CS768 Learning with Graphs **2020-21 Sem I**

Lecture 7: September 13

Scribe: Kaveri Kale *Lecturer: Abir De*

1 Supervised Methods For Link Prediction

In this lecture we saw how to design supervised link predictor. Following are the main tasks in designing supervised link predictor:

- Model formulation
- Feature construction
- Defining loss function

Designing supervised link predictor: Consider a social network $G = (V, E)$ in which each edge $e = (u, v)E$ represents an interaction between u and v at a particular time t. Split them into training and testing set.

$$G_{Training} = (V, T) = (V, E_{Training})$$

and in Test-set for each node u we will have

$$\begin{bmatrix} TestEdges \\ TestNon-Edges \end{bmatrix}$$

In last lecture we already saw Adamic Adar, Preferential Attachment, Common Neighbor which gives the score for edge (u, v),

$$\begin{aligned} s(u,v) &= AA(u,v) \\ s(u,v) &= PA(u,v) \\ s(u,v) &= CN(u,v) \end{aligned}$$

In unsupervised approach, a proximity measure is chosen and used to rank node pairs in the network. The top ranked pairs are predicted to be linked. In the supervised approach, the link prediction is handled as a classification task and different values are used as predictor attributes by a learning algorithm.

So we will define new method that is supervised methods where we will combine score from above methods and use them as features for supervised learning.

$$\begin{aligned} s_W &= W_{AA}AA(u,v) + W_{CN}CN(u,v) + W_{PA}PA(u,v) \\ &= W^T \begin{bmatrix} AA(u,v) \\ CN(u,v) \\ PA(u,v) \end{bmatrix} \end{aligned} \tag{1}$$

For each pair (u, v) we can define model,

$$s_w(u,v) = W^T \begin{bmatrix} AA \\ CN \\ PA \end{bmatrix}_{(u,v)} \tag{2}$$

Here $s_W(u,v)$ is parameterized by W and it is stack value of,

$$W^T = \begin{bmatrix} W_{AA} \\ W_{CN} \\ W_{PA} \end{bmatrix} \tag{3}$$

Don't forget to add bias vector in W.

Training the model: Learn W by training on the available training graph $G_{Training}$.
Now, how should we train W?

Example: Let consider citation graph which have 2 nodes u and v. And nodes indicates the papers.

Define features for each node.

u :- $\begin{bmatrix} f_1 \\ f_2 \\ . \\ . \\ . \\ f_n \end{bmatrix}$ & v :- $\begin{bmatrix} f_1 \\ f_2 \\ . \\ . \\ . \\ f_n \end{bmatrix}$

where features$f_1, f_2, ...f_n$ could be tf-idf of keywords present in paper.

Here we can formulate the model,

$$\begin{aligned} s_w(u,v) &= Similarity - Index(W.f_u, W.f_v) \\ &\quad or \\ s_w(u,v) &= e^{-\alpha||Wf_u - Wf_v||^2} \end{aligned} \tag{4}$$

Here Similarity-Index is some kind of similarity metric, e.g cosine similarity

$$cosine - sim(X,Y) = \frac{X.Y}{||X||_2||Y||_2} \tag{5}$$

Now define model of scores between u & v.

Given a training graph $G_{Training} = (V, E_{Training})$ we can model $s_W(u,v)$ as simple linear predictor.

$$\begin{aligned} s_W(u,v) &= W^T f_{u,v} \\ &= W^T \begin{bmatrix} AA(u,v) \\ CN(u,v) \\ PA(u,v) \end{bmatrix} \end{aligned} \tag{6}$$

In addition we can define per node features,

$$s_W(u,v) = (Wf_u, Wf_v) \tag{7}$$

For more complicated model simply use complex structure or dependency between two nodes using more "expressive" predictors.

$$\begin{aligned} s_\theta(u,v) &= NN_\theta(\{f_u\}, \{f_{u,v}\}) \\ &= FF_\theta(\{f_u\}, \{f_{u,v}\}) \end{aligned} \tag{8}$$

Lets consider $s_W(u,v)$ is linear model and $s_\theta(u,v)$is non-linear model.

We have formulated models, now how to train these models?
To train we will use supervised approach Support Vector Machine(SVM).

$$\begin{aligned}(u,v) \in E \; s_W(u,v) \geq 1 - \zeta_{u,v} \\ (u,v) \notin E \; s_W(u,v) \leq 1 - \zeta_{u,v}\end{aligned} \tag{9}$$

Loss function for SVM can be defined as,

$$L = \frac{1}{2}\lambda ||W||^2 + (1 - s_W(u,v).Y_{(u,v)}) \tag{10}$$

where Y is indicator of edge(+1) or non-edge(-1). This loss is called hinge loss.

If we combine above two constraint we will get following form,

$$Y_{(u,v)} s_W(u,v) \geq 1 - \zeta_{u,v} \tag{11}$$

Here we are trying to construct classifier which will classify edges and non edges.

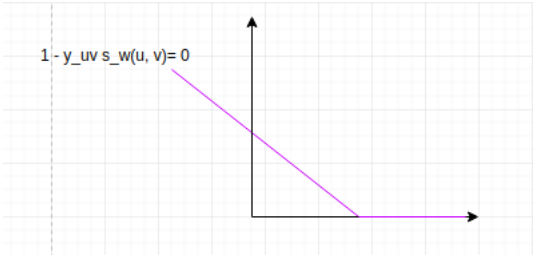

Figure 1: Hinge loss

Minimize the loss

$$\sum_{(u,v) \in Training} (1 - Y_{(u,v)} s_W(u,v))_+ = \sum_{(u,v) \in E_{Training}} (1 - s_W(u,v))_+ + \sum_{(u,v) \in NE_G / NE_{Test-set}} (1 + s_W(u,v))_+ \tag{12}$$

Once training is done test the model using test-set which contains test-edges and test-non-edges.

Caveat : In most graphs

$$\begin{aligned}|E| << |V| \times |V| \\ |E| << |NE|\end{aligned} \tag{13}$$

Data is highly imbalanced. Therefore learning becomes insensitive.
In such cases we don't compute loss by above given loss function. Rather we compute $s_W(u,v)$ for edges with $s_W(u,v)$ for non-edges (NE) in training-set.

Consider $(u,v) \in E_{Training}$ and $(u^1, v^1) \notin E_{Training}$,

$$\begin{aligned}s_W(u,v) - s_W(u^1,v^1) > 0 \\ s_W(u,v) > s_W(u^1,v^1) + \Delta\end{aligned} \tag{14}$$

Loss is greater than 0 iff $s_W(u,v) < s_W(u^1,v^1) + \Delta$