A Unified Network Embedding Algorithm for Multi-type Similarity Measures

Rui Feng[1], Yang Yang[1], Yizhou Sun[2], Chunping Wang[3]
[1]Zhejiang University
[2]University of California, Los Angeles, [3]PPDAI Group Inc.

ABSTRACT

Traditional network embedding aims to learn *representations* by capturing a predefined *vertex-to-vertex similarity measure*. However, in practice, there are different types of similarity measures (e.g., *connectivity* and *structural similarity*), which are appropriate for different downstream applications. Meanwhile, it is hard to select the "best" similarity measure that can mostly benefit the application, considering the required domain knowledge of both application scenario and network science. It sometimes requires to cooperate these similarity measures with each other for achieving better performance. Therefore, automatically integrate multiple types of similarity measures into a uniform network embedding framework is critical to obtain effective vertex representations for a downstream application.

In this paper, we address the above problem in social networks, and propose a *semi-supervised* representation learning algorithm. The general idea of our approach is to impose *social influence*, which occurs when one's opinions, emotions, or behaviors are affected by others in a social network. Particularly, we build the connection between a user's representation vector and the probability of her being influenced by another user to have a particular label (e.g., fraud, personal interest, etc.). We conduct efficient experiments based on six real-world datasets and find a clear improvement of our approach comparing with several state-of-the-art baselines.

KEYWORDS

Network embedding, Social network, Probabilistic influence model

1 INTRODUCTION

Network embedding has turned to be an effective methodology to represent graph structure data. Its basic idea is to project vertices into a low-dimensional latent space and keep similar vertices close to each other. The learned vector representations can be incorporated into feature-based machine learning algorithms, and then easily applied for downstream applications like vertex classification, link prediction, and many other network analysis tasks.

How to model multiple types of similarity measures in a uniform framework and learn effective node representations for a given application task? This is the major question we aim to answer.

In this paper, we consider the above question on networks where only a small set of vertices are labeled, and the learned representations are expected to be applied on vertex classification tasks. The application task is thus a typical problem of semi-supervised learning on graph-structured data.

In order to mix different similarity measures in a principled approach, we propose a probabilistic graph embedding framework based on *social influence* between vertices. Social influence occurs when one's opinion, emotion, or behaviors are effected by others in a social network. It provides a probabilistic interpretation of the embedding, based on which we may leverage the Expectation-Maximization approach to infer vertex labels and incorporate multi-type similarity measures.

Essentially, our embedding algorithm is based on the assumption that a vertex's label is influenced by its neighbors. Since we explicitly modeled the data generation mechanism, our approach works well for any graph, such as citation networks, that satisfies the assumption in addition to social networks.

There are some significant advantages of using our algorithm. First, the model is derived based on some assumptions of the data generation mechanism that are robust with the verification by by many researchers on social networks. It is natural that on social networks and other graphs that satisfy the assumption, our approach performs better than traditional algorithms who based their model on loose assumptions about the smoothness of the latent vector space. Second, leveraging different similarity measures together makes our algorithm is able to learn from limited labels more effectively, which leads a clear improvement comparing with several state-of-the-art semi-supervised classification methods on network. Third, our method is interpretable for the classification results as the social influence module provides a detailed who-influence-who trace when a label is inferred.

It is worthwhile to summarize our contributions as follows:

- We propose a novel network representation learning algorithm that is able to leverage multiple types of similarity measures in a uniform framework.
- We explicitly models the data generation mechanism with a probabilistic influence model.
- We conduct extensive experiments and find that our method outperforms state-of-the-art baselines.

2 PRELIMINARIES

2.1 Problem Definition

We use a *multi-view graph* to represent different similarity measures on graphs. Specifically, we denote a multi-view graph as $G = (V, E_1, \ldots, E_K)$, where $V = (v_1, \ldots, v_N)$ is the set of vertices (e.g., users in social networks) and each E_i is an edge set that represents a similarity measure on V. For each vertex i and a similarity j, we denote the τ-order neighborhood of i in E_j as $N_j^\tau(i)$, which is the set of τ-order neighbors of i when only edges in E_j is considered. Thereby, each similarity induces a graph $G = (V, E_j)$. In this paper, we call it the *similarity graph* w.r.t. similarity j.

