APRIL 5, 2018

node2vec

SCALABLE FEATURE LEARNING FOR NETWORKS

APOORVA VINOD GORUR
GITHUB.COM/APOORVAVINOD/NODE2VEC

Table of Contents

1. Introduction

Graph and Network data format is popularly used for applications involving social network groups, protein – protein interaction networks and other similar applications. Analysis of such networks are often used to derive important insights into the data. Recent research in Natural Language Processing to learn latent representations [1] for words in a vocabulary has influenced the researchers in graph and network analysis to adopt this technique. DeepWalk [2] was the first model to incorporate the concept of embeddings for the nodes in a graph. node2vec [3] improves over DeepWalk by incorporating efficient traversal techniques for generating random walks in a graph.

2. Methodologies

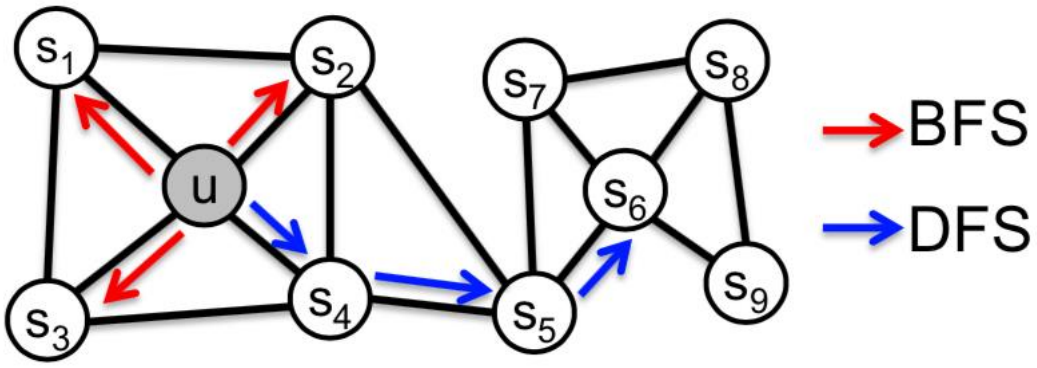

Figure 1 BFS and DFS search strategies

Two main properties that are desirable to capture in a graph are homophily and structural equivalence. Homophily refers to the property that nodes that are similar and belong to similar network clusters tend to have similar embeddings. In figure 1, the homophily property can be seen for nodes *u, s1, s2, s3* and *s4*. Structural equivalence refers to the property of nodes with similar roles have similar structural properties in the graph. In figure 1, the structural equivalence property can be seen for nodes *u* and *s6* which both act as hubs in the network. The next subsection explores how both these properties can be captured in the node embeddings.

2.1. Sampling Strategies

The random walk strategy for node2vec incorporates two sampling techniques: Breadth-First Search (BFS) and Depth-First Search (DFS). Figure 1 shows both the BFS and DFS sampling techniques on the given graph. These sampling techniques are used to sample the neighborhood of a given node *u* given by $N_s(u)$. The BFS sampling technique tends to restrict the sampling to nodes that are in the immediate vicinity of the source. The BFS sampling from source node *u* results in the neighbors *s1, s2* and *s3* which are the immediate neighbors of *u.* DFS sampling results in the nodes that are at increasing distances from the source. DFS sampling starting from

node *u* gives the nodes *s4, s5* and *s6* which are sequentially at increasing distances from source node.

BFS captures the local neighborhood of the node which means that it captures the structural equivalence between the nodes. DFS captures the relationship between different nodes which means that it tends to find bridges between different community structures in the network. BFS gives a micro-view of the neighborhood and DFS, on the other hand gives a macro-view of the neighborhood. DFS sampling leads the walk to explore larger parts of the network by distancing itself from the source node. A network tends to have a mixture of both homophily and structural equivalence and hence it is important to incorporate both BFS and DFS sampling for generating random walks. The next subsection talks about how both the techniques can be used to introduce a bias in the random walk generation.

