Temporal Link Prediction with Network Embedding: Solution of Team NodeInGraph

Manting Shen
Team: NodeInGraph
Beijing, China
3230391@gmail.com

ABSTRACT

Real world networks contains entities and there relationship are often described by various graphs since the interactions between entities are like the edges linked by nodes. However, a static graph is not enough for expressing the networks which are in practice temporal, with new edges coming in or disappearing as the passage of time. As the consequence, traditional link prediction tasks which ask if an edge exists between two nodes on a partially observed graph is not appropriate for temporal networks and alternative it is more useful asks if an edge will exist between two nodes within a given time span. In this paper, we will introduce our method to address the temporal link prediction problem in WSDM Cup 2022 [1], which aims at predicting link over temporal graphs within a given time period. As a participant (team NodeInGraph) in the competition, we explore the solution with graph-based method and find that only some network embedding methods can achieve a satisfying result. As a result, our team won the third prize with a T-Score of 0.585137 in the final leaderboard.

KEYWORDS

Temporal Link Prediction, Network Embedding, Supervised Learning

1 METHODOLOGY

Here we will introduce the overview of our proposal, our method simply calculates the edge probability based on LINE embedding, the code is established in github [2]. In the following, the details of LINE are first introduced, then we present the overall flow of data processing and model inference, finally the model results are shown.

1.1 Model Description

As well known, a real word network can be represented by a graph using verticals and edges. Currently, most graph embedding models only treat edges as undirected, which are not able to scale to large, arbitrary types of networks especially when the network has little local structures(links between neighbours). A desirable embedding model for the temporal link prediction task must satisfy several requirements: first, it must be able to preserve the proximity between different kinds of nodes linked by various edges; second, it must scale for very large networks, say millions of vertices and billions of edges; third, it can deal with networks with arbitrary types of edges: directed, undirected and/or weighted. Considering of requirements above, we use a novel network embedding model called the "LINE" to predict for the temporal networks which preserves both the local and global network structures.

LINE [1] is a graph embedding method which solves problems in areas of link predicting, visualization and embedding very large information networks into low-dimensional vector spaces with local and global graph structures captured. A local structure in practice contains verticals and linked edges.

In practice, the information network is defined as $G = (V, E)$, where V is the set of vertices(each representing a data object) and E is the set of edges(link the vertices in the graph indicate relationship between data objects). Each edge $e \in E$ is an ordered pair $e = (u, v)$. Since some relationship are strong and some are not that important to model the network, edges are associated with weights w > 0 to describe the strength of the relations. If G is undirected, then the $(u, v) \equiv (v, u)$ and $w_{uv} \equiv w_{vu}$.

For each pair of vertices linked by an edge (u, v) (the weight on that edge can be represented by w_{ij} if edges are of different weights) , there is a local proximity between u and v and the proximity is defined 0 when there is no edge observed between u and v.

The LINE defines a distribution p(·,·) over the space V × V and its empirical probability $pe_{ij} = \frac{w_{ij}}{W}$ where $W = \sum_{(i,j)\in E} w_{ij}$ by equation:

$$p(v_i, v_j) = \frac{1}{1 + exp(-u_i^{\mathrm{T}} \cdot u_j)} \tag{1}$$

where $u_i \in R_d$ is the low-dimensional vector representation of vertex v_i and a v_j is the neighborhood of v_i linked by edge.

To preserve the proximity mentioned above, objective is carefully designed since optimizing it for a very large network is challenging. One approach that attracts attention in recent years is using the stochastic gradient descent for the optimization. However, directly deploying the stochastic gradient descent is problematic for this task because edges are weighted and the weights in this task present a high variance. As a consequence, these weights of the edges will be multiplied into the gradients, resulting in the explosion of the gradients and thus compromise the performance. To address this, LINE propose a novel edge-sampling method, which improves both the effectiveness and efficiency of the inference. We sample the edges with the probabilities proportional to their weights, and then treat the sampled edges as binary edges for model updating. The objective we use of this task according to LINE is to minimize the following objective function:

$$O = d(pe(\cdot, \cdot), p(\cdot, \cdot)) \tag{2}$$

There are various methods minimizing distance between distribution pe and distribution p, and KL-divergence is chosen.As the consequence, this d(·,·) in Equ(2) is replaced by the KL-divergence. After some constants omitted, the objective can finally represented

[1] https://www.dgl.ai/WSDM2022-Challenge/
[2] https://github.com/gybjyjpku/NodeInGraph

by:

$$O = \sum_{(i,j)\in E} w_{ij} \log p(v_i, v_j) \tag{3}$$

In this task, the scales of the gradients diverge and it is very hard to find a good learning rate because some nodes co-occur many times while some co-occur only a few times. In such networks. If a large learning rate according to the edges with small weights selected, the gradients on edges with large weights will explode while the gradients will become too small if we select the learning rate according to the edges with large weights.

