A Research of Link Prediction on Directed Signed Graph

Mingyang Jiang
Shanghai Jiaotong University
F1903802
519030910384

Wenzhuo Zheng
Shanghai Jiaotong University
F1903801
519030910360

Jinghao Feng
Shanghai Jiaotong University
F1903801
519030910361

Abstract—This project presents an link prediction algorithm that expands Node2Vec, which was covered in class, onto directed signed graph. Starting from the basic idea of node embedding, we design asymmetric features to support directed edges, and apply attention-like mechanism to deal with positive and negative edges respectively. We apply a simple neural network to classify positive and negative edges based on node embedding. After many iterations and modifications, our model achieves fairly well performance on Wiki-RfA, a public dataset that records Wikipedia editor voting. Our code is available at https://github.com/Raingate/LinkPrediction-on-Directed-Signed-Graph.git

Index Terms—link Prediction, directed signed graph, node embedding

I. Introduction

In class, we learnt some classical models dealing with link prediction problem. Among which, embedding-based algorithms such as DeepWalk and Node2Vec, aroused our interests. In homework 2, we implemented these algorithms to predict links in an simple undirected, unsigned graph. We believe that based on this, node embedding methods can be modified to adapt new applications.

Consider one particular scenario: There are a group of people, within some period of time they initiated many votings. In each voting, one person becomes candidate, and a small group of other people vote. They can choose to support, oppose or abstain. Now given a possible future vote, we want to predict its class. Or equivalently, given a possible candidate, we want to predict whether he will be elected.

The scenario above can be modeled as a directed signed graph, where each node represents a person, each edge starts from a voter and points to a candidate. Based on the vote's class, the weight can be 1(support), 0(abstain) or -1(oppose).

To predict link class, traditional Node2Vec or DeepWalk cannot be competent anymore. After careful analysis, we decided that the features below must be added:

1. asymmetric embedding feature to distinguish source and target nodes.

2. a proper mechanism to encode link weight information into the embedding.

3. a properly fused classifier to predict link weight based on the embedding.

In this project, we maintain two different embedding vector for each node, namely source embedding and target embedding. When an edge feature is needed, we combine source embedding of the source node and target embedding of the target node. Specific combination methods will be presented in IV. As for embedding training, we retain basic idea of previous methods, using combination of neighboring nodes' embedding to represent a node, and optimize the distance between embedding and combination. When training, for each participating node, exactly one of its source and target embedding will be updated, according to the node's character in the vote.

As for edge classifier, we apply a light-weight network that takes source embedding of one node and target embedding of another and output weight prediction.

Although the network itself is simple, the tricky part is how to fuse it into the whole system. To bring this assumption to reality, we apply an attention-like mechanism, specifically two different weight vectors to the embedding combination part, guaranteeing separate training between positive and negative edges. This will be presented in IV.

II. Related Work

A. Network Embedding

Network embedding is the basic idea of graph representation in this project. Classic methods such as DeepWalk inherit the main idea of CBOW(Continuous Bag of Words, which is a classic model to predict text based on context)[1] in natural language processing, using feature vectors of fixed length to represent nodes, then optimize vector distance between endpoints of edges. To capture local information, Deepwalk[2] generate graph sentences with random walks, representing each node by its neighbors or sub-neighbors' embedding. Other recent works include LINE, SDNE and so on. However, these methods can only address unsigned and homogeneous networks.

B. Signed Link Prediction

In social networks, we often need to predict whether a link is positive or negative. In signed graph link prediction, Leskovec. et al. proposed signed triads as features for prediction based on structural balance theory[3][4]. Yuan et al. proposed to use separate features to represent source and target feature respectively, and apply separate weight vectors for positive and negative edges[5]. These works inspire our idea of using two different embedding for each node to address different node characters, and apply attention-like Linear layer to process positive and negative edges.

