GEMSEC: Graph Embedding with Self Clustering

Citation for published version:

Digital Object Identifier (DOI):
10.1145/3341161.3342890

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
ASONAM '19: Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
GEMSEC: Graph Embedding with Self Clustering

Benedek Rozemberczki, Ryan Davies, Rik Sarkar and Charles Sutton
School of Informatics, The University of Edinburgh
{benedek.rozemberczki,ryan.davies}@ed.ac.uk, {rsarkar,csutton}@inf.ed.ac.uk

Abstract—Modern graph embedding procedures can efficiently process graphs with millions of nodes. In this paper, we propose GEMSEC – a graph embedding algorithm which learns a clustering of the nodes simultaneously with computing their embedding. GEMSEC is a general extension of earlier work in the domain of sequence-based graph embedding. GEMSEC places nodes in an abstract feature space where the vertex features minimize the negative log-likelihood of preserving sampled vertex neighborhoods, and it incorporates known social network properties through a machine learning regularisation. We present two new social network datasets and show that by simultaneously considering the embedding and clustering problems with respect to social properties, GEMSEC extracts high-quality clusters competitive with or superior to other community detection algorithms. In experiments, the method is found to be computationally efficient and robust to the choice of hyperparameters.

Index Terms—community detection, clustering, node embedding, network embedding, feature extraction.

I. INTRODUCTION

Community detection is one of the most important problems in network analysis due to its wide applications ranging from the analysis of collaboration networks to image segmentation, the study of protein-protein interaction networks in biology, and many others [1], [2], [3]. Communities are usually defined as groups of nodes that are connected to each other more densely than to the rest of the network. Classical approaches to community detection depend on properties such as graph metrics, spectral properties and density of shortest paths [4]. Random walks and randomized label propagation [5], [6] have also been investigated.

Embedding the nodes in a low dimensional Euclidean space enables us to apply standard machine learning techniques. This space is sometimes called the feature space – implying that it represents abstract structural features of the network. Embeddings have been used for machine learning tasks such as labeling nodes, regression, link prediction, and graph visualization, see [7] for a survey. Graph embedding processes usually aim to preserve certain predefined differences between nodes encoded in their embedding distances. For social network embedding, a natural priority is to preserve community membership and enable community detection.

Recently, sequence-based methods have been developed as a way to convert complex, non-linear network structures into formats more compatible with vector spaces. These methods sample sequences of nodes from the graph using a randomized mechanism (e.g. random walks), with the idea that nodes that are “close” in the graph connectivity will also frequently appear close in a sampling of random walks. The methods then proceed to use this random-walk-proximity information as a basis to embed nodes such that socially close nodes are placed nearby. In this category, Deepwalk [8] and Node2Vec [9] are two popular methods.

While these methods preserve the proximity of nodes in the graph sense, they do not have an explicit preference for preserving social communities. Thus, in this paper, we develop a machine learning approach that considers clustering when embedding the network and includes a parameter to control the closeness of nodes in the same community. Figure 1(a) shows the embedding obtained by the standard Deepwalk method, where communities are coherent, but not clearly separated in the embedding. The method described in this paper, called GEMSEC, is able to produce clusters that are tightly embedded and separated from each other (Fig. 1(b)).

A. Our Contributions

GEMSEC is an algorithm that considers the two problems of embedding and community detection simultaneously, and as a result, the two solutions of embedding and clustering can inform and improve each other. Through iterations, the embedding converges toward one where nodes are placed close

\[ \text{embedding converge} \]
to their neighbors in the network, while at the same time clusters in the embedding space are well separated.

The algorithm is based on the paradigm of sequence-based node embedding procedures that create \( d \) dimensional feature representations of nodes in an abstract feature space. Sequence-based node embeddings embed pairs of nodes close to each other if they occur frequently within a small window of each other in a random walk. This problem can be formulated as minimizing the negative log-likelihood of observed neighborhood samples (Sec. III) and is called the skip-gram optimization \([11]\). We extend this objective function to include a clustering cost. The formal description is presented in Subsection III-A. The resulting optimization problem is solved with a variant of mini-batch gradient descent \([12]\).

The detailed algorithm is presented in Subsection III-B. By enforcing clustering on the embedding, GEMSEC reveals the natural community structure (e.g. Figure 1). Our approach improves over existing methods of simultaneous embedding and clustering \([13]\), \([14]\), \([15]\) and shows that community sensitivity can be directly incorporated into the skip-gram style optimization to obtain greater accuracy and efficiency.

