Final Report: Link Prediction on Network Structured Data

Eric L. Lee, Shu-Hao Chang, Yu-Cheng Weng, Can Jiang

November 17, 2019

1 Abstract

In this report, we focus on solving link prediction problems. The main difficulty of link prediction is to extract or generate features for each link for the machine learning algorithms to work. We surveyed several methods and categorized these methods into three categories in our proposal: heuristic method, latent feature method and subgraph sampling method.

Due to the time limit, we haven't implemented subgraph sampling method by the time when we submitted the report. However, we've implemented heuristic method and latent feature method. Besides, we do several experiments on 7 different data sets and find several interesting facts. In addition, we find a mistake about the result of matrix factorization in a series of paper published in KDD 2017 and NIPS 2018. We have notified the author in prevention of mistakes in the future publication. Through the conversation with the authors, we find it a easily made mistake if we don't pay attention to the implementation of machine learning package. In fact, these two paper have 80 citations in total and no one found this mistake. We will point out this mistake in the Matrix Factorization section.

Our report will be arranged in the following way. Section 2 is the introduction of our problem and section 3 is a section regarding how to reproduce our result. Section 4 contains some data analysis we've done on our dataset. In section 5 to section 6, we will talk about some simple heuristics and our evaluation methods. In section 7, we will use logistic regression and random forest to ensemble the heuristic model. In section 8, we apply matrix factorization model to link prediction problem and point out mistakes that made in two papers. In section 9, we summarize all the interesting finding and make our conclusion.

2 Introduction to Link Prediction Problem

Link prediction is a task to predict whether there will be an edge between two nodes. Link prediction is actually very different from a usual machine learning task by the following three characteristics:

1. We don't have any feature vector predefined in our graph.

2. Link prediction is a ranking problem.

3. The negative examples are much more than the positive.

The first characteristics is seldom encountered in machine learning problems, which makes link prediction very unique.
To make link prediction a machine learning problem. There are three ways to generate (or learn) feature vector.
1. Heuristic
This is the most simple way, but it is actually a very effective way. And there are some very famous heuristics such as Common Neighbors, Katz similarity, Preferential Attachment, ... etc. We will discuss these methods from section 5 to section 7.

2. Latent Feature Method
This method is to use factorization model(a.k.a Matrix Factorization) [3]. We give each node a latent feature vector and use the inner product be the probability whether the node will have the link. This method has a great performance in recommendation problem (link prediction in bipartite graph). We want to test whether it will have the same great performance when it comes to a much more general link prediction problem.
We will discuss this method in section 8.

3. Subgraph Sampling Method
Subgraph Sampling approach is to sample a set of subgraph and enumerate all the nodes in each subgraph. As a result, the adjacency matrix in each subgraph of these subgraphs can be the feature vector for machine learning approach. In fact, there are two recent works proposed in KDD 2017 [12] and NIPS 2018 [13] using this method.
To illustrate the idea of subgraph sampling, let us draw an example in Figure 1.
The second and third characteristics will be done by an unique way to evaluate models, and we will discuss this in Chapter 5.2.

3 How to Reproduce Our Result

We believe that reproducibility is very important when it comes to research. Thus, we carefully recorded all of our experiment results on Github. All the results presented in the following sections can be easily reproduced using this github repository:

https://github.com/miamiasheep/Purdue_ML_Course_Project

And all of our experiments can be reproduced by following the instruction of README. If you have any problems of reproducing the experiments, please feel free to contact us by posting an issue in the repository.

