Supervised Link Prediction with Path Scores

Wanzi Zhou
Stanford University
wanziz@stanford.edu

Yangxin Zhong
Stanford University
yangxin@stanford.edu

Yang Yuan
Stanford University
yyuan16@stanford.edu

1. Introduction

Link prediction has long been a classic and fundamental topic in network analysis. The problem is basically about given a network, how can we infer the possible links that will be formed in the future, or which links are missing in the current record. Link prediction is important not only for understanding the dynamics of networks, but also for completing current networks. It has many applications in different areas, from automatic recommendation systems in social networks and e-commerce, to finding the interactions between cells or proteins in bioinformatics. [5]

To predict potential or missing links in the network, attributes of nodes, edges and network topology give some clues. And machine learning (ML) based methods such as [1] make it easier to incorporate different measurements into one model. Supervised random walks (SRW) [2] combines ML methods with the graph algorithm PageRank to get a higher performance. In this project, we leverage the idea of ML methods and SRW to propose a new method for link prediction. Similar to SRW, our method also utilizes the graph information (the paths between two nodes) to train a link predicting model. But unlike SRW, it does not involve the high-cost and unscalable PageRank algorithm/update. Instead, we try to use a simpler way to extract features for each path and calculate a potential link score for two nodes directly. Experiments show that our method can train much faster than SRW with much smaller memory space, and meanwhile gets a similar performance as SRW.

We review some related work in Section 2. Our model is elaborated in Section 3. The progress until this point including data preprocessing and preliminary experiments of baseline models are introduced in Section 4 and 5. Finally, we discuss future work and challenges in Section 6.

2. Related Work

In recent years, there are more and more academic and industrial studies on the link prediction problem [5]. About 10 years ago, [4] proposed proximity-score based methods. The main idea is to assign a proximity score for each pair of nodes that are not currently connected, and rank those potential edges in decreasing order of the score. Pairs with highest scores are considered most likely to be linked. The proximity score can be different measurements on graph distance, node neighborhoods and the ensemble of all paths. This paper gave a clear overview of the link prediction problem. Some models outperformed random prediction a lot. The proximity scores deployed in those models are thus proved to be good indicators for link prediction. But the method is too inflexible to incorporate more than one score.

With the arise of machine learning, [1] formulated the link prediction problem as a supervised classification task. They extracted different features including proximity features, aggregated features and topological features. Based on the features, they trained different types of classifiers on training set, including decision tree, SVM, KNN, etc. and evaluated different models on test set. It turned out decision tree and SVM model performed the best, and some features are more helpful than others. The supervised learning method takes in multiple features in one model. However, one drawback of this approach is that they left alone the detailed structure of network. Although they used some network features and statistics, it still lost a large amount of information about the network.

In 2011, [2] used supervised random walk to perform link prediction. It formulated a supervised learning task to learn a function that assigns strengths to edges in the network such that random walk from a source node is biased according to these strengths. The learning procedure is similar to the iterative power iteration algorithm with derivatives to update the weights in the model. This model naturally combines both network structure information and attributes of nodes and edges (features explored in [4], [1]) inside the network to make reliable predictions. However, it suffers from high memory and computational complexity due to the iterative update on the whole transition matrix.

3. Approach

3.1. Problem Formulation

The link prediction task can be formulated as below: given a graph $G = (V, E)$, where V is the node set and

E is the edge set, let $M = \{(i,j) : i,j \in V, (i,j) \notin E\}$, try to find a function $F : M \to \{0,1\}$ s.t. $F(i,j) = 1$ if there is a potential edge from i to j; $F(i,j) = 0$, otherwise.

One possible application: $G = (V,E)$ is a snapshot of an evolving graph at time t, and $G' = (V,E')$ is the snapshot of the same evolving graph at time t' ($t' > t$); then we can use $F(i,j)$ to predict if $(i,j) \in E'$.

Instead of predicting for all node pairs in M, the work in [2] only predicts for a subset of M. More concretely, for each node $i \in V$, they defined a candidate set $T_i = \{j \in V : (i,j) \in E, distance(i,j) = 2\}$, where $distance(i,j)$ is the length of shortest path between i and j in G. Any $j \in T_i$ is called positive candidate if $(i,j) \in E'$; otherwise, it is called negative candidate. Then for each node i, they only predicts whether there is a potential edge from i to j, where $j \in T_i$. They argued that it's reasonable to only consider those 2-hops away nodes as candidates because they found that 92% new edges formed in the future close a path of length two in a real-world network.

We will follow a similar way of evaluation as they did. That is, we will only predict for those 2-hops away candidates and thus form our dataset as $C = \{(i,T_i) : i \in V\}$. We will split it evenly into training set D and test set H. More details of data processing can be found in Section 4.

3.2. Supervised Link Prediction with Path Scores

Inspired by Supervised Random Walks (SRW) [2], we develop an algorithm, Supervised Link Prediction with Path Scores (SLPPS), which utilizes the machine learning technique and also the network structure to solve the link prediction problem. One issue of SRW is that in each learning step, it requires many iterations of matrix multiplication to get the PageRank vector and the gradients to learn the edge weight model, which makes the algorithm not scalable. In our method, we get rid of the random walks algorithm and calculate a similar score as in SRW. Our approach is shown in Algorithm 1.