In social networks, nodes in the same community tend to have similar groups of friends, which is expressed as high neighborhood overlap. This fact can be leveraged to produce clusters that are better aligned with the underlying communities. We achieve this effect using a regularization procedure – a smoothness regularization added to the basic optimization achieves more coherent community detection. The effect can be seen in Figure 3, where a somewhat uncertain community affiliation suggested by the randomized sampling is sharpened by the smoothness regularization. This technique is described in Subsection III-C.

In experimental evaluation we demonstrate that GEMSEC outperforms – in clustering quality – the state of the art neighborhood based \([8]\), \([9]\), multi-scale \([16]\), \([17]\) and community aware embedding methods \([13]\), \([14]\), \([15]\). We present new social datasets from the streaming service Deezer and show that the clustering can improve music recommendations. The clustering performance of GEMSEC is found to be robust to hyperparameter changes, and the runtime complexity of our method is linear in the size of the graphs.

To summarize, the main contributions of our work are:

1) GEMSEC: a sequence sampling-based learning model which learns an embedding of the nodes at the same time as it learns a clustering of the nodes.
2) Clustering in GEMSEC can be aligned to network neighborhoods by a smoothness regularization added to the optimization. This enhances the algorithm’s sensitivity to natural communities.
3) Two new large social network datasets are introduced – from Facebook and Deezer data.
4) Experimental results show that the embedding process runs linearly in the input size. It generally performs well in quality of embedding and in particular outperforms existing methods on cluster quality measured by modularity and subsequent recommendation tasks.

We start with reviewing related work in the area and relation to our approach in the next section. A high-performance Tensorflow reference implementation of GEMSEC and the datasets that we collected can be accessed online\(^1\).

II. RELATED WORK

There is a long line of research in metric embedding – for example, embedding discrete metrics into trees \([18]\) and into vector spaces \([19]\). Optimization-based representation of networks has been used for routing and navigation in domains such as sensor networks and robotics \([20]\), \([21]\). Representations in hyperbolic spaces have emerged as a technique to preserve richer network structures \([22]\), \([23]\), \([24]\).

Recent advances in node embedding procedures have made it possible to learn vector features for large real-world graphs \([8]\), \([9]\), \([9]\). Features extracted with these sequence-based node embedding procedures can be used for predicting social network users’ missing age \([17]\), the category of scientific papers in citation networks \([17]\) and the function of proteins in protein-protein interaction networks \([9]\). Besides supervised learning tasks on nodes the extracted features can be used for graph visualization \([7]\), link prediction \([9]\) and community detection \([13]\).

Sequence based embedding commonly considers variations in the sampling strategy that is used to obtain vertex sequences – truncated random walks being the simplest strategy \([8]\). More involved methods include second-order random walks \([9]\), skips in random walks \([17]\) and diffusion graphs \([25]\). It is worth noting that these models implicitly approximate matrix factorizations for different matrices that are expensive to factorize explicitly \([26]\).

Our work extends the literature of node embedding algorithms which are community aware. Earlier works in this category did not directly extend the skip-gram embedding framework. M-NMF \([14]\) applies computationally expensive non-negative matrix factorization with a modularity constraint term. The procedure DANMF \([15]\) uses hierarchical non-negative matrix factorization to create community-aware node embeddings. ComE \([13]\) is a more scalable approach, but it assumes that in the embedding space the communities fit a gaussian structure, and aims to model them by a mixture of Gaussians. In comparison to these methods, GEMSEC provides greater control over community sensitivity of the embedding process, it is independent of the specific neighborhood sampling methods and is computationally efficient.

III. GRAPH EMBEDDING WITH SELF CLUSTERING

For a graph \( G = (V, E) \), a node embedding is a mapping \( f : V \to \mathbb{R}^d \) where \( d \) is the dimensionality of the embedding space. For each node \( v \in V \) we create a \( d \) dimensional representation. Alternatively, the embedding \( f \) is a \( |V| \times d \) real-valued matrix. In sequence-based embedding, sequences of neighboring nodes are sampled from the graph. Within a sequence, a node \( v \) occurs in the context of a window \( \omega \) within

\(^1\)https://github.com/benedekrozemberczki/GEMSEC
the sequence. Given a sample \( S \) of sequences, we refer to the collection of windows containing \( v \) as \( N_S(v) \). Earlier works have proposed random walks, second-order random walks or branching processes to obtain \( N_S(v) \). In our experiments, we used unweighted first and second-order random walks for node sampling \([3], [9]\).

