Graph mining - WS 2016 - Project Report

Maxi Fischer, Sören Tietböhl

Due date: 26 January 2017

Summary

Analysis of the Spotify Artist Network using an ego-based community detection algorithm.

1 Summary of the data

Spotify is a music streaming platform, hosting over 30 million songs and more than 2 million artists. The artists are associated with each other via the 'related artists' feature. This relation induces a graph, which is the subject of this project.

Because the related artists have to be queried one-by-one from the Spotify API we are only using a subset of the complete graph. Our subgraph is constructed by starting at an arbitrary artist (we chose the "Red Hot Chili Peppers") and then using breadth first traversal. The total number of nodes in the subgraph is 43907, the number of edges is 172350. The 'related artist' relation goes both ways, so its an undirected graph. We can assume that in reality the graph is dynamic, because new songs and thus artists will be added all the time. The nodes are not labelled, but various information on the artist relation can be requested, such as genre and popularity.

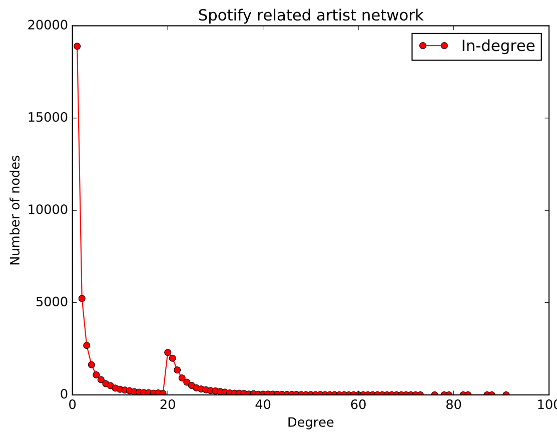

Figure 1: Degree Distribution

The number of triangles is 1105986. The average clustering coefficient is 0.2522.

2 Description of the algorithm

We chose two community detection algorithms: the Louvain-algorithm [2] and the k-Clique-Communities [3]. For the Louvain algorithm we chose the bestpartition function and for the k-Clique-Communities k = 2 to get the maximum number of partitions.

Briefly, the Louvain algorithms is a hierarchical clustering algorithm that greedy deletes edges that cause high modularity. In the resulting dendogram it chooses the best partition possible of the graph. The Louvain algorithm implementation takes a graph as an input and returns a dictionary with all nodes as keys and their community assignment as values.

The k-Clique-Communities algorithm looks for all cliques of size k reachable by adjacent k-cliques. It takes the graph as an input and returns a list of sets of nodes, each set is a community.

They are both usable on undirectional graphs.

3 Preprocessing and storing

In order to obtain the graph we query the Spotify API at [1]. The related artist query returns 20 artists that are related to the given input artist. From there we continue with breadth first traversal and query the API for all the new artists. The nodes are named by the artists id on spotify, which is returned by the API. Additionally we also store all the artist related information (e.g. genre, popularity etc) in a dictionary that maps from artist id to artist info. The graph is then written to disk as an adjacency list in plain text. The artist information is stored in json format.

Before the algorithms are run, they will load all the stored information into main memory. This is done to save time, as rebuilding the graph via API takes a long time.

4 Network analysis

Our goal is to detect communities in the graph and then compare the resulting communities with the genre distribution of the artists. We want to evaluate which community detection algorithm obtains the best genre-consistent communities. That knowledge is useful to find anomalies in the existing genre assignment and to predict the genre of an artist in the same community. Genre prediction of songs is a complex research topic in Music Information Retrieval. By using the existing assignment of the Spotify artist relations, we can simplify this prediction.

After calculating the communities with the given algorithms, we analyse the number of artists, null values (artists without an assigned genre) and the genre distribution in a community.

4.1 Qualitative Results

We look at the accuracy of the calculated communities by the algorithms. The accuracy is the number of the most present genre in the community divided by all artists in the community without artists with no genre. The goal is to get informations about the genre consistency of a community. If every artist of a community is part of the most frequent genre, you can say that the whole community is part of this genre.

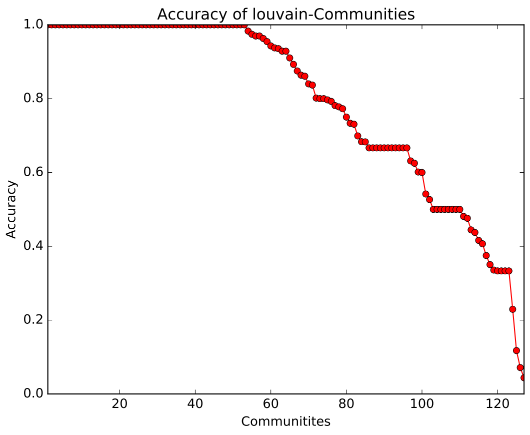

Figure 2: Accuracy of Louvain-communities

In the result of the Louvain algorithm most of the communities are relatively high, so that the communities are pretty consistent.

