreover, adding more genres seem to make them less distinguishible by using this graph calculation. More clever use of the features would probably help. For example, by using machine learning, we could tweak the weight of each feature to better separate more genres.

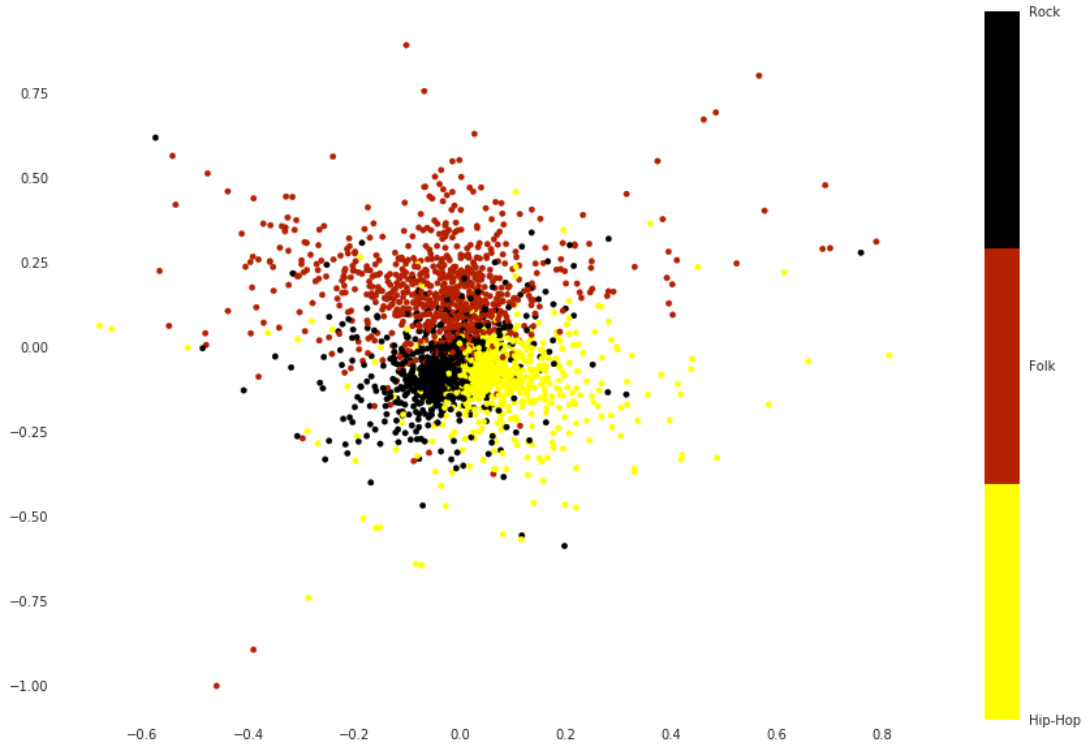

FIGURE 1 – Spring layout coloured by genre

3 Including our own songs

To include our own music into the graph we first need to extract its features, this is done by using the provided *features.py* script(TODO REF). There are 518 features in total. Then we can include the song in two ways :

— Recomputing the graph adjacency matrix with the new songs included.

— Creating a KD-Tree of the graph, then searching for the nearest points in the K-features space for the included songs

3.1 Inclusion by recomputation

After having added the songs and their extracted features to the dataset, we simply generate a new graph in the same way. As expected it is a very time consuming method, getting worse as the dataset grows bigger. This led us to the next method.

3.2 Inclusion by nearest neighbour search

Instead of recomputing the adjacency matrix and the corresponding graph, both having a high computational cost, we can estimate the new nodes position. To do so, we generate another kind of graph directly from the features, in our case we chose a KD-Tree. An example of a 2D-Tree is presented in the figure (2)

This kind of structure is conceived to quickly find the nearest neighbours in an efficient way. We decided to consider that the closest neighbour to our newly included song, in the features space, is going to be assimilated to the new node. Hence, when we want to extract a playlist, this existing node will be used.

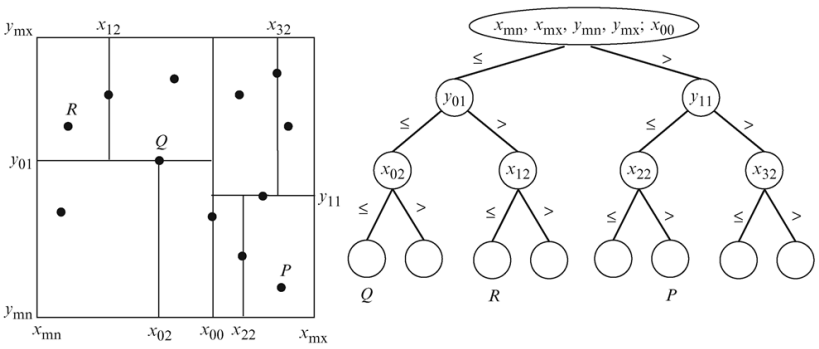

FIGURE 2 – Example of a 2D-Tree

4 Generating the playlist

To generate a playlist we explored three distinct methods :

— Nearest neighbours in the features space

— Nearest neighbours in the 2D-space of the spring layout graph

— Hottest nodes from a heat diffusion in the graph

When listening to a playlist, it is nice to be able to tell how adventurous we feel regarding the inclusion of other styles or genres. Therefore we let the user the freedom to choose its own preferred generation method.