Our goal is to minimize the negative log-likelihood of observing neighborhoods of source nodes conditional on feature vectors that describe the position of nodes in the embedding space. Formally, the optimization objective is:

\[
\min_f \sum_{v \in V} -\log P(N_S(v)|f(v))
\]

for a suitable probability function \( P(\cdot|\cdot) \). To define this \( P \), we consider two standard properties (see \([9]\)) expected of the embedding \( f \) in relation to \( N_S \). First, it should be possible to factorize \( P(N_S(v)|f(v)) \) in line with conditional independence with respect to \( f(v) \). Formally:

\[
P(N_S(v)|f(v)) = \prod_{n_i \in N_S(v)} P(n_i \in N_S(v) | f(v), f(n_i)).
\]

Second, it should satisfy symmetry in the feature space, meaning that source and neighboring nodes have a symmetric effect on each other in the embedding space. A softmax function on the pairwise dot products of node representations with \( f(v) \) to get \( P(n_i \in N_S(v) | f(v), f(n_i)) \) express such a property:

\[
P(n_i \in N_S(v) | f(v), f(n_i)) = \frac{\exp(f(n_i) \cdot f(v))}{\sum_{u \in V} \exp(f(u) \cdot f(v))}.
\]

Substituting (2) and (3) into the optimization function, we get:

\[
\min_f \sum_{v \in V} \left[ \ln \left( \sum_{u \in V} \exp(f(u) \cdot f(v)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right].
\]

The partition function in Equation (4) enforces nodes to be embedded in a low volume space around the origin, while the second term forces nodes with similar sampled neighborhoods to be embedded close to each other.

A. Learning to Cluster

Next, we extend the optimization to pay attention to the clusters it forms. We include a clustering cost similar to \( k \)-means, measuring the distance from nodes to their cluster centers. This augmented optimization problem is described by minimizing a loss function over the embedding \( f \) and position of cluster centers \( \mu \), that is, \( \min_{f, \mu} \mathcal{L} \), where:

\[
\mathcal{L} = \sum_{v \in V} \left[ \ln \left( \sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right] + \gamma \cdot \min_{v \in V, c \in C} \|f(v) - \mu_c\|_2.
\]

In Equation (5) we have \( C \) the set of cluster centers – the \( c \)-th cluster center is a \( d \)-dimensional vector in the embedding space. The idea is to minimize the distance from each node to its nearest cluster center. The weight coefficient of the clustering cost is given by the hyperparameter \( \gamma \). Evaluating the partition function in the proposed objective function for all of the source nodes has a \( \mathcal{O}(|V|^2) \) runtime complexity. Because of this, we approximate the partition function term with negative sampling which is a form of noise contrastive estimation \([11], [27]\).

\[
\frac{\partial \mathcal{L}}{\partial f(v^*)} = \frac{\sum_{u \in V} \exp(f(v^*) \cdot f(u)) \cdot f(u)}{\sum_{u \in V} \exp(f(v^*) \cdot f(u))} - \sum_{n_i \in N_S(v^*)} f(n_i)
\]

\[
\frac{\partial \mathcal{L}}{\partial \mu_c} = -\gamma \cdot \sum_{v \in V_c} \|f(v) - \mu_c\|_2.
\]
Data: $G = (V, E)$ – Graph to be embedded.
$N$ – Number of sequence samples per node.
$l$ – Length of sequences.
$\omega$ – Context size.
$d$ – Number of embedding dimensions.
$|C|$ – Number of clusters.
$k$ – Number of noise samples.
$\gamma_0$ – Initial clustering weight coefficient.
$\alpha_0, \alpha_F$ – Initial and final learning rate.