III. Dataset Establishment

After wide searching, we decided Wiki-RfA is the best-fit dataset for our model. It's a directed signed dataset which contains adequate data and is relatively balanced. For a Wikipedia editor to become an administrator, a request for adminship (RfA) must be submitted, either by the candidate or by another community member. Subsequently, any Wikipedia member may cast a supporting, neutral, or opposing vote. Wiki-RfA (kindly provided by Jure Leskovec on http://snap.stanford.edu) recorded 198275 votes(141521 support, 40363 oppose) among 11381 users. The graph is dense enough.

We wrote the python script parse_set.py to extract vote triads (voter, candidate, vote class), and turned user names into continuous integer ids, then store them in parsed.csv.

IV. Our Model

We present our network embedding and classifier model for directed signed networks.

A. Problem definition

Formally, a directed signed network is defined as $G = (V, E_+, E_-)$, where V is the set of vertices and E_+ (E_-) is the set of positive(negative) edges. Each edge $e \in E_+ \cup E_-$ is represented as $e_{uv} = (u, v, \epsilon_{uv})$, where $u, v \in V$ and $\epsilon_{uv} \in \{1, -1\}$(here 0 is omitted as abstain votes provides so little information) indicates the sign value(weight) of edge e. Our goal is to learn node embedding for each vertex in a directed signed network, and a classifier that can tell positive and negative edges based on the embedding.

B. Network Embedding Part

We develop out directed signed network embedding by adapting attention-like model.

Attention-like model: Given a sample sequence of node v_i's neighbors who points to v_i (BFS random walk), with their source embedding vectors, the attention-like model firstly computes the predicted target embedding of v_i by two different linear weight layer A_+ and A_-:

$$\hat{v}_i = \frac{1}{2}[\frac{1}{|N_+|}A_+(N_+) + \frac{1}{|N_-|}A_-(N_-)] \tag{1}$$

, where$A_+()$ ($A_-()$) denotes positive(negative) attention-like weight layer, and N_+ (N_-) denotes embedding vectors of neighbors that points to v_i with a positive(negative) edge. Note that we apply normalization coefficients to guarantee balance training between positive and negative edges, as positive edges are usually far more than negative edges in natural datasets. Note that within this process, each sample sequence contributes to exactly one target embedding vector (the certer node's) and a few source embedding vectors (neighbors'). This is shown in Fig.1

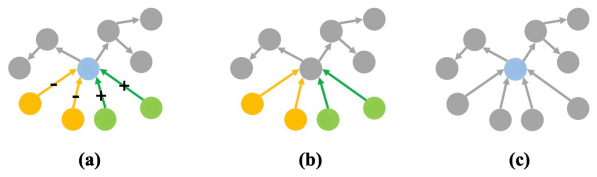

Fig. 1. (a) shows one training sequence (green / yellow means positive / negative edge or neighbor, this manner applies to all figures in this report). In embedding training, source embedding of context nodes (b) and target embedding of center node(c) are updated.

The reason we only sample neighboring nodes is that sub-neighbor is hard to define semantically. For example, if user a votes oppose to b and b votes oppose to c, we cannot state that a tend to support c. Our model only define binary relationship.

Then of similar considerations to CBOW, we apply a loss function taking in target embedding vector of v_i, and $\hat{v}_i$ as input:

$$L_1 = -\log\left(\sigma(d_{cos}(v_i, \hat{v}_i))\right) \tag{2}$$

where σ denotes Sigmoid function and d_{cos} denotes cosine distance between two vectors. Apparently, cosine distance is a symmetric distance metric, thus cannot address directed edge problem. However, attention-like layers guarantee that direction information is encoded into embedding and A_+ (A_-) beforehand. In conclusion, network embedding part trains node embedding v_+ (v_-) and attention-like layers A_+ (A_-) by optimizing the objective function similar to CBOW.

C. Edge Classifier Part

With embedding information, we propose to classify edges into class positive or negative with a light-weight neural network. Specifically, our neural network combines 3 linear layers in series, who takes in source embedding vector of start-point of the edge to predict and target embedding vector of end-point, and outputs a real number prediction between -1 and 1. Finally, apply MSE(mean square error) as loss function:

$$L_2 = MSE(\hat{w}, label) \tag{3}$$

where $\hat{w}$ denotes prediction and label is the ground truth edge weight (1 or -1).