2.2. Random walk

The random walk generation described below interpolates between BFS and DFS sampling to generate context. Let c_i denote the i^{th} node in the walk, starting with $c_0 = u$. The sampling of nodes uses a probability distribution given by:

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

where π_{vx} is the unnormalized transition probability between nodes *v* and *x* and *Z* is the normalizing constant. The transition probability is computed by $\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$ where

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Here w_{vx} is the weight of the edge between nodes *v* and *x*. For unweighted graphs, w_{vx} will have a default value of 1.0. $\alpha_{pq}(t,x)$ is called the search bias between nodes *t* and *x* with parameters p and q which are called Return parameter and In-out parameter respectively. d_{tx} represents the shortest distance between nodes *t* and *x*. Note that the value of d_{tx} is limited to {0, 1, 2} which limits the number of parameters to two. The parameters *p* and *q* are used to interpolate between BFS and DFS sampling techniques respectively.

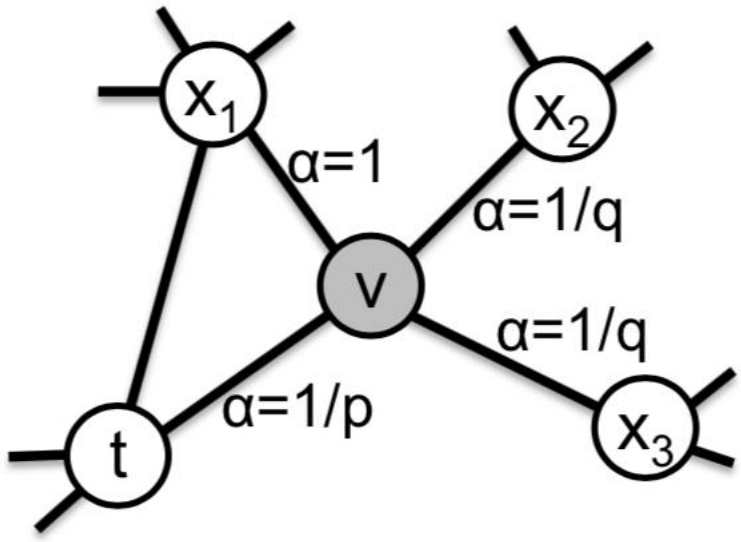

Figure 2 Transition probabilities α for the next step after the walk rooted at t transitioned to v.

The return parameter *p* is used to control the likelihood of the walk to revisit the previous node. Setting *p* to a high (*> max(q,1)*) value will make the walk less likely to resample an already visited node. Conversely setting *p* to a low value (*< min(q,1)*) makes the walk more likely to return to the previous node, hence simulating a BFS traversal. Setting $q > 1$ will make the walk more likely to sample nodes closer to source node and setting $q < 1$ will make the walk to sample nodes further from the source node, hence leading to an outward exploration. Thus, *q* is used to simulate DFS traversal. The random walks generated by this strategy is then fed to word2vec [1]. This treats the random walks as sentences and generates embeddings for each node in the network.

3. Experiments

The experiments are conducted for the task of multi-label classification and compared with the following models: Spectral Clustering [4], DeepWalk [2], LINE [5] and DNGR [6]. The default values provided in the node2vec paper are used for all the models.

3.1 Dataset

BlogCatalog [7]: The nodes in the network represent Blog authors and the edges represent the friendship between the users. There are 39 groups to which users can subscribe to. The network has 10,312 nodes and 333,983 edges.

20-NewsGroup [8]: This dataset contains around 20,000 documents related to 20 different topics. TF-IDF vectors are computed for each document. These vectors are used as nodes in the network and the cosine-similarity between the documents are the edge weights between them. The experiments are conducted on three subgraphs of the whole dataset which are represented

by 3NG, 6NG and 9NG. The subgroups are defined in the same convention used by Tian et al. (2014) [10].

3.2 Results

Table 1 shows the comparison of results achieved for the BlogCatalog dataset. It lists Macro-F1 scores for multi-label classification. node2vec's *p* and *q* parameters used for this dataset are 0.25 and 0.25 respectively.