Result: $f(v)$, where $v \in V$
$\mu_c$, where $c \in C$

1. Model $\leftarrow$ Initialize Model($|V|, d, |C|$)
2. $t \leftarrow 0$
3. for $n$ in $1:N$ do
   4. $\hat{V} \leftarrow$ Shuffle($V$)
   5. for $v$ in $\hat{V}$ do
      6. $t \leftarrow t + 1$
      7. $\gamma \leftarrow$ Update $\gamma$ ($\gamma_0, t, w, l, N, |V|$)
      8. $\alpha \leftarrow$ Update $\alpha$ ($\alpha_0, \alpha_F, t, w, l, N, |V|$)
      9. Sequence $\leftarrow$ Sample Nodes ($G, v, l$)
     10. Features $\leftarrow$ Extract Features (Sequence, $\omega$)
     11. Update Weights (Model, Features, $\gamma, \alpha, k$)
   end
end

Algorithm 1: GEMSEC training procedure

In Equation 7 we see that the gradient moves the cluster center by the sum of coordinates of nodes in the embedding space that belong to cluster $c$. Second, if a cluster ends up empty it will not be updated as elements of the gradient would be zero. Because of this, cluster centers and embedding weights are initialized with the same uniform distribution. A wrong initialization just like the one with an empty cluster in Subfigure 2B can affect clustering performance considerably.

B. GEMSEC algorithm

We propose an efficient learning method to create GEMSEC embeddings which is described with pseudo-code by Algorithm 1. The main idea behind our procedure is the following. To avoid the clustering cost overpowering the graph information (as in Fig. 2a), we initialize the system with a low weight $\gamma_0 \in [0, 1]$ for clustering, and through iterations anneal it to 1.

The embedding computation proceeds as follows. The weights in the model are initialized based on the number of vertices, embedding dimensions and clusters. After this, the algorithm makes $N$ sampling repetitions in order to generate vertex sequences from every source node. Before starting a sampling epoch, it shuffles the set of vertices. We set the clustering cost coefficient $\gamma$ (line 7) according to an exponential annealing rule described by Equation (8). The learning rate is set to $\alpha$ (line 8) with a linear annealing rule (Equation (9)).

$$
\gamma = \gamma_0 \cdot \left(10^{-\frac{t \cdot \log_{10} \gamma_0}{\omega \cdot |V| \cdot N}}\right) \quad (8)
$$

$$
\alpha = \alpha_0 - (\alpha_0 - \alpha_F) \cdot \frac{t}{\omega \cdot l \cdot |V| \cdot N} \quad (9)
$$

The sampling process reads sequences of length $l$ (line 9) and extracts features using the context window size $\omega$ (line 10). The extracted features, gradient, current learning rate and clustering cost coefficient determine the update to model weights by the optimizer (line 11). In the implementation we utilized a variant of stochastic gradient descent – the Adam optimizer [12]. We approximate the first cost term with noise contrastive estimation to make the gradient descent tractable, drawing $k$ noise samples for each positive sample. If the node sampling is done by first-order random walks the runtime complexity of this procedure will be $O((\omega \cdot k + |C|) \cdot l \cdot d \cdot |V| \cdot N)$ while DeepWalk with noise contrastive estimation has a $O(\omega \cdot k \cdot l \cdot d \cdot |V| \cdot N)$ runtime complexity.

C. Smoothness Regularization for coherent community detection

We have seen in Subsection III-A that there is a tension between what the clustering objective considers to be clusters and what the real communities are in the underlying social network. We can incorporate additional knowledge of social network communities using a machine learning technique called regularization.

We observe that social networks have natural local properties such as homophily, strong ties between members of a community, etc. Thus, we can incorporate such social network-specific properties in the form of regularization to find more natural embeddings and clusters.

This regularization effect can be achieved by adding a term $\Lambda$ to the loss function:

$$
\Lambda = \lambda \cdot \sum_{(v,u) \in E_S} w_{v,u} \cdot \|f(v) - f(u)\|_2, \quad (10)
$$

where the weight function $w$ determines the social network cost of the embedding with respect to properties of the edges traversed in the sampling. We use the neighborhood overlap of an edge – defined as the fraction of neighbors common to two nodes of the edge relative to the union of the two neighbor sets. In experiments on real data, neighborhood overlap is known to be a strong indicator of the strength of relation between members of a social network [28]. Thus, by treating neighborhood overlap as the weight $w_{v,u}$ of edge $(v, u)$, we can get effective social network clustering, which is confirmed by experiments in the next section. The coefficient $\lambda$ lets us tune the contribution of the social network cost in the embedding process. In experiments, the regularized version of the algorithms is found to be more robust to changes in hyperparameters.