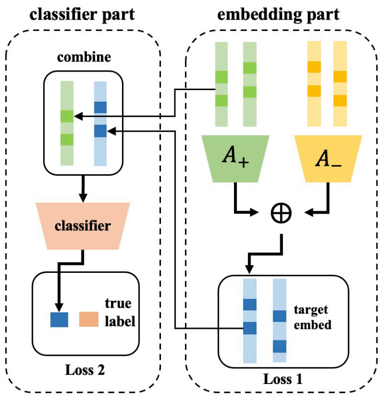

Fig. 2. In coupled version, the classifier takes in source embedding of context nodes and target (predicted) embedding of center node, and updates attention-like layer twice in each epoch

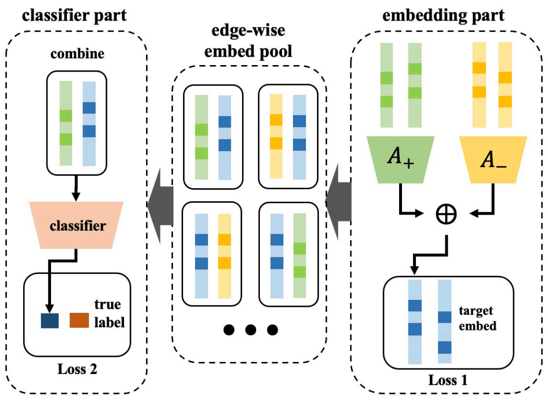

Fig. 3. In decoupled version, we train embedding first, constructing edge-wise embedding pair pool. The pool remains static ever since. The classifier is trained on these labeled pairs

This network seems simple, but the tricky part is how to fuse it into network embedding part. During our study, we proposed two fuse schemes. Here we give a discuss respectively.

1) Coupled version: This is an early version of our model (shown in fig.2), where in each epoch, for each sample sequence, first implement embedding part and compute L_1, then implement classifier part to compute L_2. Finally implement back propagation with L_1 and L_2. Specifically, BP (Back Propagation) of L_1 updates parameters of embedding and attention-like layer. BP of L_2 updates parameters of classifier network and attention-like layer. The reason for this design is that we naturally didn't believe that attention-like layer can learn weight information well without supervision of the classifier. However, two updates of attention-like layer in each epoch is tedious. In fact, coupled version cannot achieve acceptable performance. After experiments and careful analysis, we inferred that in early stage of training, the first and second updates will destroy each other's learnt information, causing the whole model get lost in parameter space.

2) Decoupled version: To deal with the problem above, we decided to accept the assumption that the embedding part can learn weight information well without supervision of classifier. Accordingly, we decoupled embedding and classifier parts (shown in fig.3). In this version, we train embedding and classifier parts separately. After training embedding, we construct an edge-wise embedding pool, where each element contains source and target embedding of two end-points of an edge, as well as the true label of the edge. Classifier takes in pool elements as its input instead.

It turns out that this modification achieves an obvious improvement of performance. This actually surprised us. That means the attention-like layer succeeds to encode edge weight information without supervision of classifier. We infer that back propagation isolation between positive and negative samples made it to divide themselves in parameter space. To conclude, the embedding part is an unsupervised system! That's to say, the attention-like layer really reached its design goal. All experiment results will be shown in V.

V. Experiments

In this section, we evaluate the performance of our proposed model on Wiki-RfA dataset. We first introduce our quality metrics, and parameter settings for experiments, then present the experimental results of our algorithm.

A. Quality metrics

We use AUC and f1-score as our quality metrics. AUC is a classic metric for binary classification problem, which is the area under ROC curve. f1-score is a widely-used (e.g. in [6]) metric in link prediction works, which is the harmonic average of precision and recall.