In our algorithm, the path length limit d is useful when we try to find all the paths from node s to node t, i.e. we only add those paths with length $\leq d$ to the path set. Note that each path we found above does not contain any loop. Then for each source node s and its candidate node t, we calculate the visit score q_{st}, which can be interpreted as the likelihood score that s will form an edge with t in the future. Note that q_{st} is the sum of path scores q_l, where l is any possible path we found. Path scores are calculated by a function shown in Algorithm 2.

To calculate the score of a path l, for each edge (u,v) in l, we extract the feature vector x_{uv} for node pair (u,v). And then we average the feature vectors for all (u,v) to get feature vector x_l of path l. Finally, we input the feature vector x_l to a two-layer neural network to get the path score q_l for l.

Algorithm 1 Training of SLPPS

Input: graph G, training set D, path length limit d, learning rate α, regularization weight λ
, loss margin b **Output:** trained parameter θ

θ = random parameters for path weight model
repeat
  for (source node s, candidate set T_s) in D **do**
    $Q_s = \{\}$
    for candidate node t in T_s **do**
      paths set L = FindPaths(G, s, t, d)
      visit score $q_{st} = 0$
      for path l in L **do**
        $q_l = \text{PathScore}(G, l, \theta)$
        $q_{st} = q_{st} + q_l$
      $Q_s[t] = q_{st}$
    loss $\mathcal{L} = \lambda||\theta||^2$
    for positive candidate node t_p in T_s **do**
      for negative candidate node t_n in T_s **do**
        $q_p = Q_s[t_p], \quad q_n = Q_s[t_n]$
        $\mathcal{L} = \mathcal{L} + I[q_p < q_n + b] \cdot (q_p - q_n - b)^2$
    $\theta = \theta - \alpha \cdot \frac{d\mathcal{L}}{d\theta}$
until convergence
return θ

Algorithm 2 Path Score Function in SLPPS

Input: graph G, path l, parameter θ
Output: path score q_l

$s = l$[start], $t = l$[end], $x_l = \vec{0}$
for edge (u,v) in l **do**
  $x_{uv} = \text{ExtractFeatureVector}(G, u, v)$
  $x_l = x_l + x_{uv}$
$x_l = x_l$ / length(l)
$q_l = \text{TwoLayerNN}(x_l, \theta)$
return q_l

We learn the parameter θ by gradient descent. Our loss function $\mathcal{L}$ consists of two part: (1) the regularization part $\lambda||\theta||^2$; (2) the hinge-like loss $I[q_p < q_n + b] \cdot (q_p - q_n - b)^2$ for all pairs of positive and negative candidate, where q_p and q_n are the visit scores of the positive and negative candidates respectively and $I[q_p < q_n + b]$ is the indicator function. Then we apply the gradient descent update for parameter θ, which is $\theta = \theta - \alpha \cdot \frac{d\mathcal{L}}{d\theta}$.

The gradient $\frac{d\mathcal{L}}{d\theta}$ is not complex to compute since the loss function $\mathcal{L}$ consists of only addition, multiplication, indicator, square and inner product. The calculation can be done by backpropagation, which is commonly used to calculate gradient in a neural net.

Actually in our algorithm, it's convenient to change the loss and the path score function. Currently we average the

feature vectors of all edges in the path to get the path feature vector. And then apply a two-layer neural net to it. But we can also use other smarter way to utilize those edge feature vectors. For example, we can use a Bi-LSTM to encode the ordering of edges in path. Or we can use a neural net model to calculate edge scores and multiply them to get the path score. We also have other choices of loss function such as Huber loss and WMW loss mentioned in [2]. We try some options above such as WMW loss in the experiments but will leave others to the future work.

4. Data Preprocessing

4.1. Original Dataset

The paper collaboration network is broadly used in link prediction research, and the year of the paper can be naturally perceived as the edge formation time. In our project, we will use the DBPL and ACM Citations Network dataset provided by cs224w course staff. In particular, we are using the DBLP-Citation-network V10 dataset which contains 3,079,007 papers and 25,166,994 citation relationships. Some statistics about the original dataset is shown in table 1. Per 1,000,000 papers are organized into a file. In each file, each line represents a paper, which is in JSON schema. By processing the json file we can load the database.

number of nodes	3,079,007
number of edges	25,166,994
year range	1940-2018

Table 1. Statistics of the original dataset

4.2. Processing Dataset

4.2.1 Extracting Dataset

As shown above, the size of the dataset is huge. To accommodate our needs for the project, among all the data from year 1940 to year 2018, we first chose papers from 1990 to 2004 and filtered out all other papers. We split the dataset into two parts: papers from 1990-1999 and from 2000-2004. The first stage of the dataset (1990-1999) is to provide the training graph where we will be computing all the features from. The second stage (2000-2004) is to provide new-link labels, i.e, we will be looking at which new links a node will make in the second stage.

We have adjusted our dataset several times. The main reason is that the time per training cycle for the machine learning algorithms we have tried is very slow, through our estimation, if we were to use the current dataset (which includes authors from papers between 1990 to 1999), the main memory needed for training Supervised Random Walk (SRW) would be no less than 50 G and the training period, thus to make the project realizable, we further compress our dataset. We choose our graph nodes to be authors from the paper only in year 1990, then, we build edges of the graph based on all the collaborations between these authors over the years between 1990 to 1999, i.e, while choosing nodes only from year 1990, we still build edges upon a ten-year period. In this way, we can effectively shrink the size of the graph, while still include the features that form during a rather long time period. Then we filtered out all nodes that have degree less than 3, as there is less meaning to predict new links for authors that do not tend to cooperate with other researchers. The statistics of training graph is shown in Table 2

number of nodes	9255
number of edges	28001

Table 2. Statistics of Training Graph

4.2.2 Choosing Source Nodes and Candidates

Similar to the work in [2], we also use the concept of active nodes or source nodes in evaluation. More concretely, **source nodes** are nodes for which we will focus on predicting new links (we do not predict links for the rest of the nodes, but they help forming the entire network and thus are important too). Then for each source node, we define **destination** or **target** nodes to be nodes which it made new links to in the second stage (2000-2004) of the dataset.