Each vertex v_i corresponds to a discrete label $y_i \in \{0, 1, \ldots, n_k\}$ provided by the downstream application. y_i is known for only

$1 \leq i \leq M$ and $M \ll N$. Our goal is to infer the probability distribution of unknown labels y_j for $M+1 \leq j \leq N$.

Network embedding seeks to learn low-dimensional and distributed representation of vertices in the graph. Since our graph contains multiple edge sets and thus represent different similarities to be embedded into the low dimensional space, for each similarity measure k corresponding to E_k, we are to find a mapping $\Phi_k : V \rightarrow \mathbb{R}^m, v_i \mapsto u_i^k$, where $v_i \in V$ and $u_i^k \in \mathbb{R}^m$. Φ_k is parameterized by Θ_k, and we denote Θ as the union of $\{\Theta_k\}_{k=1}^K$.

In the following of this paper, for simplicity of notations, we may ignore indexes and use the notation Θ and N to represent embedding parameters and neighborhood without causing any confusion.

3 OUR APPROACH

In this section, first, we consider the estimation of y_i using the influence of the τ-order neighborhood; then, we integrate different similarities into one estimator.

3.1 Social-influence-based Probabilistic Network Embedding Framework

Modeling social influence. In this paper, we denote $v_j \rightsquigarrow v_i$ as "v_j influences v_i" and formulate it in the probabilistic language:

$$\mathcal{P}\left(y_i = 1 | v_j \rightsquigarrow v_i\right) = \mathcal{P}\left(y_j = 1\right) \tag{1}$$

where we may use embeddings to estimate the influence and assume that

$$\mathcal{P}\left(v_j \rightsquigarrow v_i | N^\tau(i), \Theta\right) = \frac{S(v_i, v_j|\Theta)}{\sum_{k \in N^\tau(i)} S(v_i, v_k|\Theta)} \tag{2}$$

where $S(v_i, v_j|\Theta)$ is an embedding-dependent term.

To estimate y_i, we need to rely on v_i's neighborhood, $N^\tau(i)$. Under mild assumptions we may derive that

$$\begin{aligned} & \mathcal{P}(y_i = 1 | N^\tau(i), \Theta) \\ = & \sum_{j \in N^\tau(i)} \mathcal{P}(v_j \rightsquigarrow v_i | N^\tau(i)) \mathcal{P}(y_i = 1 | v_j \rightsquigarrow v_i, N^\tau(i)) \\ = & \frac{\sum_{j \in N^\tau(i)} S(v_i, v_j|\Theta) \mathcal{P}(y_j = 1)}{\sum_{k \in N^\tau(i)} S(v_i, v_k|\Theta)} \end{aligned} \tag{3}$$

The embedding is used to approximate the influence. The embedding is generated by a two-layer neural network, where the first layer, called the embedding layer, is a linear layer whose weight vectors u_i are the embedding of v_i. The second hidden layer is composed of a distinctive set of vectors u_i', the weight corresponding to v_i. The influence is then computed as

$$S(v_i, v_j|\Theta) = \sigma(u_i \cdot u_j') = \frac{1}{1 + \exp\left(-u_i \cdot u_j'\right)} \tag{4}$$

where u_i is the embedding vector for v_i and u_j' is the weight corresponding to v_j.

Training of the embedding. The local loss function for each vertex pair v_i, v_j is then:

$$\begin{aligned} & - \quad \mathcal{P}(v_j \rightsquigarrow v_i | N^\tau(i), \Theta) \delta(i, j) \\ & - \quad \left(1 - \mathcal{P}(v_j \rightsquigarrow v_i | N^\tau(i), \Theta)\right) (1 - \delta(i, j)) \end{aligned} \tag{5}$$

which is the commonly used cross-entropy loss. Here, $\delta(i, j) = \sum_k y_{ik} y_{jk}$, where k is taken across all categories, which yields the probability that $y_i = y_j$, and we use it as a surrogate of the ground-truth probability that j influences i, which is hard to infer from a static graph.