2Neighbor sets $N(a)$ and $N(b)$ of nodes $a$ and $b$, the neighborhood overlap of $(a, b)$ is defined as the Jaccard similarity $\frac{N(a) \cap N(b)}{N(a) \cup N(b)}$. 
The effect of the regularization can be understood intuitively through an example. For this exposition, let us consider matrix representations of the social network describing closeness of nodes. In fact, other skip-gram style learning processes like [8], [9] are known to approximate the factorization of a similarity matrix $M$ such as [26]:

$$M_{u,v} = \log \left( \frac{\text{vol}(G)}{\omega} \sum_{r=1}^{\omega} \frac{1}{\deg(v)} \prod_{a \in P_{v,u}^r} \frac{1}{\deg(a)} \right) - \log(k)$$

where $P_{v,u}^r$ is the set of paths going from $v$ to $u$ with length $r$. Elements of the target matrix $M$ grow with number of paths of length at most $\omega$ between the corresponding nodes. Thus $M$ is intended to represent level of connectivity between nodes in terms of a raw graph feature like number of paths.

The barbell graph in Figure 3a is a typical example with an obvious community structure we can use to analyze the matter. The optimization procedure used by Deepwalk [8] aims to converge to a target matrix $M_{u,v}$ shown in Figure 3b. Observe that this matrix has fuzzy edges around the communities of the graph, showing a degree of uncertainty. An actual approximation by running the Deepwalk is shown in Figure 3c which naturally incorporates further uncertainty due to sampling. A much more clear output with sharp communities can be obtained by applying a regularized optimization. This can be seen in Figure 3d.

### Table I

| Source     | Dataset     | |V| | Density | Transitivity |
|------------|-------------|--------|--------|---------|-------------|
| Facebook   | Politicians | 5,908  | 0.0024 | 0.3011 |
|            | Companies   | 14,113 | 0.0005 | 0.1532 |
|            | Athletes    | 13,866 | 0.0009 | 0.1292 |
|            | Media       | 27,917 | 0.0005 | 0.1140 |
|            | Celebrities | 11,565 | 0.0010 | 0.1666 |
|            | Artists     | 50,515 | 0.0006 | 0.1140 |
|            | Government  | 7,057  | 0.0036 | 0.2238 |
|            | TV Shows    | 3,892  | 0.0023 | 0.5966 |
| Deemer     | Croatia     | 54,573 | 0.0004 | 0.1146 |
|            | Hungary     | 47,538 | 0.0002 | 0.0929 |
|            | Romania     | 41,773 | 0.0001 | 0.0752 |

### IV. Experimental Evaluation

In this section we evaluate the cluster quality obtained by the GEMSEC variants, their scalability, robustness and predictive performance on a downstream supervised task. Results show that GEMSEC outperforms or is at par with existing methods in all measures.

#### A. Datasets

For the evaluation of GEMSEC real-world social network datasets are used which we collected from public APIs specifically for this work. Table I shows these social networks have a variety of size, density, and level of clustering. We used graphs from two sources:

- Facebook page networks: These graphs represent mutual like networks among verified Facebook pages – the types of sites included TV shows, politicians, athletes, and artists among others.
- Deezer user-user friendship networks: We collected friendship networks from the music streaming site Deezer and included 3 European countries (Croatia, Hungary, and Romania). For each user, we curated the list of genres loved based on the songs liked by the user.

#### B. Standard parameter settings

A fixed standard parameter setting is used our experiments, and we indicate any deviations. Models using first order random walk sampling strategy are referenced as GEMSEC and Smooth GEMSEC, second order random walk variants are named as GEMSEC$_2$ and Smooth GEMSEC$_2$. Random walks with length 80 are used and 5 truncated random walks per source node were used. Second-order random walk control hyperparameters [9] _return_ and _in-out_ were chosen from $\{2^{-2}, 2^{-1}, 1, 2, 4\}$. A window size of 5 is used for features. Each embedding has 16 dimensions and we extract 20 cluster centers. A parameter sweep over hyperparameters was used to obtain the highest average modularity. Initial learning rate values are chosen from $\{10^{-2}, 5 \cdot 10^{-3}, 10^{-3}\}$ and the final learning rate is chosen from $\{10^{-3}, 5 \cdot 10^{-4}, 10^{-4}\}$. Noise contrastive estimation uses 10 negative examples. The initial clustering cost coefficient is chosen from $\{10^{-1}, 10^{-2}, 10^{-3}\}$. The smoothness regularization term’s hyperparameter is 0.0625 and Jaccard’s coefficient is the penalty weight.

