REPORT

Information Retrieval and Extraction

Major Project

Understanding Large Social Networks

Mentor
Ganesh Jawahar

Team Number
57

Team Members
Madan Jhawar (201202018)
Raj Patel (201301048)
Kishan Vadalia (201505621)

Abstract

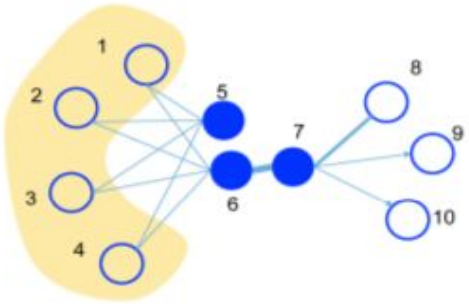
In very large networks, containing millions of nodes and billions of edges, some application like visualization, node classification, and link prediction become extremely complex. Therefore, an efficient and scalable solution is to represent every node in a network into low-dimensional vector spaces. These embeddings can further be used for tasks such as visualization, node classification, link prediction, etc. Several approaches such as DeepWalk, LINE, etc., have been proposed for this task. For this project, we focus mainly on LINE model, which is suitable for arbitrary types of information networks such as undirected, directed, and/or weighted. The method optimizes a carefully designed objective function that preserves both the local and global network structures. Empirical experiments prove the effectiveness of the LINE on a variety of real-world information networks, including language networks, social networks, and citation networks.

1. Introduction

There are several existing real-world information networks such as WWW, Flickr, YouTube, Facebook, etc. The size of these Information networks ranges from hundreds of nodes to billions of nodes. In the baseline paper, a network embedding model, called the "LINE", is proposed. The model is able to scale to very large, arbitrary types of networks such as undirected, directed and/or weighted.

The paper explores two important concepts: *First-Order Proximity* and *Second-Order Proximity.* The First-Order Proximity captures the observed tie strength between the nodes, whereas, the Second-Order Proximity takes into account the shared neighbourhood structures of the vertices. The general notion of the second-order proximity can be interpreted as nodes with shared neighbours being likely to be similar.

The effectiveness of the learned embeddings is evaluated within multiple data mining tasks, including word analogy, text classification, and node classification. The results suggest that the LINE model outperforms other competitive baselines in terms of both effectiveness and efficiency. It is able to learn the embedding of a network with millions of nodes and billions of edges in a few hours on a single machine.



A toy example of information network. Edges can be undirected, directed, and/or weighted. Vertex 6 and 7 should be placed closely in the low-dimensional space as they are connected through a strong tie. Vertex 5 and 6 should also be placed closely as they share similar neighbours

2. LINE: Large-Scale Information Network Embedding

A desirable embedding model for real world information networks must satisfy several requirements. First, it must be able to capture both the local and the global structure of the network. Second, it must scale for very large networks, containing millions of vertices and billions of edges. Third, it should be applicable to networks with arbitrary types of edges such as directed, undirected and/or weighted. LINE presents a novel network embedding model which satisfies all the three requirements.

2.1. First-order Proximity

The first-order proximity refers to the local pairwise proximity between the vertices in the network. To model the first-order proximity, for each undirected edge (i, j), we define the joint probability between vertex v_i and v_j as follows:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}$$

Whereas, Empirical Probability is defined by:

$$\hat{p}_1(i, j) = \frac{w_{ij}}{W}$$

$W = \sum_{(i,j)\in E} w_{ij}$ where

We minimize the KL-divergence of two probability distributions, given by:

$$O_1 = -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$

Here, we are trying to minimize the difference between the two probability distributions such that estimated probability value comes as close as possible to the empirical value.

2.2. Second-order Proximity

The second-order proximity assumes that vertices sharing many connections to other vertices are similar to each other. In this case, each vertex is also treated as a specific "context" and vertices with similar distributions over the "contexts" are assumed to be similar. Therefore, each vertex plays two roles: the vertex itself and a specific "context" of other vertices.