The objective is trained by stochastic gradient descent. Following the procedure adopted by Perozzi et al. [5], we use random walks to generate sequences of vertices called the "corpus", and consequently draw vertex pairs from the corpus to feed into the model. For each vertex pair, a gradient step can be taken for the objective (5).

Local negative sampling. Inspired by the technique proposed by Mikolov et al. [4] for efficient training of the language model, we use *local negative sampling* to approximate the computation of (2).

In practice, we draw N_k negative vertices, $\{v_{s_{i,r}}\}_{r=1}^{N_k}$ from the neighborhood of v_i's. Then, (2) is approximated by

$$\log(\sigma(u_i \cdot u_j')) - \sum_{r=1}^{N_k} \log(\sigma(-u_i \cdot u_{s_{i,r}}'))$$

Sampling negative vertices only from a vertex's neighborhood follows from the formulation of (2).

3.2 Integrating Different Similarity Measures

Now suppose that we have multiple similarity measures, $E_1, \ldots, E_K$, and for each E_k the embedding parameters are Θ_k. In this circumstance, we assume that

$$\mathcal{P}(y_i = 1|\Theta) = \sum_{k=1}^{K} w_k \mathcal{P}(y_i = 1|\Theta_k, N_k^\tau(i)) \tag{6}$$

With the constraint: $\sum_k w_k = 1$. w_k has a probabilistic interpretation: it is the probability that y_i is determined by the kth similarity measure.

The estimation of w_k, which determines the influence of each similarity, can be derived by taking the expectation:

$$\mathbb{E}(w_k) = \frac{\sum_{j \in N_i^\tau(i)} S\left(i, j|\Theta_k\right) \delta(i, j)}{\sum_q \sum_{j \in N_q^\tau(i)} S\left(i, j|\Theta_q\right) \delta(i, j)} \tag{7}$$

Though (7) gives an unbiased estimation, in practice, we only estimate the weights for vertices with known labels, and use the average of the estimated values as the weights for vertices with unknown labels.

The training process is summarized in Algorithm 1. Specifically, in each iteration, we first optimize embeddings for each similarity sufficiently and then we incorporate the prediction based on different similarities by taking the expected value of w_k.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

Datasets. We conduct experiments on the following six datasets:

- Agent, Debt. These data are provided by PPDai, the largest unsecured micro-credit loan platform in China. The vertices are the users, and the edges are the call logs between them, weighted by the number of calls. For Agent, each user is classified as having

Algorithm 1 The training process.

N_t is the number of iterations, N_b is the batch size for sampling, N_k is the number of negative vertices in negative sampling.
for p in $1 \ldots N_t$ **do**
  Optimize embeddings.
  for each similarity measure t **do**
    repeat
      Sample a batch of center-context pairs $\{v_{n_i}, v_{c_i}\}_{i=1}^{N_b}$.
      For v_{n_i}, sample N_k negative vertices from $N_t^{\tau}(i)$.
      Compute the loss by $\sum_{i=1}^{N_b} \ell_t(v_i, v_j)$ and its gradient, and perform a step with Stochastic Gradient Descent.
    until the convergence of Θ_t
  end for
  Estimate the similarity-specific values of y for unlabeled vertices by (3).
  Estimate similarity weight by (7) for labeled vertices.
  Estimate the final value of y for unlabeled vertices by (6).
end for

Dataset	Agent	Debt	Investor	Cora	Citeseer	Wiki
Vertices	10678	25537	9953	2708	3312	2405
Edges	25944	26997	373923	5429	4732	17981
Categories	2	2	2	7	6	19

Table 1: Statistics of datasets.

defaulted for more than 90 days, while for Debt, each vertex is classified as being a fraudulent user or not.