Inspired by [2], we define our source nodes to be nodes from the first stage (nodes from paper in year 1990 building edges throughout year 1990 to 1999) which made at least 3 two-hops-away new links (closing triangle) in the second stage (2000 - 2004). For every source node, we further constrain our destination nodes to be nodes which 1) form new link with the source node, 2) and are originally two-hops away from the source nodes. Similarly, we only consider two-hops-away candidates nodes, i.e. we only run our algorithms on these source nodes and only predict new links between a source node and nodes that are two-hops away from it. We do these processing for two reasons. First, in real social networks people tend to make friends with friends and close a triad, so it is intuitive to choose candidates from nodes that are 2-hops away. Second, we need enough destination nodes for a source node as labels. Thus, we only choose nodes that have made at least 3 new links with two-hop-away nodes as source nodes. After applying these processing, we obtain 278 source nodes in total.

4.2.3 Training and Testing Dataset

Inspired by [2], we randomly choose 60% of the source nodes as our train set, 20% as the validation set and 20%

as our test set.

4.2.4 Feature Set

Learning from [1] and [4] collectively, for every edge in the training graph, we have extracted the following features:

Number of Collaborations Intuitively, if two authors collaborate more papers, the edge between them should have more strength.

Sum of papers It is more likely that an author will collaborate with others if he writes more papers.

Sum of keyword count It is shown that researchers that have a wide range of interests or those who work on interdisciplinary research usually use more keywords [1].

Keyword match count One can expect that the higher the keyword match count between two authors, the more possible that they are to work in similar fields and the more likely that they will cooperate with each other.

Keyword jaccard similarity Another feature that is used in terms of keywords besides keyword match count.

Sum of common neighbors Now we use $T(x)$ to represent the neighbor set of node x. For node x and y, this will be $|T(x) \cap T(y)|$.

Sum of neighbors $|T(x)| + |T(y)|$.

Neighbor jaccard coefficient $\frac{T(x) \cap T(y)}{|T(x) \cup T(y)|}$

Adamic/Adar $\sum_{z \in T(x) \cap T(y)} \frac{1}{\log |T(z)|}$ [4] This feature weakens the weight of common neighbors that have many neighbors themselves. Such people cooperate with many other authors so that having common neighbors of such researchers does not strongly imply the possible future cooperation between x and y as compared to having a common neighbor who has rarely cooperated with other researchers.

Sum of log secondary neighbor count If a person is connected to a node that has very high connectivity, chances are that she/he might co-author with a distant node through a middle node [1]. This is represented as $\sum_{z \in T(x) \cap T(y)} T(z)$

Preferential attachment $T(x) \cdot T(y)$

Sum of Clustering index People with higher clustering index tend to cooperate with others more.

PageRank sum $r(x) + r(y)$ While computing vanilla PageRank we only get one static PageRank vector for the whole graph, computing PageRank with restarts for every node will be very costly, thus for now we will use the general PageRank vector. The PageRank with restarts is considered in the future if needed.

All features are normalized to zero-mean and unit-standard-deviation before fed into SRW and our method.

4.2.5 Small Dataset Statistics

After all the preprocessing procedures, we list the statistics of our final dataset in Table 3.

N	E	S	$\overline{D}$	$\overline{C}$	$\overline{D}/\overline{C}$
9255	28001	278	4.6187	127.7410	0.0362

Table 3. **Dataset Statistics.** N: number of nodes, E: number of edges, S: number of source nodes, $\overline{C}$: avg. number of destination nodes $\overline{D}$: avg. number of candidates for every source nodes

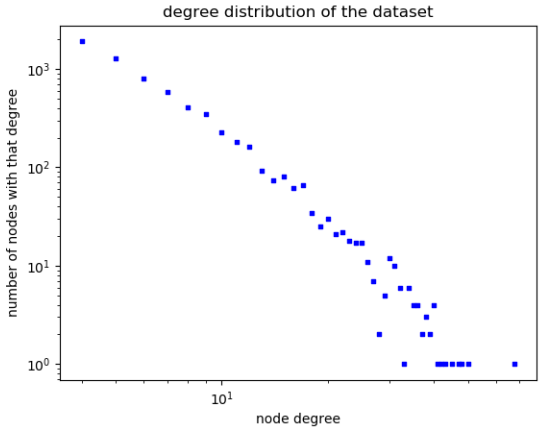

Figure 1. Degree Distribution of the Dataset

Degree distribution of the dataset is shown in Figure 1. As we can see, the degree distribution is a natural power-law distribution. Most of the nodes have degree below 10.

For the source nodes we selected, we also compute distribution of the number of candidate nodes (2-hops-away nodes) and number of destination nodes (positive candidate nodes) in Figure 2 and Figure 3, respectively. As we can see, most of source nodes have less than 400 candidate nodes and less than 10 destination nodes.