B. Parameter settings

In embedding part, each sample sequence contains the center node and at most 10 positive (negative) neighbors (here positive/negative neighbor means neighbor pointing to center node with positive/negative edge). If less than 10, we don't over-sample. Using optimizer Adam with learning rate 1e-2 and embedding vector length 256, it takes about 15 epochs for loss 1 to converge to about 0.33. So we fix this number of epochs as 20. The attention-like layers' input and output are consistent with embedding vector length.

In classifier part, we combine input source and target embedding vectors with different methods. This will be discussed later. As for network structure, we apply 3 ReLU-activated linear layers to down-sample input from size 64 to 16 to 1, and a tanh function at last. Using optimizer Adam with learning rate 1e-3, it takes about 20 epochs for loss 2 to converge. So we set this number of epochs as 25.

For testing part, we randomly sampled 1000 positive and 1000 negative edges from whole graph. When training, in each epoch we randomly sample equivalent number of positive and negative edge pairs from edge-wise embedding pool. Different numbers of training pairs per epoch are tried, and we will discuss this later.

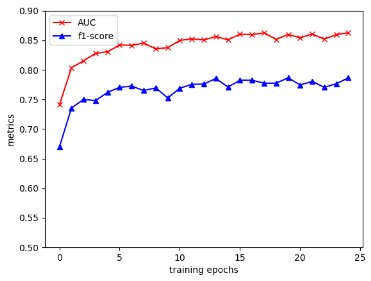

Fig. 4. one testing result trajectory of decoupled version model. Both AUC and f1 score is shown in the figure

C. Performance of our model and comparisons

The best performance is made by decoupled version model. A particular testing result is shown in Fig.4.

As shown above, the best AUC we reached is 87%, the best f1 score 0.79. This is a relatively acceptable performance.

Besides, we compare the decoupled version with coupled version and plain Node2Vec model. Here we construct Node2Vec model in an intuitive way: first construct 'positive (negative) graph' using only positive (negative) edges. Omitting edge direction, train 2 set of Node2Vec embedding on positive and negative graph respectively. When deciding an edge's class, we first compute its scores with two (+ and -) embedding, denoted as $score_+$ and $score_-$. Then we decide its class by thresholding

$$s = \frac{score_+}{score_+ + score_-} \tag{4}$$

For coupled version model, we use the same parameters as decoupled version. We compare these 3 models with different training set size, ranging from (2000+,2000-) to (20000+,20000-). We use (20000+,20000-) as the standard training set size (1.0) here.

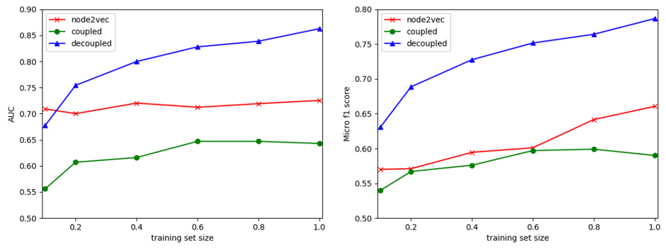

Fig. 5. Comparisons of AUC and f1 score of plain Node2Vec, coupled and decoupled version. Different training set sizes is used (0.1, 0.2, 0.4, 0.6, 0.8, 1.0)

We can see from figure above that coupled version only reaches AUC less than 65% and f1 score less than 0.6. The plain Node2Vec, to our surprise, reaches AUC 70% and f1 score 0.65. As analyzed in III, coupled structure of embedding network and classifier really does harm to parameter learning of attention-like layers.

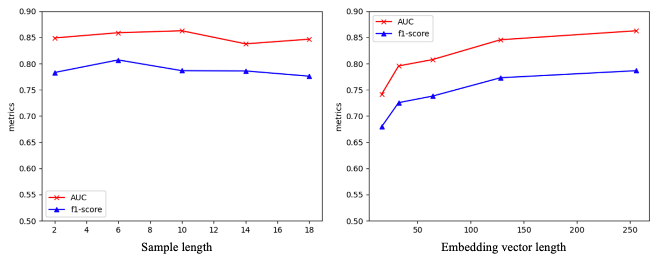

Fig. 6. Fig on the left shows effects of different sample length on AUC and f1 score, and fig on the right shows effects of different embedding vector length

D. Parameter sensitivity

In this subsection we examine how different choices of parameters affect the performance of our model. When one parameter is testing, others are set as default.