- Investor. This dataset is provided by PPDai. Each vertex represent a registered investor in Shanghai, China, and edges represent the geographical distances between them.
- Cora[1] and Citeseer[2]. These are academic citation networks where each vertex represent a paper and edges represent the citations between them. Each paper is categorized into several classes indicating their topic.
- Wikipedia[3]. This is extracted from Wikipedia containing webpages and edges represent references. This is a much denser network than Cora and Citeseer.

Data statistics, including the number of vertices, edges, and categories, are listed in Table 1.

Similarity graphs. In application, we are in need of similarity measures other than the one provided by the original graphs. For example, vertices with structurally similar neighborhoods could share similar virtues. For each graph in our experiment, we use the method proposed by Ribeiro et al. [6] to generate a "structural graph", with the same set of vertices and edges representing the structural similarity between them. The original graph, representing the connectivity between vertices, is called the "connectivity graph". Later on, we will use the integrate model mentioned before to combine the prediction based on these two similarity graphs.

[1]We used the same dataset as preprocessed by Kipf and Welling [3], provided in https://github.com/tkipf/pygcn/tree/master/data/cora.
[2]Aquired from http://csxstatic.ist.psu.edu/downloads/data.
[3]We used the same dataset used by Tu et al. [8], provided in https://github.com/thunlp/MMDW.

	Agent	Debt	Investor	Cora	Citeseer	Wiki
Deepwalk	8.77	29.51	17.02	43.2	67.2	52.03
GreRep	10.28	22.69	23.68	74.79	50.97	55.16
GCN	**12.21**	33.20	25.01	70.65	53.02	50.72
Ours	10.01	**35.40**	**23.57**	**75.32**	**57.39**	**57.49**

Table 2: Performance of the influence model in terms of Micro-F1.

	Agent	Debt	Investor	Cora	Citeseer	Wiki
N-GCN	22.35	33.17	23.90	72.54	55.11	52.70
Ours	**30.19**	**35.81**	**25.12**	**76.28**	**57.56**	**55.23**

Table 3: Performance of the intergrate model in terms of Micro-F1.

Baseline methods. The following graph embedding algorithms are compared to ours:

- Deepwalk [5], LINE [7], SDNE [9], GraRep [2]. These are unsupervised graph embedding algorithms for social networks utilizing different local and global information.
- GraRep [2]. This is an unsupervised graph embedding algorithms for social networks utilizing different local and global information.
- GCN[4]. It stands for the Graph Convolutional Network by [3]. It is a semi-supervised learning algorithm which provides both vector representations and end-to-end classification results.
- N-GCN. It is an extension of GCN by Abu-El-Haija et al. [1] to support multiview graphs. Specifically, they concatenated the vector representations of each vertex learned by multiple GCNs on different relational graphs, which is, in our language, graphs induced by similarity measures.

4.2 Model Parameters

For all our experiments, the embedding dimension is 32. For Cora, Citeseer and Wikipedia, we set τ, the order of neighborhood, as 2, and for Agent, Debt, and Investor it is 4. For GCN and Deepwalk, we used the model parameters recommended by the authors.

4.3 Comparison Results

Table 2 shows the classification results of baseline methods. In this table, only the *connectivity graph* is used for prediction. In 5 out of 6 datasets, we've achieved the best performance. The performance of GCN in this paper is worse compared to what was reported by Kipf and Welling [3] It is because that, while GCN supports vertex features, in our experiments, no vertex feature is used for any method, including GCN, for the sake of fair comparison.

Table 3 shows the performance of the integrate model. We compared our method with N-GCN, Abu-El-Haija et al. [1]'s extension of GCN to multi-view graphs. We report that our method outperforms N-GCN in most cases. Note that for Wiki dataset, the combined model performs worse than only using the connectivity graph.