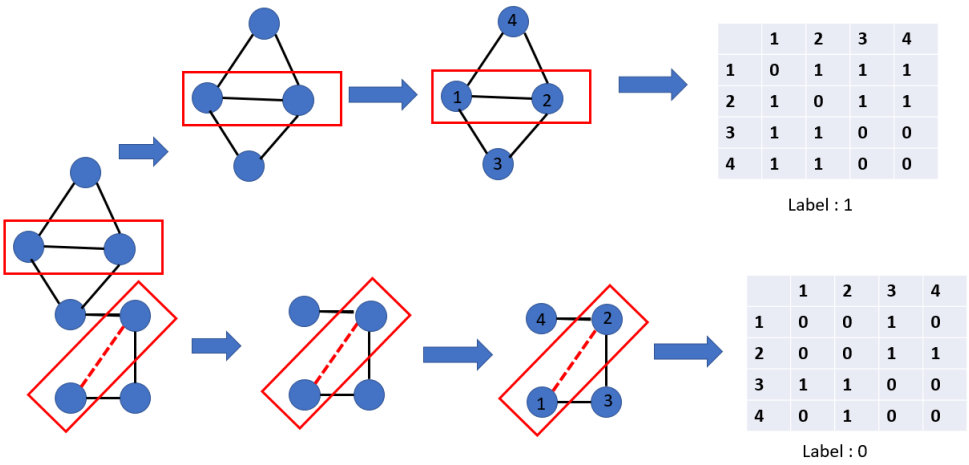

Figure 1: subgraph sampling method

4 Data Analysis

4.1 Data Set

We collected 7 different datasets which are all unique and have different characteristics. Table 1 shows basic statistics of each data set.

4.1.1 Facebook

This is a dataset downloaded from SNAP(Stanford Network Analysis Project)[5]. The data is an egonet in facebook. Each node represents an account in Facebook and each link represents friendship in Facebook.

4.1.2 Power

Power is an power grid of western US.[11] For the power grid, nodes represents generators, transformers and substations, and links represent high-voltage transmission lines between them.

4.1.3 NS

NS is a collaboration network of researchers who publish paper on network sciences [7]. The network maps coauthorships between 379 scientists whose research centers on the properties of networks of one kind or another, where each node represents an author in the network, and the link indicate coauthorships between the two authors.

dataset	nodes	edges	average degree	max degree
Facebook	4039	88234	21.85	1045
Power	4941	6595	1.33	19
NS	1461	2742	1.87	34
Router	5022	6258	1.26	106
USAir	332	2126	6.40	139
Yeast	2375	11683	4.91	118
Celegan	297	2148	7.23	134

Table 1: Basic Information of different dataset

4.1.4 Router

Router is a router-level Internet.[9]Each node represents a router and each edge represents links between two networks, which identified by DNS names.

4.1.5 USAir

USAir is a network of US air lines. [2]Each node represents an airport and each edge represents airline among two US airports.

4.1.6 Yeast

Yeast is a protein-protein interaction network in yeast[10]. The dataset is obtained by applying the spectral analysis method to complicated protein-protein interaction networks and identified interesting topological structures. Each node represents a protein and each link represents interaction between proteins.

4.1.7 Celegan

Celegan is a neural network of Caenorhabditis elegans.[11] For C. elegans, a link joins two neurons (nodes) if they are connected by either a synapse or a gap junction. All links are treated as undirected and unweighted, and all nodes as identical, recognizing that these are crude approximations.

4.2 Visualizing Our Dataset

We implemented a tool to visualize our datasets. And all of our visualization graphs can be seen in the following link:

https://github.com/miamiasheep/Purdue_ML_Course_Project/tree/master/graph

Since the network is too big and it is hard to find any clue regarding the network structure if we draw all the edges and nodes, we only sampled a small portion of the network on every dataset.

By visualizing our data sets, we can see that the characteristic of each data is actually very different. Let's take Facebook (figure2) and Router (figure3) as examples. Facebook dataset has multiple triangles in the middle of the network. In contrast, Router has several star-stuctured networks.