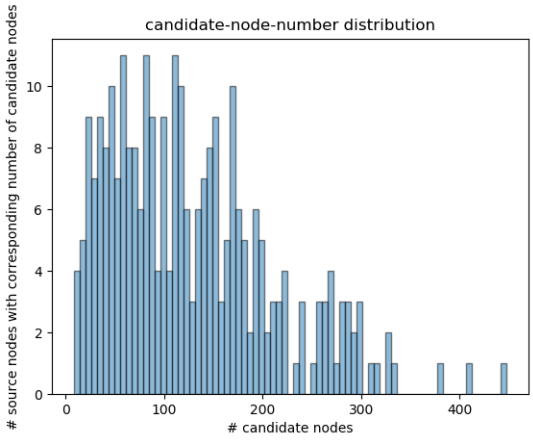

Figure 2. Degree Distribution of candidate (2-hops-away) nodes

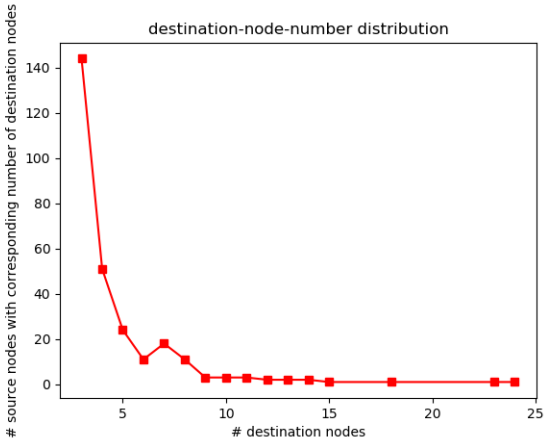

Figure 3. Degree Distribution of destination nodes

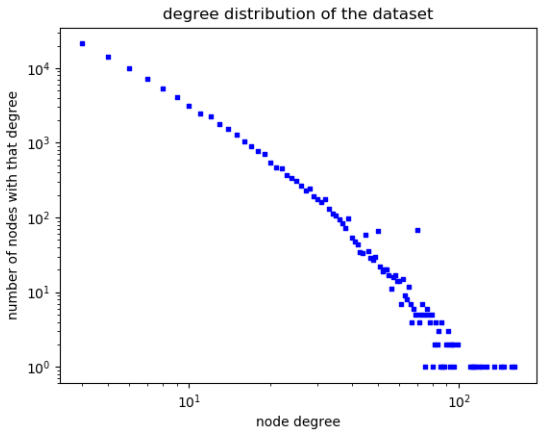

Figure 4. Degree Distribution of the Big Dataset

4.2.6 Big Dataset Statistics

While we have obtained a small dataset with 278 source nodes as above, we would still test methods other than Supervised Random Walk (SRW) on the big dataset, where we include all the nodes from papers in 1990 to 1999 and include all the collaborations of these nodes between 1990 and 1999. We still use year 2000 to 2004 as the second stage.We list the statistics of our final dataset in Table 4.

N	E	S	$\overline{D}$	$\overline{C}$	$\overline{D}/\overline{C}$
116390	411159	3416	4.4300	166.7951	0.0266

Table 4. **Dataset Statistics.** N: number of nodes, E: number of edges, S: number of source nodes, $\overline{C}$: avg. number of destination nodes $\overline{D}$: avg. number of candidates for every source nodes

Degree distribution of the dataset is shown in Figure 4. Distribution of number of candidate nodes is shown in Figure 5. Distribution of number of destination nodes is shown in Figure 6.

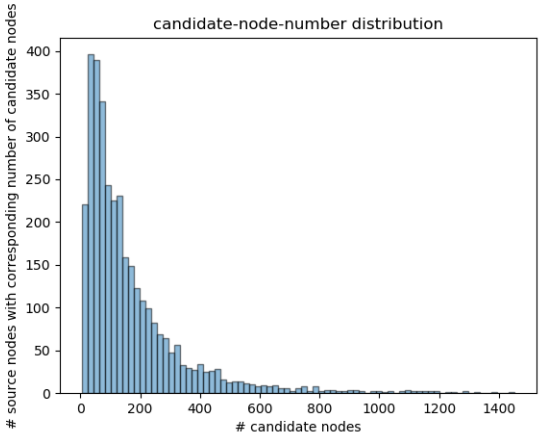

Figure 5. Degree Distribution of candidate (2-hops-away) nodes in the big dataset

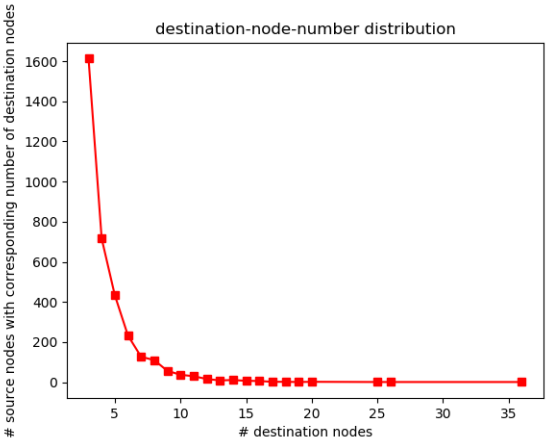

Figure 6. Degree Distribution of destination nodes in the big dataset

5. Experiments and Evaluation

5.1. Experiment Setup and Evaluation Method

We will evaluate our algorithms on stage 2 of the dataset (2000-2004). For every source node in test set, we compute the proximity/PageRank/visit scores (corresponding to different algorithms) between source node and all candidate nodes (2-hops away nodes). Two measurements are used for evaluation. We fist compute top-20 precision, where for each source node, we choose the top 20 candidate nodes with the highest scores as our predictions and compute the proportion of destination nodes among them. Second, we vary the number of predictions from 0 to maximum number of candidate nodes, draw the ROC curve and compute AUC for each method. For the ROC curve, the x-axis represents

the fraction negatives (fall-out), and y-axis represents the fraction positives (recall), illustrated as below

$$\text{recall} = \frac{|\{\text{positive candidate nodes}\} \cap \{\text{predicted nodes}\}|}{|\{\text{positive candidate nodes}\}|}$$

$$\text{fall-out} = \frac{|\{\text{negative candidate nodes}\} \cap \{\text{predicted nodes}\}|}{|\{\text{negative candidate nodes}\}|}$$