#### C. Cluster Quality

Using Facebook page networks we evaluate the clustering performance. Cluster quality is evaluated by modularity – we assume that a node belongs to a single community. Our results are summarized in Table II based on 10 experimental repetitions and errors in parentheses correspond to two standard deviations. The baselines use the hyperparameters from the respective papers. We used 16-dimensional embeddings throughout. The embeddings obtained with non-community-aware methods were clustered after the embedding by $k$-means clustering to extract 20 cluster centers. Specifically, comparisons are made with:

1) **Overlap Factorization** [29]: Factorizes the neighborhood overlap matrix to create features.
2) **DeepWalk** [8]: Approximates the sum of the adjacency matrix powers with first order random walks and implicitly factorizes it.
3) **LINE** [16]: Implicitly factorizes the sum of the first two matrix powers with first order random walks and implicitly factorizes it.
4) **Node2vec** [9]: Factorizes a neighbourhood matrix obtained with second order random walks. The _in-out_ and _return_ parameters of the second-order random walks
were chosen from the \( \{2^{-2}, 2^{-1}, 1, 2, 4\} \) set to maximize modularity.

5) **Walklets [17]**: Approximates with first order random walks each adjacency matrix power individually and implicitly factorizes the target matrix. These embeddings are concatenated to form a multi-scale representation of nodes.

6) **ComE [13]**: Uses a Gaussian mixture model to learn an embedding and clustering jointly using random walk features.

7) **M-NMF [14]**: Factorizes a matrix which is a weighted sum of the first two proximity matrices with a modularity based regularization constraint.

8) **DANMF [15]**: Decomposes a weighted sum of the first two proximity matrices hierarchically to obtain cluster memberships with an autoencoder-like non-negative matrix factorization model.

Smooth GEMSEC, GEMSEC\(_2\) and Smooth GEMSEC\(_2\) consistently outperform the neighborhood conserving node embedding methods and the competing community aware methods. The relative advantage of Smooth GEMSEC\(_2\) over the benchmarks is highest on the Athletes dataset as the clustering’s modularity is 3.44\% higher than the best performing baseline. It is the worst on the Media dataset with a disadvantage of 0.35\% compared to the strongest baseline. Use of smoothness regularization has sometimes non-significant, but definitely positive effect on the clustering performance of Deepwalk, GEMSEC and GEMSEC\(_2\).

### D. Sensitivity Analysis for hyperparameters

We tested the effect of hyperparameter changes to clustering performance. The Politicians Facebook graph is embedded with the standard parameter settings while the initial and final learning rates are set to be \( 10^{-2} \) and \( 5 \cdot 10^{-3} \) respectively, the clustering cost coefficient is 0.1 and we perturb certain hyperparameters. The second-order random walks used in-out and return parameters of 2. In Figure 4 each data point represents the mean modularity calculated from 10 experiments. Based on the experimental results we make two observations. First, GEMSEC model variants give high-quality clusters for a wide range of parameter settings. Second, introducing smoothness