4.1 518D features exploration

First, we can create a playlist directly from the euclidean distance between features. This property should result in similar features, such as rythm, or height of voice of example, but being more adventurous in regards to genres. One could for example end up listening to hip-hop when starting from rock.

4.2 2D graph exploration

Secondly, we select the closest songs around the selected music (node) in the graph. This effectively results in a kind of circle around a point. This type of playlist will thus be very targeted towards a specific kind of song. For this purpose we use the simple euclidean distance in our 2D graph.

4.3 Graph heat diffusion

Finally, we generate a heat diffusion around selected nodes, and create a playlist from the "warmest" nodes. To simulate this heat diffusion in the graph, we use the functions of PyGSP allowing us to generate the heat kernel filter and apply it to our graph. We provide a graph in node domain but the filtering is performed in frequency domain. The heat diffusion parameter "tau" can be tweaked by the user.

With this method it is possible to mix the diffusion from each starting node which results in a playlist being between every given song. For example, if we give a rock song and a rap song, we will get some songs which are a fusion of rock and rap.

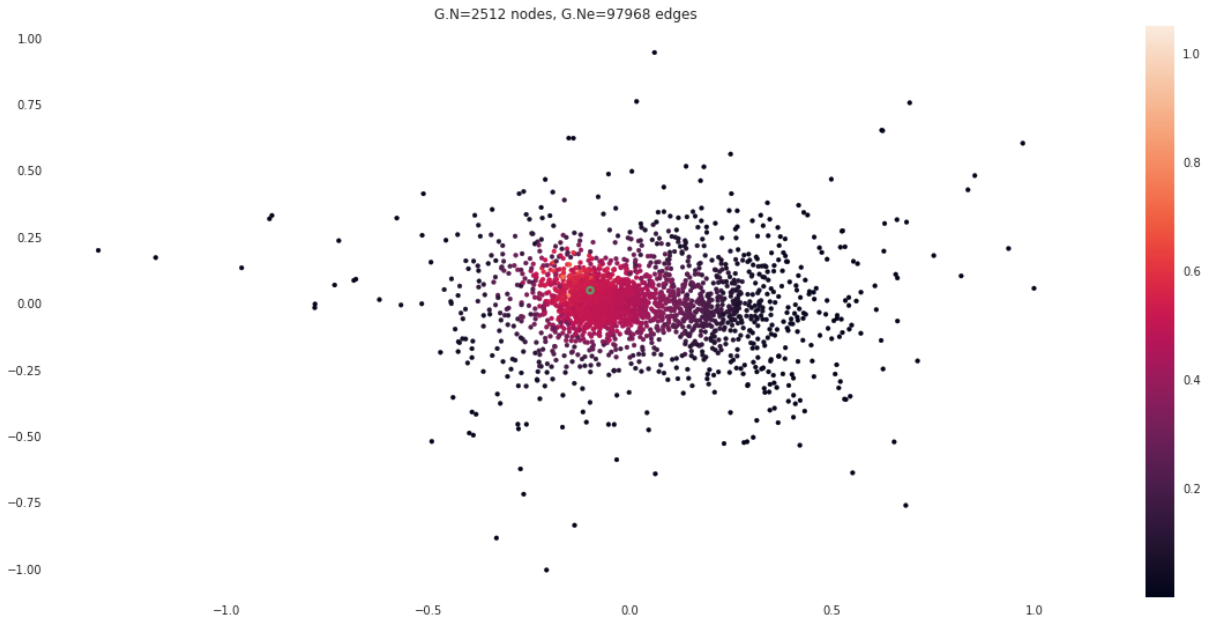

FIGURE 3 – Heat spreading starting from one node(green marker)

5 Conclusion

By listening to the generated playlists, we can observe that the extracted songs are related in some way to the input songs, however the difference between the different methods is quite subtle and subjective. The heat spreading method gives interesting results. It is interesting to note that using only the 2D graph, we get good results, confirming a good classification of the songs.

By using what we learned during the course and the project milestones, we were able to gain and use some knowledge on graphs and Python to generate awesome playlists.