As an consequence, as the amount of edge we use is large, there will be a edge sampling to train the model. In detail. we unfold a weighted edge into multiple binary edges and sample the binary edges to describe the original weighted edge as it probabilities proportional to the original edge weights..

Let $W = (w1, w2...w)$ denote the sequence of the weights of the edges. Firstly, we calculate the sum of weights defined as w_sum, and then we sample a random value within the range of $[0, w_sum]$ to see which interval the random value falls into. After a interval is chosen, the interval weight sum is used.

1.2 Model Inference

Based on the node embedding generated by LINE, the existence probability of each edge is defined as the dot product of the two corresponding node embeddings. Concretely, the probability of edge $< i, j >$, $e_{i,j}$ is scored as following:

$$e_{i,j} = u_i^{\mathrm{T}} \cdot u_j \tag{4}$$

Where u_i and u_j are the node embeddings of node i and j.

2 DATA PROCESSING

2.1 EMBEDDING NODES

We first generate node embedding using local structures represented by node pairs linked by weighted edges the task provides with the open source code [3].

The input file we generated contains lines in triples like:
source_id1 dest_id1 weight1
source_id2 dest_id2 weight2
source_id3 dest_id3 weight3
...

Note that, the weight is the occurrence frequency of the source node and the destination node.

2.2 CALCULATING SIMILARITY

After node embedding calculated, the graph is preserved and contains the information between neighbours(node pairs) by the shell command: ./line -train train_file -output embedding_file -size 128 -order 1 -negative 5 -samples 100 -rho 0.025 -threads 10 Then, we calculate the similarity based on node embedding of the head and tail node as the prediction score. (as shown in model_predict.py). The file model_predict.py is to calculate the dot similarity of two target nodes and give the outputs.

[3] https://github.com/tangjianpku/LINE

In summary, we generate the edge occurrence, train a LINE model, and calculate the node similarity, the overall pipeline is illustrated in Figure 1.

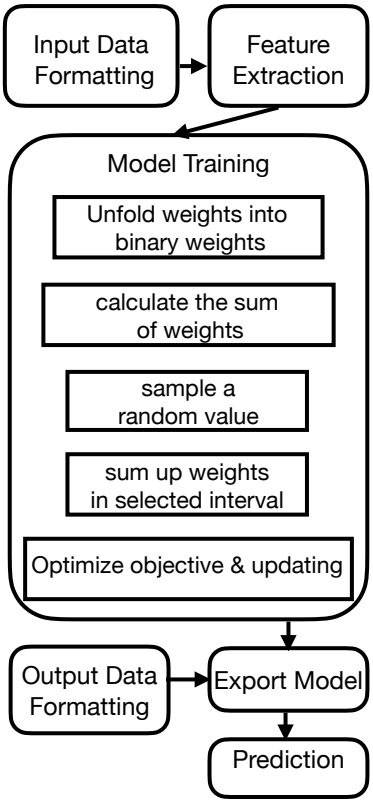

Figure 1: Overall pipeline of our model.

3 MODEL RESULTS

The competition employ the T-score as the evaluation metric, which is calculated based on AUC score [4]:

$$TScore = (AUC - mean(AUC))/std(AUC) * 0.1 + 0.5 \tag{5}$$

Where $mean(AUC)$ and $std(AUC)$ represents the mean and standard deviation of AUC of all participants. In addition, the score for ranking will be the average TScore value on Dataset A and B, that is, $(TScore_A + TScore_B)/2$.

As illustrated in Table 1, it can be learned that simply employing network embedding method to measure the node similarity could achieve satisfying results. Our method has a AUC score of 0.628 in Dataset A and 0.866 in Dataset B, and an average TScore of 0.585.

[4] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Table 1: Competition results.

Team Name	AUC (Dataset A)	AUC (Dataset B)	Average of T/100	Ranking
AntGraph	0.666001	0.901961	0.630737	1
nothing here	0.662482	0.906923	0.628942	2
NodeInGraph	0.627821	0.865567	0.585137	3
We can [mask]!	0.603621	0.898232	0.572372	4
IDEAS Lab UT	0.605264	0.873949	0.566849	5
SLi-Rec	0.583935	0.892547	0.552647	6

REFERENCES

[1] 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*. 1067–1077.