---

**TABLE II**

<table>
<thead>
<tr>
<th>Politicians</th>
<th>Companies</th>
<th>Athletes</th>
<th>Media</th>
<th>Celebrities</th>
<th>Artists</th>
<th>Government</th>
<th>TV Shows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overlap Factorization</strong></td>
<td>0.810 (±0.008)</td>
<td>0.553 (±0.010)</td>
<td>0.601 (±0.020)</td>
<td>0.471 (±0.016)</td>
<td>0.551 (±0.01)</td>
<td>0.474 (±0.018)</td>
<td>0.008 (±0.024)</td>
</tr>
<tr>
<td><strong>DeepWalk</strong></td>
<td>0.840 (±0.015)</td>
<td>0.637 (±0.012)</td>
<td>0.649 (±0.012)</td>
<td>0.481 (±0.022)</td>
<td>0.631 (±0.011)</td>
<td>0.508 (±0.029)</td>
<td>0.680 (±0.024)</td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>0.841 (±0.014)</td>
<td>0.651 (±0.009)</td>
<td>0.665 (±0.007)</td>
<td>0.558 (±0.012)</td>
<td>0.642 (±0.010)</td>
<td>0.557 (±0.014)</td>
<td>0.690 (±0.017)</td>
</tr>
<tr>
<td><strong>Node2Vec</strong></td>
<td>0.846 (±0.012)</td>
<td>0.664 (±0.008)</td>
<td>0.690 (±0.007)</td>
<td>0.565 (±0.011)</td>
<td>0.643 (±0.013)</td>
<td>0.566 (±0.010)</td>
<td>0.692 (±0.017)</td>
</tr>
<tr>
<td><strong>Walklets</strong></td>
<td>0.843 (±0.014)</td>
<td>0.655 (±0.012)</td>
<td>0.664 (±0.007)</td>
<td>0.562 (±0.009)</td>
<td>0.621 (±0.043)</td>
<td>0.548 (±0.016)</td>
<td>0.689 (±0.019)</td>
</tr>
<tr>
<td><strong>ComE</strong></td>
<td>0.830 (±0.008)</td>
<td>0.654 (±0.005)</td>
<td>0.665 (±0.007)</td>
<td>0.573 (±0.005)</td>
<td>0.635 (±0.010)</td>
<td>0.560 (±0.011)</td>
<td>0.696 (±0.011)</td>
</tr>
<tr>
<td><strong>M-NMF</strong></td>
<td>0.816 (±0.014)</td>
<td>0.646 (±0.007)</td>
<td>0.655 (±0.008)</td>
<td>0.561 (±0.004)</td>
<td>0.628 (±0.006)</td>
<td>0.535 (±0.021)</td>
<td>0.668 (±0.011)</td>
</tr>
<tr>
<td><strong>DANMF</strong></td>
<td>0.810 (±0.020)</td>
<td>0.648 (±0.009)</td>
<td>0.650 (±0.009)</td>
<td>0.560 (±0.006)</td>
<td>0.628 (±0.011)</td>
<td>0.532 (±0.019)</td>
<td>0.673 (±0.015)</td>
</tr>
<tr>
<td><strong>Smooth DeepWalk</strong></td>
<td>0.859 (±0.017)</td>
<td>0.667 (±0.007)</td>
<td>0.669 (±0.007)</td>
<td>0.541 (±0.006)</td>
<td>0.643 (±0.008)</td>
<td>0.523 (±0.020)</td>
<td>0.707 (±0.018)</td>
</tr>
<tr>
<td><strong>GEMSEC</strong></td>
<td>0.851 (±0.009)</td>
<td>0.662 (±0.013)</td>
<td>0.674 (±0.009)</td>
<td>0.536 (±0.011)</td>
<td>0.636 (±0.014)</td>
<td>0.528 (±0.020)</td>
<td>0.705 (±0.019)</td>
</tr>
<tr>
<td><strong>Smooth GEMSEC</strong></td>
<td>0.855 (±0.006)</td>
<td>0.683 (±0.009)</td>
<td>0.692 (±0.009)</td>
<td>0.567 (±0.009)</td>
<td>0.649 (±0.008)</td>
<td>0.559 (±0.011)</td>
<td>0.710 (±0.008)</td>
</tr>
<tr>
<td><strong>GEMSEC(_2)</strong></td>
<td>0.852 (±0.010)</td>
<td>0.667 (±0.008)</td>
<td>0.683 (±0.008)</td>
<td>0.551 (±0.008)</td>
<td>0.638 (±0.009)</td>
<td>0.562 (±0.020)</td>
<td>0.712 (±0.010)</td>
</tr>
<tr>
<td><strong>Smooth GEMSEC(_2)</strong></td>
<td>0.859 (±0.006)</td>
<td>0.684 (±0.008)</td>
<td>0.692 (±0.007)</td>
<td>0.571 (±0.010)</td>
<td>0.649 (±0.011)</td>
<td>0.562 (±0.017)</td>
<td>0.712 (±0.010)</td>
</tr>
</tbody>
</table>
regularization makes GEMSEC models more robust to hyper-parameter changes. This is particularly apparent across varying the number of clusters. The length of truncated random walks and the number of random walks per source node above a certain threshold has only a marginal effect on the community detection performance.