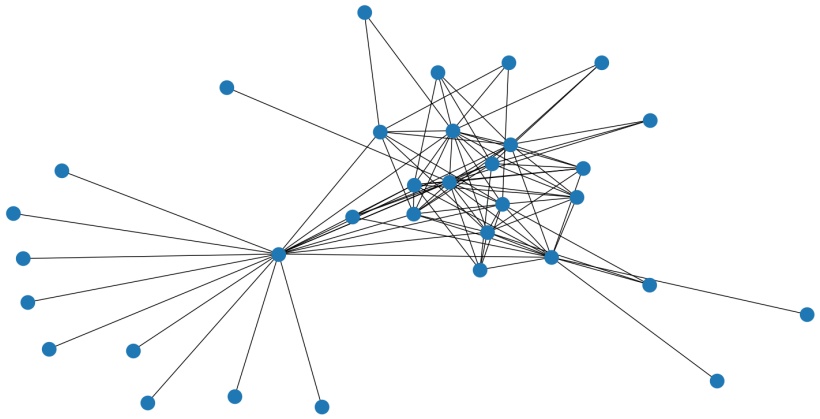

Figure 2: facebook

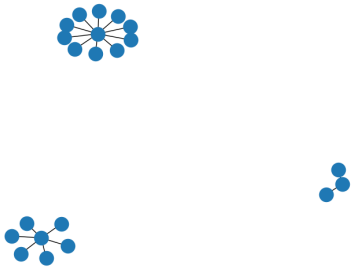

Figure 3: Router

5 Heurist Method and Evaluation

5.1 Heuristic

For the following sections, we denote x and y as two nodes, between which we want to predict if there is a link. And we denote the set of neighbors of x as N(x). We implemented six very common heuristics. And in table 2, we organized the formulas of the six heuristics.

5.1.1 Common Neighbors(CN)

It is a very simple heuristic. The intuition is that the more common neighbors the two nodes have, the more likely that there is a link between them. The score can be calculated as $|N(x) \cap N(y)|$.

5.1.2 Jaccard(JC)

It is very similar to common neighbors except that Jaccard considers the fact that two higher degree nodes unsurprisingly tend to have more common nodes. Thus, if two lower degree nodes have common neighbors, maybe they really have a strong relationship. The score can be calculated as $\frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|}$

5.1.3 Adamic Adar(AA)

It is also very similar to common neighbors. It assumes that a low degree common neighbors have more contribution to the probability of a link between x and y. Thus, the score divides the weight of each common neighbor by the logarithm of its degree. It is actually very similar to TFIDF. We can calculate Adamic Adar by $\sum_{z \in N(x) \cap N(y)} \frac{1}{log(|N(z)|)}$

5.1.4 Total Neighbors(TN)

It is the simplest baseline, which is just summing up the total size of neighbors of x and y. It can be calculated as $|N(x)| + |N(y)|$

5.1.5 Preferential Attachment(PA)

It is assumed that a high degree node is more likely to link with other nodes. Preferential Attachment calculate the product of degree of x and y. It can be calculated as $|N(x)| * |N(y)|$

5.1.6 Page Rank(PG)

It is very similar to PA. But instead of using the product of nodes' degrees, it uses the product of two nodes' scores derived from performing page rank on our network. Since the product can be very small, we use the sum of logarithm instead. Let r_x be the score of x after performing page rank and r_y be the score of y after page rank. It can be calculated as $\log r_x + \log r_y$

5.1.7 Katz

The heuristic counts all the paths between given pairs of nodes, with shorter paths weighted more heavily. It can be calculated as $\sum_{l=1}^{\infty} \beta^l |paths_{xy}^{<l>}|$, where β is a free parameter and $|paths_{xy}^{<l>}|$ is the number of paths between x and y at given length l.

5.1.8 Random Walk with Restart (RWR)

It is an adaptation of Page Rank algorithm. Consider random walker starting from node x and periodically, with probability α, returning to x. Let q_x be a stationary distribution of Markov chain describing this walker. From definition of stationary distribution: $q_x = (1-\alpha)P^T q_x + \alpha e_x$, where e_x is a unit vector with 1 on position corresponding to node x, and P is a transition matrix describing ordinary random walker. The RWR index is then defined as $q_{xy} + q_{yx}$, where q_{xy} means the yth element of q_x. Since the sum can be very small, we calculated it as $\log q_{xy} + \log q_{yx}$

method	description
Common Neighbor(CN)	$\|N(x) \cap N(y)\|$
Jaccard(JC)	$\frac{\|N(x)\cap N(y)\|}{\|N(x)\cup N(y)\|}$
Adamic Adar(AA)	$\sum_{z\in N(x)\cap N(y)} \frac{1}{log(\|N(z)\|)}$
Total Neighbor(TN)	$\|N(x)\| + \|N(y)\|$
Preferential Attachment (PA)	$\|N(x)\| * \|N(y)\|$
Page Rank(PR)	$\log r_x + \log r_y$
Katz	$\sum_{l=1}^{\infty} \beta^l \|paths_{xy}^{<l>}\|$
Random Walk with Restart (RWR)	$\log q_{xy} + \log q_{yx}$