And AUC will be the area that is covered under the ROC curve. Usually, an area of 1 represents a perfect test; an area of 0.5 represents a poorest test.

5.2. Algorithms and Settings

Proximity-based method [4] is implemented as our baseline. In the experiment, we try two different measurements as the proximity score: 1) number of common neighbors, and 2) Adamic/Adar. According to [4], Adamic/Adar yields the best performance in all typical proximity measures for predicting.

We implement Supervised Random Walks (SRW) [2] based on the infrastructure of a previous CS224w project found on GitHub [3]. We make several optimization and adjustments to the existing code base, specifically by using several vectorization tricks in numpy and managing the memory better to fit our need for training more data than the code was used to. Although improvements can still be applied, our modified version is much more efficient and less space-consuming than the original one. We use logistic edge strength as suggested in [2]. We tried hinge square loss and WMW loss mentioned in [2] but only report results of the better one (square loss). For optimizer, we tried L-BFGS algorithm and batch gradient descent and found batch gradient descent perform better. Features in training are listed in Section 4.2.4. Our best result comes from setting the gradient descent learning rate to be around 0.1, regularization parameter λ to be 1, margin in the squared loss function to be 0.01, and teleport rate α to be 0.3.

We implement our method, Supervised Link Prediction with Path Scores (SLPPS), using Snap.py and PyTorch package. The algorithm is shown in Alg 1 and the features we use are listed in Section 4.2.4. The path score in Alg 2 comes from a two-layer neural network, where the first layer has 128 hidden units and the second layer has 64 hidden units. We tried hinge square loss and WMW loss in [2] as well and report the performance of both settings. We tried gradient descent and Adam optimizer and found the latter yield a higher performance. For small dataset, we found a mini-batch update of size 5 yields the best performance, and a mini-batch update of size 10 is better for big dataset. We also vary the learning rate from 0.01 to 0.001 and change the path limit d in Alg 1 to analyze how they impact the training process and test performance. The regularization parameter $\lambda = 1$, and the margin in the squared loss is 0.5.

5.3. Experiment Results on Small Dataset

5.3.1 Training Loss

The training loss of SLPPS and SRW on small dataset are shown in Figure 7 and Figure 8, respectively. Note that the magnitude of two losses are different because the visit score in SLPPS and PageRank score in SRW have different scales. But we can see that both losses are decreasing as training time goes larger, which indicates the success of training of both algorithms. We also notice that the curve of SRW is smoother than SLPPS. This is because we use a batch update in SRW and a mini-batch update in SLPPS. Usually the batch update will have a smoother gradient change among different iterations than mini-batch because it averages over more data points. One last interesting thing is that there exists some extremely sharp spikes in the training curve of SLPPS. This is because we use an Adam optimizer in SLPPS. It is a known problem of Adam that the training curve will have spikes when the gradient occasionally increases after being around zero for a long time.

Figure 7. Training loss vs. time of SLPPS on small dataset

Figure 8. Training loss vs. time of SRW on small dataset

5.3.2 Training and Validation Performance Curves

We also analyze how the predicting performance change as training time goes larger. More concretely, we evaluate SLPPS and SRW with top-20 precision and AUC throughout the training process on both training set and validation set. The results are shown in Figure 9, 10, 11, and 12.

For SLPPS, we notice that the curves of both top-20 precision and AUC on training set keeps increasing during the whole training process in general. There exists some spikes due to Adam optimizer as we explained above, but the predicting performance becomes better and better in general. This indicates that minimizing the loss function can indeed improve the prediction. However, the curves of both measures on validation set do not increase all the time. They both first increased rapidly and then wiggled around a steady value. This issue is called overfitting in machine learning. And the overfitting here may come from not enough data in small dataset or a small regularization parameter λ. Actually we have increased λ from 0.01 up to 1 but the overfitting issue still exists. And if we increase λ up to 10, then an underfitting issue will occur instead. If time allowed, we would tweak the parameter more carefully, but we believe not enough data is probably the cause instead.

