NGSA Project

Experimental evaluation of community detection algorithms for musical tastes predictions

Florian Bettini – Maxime Lutel – Gabriel Muller – Benjamin Pommier

Abstract:

Our project aims to evaluate multiple community detection algorithms when applied to predicting musical tastes. We will therefore compare two approaches: one using classical clustering community detection algorithms, the other combining these algorithms with node embeddings.

Performances of both approaches will be evaluated on data collected from Deezer in November 2017 in 3 different countries: Romania, Croatia and Hungary. These datasets contain individuals (nodes) who have relationship (edges) and musical tastes (node attributes). Our approach is motivated by the increasing need in performance within an industry driven by a strong growth and an overwhelming pressure to generate profits.

1. Motivation and Problem Definition

1.1. The Problem

Improving the identification of communities within a musical network could benefit to both platforms and customers by drastically improving the quality and relevance of musical suggestions. The difficulty of assigning individuals to musical communities is that a relationship may not reflect a potential taste for a given genre of music. Consequently, we want to implement a node embedding in addition to a community detection algorithm, and then compare the results with classical community detection algorithms.

1.2. Applications

Algorithms of community detection can be used for both:

- Suggesting new genres to individuals. For example, if someone belongs to a 'pop' community, but is not yet interested in 'pop', it could be relevant to suggest him new pop songs.
- Suggesting new relationships to individuals, thus it could be relevant to suggest a new friend to a user when both are within a same community.

1.3. Related work

Community detection aims to discover groups of nodes on a graph such that the intra-group connections are denser than the intergroup ones. In social network graphs, communities are often overlapped. Hence classical algorithms such as Louvain or Girvan-Newman cannot address the issue of soft-clustering.

New algorithms were developed to address this topic, such as CPM (Clique Percolation Methods), Fuzzy Detection or NMF and PCA based methods, as mentioned in [6].

With the recent development of neural networks and deep learning, node embeddings are used for community detection. The general idea is to benefit from additionnal information from the nodes to generate a vector representation of the node and then learn the community by using unspervised machine learning techniques.

2. Methodology

Our approach will be structured as in this diagram:

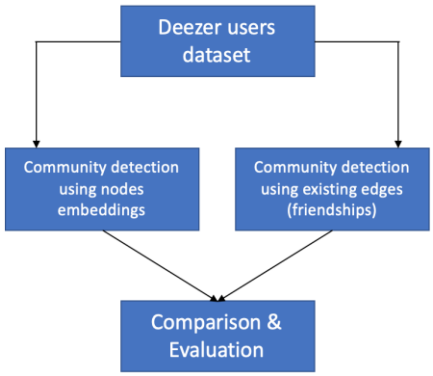

Figure 1 – Methodology

As a first step, we will rely on classical clustering models to detect communities. This approach will be computed thanks to different kind of algorithms:

- Hierarchical clustering (FastGreedy)
- Spectral clustering
- Dynamic algorithms (Walktrap and Label propagation)
- Louvain method

The performance of these algorithms will be compared based on the modularity criteria.

As a second step, we will implement methodologies that learn node embeddings and use them to improve the clustering methods. The embeddings learning step will be performed with methods such as Node2Vec and Deep Walk, and the community detection will rely on a method combining these embeddings with clustering and regularization, which is called GEMSEC (Graph Embedding with Self Clustering). We will test variations of these methods to find the one that is the most suitable for community detection.

The comparison of these two approaches will allow us to understand whether embeddings improve community detection or not, and to quantify its impact on clustering and classification tasks.

3. Evaluation

The Deezer dataset (available *here*) will be used for the entire project. On this dataset, each node has attributes (musical taste: between 1 and 84 per user) which can be predicted with a supervised method thanks to the embeddings learned for each node.

Our methodology results in two distinct outputs: clustering and embeddings. Each of them will be evaluated in a different way:

- Clustering quality will be evaluated thanks to modularity, which will be high if there are dense connections within communities and sparse ones between communities.

- Embeddings quality will be evaluated by using them to perform a multi-label classification task to predict the attributes of each node. We will compare the F1 score obtained when training machine learning models with these embeddings as features in order to determine which method gives the best embedded representation of nodes.

