CS 768 Learning with Graphs **Autumn 2020**

Lecture 2

Instructor: Prof Abir De *Scribe: Chitrank Gupta, 170050024*

Link Prediction (LP)

The Fundamental Problem

Given an evolution sequence of a graph $\mathcal{G}$, $\mathcal{S}_{\mathcal{G}} = \{ \mathcal{G}_t$ for t from 1 to T$\}$ where each of $\mathcal{G}_t$ is a snapshot in the evaluation of the graph, the link prediction problem is to accurately predict what edges are going to be added in the future[1].

Our approach to tackle this problem in this course will be roughly based on

1 Understanding statistical properties of the graph based on $\mathcal{S}_{\mathcal{G}}$

2 Understanding the domain of the graph $\mathcal{G}$

Understanding the challenge

1 **Independence assumption of data samples**. Many standard supervised ML models, assume that the prediction on one sample of the data is independent of the prediction on other samples. However, in the case of graphs, the data under consideration – the edges – no longer remain independent. This is because, once we discover that an edge exists, then it can affect possibly effect the chances that a nearby edge will also exist.

2 **How to get train-test split when $\mathcal{S}_{\mathcal{G}}$ is unavailable?** For most of the real-life datasets, the dynamics of a dynamic graph are not available. And so we are required to work with a single snapshot of the graph (henceforth called $\mathcal{G}$). Standard ML algorithms require a test split to evaluate and so an ML algorithm for LP would require a test split from which we can collect edges which are not present in the train

[1] we assume that edges don't disappear in graph $\mathcal{G}$ during its evolution.

split and apply our LP model trained on the train split to find out how well is it able to predict node-pairs in test split as edges.
Had $\mathcal{S}_\mathcal{G}$ been available, one could have constructed train split from the first k% (say 80%) of the $\mathcal{S}_\mathcal{G}$ and the last 100-k% would have been kept as the test split. In this way we could have accurately captured the evolution of the graph $\mathcal{G}$ while training our model. But because we only have $\mathcal{G}$, one has to collect node pairs in ($\mathcal{V}$ x $\mathcal{V}$) for test split $\mathcal{E}_{test}$ (and also remove all the edges in the test split from the graph $\mathcal{G}$) and then train and evaluate. The problem with this $\mathcal{E}_{test}$ is that many of the edges in $\mathcal{E}_{test}$ could have actually been introduced in the early stages of $\mathcal{G}$. This way we are trying to train our model on the later stages of $\mathcal{G}$ and evaluate on its earlier stages, hence inaccurately learning the evolution dynamics of $\mathcal{G}$.
One way to minimize this problem is to create several train-test splits and learn-evaluate your model on them separately.

Prediction of Links

Hard Prediction

In hard prediction, the LP algorithm directly predicts whether a node pair in $\mathcal{E}_{test}$ is going to be an edge or not. To evaluate such a model we calculate total number of right (or wrong) predictions.

Soft Prediction

In soft prediction, the LP algorithm assigns a probability or score corresponding to the edge-ness of the node pair. Most of the times, it is better to work with soft prediction because this way we can also know about the uncertainty that a model has in prediction of edges.
With soft predictions at our disposal, how can we actually predict which node-pair is actually going to be an edge? This is somewhat a difficult question than answering suppose, predict some K node-pairs as edges. This can be easily answered by selecting the top K node-pairs sorted (in decreasing order) by their soft scores.
Coming back to the question of hard assigning a node pair as edge or non-edge, we can apply a similar approach by using a hyper-parameter - δ (threshold). If the score of a node-pair exceeds δ then we predict that node-pair as edge otherwise not.

Evaluation on Aggregated list vs Per-node list

Suppose you have to report some K link predictions or you want to tune your δ threshold (which somewhat reduces to finding a good K), or you want to recommend 20 movies on

Netflix to a user, how would one go about applying an evaluation protocol on an LP algorithm?

If one evaluates the LP algorithm by seeing how many of the top K link predictions are actually correct, then this could be a misleading evaluation protocol. The reason is that most of those top-most correct predictions could actually be related to a single node or few. So the result is that the LP algorithm could be performing well in some small part of the graph but poorly in other regions. To mitigate this issue, we instead have separate $\mathcal{E}_{test}$ for each node in the graph and evaluate the model on each of these lists separately.

Sampling Training(or Test) Data from a graph

The graph in reality could have non-uniform density, i.e., there could be nodes linked with many other nodes, while some which are poorly connected with other nodes. Suppose you have to create a $\mathcal{E}_{test}$. If you randomly sample edges across the graph, then there could be non-uniformity in ratio of the number of true edges and number of no-edges associated with a node. To avoid this, we simply sample the edges and non-edges for each node separately.