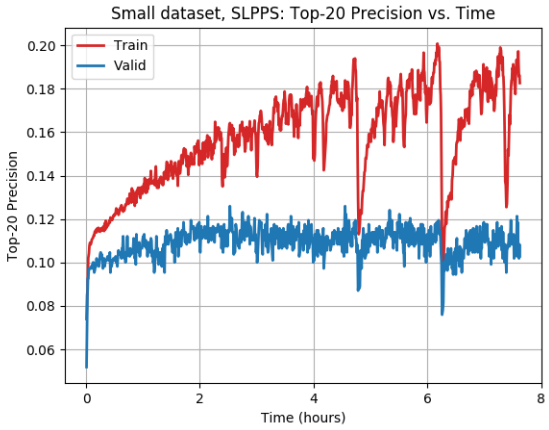

Figure 9. Top-20 Precision vs. time of SLPPS on small dataset

For SRW, although the training loss in Figure 8 decreases all the time, its training performance in Figure 10 and 12 plateaued and even decreased in the second half of training. This indicates that minimizing the loss function cannot improve the prediction in the second half of training. To fix this issue, we may change the loss function (e.g. tweak the margin parameter in square loss more carefully or use another loss function), or train for a even longer time. But since SRW has a high time complexity, more careful parameter tweaking and longer training are not allowed under the limited time. Note that the top-20 precision on validation set became better in the second half of training, which

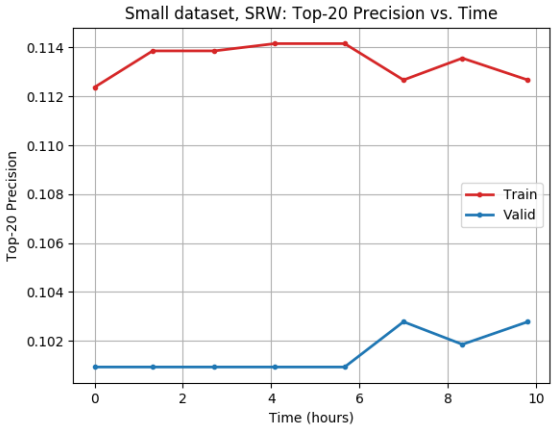

Figure 10. Top-20 Precision vs. time of SRW on small dataset

means that the model actually learnt/generalized something useful for prediction from the training data. Another interesting thing is that AUC on validation set is higher than AUC on training set. This indicates that on validation set, fewer positive candidates are in the top-20 predictions but many of them have high rankings (maybe top-30 or top-40). On validation set, as the top-20 precision goes higher, the AUC decreases instead. This might indicate that a better loss function is required to improve both measures at the same time.

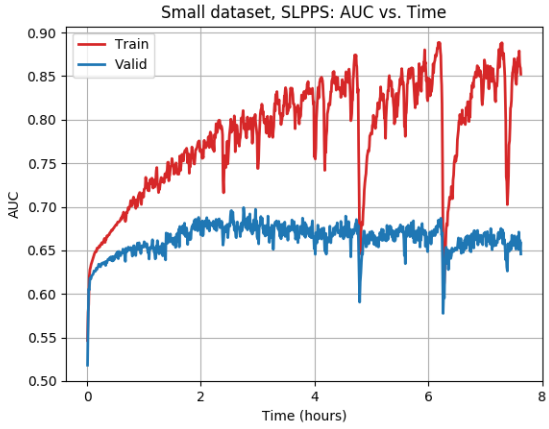

Figure 11. AUC vs. time of SLPPS on small dataset

Compared to SLPPS, the predicting performance of SRW didn't improve too much after training. This is because SRW is basically a graph method similar to PageRank. The edge weight model learnt from data might improve the performance a little, but not too much. And SLPPS does not depends on the graph methods heavily, so it can be trained much faster and also for more iterations to get a larger improvement from the initialized model.

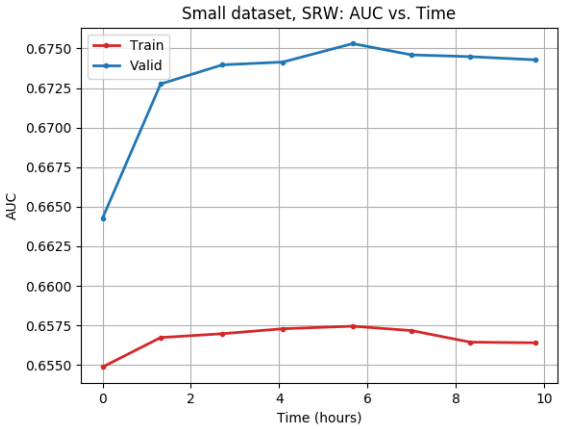

Figure 12. AUC vs. time of SRW on small dataset

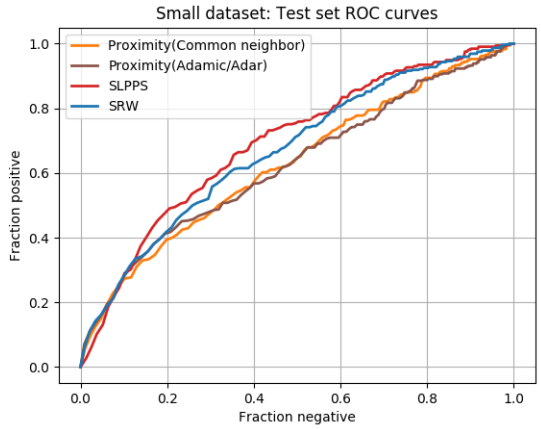

Figure 13. ROC curves on small dataset

Method	Time (h)	AUC	Prec@20
Prox (Common Nbr)	/	0.62813	0.0875
Prox (Adamic/Adar)	/	0.62544	0.0875
SRW	7.00	0.67084	0.0964
SLPPS (Square loss, d=2, lr=0.01)	4.55	**0.69332**	**0.1017**
SLPPS (WMW loss, d=2, lr=0.01)	3.48	0.67928	0.0979
SLPPS (Square loss, d=3, lr=0.01)	12.23	0.67479	0.0955
SLPPS (Square loss, d=2, lr=0.001)	9.40	0.67024	0.0946