4. Experiments & Results

As explained above, various experiments will be realized in order to attain an optimal clustering of the users:

- Multiple community detections algorithms will be evaluated (Girvan-Newman, Louvain, spectral clustering, …).
- Overlapping and non-overlapping community detection methods will be tested. Only the most fitted to the dataset will be conserved.
- With GEMSEC, new graphs will be generated based on the users' music tastes. Continuous edge creation (each edge has a weighted corresponding to a similarity score) and discrete edge creation (an edge exists only if a similarity score is higher to a threshold (typically 0.5)) methods will be tested and compared.

4.1 GEMSEC: Theory

4.1.1 Graph embeddings: GEMSEC

The purpose of graph embedding is to create a representation of the graph in a vector space which can later be used for multiples machine learning applications, such as labelling, links predictions or graph visualization. Although there are many methods of graph embeddings (such as Deepwalk, Node2vec, Line, Come…) we decided on this project to focus on a specific method called GEMSEC (Graph Embeddings with Self Clustering, *Rozemberczki et al*, 2019) which purposely embedded graphs while conserving community structure. The idea behind GEMSEC is that nodes from a same community will be represented close to each other in the vector space while communities will be clearly separated (see image below).

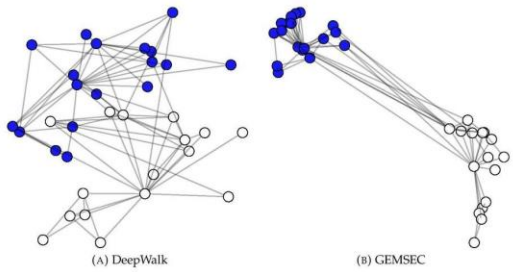

Figure 2 – Illustration of GEMSEC

GEMSEC is thus comprised of three successive phase which will affects the graph embeddings at each iteration:

- A classic embedding phase similar to deepwalk which will generate embeddings consistent with the graph structure.

- A clustering phase which will tends to gather nodes around the cluster centers, thus separating communities while tightening same-community nodes.

- A regularization phase based on graph property adding knowledge of social network communities, thus improving the clustering of communities.

$$Gemsec = Deepwalk + Clustering + Regularisation$$

We will now detail the mechanisms behind the three steps:

4.1.2 DeepWalk

In many ways, the embedding of a graph is similar to the embedding of words document. Indeed, the idea behind Deepwalk is to generate a corpus of graph sequences (similar to sentences in the word2vec model) which will later be used as inputs in a skip-gram model in order to generate embeddings (see figure below).

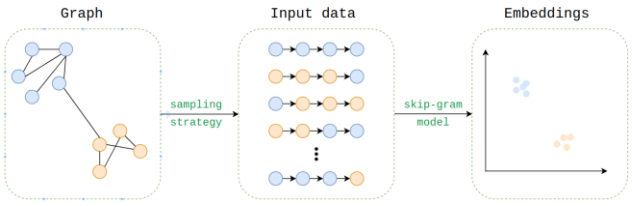

Figure 3 – Process of graph embedding

In order to generate those nodes sequences, we perform a random walk of constant length starting from each node of the graph, and repeat it multiple times, using index of the nodes as tokens. We now have a tokenized corpus made of $Number\ of\ repetitions * Number\ of\ nodes$ walks (~sentences) which can be directly used as inputs to the skip-gram model which will generate embedding through the use of context window features extraction, negative sampling and a softmax function.

The loss of this step will therefore be similar to the skip-gram model one:

$$\mathrm{L}_D = \sum_{v \in V} [\ln (\sum_{u \in V} \exp(f(v).f(u))) - \sum_{n_i \in N_s(V)} f(n_i).f(u)]$$

With:

- V the vertices ensemble of the graph
- $f(v)$ the embedding of vertice v
- $N_s(v)$ the windows of v.

