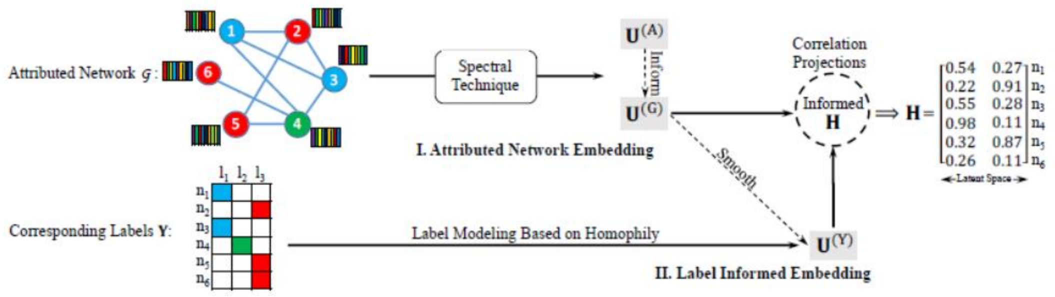
Paper One

Label Informed Attributed Network Embedding

Motivation

Nowadays, many researchers focus their attention on network embedding, which is considered to be very useful in many fields and future applications. However, the existing methods to learn low-dimensional vector representations follow an unsupervised manner, ignoring the abundant label information in real work data. Therefore, the authors of this paper are eager to find a novel algorithm in order to incorporate the known labels into the original model, thus getting the joint vector representations.

Main Ideas



As shown in the above picture, the LANE algorithm contains two important modules: Attributed Network Embedding and Label Informed Embedding Module.

- The former one, Attributed Network Embedding, aims to learn the vector representations that preserve the geometrical information and attribute information in $U^{(G)}$ and $U^{(A)}$. In order to preserve geometrical information, our goal is to $\underset{\mathbf{U}^{(G)}}{\text{minimize}} \quad \frac{1}{2}\sum_{i,j=1}^{n} s_{ij}\|\frac{\mathbf{u}_i}{\sqrt{d_i}} - \frac{\mathbf{u}_j}{\sqrt{d_j}}\|_2^2.$, where s_{ij} is the cosine measurement of g_i and g_j, namely $s_{ij} = \frac{g_i \cdot g_j}{\|g_i\|\|g_j\|}$. And $d_i = \sum_j s_{ij}$, u_i and u_j are the learned representations of node I and node j that we want to get. Actually, the above objective function is easy to understand but is not easy to calculate, thus the authors reformulate it into this one: $\max_{\mathbf{U}^{(G)}} \mathcal{J}_G = Tr({\mathbf{U}^{(G)}}^T \mathcal{L}^{(G)} \mathbf{U}^{(G)})$,
where ${\mathbf{U}^{(G)}}^T \mathbf{U}^{(G)} = \mathbf{I}$ and $\mathcal{L}^{(G)} = \mathbf{D}^{(G)-\frac{1}{2}}\mathbf{S}^{(G)}\mathbf{D}^{(G)-\frac{1}{2}}$.
Similarly, we can get the objective function to preserve the attribute information:

$$\underset{\mathbf{U}^{(A)}}{\text{maximize}} \quad \mathcal{J}_A = \mathrm{Tr}(\mathbf{U}^{(A)^T}\mathcal{L}^{(A)}\mathbf{U}^{(A)})$$,

where $\mathbf{U}^{(A)^T}\mathbf{U}^{(A)} = \mathbf{I}.$ and $\mathcal{L}^{(A)} = \mathbf{D}^{(A)-\frac{1}{2}}\mathbf{S}^{(A)}\mathbf{D}^{(A)-\frac{1}{2}}$.

What's more, to incorporate $U^{(A)}$ into $U^{(G)}$, the authors project $U^{(A)}$ into the space of $U^{(G)}$ and define the following formula as the correlations:

$\rho_1 = \mathrm{Tr}(\mathbf{U}^{(A)^T}\mathbf{U}^{(G)}\mathbf{U}^{(G)^T}\mathbf{U}^{(A)}).$ And the entire goal is to maximize $J_G + \alpha J_A + \alpha\rho_1$.