Table 5. Test performance of different methods on small dataset. Time: training time to achieve the best validation performance, Prec@20: top-20 precision on test set

5.3.3 Test Performance

To evaluate the performance of SLPPS and SRW on test set, we select the model parameters that achieve the highest validation performance (top-20 precision) during training process as our final model parameters. Then we evaluate the top-20 precision and AUC on test set using these parameters. We report the test performance of different methods in Table 5 and plot their ROC curves in Figure 13. It is clear that SLPPS with appropriate model setting achieves the highest test performance among all algorithms. For two baseline methods, common neighbor is doing slightly better than Adamic/Adar, and the proximity based methods perform the worst in small dataset. SRW performs much better than proximity based method due to the effective combination of PageRank algorithm and supervised learning. But we can find that SLPPS with some slightly worse model setting can also achieve comparable performance as SRW. This is because SLPPS can train faster and for more iterations to learn better model parameters to fit the data.

We also explored how different model settings or hyperparameters impact our algorithm as shown in Table 5. We found that square loss is more suitable for our dataset than WMW loss. We believe both loss functions make sense, but are still not tailored enough for our ultimate evaluating standard. And on our dataset, the squared loss happens to be a bit better than WMW. When we changed the path limit d in SLPPS from 2 to 3, the test performance decreased. We found that this modification involves 20x or more paths compared to $d = 2$, which introduces more noise and make the model much harder to generalize from data. Meanwhile, it requires 5x more time to train in each iteration. As a result, even after 12 hours, the training still didn't completely converge. We also tried a smaller learning rate (0.001), but the training loss decreased much more slowly. And after 9 hours, its training loss and predicting performance didn't change a lot. It is probably because it was trapped in some local optima that the small learning rate could not break through. As a result, it yields the worst performance among all settings.

In Table 5, the training time to achieve the best validation performance is also reported. It took SRW 7 hours but actually it is only the fifth iteration. As we mentioned before, SRW requires a longer time to train in each iteration. So we believe if we can try more settings and train for a longer time, its performance may be better. SLPPS with appropriate model settings can always learn good parameters in a short time since with the same training time, it updates much more iterations than SRW (with 7 hours, it updates 800 iterations in total, so it is about 160x faster than SRW). Also, we still have an overfitting issue in SLPPS as mentioned above, which makes the algorithms achieve the best validation performance earlier.

5.4. Experiment Results on Big Dataset

We also evaluate our method on the big dataset. However, we are not able to evaluate SRW on big dataset. This is again because SRW is essentially a time-consuming algorithm that involves power iteration in both forward and backward update. It actually also has a high space complexity even when considering storing the n by n transition matrix only. It takes around 1.5 hour to train one iteration on a 2.27GHz CPU, and takes up around 5.5G memory when training on the small dataset with 9255 nodes. And on the large dataset with 116k nodes, it will require at least more than 50G memory, which exceeds the computational resources we have (to compare, SLPPS requires only 2.1G memory to train small dataset, and 18.5G memoery to train big dataset).

The training loss of SLPPS on big dataset is shown in Figure 14. Compared to the training loss on small dataset, this time the curve is much smoother. This is because we increase the mini-batch size this time (from 5 to 10). And the number of candidates of each source node becomes larger compared to small dataset, which is equivalent to having more data per source node. As a result, the gradients among different iterations become more stable.

Figure 14. Training loss vs. time of SLPPS on big dataset

The predicting performance (top-20 precision and AUC) on training set and validation set during the training process is shown in Figure 15 and 16. We found that the overfitting issue becomes more severe compared to small dataset regardless we have more data in large dataset. Actually we can find that the training loss is not converged at all even after 23 hours of training. And in small dataset, the predicting performance (AUC) can achieve around 0.9 when the training is converged but this time the AUC on training set is only around 0.63. In fact, due to the size of big dataset, the training becomes 15x slower compared to small dataset, which gives us much fewer training iterations this time. As a result, we believe that such training time is still not long enough to learn/generalize a good model.

Notice that the validation performance increased rapidly at the very beginning and then decreased slowly in the rest of time. This indicates that the model generalized from training set did not do well in validation set. This may be due to the model has not been generalized well enough yet as mentioned above, or due to the difference between training set and validation set. Note that the graph of big dataset is 12x larger than small dataset. And the nodes (authors) come from different generations (to compare, in small dataset, we only include authors that published articles in the first year). As a result, big dataset is much noisier than the small one, and the nodes may behave much more differently from each other. All these issues make it harder to learn/generalize a good model from the big dataset. However, due to the slow training on big dataset, we can neither train longer or perform too much parameter tweaking. We have to leave this to future work.