4.1.3 Clustering

Following the embedding phase, a clustering phase is applied to the walk at each step of the algorithm. During this phase, we measure the distance from the nodes to each clusters center and use it as a clustering cost.

$$L_c = \sum_{v \in V} \min_{c \in C} ||f(v) - \mu_c||_2$$

This clustering cost is then added to the total loss, with a weight γ (which is updated at each step following an exponential annealing rule):

$$L = L_D + \gamma . L_C$$

The idea is behind this additional clustering cost is to minimize the distance from each node to its nearest cluster center.

4.1.4 Regularization

In order to add more graph related information to the model, we add another regularization phase following the clustering phase. This regularization cost is described as follow:

$$\mathrm{L_R} = \sum_{(v,u) \in E_S} w_{v,u} . ||f(u) - f(v)||_2$$

where the weight functions w determines the social network cost of the embeddings.

Here $w_{v,u}$ is chosen as the Jaccard similarity coefficient between node v and node u. Indeed, as Jaccard's coefficient is a strong indicator of node relationship in a graph, this regularization cost will enable the algorithm to find embeddings and clusters more natural and community based.

The regularization cost is also added to the loss function with a weight λ leading to the global loss:

$$L = L_D + \gamma . L_C + \lambda . L_R$$

Finally, the features extracted from the deepwalk phase, the clustering cost coefficient from the clustering phase and the regularization cost from the regularization phase determine the update to model weights by the optimizer (Adam), leading to consistent community-based graph embeddings.

4.1.5 Algorithm

The GEMSEC algorithm can therefore be described as follow:

- Initialization of the model
- t = 0
- $Repetitions * Number\ of\ nodes$ random walks are performed and store in a variable "walk".
- For rep in $Repetitions$:
  - For nodes in $Number\ of\ nodes$:
    - t = t+1
    - Upgrade γ
    - Upgrade learning rate (linear annealing rule)
    - $L_D, \text{Features}_t = \text{Deepwalk}(\text{walk}_t)$
    - $L_C = Clustering(walk_t, features_t)$
    - $L_R = Regularisation(walk_t, features_t)$
    - Update Weights
  - End
- End

4.2. Impact of each layer

4.2.1 Impact on modularity

This part aims to evaluate the performance of the GEMSEC algorithm in terms of classification. To do so, a modularity score is computed for each algorithm among (from the simplest to most sophisticated):

1) DeepWalk
2) DeepWalk with Regularization
3) GEMSEC
4) GEMSEC with Regularization

However, we need to know if a node belongs to a specific community in order to compute the modularity of a graph (later expressed as C_i). Indeed, the definition of the modularity score Q is:

$$Q = \frac{1}{2m}\sum_{ij}(A_{ij} - \frac{k_i k_j}{2m})\delta(C_i, C_j)$$

with:

- A the adjacency matrix
- k_i the degree of node i
- m the number of edges in the graph
- C_i the community of node i
- $\delta(.)$ the Kronecker function

Fortunately, GEMSEC algorithms (3 and 4) provide the coordinates of the means of each cluster. Therefore, when computing the modularity, we assign each node to its closer cluster (eg the one with the closest cluster's mean).

On the other hand, for evaluation purpose, we use a KMeans algorithm for DeepWalk algorithms (1 and 2) to provide communities to each node.

Therefore, the modularity score we compute is a great indicator of the clustering performance: the closer it is to 1, the stronger the clusters (and embeddings) are.

We performed this modularity analysis on 3 datasets: Deezer Croatia, Deezer Hungary, and Deezer Romania.

Results are shown in Tables 1, 2 and 3.

Algorithm	Dataset	Modularity
DeepWalk	Deezer Croatia	0.235
DeepWalk with Regularization	Deezer Croatia	0.287
GEMSEC	Deezer Croatia	0.354
GEMSEC with Regularization	Deezer Croatia	**0.407**

Table 1 – Modularity analysis – Deezer Croatia

Algorithm	Dataset	Modularity
DeepWalk	Deezer Hungary	0.275
DeepWalk with Regularization	Deezer Hungary	0.292
GEMSEC	Deezer Hungary	0.351
GEMSEC with Regularization	Deezer Hungary	**0.409**

Table 2 – Modularity analysis – Deezer Hungary

Algorithm	Dataset	Modularity
DeepWalk	Deezer Romania	0.468
DeepWalk with Regularization	Deezer Romania	0.484
GEMSEC	Deezer Romania	0.543
GEMSEC with Regularization	Deezer Romania	**0.549**

Table 3 – Modularity analysis – Deezer Romania

Impact of random walk's order

To go even further, we decide to validate, or not, the results in [1] with regards to the order of the random walk. Indeed, results are shown to be better for the second order.