We introduce two vectors u_i and u_i', where u_i is the representation of v_i when it is treated as a vertex while u_i' is the representation of v_i when it is treated as a specific "context". For each directed edge (i, j), we first define the probability of "context" v_j generated by vertex v_i as:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)}$$

The empirical distribution $\hat{p}_2(\cdot|v_i)$ is defined as

$$\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{d_i}$$

where w_{ij} is the weight of the edge (i, j) as $\hat{p}_2(v_j|v_i) = d_{ij}$ and d_i is the out-degree of vertex i, i.e.

$$d_i = \sum_{k\in N(i)} w_{ik},$$

We, therefore, want to minimize the KL-Divergence between the two probability distributions,

$$O_2 = -\sum_{(i,j)\in E} w_{ij} \log p_2(v_j|v_i)$$

To embed the networks by preserving both the first-order and second-order proximity, a simple and effective way we find in practice is to train the LINE model which preserves the first-order proximity and second-order proximity separately and then concatenate the embeddings trained by the two methods for each vertex.

2.3. Model Optimization

If we carefully analyze O_2, it requires the summation over the entire set of vertices when calculating the conditional probability $p_2(\cdot|v_i)$. This becomes computationally expensive if the number of vertices in the network becomes too high. To address this problem, we adopt the approach of **Negative Sampling**, which samples multiple negative edges according to some noisy distribution for each edge (i, j). More specifically, it specifies the following objective function for each edge (i, j):

$$p_2(v_j|v_i) = \log \sigma(\vec{u}_j'^T \cdot \vec{u}_i) + \sum_{i=1}^{K} E_{v_n \sim P_n(v)} [\log \sigma(-\vec{u}_n'^T \cdot \vec{u}_i)]$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function. The first term models the observed edges, the second term models the negative edges drawn from the noise distribution and K is the number of negative edges. We set $P_n(v) \propto d_v^{3/4}$, where d_v is the outdegree of vertex v.

For the objective function, there exists a trivial solution: $u_{ik} = \infty$, for i = 1, . . . , |V| and k = 1, . . . , d. To avoid the trivial solution, we can still utilize the negative sampling approach by just changing $u_j'^T$ to u_j^T.

We adopt the asynchronous stochastic gradient algorithm for optimizing the equation. In each step, the ASGD algorithm samples a mini-batch of edges and then updates the model parameters.

3. Novelty

As mentioned in Section 2.2, the authors train the neural networks for the First-Order Proximity and the Second-Order Proximity independently, and then concatenate the embeddings learned from both the models for a node to get the final embeddings of a node.

In our project, we update this part of the model. We use the same lookup table for the node representation in both the neural networks, i.e., the node embeddings learned from the First-Order Proximity are used in the neural network for the Second-Order Proximity, instead of initializing them again to random values. The embeddings learned from the Second-Order Proximity deep-net are, then, taken as the final embeddings of a node.

4. Experimentation and Results

4.1 Dataset

We are using **Blogcatalog Dataset**, which manages the bloggers and their blogs. There are *directed edges* between the bloggers (nodes), which represent friendship amongst them. The specifications are given below:

#Nodes	10312
#Edges	333983
#Categories (Labels)	39

4.2 Results

Parameters

Feature	Value
Embedding Size	20
Number of Epochs	30
Learning Rate	0.035
Number of Negative Samples	10

Results

Metric \ Train%	60%	70%	80%
Micro-F1	38.6%	39.71%	40.34%
Macro-F1	24.47%	24.93%	25.21%

LINE Model, when separately trained and concatenated

Metric \ Train%	60%	70%	80%
Micro-F1	39.34%	39.11%	41.62%
Macro-F1	23.82%	24.13%	25.36%

LINE Model, with same lookup table

5. References

- LINE paper (Baseline Paper) - http://www.www2015.it/documents/proceedings/proceedings/p1067.pdf
- DeepWalk paper - http://arxiv.org/pdf/1403.6652.pdf
- Deep Learning Stanford Lecture notes - http://cs224d.stanford.edu/lecture_notes/LectureNotes1.pdf
- Neural network Github repository - https://github.com/torch/nn
- Sample Lua codes -
  - https://github.com/gjwhr/dl4nlp-made-easy/blob/master/word2vec/cbow.lua
  - https://github.com/gjwhr/dl4nlp-made-easy/blob/master/word2vec/skipgram.lua
- Blog Catalog Dataset - http://socialcomputing.asu.edu/datasets/BlogCatalog3