- The latter one, Label Informed Embedding Module, aims to learn the vector representations that preserve the label information. Similar to the J_G and J_A, the mathematical formula of J_Y is as followed: $\underset{\mathbf{U}^{(Y)}}{\text{maximize}} \quad \mathcal{J}_Y = \mathrm{Tr}\left(\mathbf{U}^{(Y)^T}(\mathcal{L}^{(YY)} + \mathbf{U}^{(G)}\mathbf{U}^{(G)^T})\mathbf{U}^{(Y)}\right)$,

Where $\mathcal{L}^{(YY)} = \mathbf{D}^{(Y)-\frac{1}{2}}\mathbf{S}^{(YY)}\mathbf{D}^{(Y)-\frac{1}{2}}$, and $S^{(YY)}$ is the cosine similarity of YY^T. The difference of J_Y is that the rank of matrix $S^{(YY)}$ is limited by the number of label categories k, which is usually smaller than the embedding dimension d. Thus, the authors use $\mathbf{U}^{(G)}\mathbf{U}^{(G)T}$ to smooth the equation. What's more, the final goal is to learn a matrix H to preserve all the information in a low-dimensional space, thus we need to define the correlations as followed: $\rho_2 = \mathrm{Tr}(\mathbf{U}^{(G)^T}\mathbf{H}\mathbf{H}^T\mathbf{U}^{(G)})$, $\rho_3 = \mathrm{Tr}(\mathbf{U}^{(A)^T}\mathbf{H}\mathbf{H}^T\mathbf{U}^{(A)}),$, and $\rho_4 = \mathrm{Tr}(\mathbf{U}^{(Y)^T}\mathbf{H}\mathbf{H}^T\mathbf{U}^{(Y)}).$. The cost function is $\mathcal{J}_{corr} = \rho_2 + \rho_3 + \rho_4,$

- Finally, we are supposed learn the joint representation according to the first module and second module. Therefore, the final goal is:

$$\underset{\mathbf{U}^{(\cdot)},\mathbf{H}}{\text{maximize}} \quad \mathcal{J} = (\mathcal{J}_G + \alpha_1\mathcal{J}_A + \alpha_1\rho_1) + \alpha_2\mathcal{J}_Y + \mathcal{J}_{corr}$$

Where $U^{(G)T}U^{(G)} = I$,$U^{(A)T}U^{(A)} = I$,$U^{(H)T}U^{(H)} = I$ and $H^TH = I$

Main Algorithm

According to the paper, the optimization problem is a convex problem, so it is guaranteed to converge to a local optimal point. Moreover, each updating step of four variables can be corresponding to find the top d eigenvectors:

$(\mathcal{L}^{(G)} + \alpha_1\mathbf{U}^{(A)}\mathbf{U}^{(A)^T} + \alpha_2\mathbf{U}^{(Y)}\mathbf{U}^{(Y)^T} + \mathbf{H}\mathbf{H}^T)\mathbf{U}^{(G)} = \lambda_1\mathbf{U}^{(G)}$, (13)

$(\alpha_1\mathcal{L}^{(A)} + \alpha_1\mathbf{U}^{(G)}\mathbf{U}^{(G)^T} + \mathbf{H}\mathbf{H}^T)\mathbf{U}^{(A)} = \lambda_2\mathbf{U}^{(A)}, \quad (14)$

$$(\alpha_2 \mathcal{L}^{(YY)} + \alpha_2 \mathbf{U}^{(G)} \mathbf{U}^{(G)T} + \mathbf{H}\mathbf{H}^T)\mathbf{U}^{(Y)} = \lambda_3 \mathbf{U}^{(Y)}, \quad (15)$$

$$(\mathbf{U}^{(G)} \mathbf{U}^{(G)T} + \mathbf{U}^{(A)} \mathbf{U}^{(A)T} + \mathbf{U}^{(Y)} \mathbf{U}^{(Y)T})\mathbf{H} = \lambda_4 \mathbf{H}. \quad (16)$$