Based on the Deezer's dataset in Romania, we get the following results:

	Modularity	
Random walk's order	First order	Second order
DeepWalk	**0.468**	0.452
DeepWalk with Regularization	**0.484**	0.481
GEMSEC	**0.543**	0.536
GEMSEC with Regularization	**0.549**	0.519

Table 4 – Impact of walks' order – Deezer Croatia

We then disprove the assessment made in [1] that second order random walks consistently provide better results. This assumption should be put in regard to the structure of the graph. Here, with a non-dense dataset, this assumption does not hold.

4.2.2 Impact on multi-label node classification

GEMSEC algorithm aims at improving the embedding for further clustering tasks. As we focused on the Deezer dataset, another use of these embeddings may be to predict the music that a user may like, by analyzing only its relationships. Each of the user (node) likes one or several genres (Pop, Rock, etc.), thus we face a multilabel classification problem. We adopted the same methodology as [1] to compare the different embeddings.

First, all the nodes are embedded (using GEMSEC, DeepWalk, Node2Vec, etc.) in R^{16}. The embeddings are then used separately as features for the multi-label classification problem. To be able to compare the various embeddings, we choose to use a OneVsAll and a ClassifiierChain approach with a LogisticRegression (C=1). In the OneVsAll method, a classifier is trained to distinguish one specific class vs. all of the other. Thus, there is no relationships between the predictions. However, in the classifier chain approach, It takes into account the previous predictions to predict the other class. We reduced the number of genres to 40 (instead of 88 by keeping only the most "liked") because several genres occur less than 1% of the time. This approach is enough to estimate the performance of the embeddings because we have all the most represented classes.

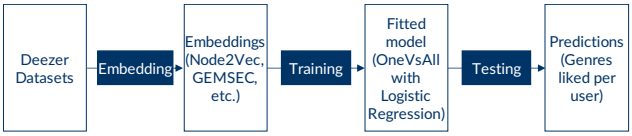

Figure 4 – Process of multilabel classification

During the training process, we might also have included a grid search. It was tried to optimize the parameters, but we didn't include it in the results because:

- Optimal parameters were not the same for all models, hence introducing a bias in the comparison
- Running a real grid search with enough parameters to significantly observe an increase in performance was too computationally expensive

The objective function we use to analyze the performance of the algorithm was the F1-Score. Micro, macro and weighted F1-Score were used. Finally, as we can observe in the table below, GEMSEC outperforms the other embeddings for the classification task. Also, a positive influence is observed for the regularization process.

Country	Croatia		
F1-Score	**Macro**	**Micro**	**Weighted**
DeepWalk	4.99%	32.56%	21.05%
DeepWalk With Regularization	4.90%	32.57%	20.88%
GEMSEC	4.83%	32.30%	20.54%
GEMSEC With Regularization	**5.00%**	**32.76%**	**21.09%**

Table 5 – Classification task on Deezer Croatia

Country	Hungary		
F1-Score	**Macro**	**Micro**	**Weighted**
DeepWalk	6.19%	37.84%	24.97%
DeepWalk With Regularization	6.16%	37.88%	25.13%
GEMSEC	6.19%	37.31%	24.62%
GEMSEC With Regularization	**6.35%**	**38.61%**	**25.71%**

Table 6 – Classification task on Deezer Hungary

Country	Romania		
F1-Score	**Macro**	**Micro**	**Weighted**
DeepWalk	4.95%	32.33%	**20.68%**
DeepWalk With Regularization	4.79%	31.93%	20.19%
GEMSEC	**4.96%**	**32.49%**	20.59%
GEMSEC With Regularization	4.88%	32.01%	20.35%

Table 7 – Classification task on Deezer Romania

4.3. Going further

4.3.1 Recoding of GEMSEC

In order to fully grasp the mechanisms behind GEMSEC, the algorithm described in [1] was recoded using simple library such as numpy and igraph. While consistent and performant for small graphs (<1000 nodes), the modularity and classification results in this project were obtained using the original code[1] for performance, especially computation times reasons.