Table 2: link prediction methods

5.2 Evaluation

Evaluation is not trivial when it comes to link prediction problems. If we want to evaluate all pairs of nodes, it will need O(N^2) predictions, where N is the size of the nodes. And even for a small graph with 1000 nodes in testing set, the evaluation time can be really long because we have to make 1000 * 999 /2 predictions. In fact, finding a fair and effective evaluation is also a research topic. Ryan and Nitech published a paper [6] about how to evaluate a link prediction problem. We choose the most common way to evaluate our model: Down-sampling negative samples. (By negative samples, it means a pair of nodes with no link between them. In link prediction problems, negative samples are usually way more than positive ones.)

First, we randomly divided our edges into training, validation and testing. The ratio is 0.8:0.1:0.1. Training set is for training while validation set is only for grid search for the best parameters. And testing set is for evaluating our model.

Second, we randomly sampled negative samples for validation and testing set. The sample size is equal to the dataset size.

Third, we train the models using training set, and tuned our model using validation set, and evaluate the models with testing set. The higher score a pair gets, the higher the ranking it is and having more chance to have a link between them. For evaluation, we used the ranking metrics. We mainly used AUC-ROC score (area under Receiver Operating Characteristics curve, abbreviated to AUC here). AUC has a very good characteristic that the sampling AUC is actually an approximation of actual AUC even if we do down-sampling.

We also implemented f1@k score as an evaluation metric. For the parameter k, we set the k equals to the size of positive samples. For this k, recall is actually equals to precision. And f1@k thus has a very intuitive physical meaning: the accuracy of finding all of the missing links given the size of missing links. In the future, we'll try to set different ks and see the influence of them.

Data Set	CN	JAC	AA	PA	TN	PG	Katz	RWR
Celegans	0.8415	0.7835	0.8566	0.7567	0.7318	0.7593	0.8416	0.8863
facebook	0.9882	0.9863	0.9893	0.8327	0.7352	0.8085	0.9879	0.9904
NS	0.973	0.9745	0.9738	0.6842	0.5287	0.5268	0.9899	0.9911
Power	0.5973	0.5973	0.5973	0.5296	0.5207	0.5613	0.7743	0.8124
Router	0.6119	0.6116	0.6119	0.9273	0.9152	0.9385	0.6756	0.717
USAir	0.9474	0.9118	0.9581	0.899	0.8616	0.8919	0.9323	0.9455
Yeast	0.902	0.9018	0.9029	0.8582	0.7945	0.8523	0.9678	0.9765

Table 3: AUC of the six baselines

Data Set	CN	JAC	AA	PA	TN	PG	Katz	RWR
Celegans	0.8472	0.7269	0.8102	0.6944	0.6481	0.6991	0.7963	0.8148
facebook	0.9724	0.9495	0.9707	0.7546	0.6796	0.7445	0.9643	0.973
NS	0.9524	0.9524	0.9524	0.6407	0.5801	0.4848	0.987	0.9913
Power	0.1947	0.1947	0.1947	0.5088	0.4779	0.4823	0.6991	0.9491
Router	0.2279	0.2279	0.2279	0.8346	0.8199	0.8382	0.6324	0.636
USAir	0.8995	0.8278	0.8947	0.8278	0.8278	0.8278	0.8517	0.9091
Yeast	0.815	0.815	0.815	0.7882	0.7364	0.7721	0.9392	0.9553

Table 4: F1 score of the six baselines

Also, something to note here is that some nodes may only exist in validation or testing set but not in training set. We discarded the pairs of nodes where at least one node is not in the training set when evaluating, i.e., we only considered pairs of which both nodes are in the training set.

6 Result and Discussion of Heuristic Methods

Table 3 shows AUC's of the six different baselines and Table 4 shows the F1 scores.