1) Embedding vector length: Embedding vector length denotes the size of vector embedded from a node in graph. A larger embedding length can imply more information but be more computationally costly. Fig.6 shows the AUCs and F1-scores using different embedding vector lengths. We set our length from 16 to 256. The experiment shows that a better performance is achieved using larger embedding length. We can have a view that too short embedding length such as 16 or 32 is not enough for implicit information embedding. Though increasing the embedding vector length can further improve the performance, larger size like 512 and 1024 are not recommended. We consider that length of 256 nearly reaches the limit of our model, and larger size can cause rather long training time.

2) Sample length: This parameter limits the max size of neighbors in a sample. For each training node, we sample up to *sample_length* positive neighbors and up to *sample_length* negative neighbors if possible. Fig.6 gives the performance with sample length of 2,6,10,14 and 18. The results shows that this parameter has a little influence on performance. This is because the difference between sample length can be narrowed if we train for more epochs to converge. However, too large sample length can cause the imbalance between positive neighbors and negative neighbors as candidates have less discommenders than supports, which make the training biased.

3) Combination methods for classifier: As is described before, we concatenate the source and target node embedding as classifier input. We also try other combination methods from [5]. The results are shown in Table I. Our method outperforms the other choices of combination. This is because concatenating maintains the

TABLE I

combination methods	AUC	F1-score
InnerProduct	0.632	0.577
Average	0.771	0.707
L1-weight	0.623	0.589
L2-weight	0.670	0.636
Concatenate(Ours)	0.863	0.787

order information for source node and target node, while other methods combine the two nodes in a symmetric form. Actually, we have noticed the asymmetry in the Wikipedia dataset, such as two voters give opposite votes for each other. While other methods fail to deal with this asymmetry, deep network can learn to handle the directed edges if we provide ordered combinations. These cases also support our choice of embedding source and target nodes differently.

VI. Conclusions

In this project we study the problem of predicting voting-style links on directed signed graph. We first designed separate embedding vectors for each node, encoding edge direction information. Then we designed an attention-like layer who address samples from positive and negative edges respectively. We constructed an embedding-classifier coupled network with attention-like layer at the intersection, which failed to achieve good performance. Later we decoupled embedding and classifier parts, training embedding in an unsupervised manner, which achieves rather good performance. We conduct extensive experiments to validate our model, and did research on how parameters and network structures affect our model's performance. Also, we compare our model with other models. During the process, we gain insights into directed signed feature designing, and how to arrange model structure to make every part of the model do their best. We extend classic methods to adapt to new problems, and that's a lot of enlightenment.

VII. Relationship between our work and data mining technologies

As is discussed in I, our work is inspired by DeepWalk and Node2Vec taught in class. We extend their idea, design our own network structure to address directed signed graph problem. This can be used for data mining applications such as vote prediction.

Acknowledgment

We would like to thank Professor Xiang and Teaching Assistants for their kind and patient help throughout this hard semester. We learnt a lot from Data Mining course, it really broadened our horizons.

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.

[2] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.

[3] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in Proceedings of the SIGCHI conference on human factors in computing systems, 2010, pp. 1361–1370.

[4] ——, "Predicting positive and negative links in online social networks," in Proceedings of the 19th international conference on World wide web, 2010, pp. 641–650.

[5] S. Yuan, X. Wu, and Y. Xiang, "Sne: signed network embedding," in Pacific-Asia conference on knowledge discovery and data mining. Springer, 2017, pp. 183–195.

[6] H. Wang, F. Zhang, M. Hou, X. Xie, M. Guo, and Q. Liu, "Shine: Signed heterogeneous information network embedding for sentiment link prediction," in Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, 2018, pp. 592–600.