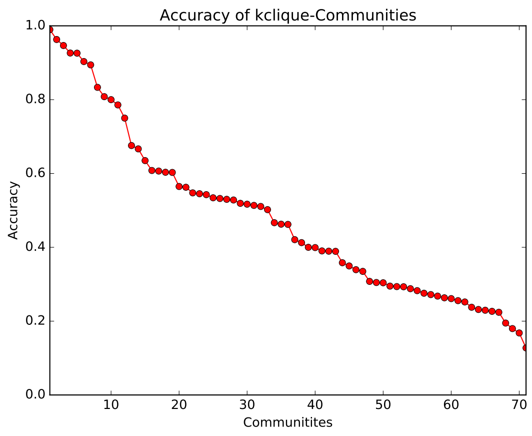

Figure 3: Accuracy of kclique-communities

In comparison the k-Clique-Communities algorithm with k = 2 produces a lot of communities with mixed genres. For this use case the detected communities are not very useful.

Now we analyse the number of null values in all calculated communities. We want to find out how much artists have no genre and can be predicted.

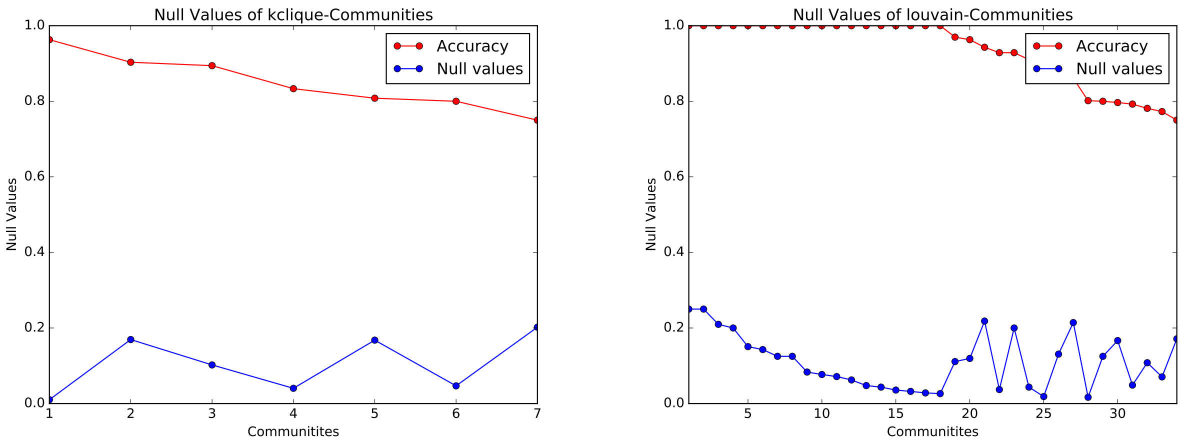

Figure 4: Null value data of kclique (left) and louvain (right)

We chose thresholds to select communities for which it is sensible to predict genres of null_values. The accuracy threshold is 75%, the null values threshold (number of null values divided by the total number) is 25%. The accuracy threshold should ensure that there are enough deciding reference points for the prediction, while the null values threshold cuts the communities which contain a lot of null values. The more null values there are , the less meaningful a prediction will be. We cut communities with no null values too, because you cannot predict any genres in them.

In Figure 4 you see the thresholded communities of both algorithms. While Louvain calculated more than 30 communities where you can predict the new genres, 2-Clique-Communities just found 7 communities.

As you may see the Louvain algorithm works much better in that use case. In general it calculates bigger communities that seem to map better on the general genre distribution.

4.2 Time and scalability

Building the Graph

The total amount of time it took to build the graph is 14 minutes 3 seconds. This includes querying the spotify API for related artists and then storing all the information in the respective data structures.

The artist information has a size of 39 MB, the graph adjacency list has a size of 4.8 MB.

Running the algorithms

The Louvain-algorithm completed in 35.6 seconds.

The k-Clique-communities completed in 92.7 seconds.

For comparison we ran the algorithms on a larger subset of the spotify graph which had 176417 Nodes. On this graph the Louvain-algorithm completed in 199.3s. The kclique algorithm did not complete because our machine did not have enough memory.(We used a small Debian VM with 1 GB RAM)

5 Limitations and interesting findings

Using the Spotify API resulted in some limitations. The graph creation speed depends on the rate limiting of the API and the internet connection. Additionally our findings are strongly limited by the quality of the "related artists" relation. In the case that Spotify has a different understanding of related artists, they could create edges that are not in any way associated with genre, thus distorting our results. Another point we have not yet considered is that you can have multiple quite similar genre labels for one artist and hierarchical connection between genres. For example Älternative Rockïs a subset of Ȑock". With our approach we would just predict Ȑock", not considering subsets.

The Louvain algorithm has a general runtime in O(n log n) so that it should scale quite good for bigger graphs.

The k-Cliques Communities algorithm may scale worse. It depends on finding k-Cliques in a network which may have an exponential runtime. In most applications the algorithm is still relatively fast, especially in a rather sparse graph like ours.

Our findings may help to improve the Spotify API and the general genre prediction of artists in other applications, like artist pages in Wikipedia.

Bibliography

[1] Spotify web api. `https://developer.spotify.com/web-api/get-related-artists/`. Accessed: 2017-01-22.

[2] P. D. Meo, E. Ferrara, G. Fiumara, and A. Provetti. Generalized louvain method for community detection in large networks. In *Proceedings of the 11th International Conference On Intelligent Systems Design And Applications*, pages 88–93, 2011.

[3] G. Palla, I. Dernyi, I. Farkas1, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. In *Nature 435*, pages 814–818, 2005.