[4]The implementation we used can be found in https://github.com/tkipf/pygcn

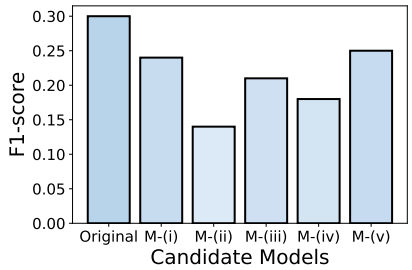
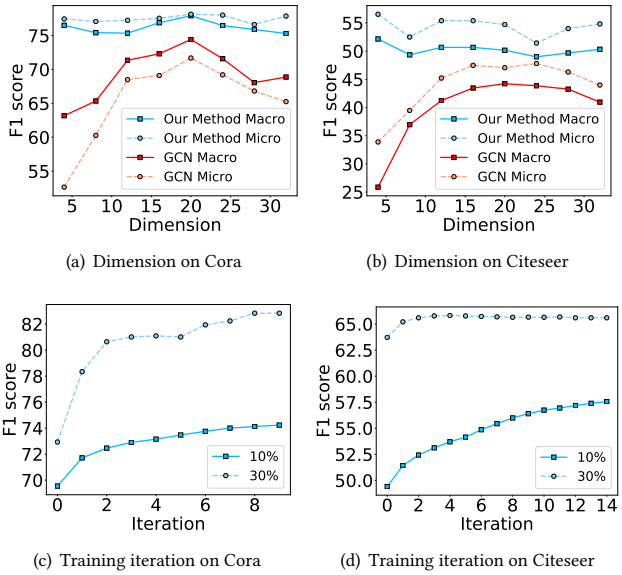


Figure 1: Performance of candidate methods in terms of F1-score on Agent.



(a) Dimension on Cora

(b) Dimension on Citeseer

(c) Training iteration on Cora

(d) Training iteration on Citeseer

Figure 2: Parameter analysis on dimension and ratio of labeled vertices. In (c) and (d), 10% and 30% are the ratio of the size of the training set.

Table 2 shows the classification results in which we compared our algorithm to other baseline methods.

4.4 Model Analysis

Deconstruction of the Model. In this section, we analyze why our model works. By removing or replacing some components of our model, we may see how each contributes to the whole model. Therefore, we compare results with the following candidate models:

(i) Not optimizing embedding by (5), but instead, by the unsupervised Skip-gram, as in [4] and [5];
(ii) Not using (3) to estimate y_i, but instead, use a logistic regression on the learned representations instead;
(iii) Not using (7) to estimate weights between similarities, but instead, use a logistic regression on the similarity-specific $y_{t,i}$'s for the prediction of y_i;
(iv) Using only one of the similarity graphs;
(v) Negative sampling from the whole graph instead of the neighborhood.

The performance of the candidate models are shown in Figure 1. By substituting each component of the model, the performance drops accordingly.

Parameter Analysis. In this section, we analyze the effect of embedding dimension and the training iterations. Figure 2(a) and 2(b) shows the effect of dimensionality on our method and GCN. Clearly, the performance of our method is stable even when the dimensionality is low. In fact, when the embedding dimension is 2, our method gives impressively 53% Macro-F1, while the GCN gives only 15.68%. Our method is more efficient with limited dimensionality because our model is meant to preserve only the local information and the "influence" between vertices. Figure 2(c) and 2(d) shows the change of Micro-F1 score when training. When the labels are scarer, it takes longer iterations for the model to achieve its best performance.

5 CONCLUSION

In this paper, we propose to learn the graph embedding that handles multi-type similarity measures for vertex. In particular, we impose *social influence* in a probabilistic and semi-supervised framework. Extensive experiments demonstrate that with the advantages of incorporating different similarity measures, our model achieves clear improvement comparing with several state-of-the-art baselines.

REFERENCES

[1] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2018. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. *arXiv preprint arXiv:1802.08888* (2018).
[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 891–900.
[3] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.
[5] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.
[6] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.
[7] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.
[8] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, Maosong Sun, et al. 2016. Max-Margin DeepWalk: Discriminative Learning of Network Representation.. In *IJCAI*. 3889–3895.
[9] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.