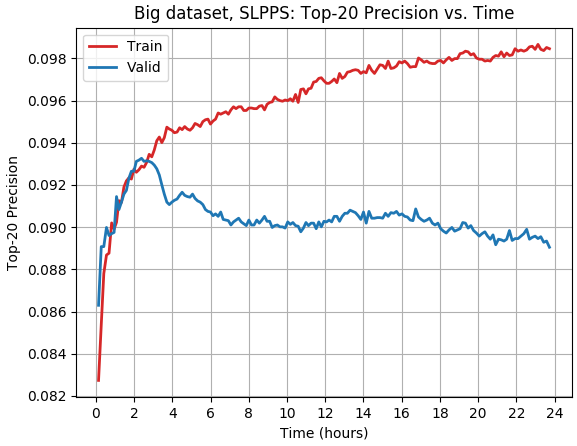

Figure 15. Top-20 Precision vs. time of SLPPS on big dataset

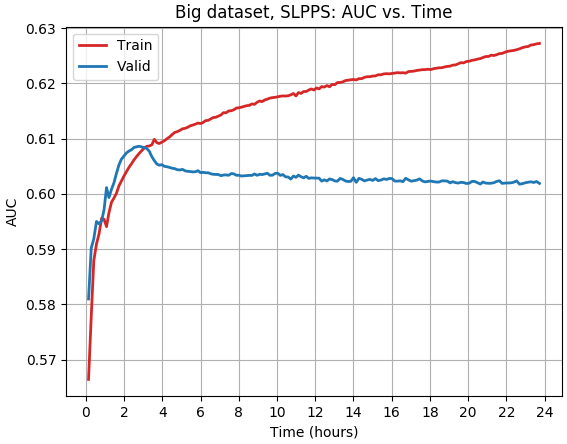

Figure 16. AUC vs. time of SLPPS on big dataset

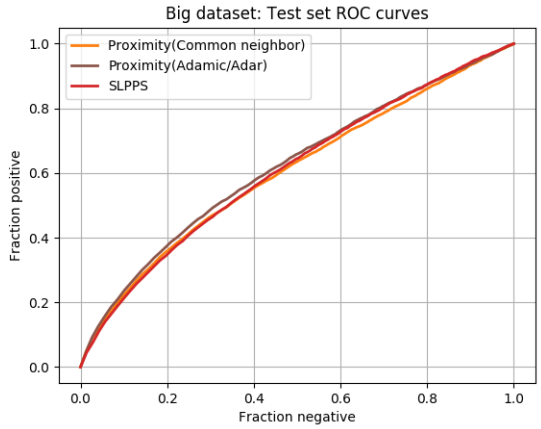

Figure 17. ROC curves on big dataset

Method	Time (h)	AUC	Prec@20
Prox (Common Nbr)	/	0.60450	0.0959
Prox (Adamic/Adar)	/	**0.62055**	**0.0992**
SLPPS (Square loss, d=2, lr=0.001)	2.51	0.61341	0.0977
SLPPS (Square loss, d=2, lr=0.01)	1.88	0.60696	0.0961
SLPPS (Square loss, d=3, lr=0.001)	11.77	0.60210	0.0948

Table 6. Test performance of different methods on big dataset. Time: training time to achieve the best validation performance, Prec@20: top-20 precision on test set

The test performance of different methods are reported in Table 6, and the ROC curves are plot in Figure 17. Note that on the big dataset, SLPPS did not perform the best. Actually, it only achieved a poor performance between two proximity based methods. We tried increasing the learning rate to accelerate the convergence of training. But unfortunately with a learning rate larger than 0.01, the training loss tends to converge to a poor value. It was probably going back and forth and cannot reach the local optima due to the large learning rate. Thus, we believe there exists a better learning rate somewhere between 0.001 and 0.01 but we did not find it within the limited time. Also, if we increase the path limit parameter d (from 2 to 3), then the training becomes even slower. After 12 hours, we only have around 20 training iterations. So we have to stop earlier and report the test performance up to that point in Table 6.

5.5. Time Complexity Analysis

Although SLPPS is shown to be much faster than SRW, the time complexity of it is still very high. Suppose the nodes of a graph have an average degree of β. Then with path limit d, for each source node, we need to do a BFS search and iterate around β^d edges. So if a training set contains N source nodes, the time complexity of each training iteration of SLPPS will be $O(N\beta^d)$. As a result, with a large graph, the average degree β may become larger, and the time complexity will grow exponentially with β. And if we increases d, then it will require β x more time in each training iteration. As a result, SLPPS is still not scalable as the graph goes larger. So when dealing with extremely large graph, we need to either reduce the graph size by proper data processing or resort to pure machine learning algorithms.

6. Conclusion and Future Work

In this project, we propose a new method, Supervised Link Prediction with Path Scores (SLPPS), for the link prediction task in social network. Our method combines the path information with the supervised learning technique to train a model for potential link prediction. The idea of combining graph method and machine learning comes from Supervised Random Walks (SRW) [2]. Compared to SRW, our method does not involve the PageRank iterative forward and backward update, so it has a lower time and space complexity. Experiments show that our methods can be 160x faster than SRW and requires less than 50% memory. Meanwhile, it can achieve similar and even higher predicting performance than SRW.

However, SLPPS tends to suffer from overfitting issue during training and requires great effort for parameters tweaking. Moreover, its time complexity is still too high and not scalable for extremely large dataset. Also, large dataset will make it harder for SLPPS to generalize a good model.

In the future, we may try more model settings and parameter tweaking for SLPPS to see if we can reduce the overfitting issue. We will also explore smarter way than averaging to combine edge features (or edge scores) into path features (or path scores). Furthermore, to speed up SLPPS and make it more scalable, we will try to combine it with some graph reduction algorithms. We will also implement some pure supervised learning methods and compare their performance and efficiency to our method.

References

[1] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki. Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*, 2006.

[2] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM, 2011.

[3] T.-P. Lee. Python implementation of supervised random walks algorithm on github. `https://github.com/tblee/Link_prediction`, 2016.

[4] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *journal of the Association for Information Science and Technology*, 58(7):1019–1031, 2007.

[5] P. Wang, B. Xu, Y. Wu, and X. Zhou. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences*, 58(1):1–38, 2015.