According to the result above, we can see that no heuristic can outperform all other heuristics, and that Adamic Adar(AA), Jaccard, and Common Neighbors perform pretty well in the Facebook and NS dataset. However, for Power and Router dataset, the results of Common Neighbor based methods are pretty bad. At first, we think the reason that AA, Jaccard and Common Neighbors perform so bad is because these three heuristics are not suitable for Power and Router datasets. However, when we carefully analyzed the predictions, we found AA, Jaccard and Common Neighbors were actually very good for the high-ranking pairs. The reason why it performed not well in AUC and F1@(sample size) is because these datas have a low average degree so that many predictions are zero. When we see the result of precision@50, we will find that Common Neighbors, AA and Jaccard are actually very good under this evaluation.

Dataset	CN	JAC	AA	PA	TN	PG	RWR	Katz
Celegan	0.98	0.62	0.98	0.84	0.8	0.56	0.3	0.2
facebook	1.0	0.92	1.0	0.76	0.8	0.44	0.86	0.52
NS	1.0	1.0	1.0	0.6	0.38	0.4	0.86	1.0
Power	1.0	1.0	1.0	0.44	0.36	0.26	1.0	1.0
Router	0.98	0.92	0.94	0.96	0.6	0.64	0.86	1.0
USAir	1.0	0.84	1.0	1.0	0.8	0.76	0.44	0.52
Yeast	1.0	0.96	1.0	1.0	0.9	0.92	0.92	1.0

Table 5: precision@50 of the six baselines

To achieve a better result, we resorted to machine learning approach.

7 Ensemble Heuristics (Machine Learning Approach)

In this section, we present the process and the results using the machine learning approach.

First, we sampled the training set and testing set. We made both sets class-balanced, which implied that half the pairs have links, and the other don't. And as we mentioned above, the features of a instance in the datasets are the heuristics generated using the heuristic method. And the label of the instance is either 1 or 0, meaning whether there is a link between the pair. We formulated it as a classification problem, but for the prediction, we predicted the probability instead of the labels.

Second, we employed logistic regression and random forest for the problem. We used grid search and 3-fold cross validation to find the best parameters for the models.

Third, we built the models with the best parameters for the testing set. We then got the probability of whether there is a link for each pair. One can see the probability as a score, just as what we obtained when using heuristic methods. The higher the score, the higher the ranking it gets. For now, we can employ AUC and F1 to inspect the model performances. Table 6 and Table 7 show the results.

One can compare the results to those heuristic methods and find that machine learning approach overall doesn't really outperform heuristic methods. We think it is because machine learning approach relatively tends to recommend higher order links, which may not be a good case for the dataset. Besides, we can see that heuristic methods already performed very well, especially the Random Walk with Restart (RWR) method. In fact, our random forest models also suggested that RWR is the most important feature for almost every dataset. But still, machine learning approach has its own value. For the Router dataset, RWR doesn't seem to be a good method. This is where machine learning approach comes into play. It may be more robust and less dataset-variant. For the future work, we are interested in generating more heuristic features for the machine learning models and imposed some constraints on

Dataset	RF	LogisticRegression
Celegan	0.8292	0.8697
facebook	0.9829	0.9866
NS	0.9925	0.9898
Power	0.5923	0.5029
Router	0.9208	0.7591
USAir	0.9377	0.9476
Yeast	0.9652	0.9702

Table 6: AUC of the machine learning models

Dataset	RF	LogisticRegression
Celegan	0.7870	0.8101
facebook	0.9691	0.9647
NS	0.9826	0.9870
Power	0.4933	0.4314
Router	0.7794	0.7095
USAir	0.8995	0.9043
Yeast	0.9258	0.9338

Table 7: F1@label size of the machine learning models

the models for better performance.

8 Matrix Factorization

In this section, we try to apply Matrix Factorization to link prediction. Matrix Factorization works by decomposing user-item interaction matrix into the product of two lower dimensionality of rectangular matrices. It is very popular in recommendation system problem. Let the user-item interaction be the matrix R. And the first matrix be P and the second matrix be Q.

$$\min_{P,Q} \sum_{(u,i)} L(R_{u,i}, P_u Q_i^T) \tag{1}$$

where L is the loss function we can define.

For example, the following equation is a example of using square error.

$$\min_{P,Q} \sum_{(u,i)} (R_{u,i} - P_u Q_i^T)^2 \tag{2}$$

However, the link prediction goal is the ranking not square error. So, we adopt Bayesian Personal Ranking[8] in this problem. Denote I_u^+ is the node that has a link with node u and I_u^- is the node that doesn't have a link with node u. And let D_s be $\{(u,i,j)|i \in I_u^+, j \in I_u^-\}$.