E. Music Genre Recommendation

Node embeddings are often used for extracting features of nodes for downstream predictive tasks. In order to investigate this, we use social networks of Deezer users collected from European countries. We predict the genres (out of 84) of music liked by people. Following the embedding, we used logistic regression with $\ell_2$ regularization to predict each of the labels and 90% of the nodes were randomly selected for training. We evaluated the performance of the remaining users. Numbers reported in Table III are $F_1$ scores calculated from 10 experimental repetitions. GEMSEC$_2$ significantly outperforms the other methods on all three countries’ datasets. The performance advantage varies between 3.03% and 4.95%. We also see that Smooth GEMSEC$^2$ has lower accuracy, but it is able to outperform DeepWalk, LINE, Node2Vec, Walklets, ComE, M-NMF and DANMF on all datasets.

F. Scalability and computational efficiency

To create graphs of various sizes, we used the Erdos-Renyi model and with an average degree of 20. Figure 5 shows the log of mean runtime against the log of the number of nodes. Most importantly, we can conclude that doubling the size of the graph doubles the time needed for optimizing GEMSEC, thus the growth is linear. We also observe that embedding algorithms that incorporate clustering have a higher cost, and regularization also produces a higher cost, but similar growth.

V. CONCLUSIONS

We described GEMSEC – a novel algorithm that learns a node embedding and a clustering of nodes jointly. It extends existing embedding modes. We showed that smoothness regularization is used to incorporate social network properties and produce natural embedding and clustering. We presented new social datasets, and experimentally, our methods outperform a number of strong community aware node embedding baselines.

VI. ACKNOWLEDGEMENTS

Benedek Rozemberczki and Ryan Davies were supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1).

REFERENCES


TABLE III
MULTI-LABEL NODE CLASSIFICATION PERFORMANCE OF THE EMBEDDING EXTRACTED FEATURES ON THE DEEZER GENRE LIKES DATASETS. PERFORMANCE IS MEASURED BY AVERAGE F1 SCORE VALUES. MODELS WERE TRAINED ON 90% OF THE DATA AND EVALUATED ON THE REMAINING 10%. ERRORS IN THE PARENTHESSES CORRESPOND TO TWO STANDARD DEVIATIONS. GEMSEC MODELS CONSISTENTLY HAVE GOOD PERFORMANCE.

<table>
<thead>
<tr>
<th></th>
<th>CROATIA</th>
<th>HUNGARY</th>
<th>ROMANIA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Micro</td>
<td>Macro</td>
<td>Weighted</td>
</tr>
<tr>
<td>Overlap Factorization</td>
<td>0.319±0.017</td>
<td>0.026±0.007</td>
<td>0.208±0.001</td>
</tr>
<tr>
<td>DeepWalk</td>
<td>0.321±0.006</td>
<td>0.026±0.001</td>
<td>0.207±0.002</td>
</tr>
<tr>
<td>LINE</td>
<td>0.331±0.013</td>
<td>0.028±0.002</td>
<td>0.212±0.004</td>
</tr>
<tr>
<td>Node2Vec</td>
<td>0.348±0.012</td>
<td>0.032±0.001</td>
<td>0.235±0.003</td>
</tr>
<tr>
<td>Walklets</td>
<td>0.363±0.013</td>
<td>0.043±0.003</td>
<td>0.270±0.002</td>
</tr>
<tr>
<td>ComE</td>
<td>0.326±0.012</td>
<td>0.028±0.002</td>
<td>0.217±0.001</td>
</tr>
<tr>
<td>M-NMF</td>
<td>0.336±0.005</td>
<td>0.028±0.001</td>
<td>0.217±0.002</td>
</tr>
<tr>
<td>DANMF</td>
<td>0.340±0.007</td>
<td>0.027±0.002</td>
<td>0.210±0.002</td>
</tr>
<tr>
<td>Smooth DeepWalk</td>
<td>0.329±0.006</td>
<td>0.028±0.002</td>
<td>0.215±0.002</td>
</tr>
<tr>
<td>GEMSEC</td>
<td>0.328±0.006</td>
<td>0.027±0.002</td>
<td>0.212±0.004</td>
</tr>
<tr>
<td>Smooth GEMSEC</td>
<td>0.333±0.006</td>
<td>0.028±0.002</td>
<td>0.215±0.004</td>
</tr>
<tr>
<td>GEMSEC2</td>
<td>0.381±0.007</td>
<td>0.046±0.003</td>
<td>0.287±0.005</td>
</tr>
<tr>
<td>Smooth GEMSEC2</td>
<td>0.373±0.005</td>
<td>0.041±0.002</td>
<td>0.276±0.006</td>
</tr>
</tbody>
</table>