CS 768 Learning with Graphs - Lecture 4

Yateesh Agrawal (170050005)

September 2020

1 Introduction

Last week, the problems and challenges of LP problems were discussed. First was how to sample the graph and split it into the test and training dataset, for which two ways of sampling, Aggregated Sampling and Query/Node Specific Sampling, were explained. Then evaluation problem of an LP algorithm was discussed. The first step to solve it was to construct a rank list of edges and non-edges using the LP algorithm provided. The rank list can be both Aggregated or Query/Node Specific. In either case, we still require an evaluation metric to compare a given rank list with the ground truth values. Therefore, in this lecture, we will design a few kinds of ranking loss.

2 Evaluation Metrics

2.1 AUC

In the rank list given to us, we know that score(1,2) > score(2,3) > score(5,6) so we expect that (1,2) is an edge (e) and as we go down the likelihood of a pair being an edge decreases, thus if (2,3) is a non-edge (ne) then (5,6) is also non-edge. If this is not the case then ranking accuracy should decrease.

$$\begin{bmatrix}(1,2)\\(2,3)\\(5,6)\\\cdot\\\cdot\end{bmatrix} \text{---}> \begin{bmatrix}1\\-1\\1\\\cdot\\\cdot\end{bmatrix}$$

Predicted Rank List and it's Ground Truth

After matching the rank list to the corresponding ground truth (which we get from data), we want this ground truth list to be sorted in the correct order. So we create a metric as follows, for every edge non-edges pair where score(e) > score(ne), we give algorithm 1 reward. For every edge non-edges pair where score(e) < score(ne), we give algorithm -1 reward. In cases of s(e) = s(ne), we give only 0.5 reward. Then we divide it by maximum possible score i.e. #[e]*#[ne].

$$\text{AUC} = \frac{\sum_{\text{e,ne}} 1[\text{s(e)}>\text{s(ne)}] - \sum_{\text{e,ne}} 1[\text{s(e)}<\text{s(ne)}] + 0.5(\sum_{\text{e,ne}} 1[\text{s(e)}=\text{s(ne)}])}{\#[\text{e}]*\#[\text{ne}]}$$

For mostly practical graphs where #[ne]>>>#[e], we see that AUC is very forgiving. Eg:- a very bad rank list with 10 e and 100 ne, arranged in such a way that we first see 10 ne then 10 e and

then the remaining 90 ne, gives AUC of (90*10-10*10)/10*100 = 80%. This is because of large number of ne.

2.2 MAP

Given a ground truth list, we define Precision@k as no. of edges retrieved up to k divided by k. So in our ground truth example on previous page, Precision@1 = 1, Precision@2 = 1/2, Precision@3 = 2/3 and so on. Average precision is then calculated over k where there is an edge (not over the entire list). For per node rank list, we calculate Mean Average Precision (MAP) by taking the average of AvP across all rank lists.

$$\text{Pr@k} = \frac{\#\text{Edges retrieved up to k}}{\text{k}}$$

$$\text{AvP} = \frac{\sum_{\text{k:(e is in k)}} \text{Pr@k}}{\text{Total no. of edges}}$$

$$\text{MAP} = \frac{\sum_{\text{q}\epsilon\text{V}} \text{AvP}_{\text{L(q)}}}{|\text{V}|}$$

In a list with just 1 edge on the top we have both AUC and AvP = 1 but when this edge goes on the second position AUC drops to 0.8 but AvP drops to 0.5, Thus we can say, AvP is harsher measure than AUC. so it better captures the goodness of ranking being sensitive to perturbations in ordering. So it is used more often than AUC.

2.3 MRR

It is a simple metric defined by one upon rank of first edge. It is also a harsh measure as to when the rank of the first edge drops from 1 to 2, MRR scores drop 50% like MAP.

$$\text{MRR} = \frac{1}{\text{Topmost edge position in the rank list}}$$

3 Further Reading

- Today we discussed 3 metrics for ranking evaluation namely AUC, MAP and MRR from which only MAP and MRR are sensitive to order.
- Homework :- Full form of AUC is Area under the curve as it has a definition corresponding to area under some curve. But we defined it in a different way, so find the other definition and prove that these two definitions are indeed equivalent.
- In next class, different LP algorithms such as Adamic Adar Heuristics, Common Neighbor Heuristics and Preferential Attachment Heuristics will be discussed.