Algorithm	Average Macro-F1 score
Spectral Clustering	0.0405
DeepWalk	0.2110
LINE	0.0784
node2vec	**0.2667**

Table 1 Comparison of Macro-F1 scores of different models on BlogCatalog dataset

From table 1, it can be seen that node2vec outperforms the other algorithms. The performance (Macro and Micro F1-scores) of node2vec for different training sizes are shown in figure 3. The Macro-F1 score goes as high as 0.3 for training data set to 90%.

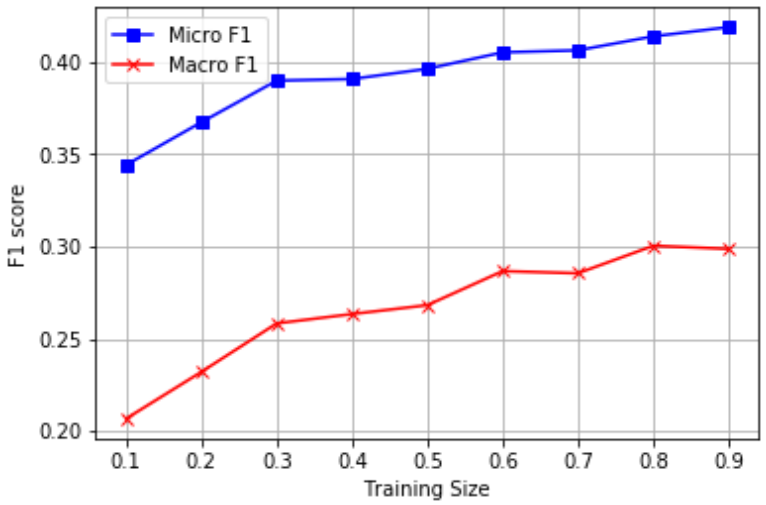

Figure 3 Micro and Macro F1 scores of node2vec for different training sizes

Table 2 shows the comparison of results achieved for the 20-NewsGroup dataset. The normalized mutual information NMI (%) are shown for each sub-graph. node2vec's *p* and *q* parameters used for this dataset are 0.25 and 0.5 respectively.

Model	NMI (%)		
	3NG	6NG	9NG
DNGR (α = 0.98)	74.78	64.38	55.6
DNGR (α = 0.95)	74.51	65.23	54.9
DNGR (α = 1.0)	71.03	**64.84**	52.1
DeepWalk (d = 128)	55.83	58.11	46.86
DeepWalk (d = 256)	56.34	58.4	46.62
DeepWalk (d = 512)	59.64	59.82	46.29
node2vec (d = 128)	**77.51**	62.99	**56.7**

Table 2 NMI(%) value comparison of different models

6. Conclusion

The results reported in section 5 clearly shows node2vec performing better than its contemporaries. It should also be noted that the random walk generation can easily be parallelized so the model is scalable to large networks. The node2vec does a good job of capturing both homophily and structural equivalence properties by interpolating between BFS and DFS sampling techniques. Apart from the multi-label classification shown above, the original authors of node2vec have also adopted node2vec to generate embeddings for edges. These edge embeddings can then be used for link prediction tasks. node2vec has seen success in various applications and has also inspired more models that have been proposed after this, such as struc2vec [9].

7. References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In ICLR, 2013.

[2] B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: Online learning of social representations. In KDD, 2014.

[3] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In ACM SIGKDD.

[4] L. Tang and H. Liu. Leveraging social media networks for classification. Data Mining and Knowledge Discovery, 23(3):447–478, 2011.

[5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: Large-scale Information Network Embedding. In WWW, 2015.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*, pages 1145–1152. AAAI Press, 2016.

[7] R. Zafarani and H. Liu. Social computing data repository at ASU, 2009.

[8] Ken Lang's repository: http://qwone.com/~jason/20Newsgroups/

[9] L. Ribeiro, P. Saverese, D. Figueiredo. struc2vec: Learning Node representations from Structural Identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385-394. KDD'17

[10] Tian, F.; Gao, B.; Cui, Q.; Chen, E.; and Liu, T.-Y. 2014. Learning deep representations for graph clustering. In AAAI.