If we apply Bayesian personal ranking on Matrix Factorization, we will get equation 3.

$$\min_{P,Q} \sum_{(u,i,j)\in D_s} \log(1 + e^{-P_u(Q_i^T - Q_j^T)}) \tag{3}$$

All of the implementation can be found in libmf[4] and we directly use libmf in our experiment.

8.1 A Miastake Using Matrix Factorization

We should be very careful when applying matrix factorization to solve link prediction problem.

When we survey related paper, we find that some papers use matrix factorization in a wrong way, and the mistake still exists in recent NIPS paper. The paper was accepted in KDD 2017 and then submitted in NIPS 2018 using this incorrect result and being accepted without noticing the bug. We then contact the author and find that we are the first to find this bug. Because these two papers already have over 80 citations in two year, we think it might be a bug that not so easy to find. So, we want to point out this mistake in the following section and our final project presentation.

8.2 Zero Predictions Problem

Matrix Factorization is very popular in solving recommendation problem which can be one of link prediction problem. The only difference is that recommendation problem focus on the bipartite graph.
Obviously, matrix factorization can also solve link prediction problem that is not bipartite, which is the most intuitive way to directly input the edges of the graph into matrix factorization model. Just like Figure 4.

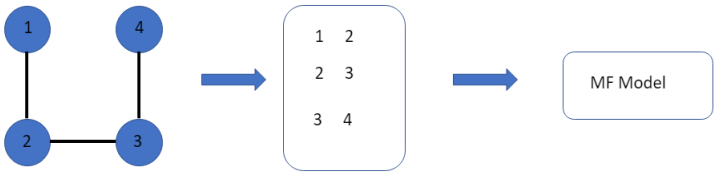

Figure 4: wrong usage of matrix factorization

However, it is not the correct way to use matrix factorization. In fact, it will yield a very bad result especially for the graph with low average degree.
The reason Matrix Factorization performs badly isn't because it's not suitable for this problem, but because it predicts many zeros. Apparently, there are something wrong here. Table 8 shows the percentage of zero prdictions of each dataset.

Dataset	zeros	testing size	percentage(%)
Celegan	42	431	9.74
facebook	1008	17556	5.74
NS	113	477	23.69
Power	338	1027	32.91
Router	282	724	38.95
USAir	96	414	23.19
Yeast	404	2192	18.43

Table 8: zero predictions in each dataset

8.3 Implementation of libMF

To figure out why zero predictions problem happen, we read the source code of one of famous matrix factorization package "libMF". And what the package is do is to perform the following stochastic gradient descent(if you use the default option). We describe the process in algorithm 1. (PS: It is a very high level description of libmf, besides perform the SGD, it actually do lots of optimization using multi-threading.)

The input format is [node1] [node2] [label]. [node1] is used as an instance in first latent matrix (P). [node2] is an instance in second latent matrix (Q). We will find that, in the example of figure 4. The latent vector of Node 4 in P has no chance to be updated because node 4 only appear in the second position[node2]. And libmf set the intial matrix to zero matrix. Therefore, the latent vector of node 4 will become a zero vector. However, it is not reasonable. Because there is actually a positive link related to node 4. This problem will happen even if we don't use bayesian personal ranking. Because node 4 only exists in the position of second node. The latent vector of node 4 will not be update even if we use other optimization goal.

8.4 Two ways to solve this problem

8.4.1 Duplicate data

The most simple way to solve the problem is to duplicate the training data. For each edge (x,y), we create a new edge (y,x) in the training set (Discribe in Figure 5). It will solve the zero predictions problem.

Although it seems to be a small modification, it will lead to a significant improvement in most of the datasets.

Table 9 shows the improvement of doing this modification. MfWrong column is the result of wrong matrix factorization method and MfDup is the result of the matrix factorization using duplicate data. We can see that it helps a lot for the graph who has low average degree.