4.4. Performance vs. other algorithms

This part aims to compare the GEMSEC algorithm to top of the art (and often-used) community detection algorithms.

4.4.1 Performance on modularity

Similarly, to the part 4.2.1, we use the modularity score to compare all algorithms. Our results were computed on a single CPU (4G RAM, Intel core i5). In the three following parts, all results are plotted on a scatter plot:

- X axis is the number of clusters. We are here interested by all algorithms capable of creating less than 84 clusters (eg the number of musical taste in Deezer).
- Y axis is the modularity score
- The size of the bubble is the computational time (from 0 to 2h30 hours). The bigger the bubble, the higher the computational time.

Croatia dataset

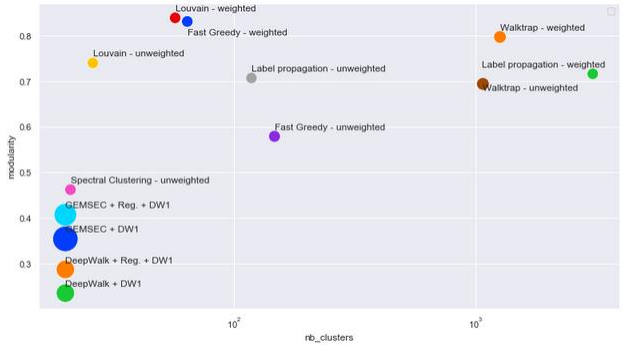

Figure 4 – Modularity analysis – Deezer Croatia

Algorithm	Time (secs)	# clusters	Modularity
Weighted			
Fast Greedy	45	64	0.831
Walktrap	728	1263	0.797
Spectral Clustering	9	1	0
Label propagation	6	3060	0.716
Louvain	6	57	0.839
Non weighted			
Fast Greedy	290	147	0.579
Walktrap	720	1073	0.694
Spectral Clustering	98	21	0.462
Label propagation	10	118	0.707
Louvain	4	26	0.740

Table 8 – Modularity analysis – Deezer Croatia

Hungary dataset

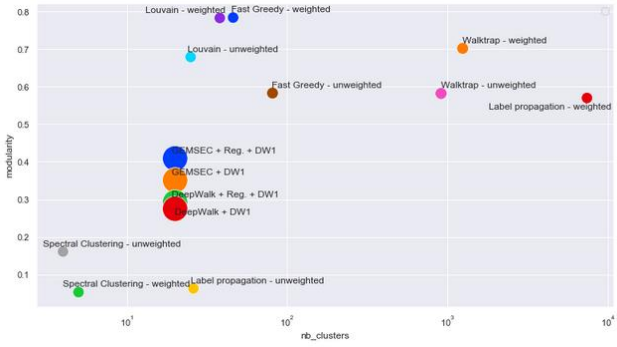

Figure 5 – Modularity analysis – Deezer Hungary

[1] https://github.com/benedekrozemberczki/GEMSEC/tree/master/src

Algorithm	Time (secs)	# clusters	Modularity
Weighted			
Fast Greedy	19	46	0.784
Walktrap	205	1241	0.702
Spectral Clustering	16	5	0.053
Label propagation	1	7394	0.570
Louvain	2	38	0.783
Non weighted			
Fast Greedy	157	81	0.583
Walktrap	224	910	0.582
Spectral Clustering	13	4	0.161
Label propagation	9	26	0.063
Louvain	4	25	0.679