And the whole algorithm is as followed:

```
   Input: $d$, $\epsilon$, $\mathcal{G}$, $\mathbf{Y}$.
   Output: Embedding representation $\mathbf{H}$.
 1 Construct the affinity matrices $\mathbf{S}^{(G)}$ and $\mathbf{S}^{(A)}$;
 2 Compute Laplacian matrices $\mathcal{L}^{(G)}$, $\mathcal{L}^{(A)}$ and $\mathcal{L}^{(Y)}$;
 3 Initialize $t = 1$, $\mathbf{U}^{(A)} = \mathbf{0}$, $\mathbf{U}^{(Y)} = \mathbf{0}$ and $\mathbf{H} = \mathbf{0}$;
 4 repeat
 5     Update $\mathbf{U}^{(G)}$ by solving Eq. (13);
 6     Update $\mathbf{U}^{(A)}$ by solving Eq. (14);
 7     Update $\mathbf{U}^{(Y)}$ by solving Eq. (15);
 8     Update $\mathbf{H}$ by solving Eq. (16);
 9     $t = t + 1$;
10 until $\mathcal{J}_t - \mathcal{J}_{t-1} \leq \epsilon$;
11 return $\mathbf{H}$.
```

My idea

In my opinion, network embedding is very powerful since it can transfer the complex topological graph into low dimensional matrices. In the matrices, each vector represents a node, thus we are very convenient to use the node vectors. We can not only apply many classical machine learning algorithms onto the graph, but also solve the graph problem such as community detection very efficiently, in which we iteratively walk every node again and again in the past but can use clustering method with node vectors right now.

Paper Two

DeepWalk: Online Learning of Social Representations

Motivation

In this paper, the authors come up with a new idea creatively that we can associate the network embedding problem with the language models such as the famous one, word2vec. Authors are going to take a random walk on the network and treat the nodes as words, the sequence of walks as sentences. After thinking about these ideas, authors try to perfect their ideas as well as make experiments to confirm their ideas.

Main Theory

The problem authors want to solve is illustrated as the following. For a given graph G(V,E) and a partial labeled network G_L=(V,E,X,Y) with attributes $X \in R^{|V| \times S}$ and labels $Y \in R^{|V|\times|y|}$, our goal is to learn $X_E \in R^{|V|\times d}$, where d is a small number. The X_E is required to preserve the original topological information as well as perform well when the network changes a little bit.

In order to achieve the above goal, the authors combine the random walk with the natural language model. As for random walk, there are many advantages: we can have local information by random walk; it is better to apply to parallel; it can only be applied to the updating parts of network…

From language model, the goal is to maximize the probability $\Pr\left(v_i \mid (v_1, v_2, \cdots, v_{i-1})\right)$ => $\Pr\left(v_i \mid (\Phi(v_1), \Phi(v_2), \cdots, \Phi(v_{i-1}))\right)$, where a mapping: $\Phi : v \in V \rightarrow R^{|V|\times d}$. But the problem is hard to solve, thus the authors transfer it into the following formula on the basis of the word2vec model's ideas:

$$\underset{\Phi}{\text{minimize}} \quad -\log \Pr\left(\{v_{i-w}, \cdots, v_{i-1}, v_{i+1}, \cdots, v_{i+w}\} \mid \Phi(v_i)\right)$$

Main Algorithm

Since the algorithm is combined with the ideas of random walk and natural language model word2vec, thus the major parts are RandomWalk and SkipGram:

```
Input: graph $G(V, E)$
    window size $w$
    embedding size $d$
    walks per vertex $\gamma$
    walk length $t$
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V|\times d}$
 1: Initialization: Sample $\Phi$ from $\mathcal{U}^{|V|\times d}$
 2: Build a binary Tree $T$ from $V$
 3: for $i = 0$ to $\gamma$ do
 4:     $\mathcal{O} = \text{Shuffle}(V)$
 5:     for each $v_i \in \mathcal{O}$ do
 6:         $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
 7:         SkipGram($\Phi$, $\mathcal{W}_{v_i}$, $w$)
 8:     end for
 9: end for
```

RadndomWalk module selects network vertices randomly and generates the number of γ vertices sentences with the length of t. And the SkipGram module is responsible for updating the representations of nodes:

```
Algorithm 2 SkipGram($\Phi$, $\mathcal{W}_{v_i}$, $w$)
1: for each $v_j \in \mathcal{W}_{v_i}$ do
2:     for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ do
3:         $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
4:         $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
5:     end for
6: end for
```

My idea

Though the authors do not introduce a completely new method to learn the network representations, I still consider the authors very creative. Because they put forward a new thought to consider network embedding and connect it with natural language model, which leave us many future work to think and do. As for the algorithm itself, it is not very perfect probably because the most important contribution of this paper is its idea. Its random walk module is too simple since it is working completely randomly. Therefore, there is no doubt that some other works are trying to fix this problem such as node2vec algorithm, which is published in KDD, 2016.