Table 9 – Modularity analysis – Deezer Hungary

Romania dataset

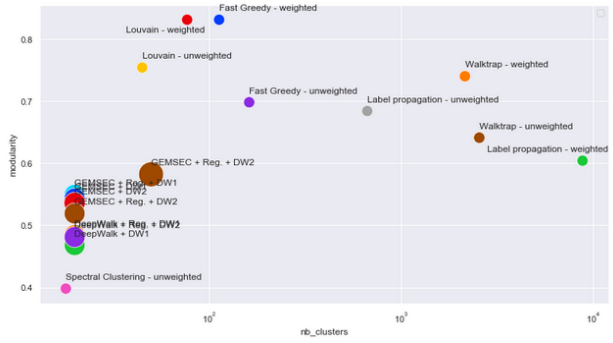

Figure 6 – Modularity analysis – Deezer Romania

Algorithm	Time (secs)	# clusters	Modularity
Weighted			
Fast Greedy	5	113	0.831
Walktrap	79	2150	0.740
Spectral Clustering	2	1	0
Label propagation	2	8803	0.604
Louvain	1	77	0.831
Non weighted			
Fast Greedy	40	162	0.698
Walktrap	80	2554	0.641
Spectral Clustering	34	18	0.398
Label propagation	4	667	0.684
Louvain	2	45	0.754

Table 10 – Modularity analysis – Deezer Romania

4.4.2 Performance on the classification task

Like in 4.2.2, we ran the same models to predict the genre liked by a user based on its embedding. This time, we choose to compare it with other state-of-the-art algorithms.

As expected, GEMSEC With Regularization offers good results, but does not clearly outperforms the other models as it was demonstrated in the paper.

Country	Croatia		
F1-Score	**Macro**	**Micro**	**Weighted**
Node2Vec	2.32%	24.03%	12.93%
Laplacian Eigenmaps	2.32%	23.80%	12.79%
MNMF	4.70%	31.72%	20.20%
DANMF	3.52%	27.20%	16.56%
GEMSEC With Regularization	**5.00%**	**32.76%**	**21.09%**

Table 11 – Classification task – Deezer Croatia

Country	Hungary		
F1-Score	**Macro**	**Micro**	**Weighted**
Node2Vec	3.92%	33.78%	18.34%
Laplacian Eigenmaps	3.97%	33.88%	18.46%
MNMF	5.40%	35.97%	22.66%
DANMF	4.74%	35.32%	20.78%
GEMSEC With Regularization	**6.35%**	**38.61%**	**25.71%**

Table 12 – Classification task – Deezer Hungary

Country	Romania		
F1-Score	**Macro**	**Micro**	**Weighted**
Node2Vec	3.90%	33.45%	18.11%
Laplacian Eigenmaps	3.92%	32.99%	17.74%
MNMF	**5.04%**	**33.74%**	**21.02%**
DANMF	4.11%	33.01%	18.37%
GEMSEC With Regularization	4.88%	32.01%	20.35%

Table 13 – Classification task – Deezer Romania

5. Reference

Some references were selected in order to support our approach. As our focus will mainly be on the comparison of algorithms on a specific dataset, we will mainly rely on [1] to implement the algorithms and compare our results.

[1] B. Rozemberczki, R. Davies, R. Sarkar and C. Sutton, "GEMSEC: Graph Embedding with Self Clustering," *ASONAM*, 2019.

[2] M. Aqib Javed , M. Shahzad Younis, L. Siddique , Q. Junaid and B. Adeel, "Community detection in networks: A multidisciplinary review," *Journal of Network and Computer Applications*, vol. 108, pp. 87-111, 2018.

[3] L. Ye, S. Chaofeng , H. Xin and Z. Yanchun, "Community Detection in Attributed Graphs: An Embedding Approach," in *AAAI-18*, 2018.

[4] C. Sandro , V. W. Zheng, H. Cai, K. Chen-Chuan Chang and E. Cambria, "Learning Community Embedding with Community Detection and Node Embedding on Graphs," in *CIKM*, Singapore, 2017.

[5] J. Chitra Devi and E. Poovammal, "An Analysis of Overlapping Community Detection Algorithms in Social Networks," in *IMCIP*, 2016.

[6] T. Cunchao, Z. Xiangkai and W. Hao , "A Unified Framework for Community Detection," *Journal Of Latex Class Files*, vol. 14, no. 8, 2015.

[7] N. Schlitter and T. Falkowski, "Mining the Dynamics of Music Preferences from a Social Networking Site," in *International Conference on Advances in Social Network Analysis and Mining*, Athens, 2009.