Algorithm 1 Matrix Factorization

procedure MATRIX FACTORIZATION(iteration)
  P ← Zero Matrix
  Q ← Zero Matrix
  for i in 1...iteration **do**
    for edge(u,i) in all edges **do**
      Sample a negative node j that have no link to u
      for k in latent features **do**
        $P_{u,k} \leftarrow P_{u,k} - \alpha * \frac{\partial Error}{\partial P_{u,k}}$
        $Q_{i,k} \leftarrow Q_{i,k} - \alpha * \frac{\partial Error}{\partial Q_{i,k}}$
        $Q_{j,k} \leftarrow Q_{j,k} - \alpha * \frac{\partial Error}{\partial Q_{j,k}}$
      end for
    end for
  end for
end procedure

Dataset	MfWrong	MfDup	Improvement(%)
Celegan	0.7502	0.7978	6.34
facebook	0.9797	0.9799	0.02
NS	0.8081	0.9364	15.88
Power	0.5307	0.6985	31.62
Router	0.7321	0.8271	12.98
USAir	0.8183	0.8193	0.12
Yeast	0.924	0.9557	3.43

Table 9: Comparison with MfWrong and MfDup in AUC

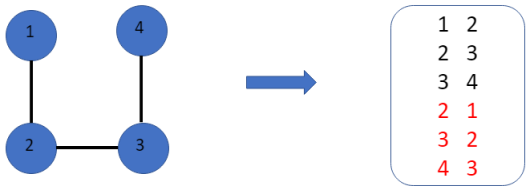

Figure 5: fix problems by duplicating data

8.4.2 Factorize Into One Matrix

We don't have to factorize the matrix into two matrices which will yield lots of unnecessary parameters. We can change the optimization into following form in equation 4. Due to the time limit, we didn't finish this part so far but still working on it.

$$\min_{P} \sum_{(u,i,j)\in D_s} \log(1 + e^{-P_u(P_i^T - P_j^T)}) \tag{4}$$

9 Summary

In this report, we reimplement several models aims at tackle link prediction problem and apply these models to 7 datasets. We actually discover something interesting, and summarize them in this section.

9.1 Common Neighbors Is A Very Good Heuristic Model

We think it's a riveting finding. People argue that machine learning model may be a better solution since no heuristic can perform consistently well on every dataset. Common Neighbors may be more suitable for data that has multiple triangles such as Facebook; Preferential Attachment is more suitable for data having lots of star structures such as Router dataset.

However, according to our experiment, it may not tell the whole story. Common Neighbors seem to perform badly in Router dataset. However, it is not because Common Neighbors is not suitable for dataset containing lots of star structures, but because lots of pairs don't have common neighbors and generate the same zero predictions. It will worsen the result of AUC of common neighbors. However, common neighbor actually have a wonderful performance in term of precision@50 even if we apply to Router or Power dataset.

When it comes to recommendation system, Precision@(small k) is actually a crucial metric. However, we found few papers about link prediction using precision@(small k) as evaluation metric. We guess it is because Common Neighbors already did well, so it's hard to improve.

9.2 Ensemble Heuristic Directly May Not Be A Good Approach

We have ensembled 7 ensemble heuristics using random forest and logistic regression in section 7. At the beginning, we think it will at least yield a result better than all the heuristics.

However, it doesn't happen.

We find that nodes are much less probable to link with nodes that are far from them. We first discover the fact when we perform katz on seven different datasets. When we do the grid search, we find the best β tend to be minute(ex: 10^-7). Ensemble heuristic render more chance to predict nodes that are far away from each other in our model, since Total Neighbors and Preferential Attachment features assign values to every pair. However, we've tried to remove these two features and the performance seems to be slightly better.

One of the state-of-the-art method is subgraph sampling which mentioned in section 2. A reason that it performs may because it won't recommend nodes that are far from the source nodes.

9.3 Carefully Check Implmentation Is Very Important

If we didn't check the implementation of matrix factorization, we might think matrix factorization performs terribly. In fact, considering matrix factorization only need O(k)(k is the size of latent vector) time in prediction, matrix factorization is actually a terrific model in practice. It can achieve a decent performance with a good prediction speed.

9.4 Ensemble Heuristic Directly May Not Be A Good Approach

We have ensembled 8 ensemble heuristic using random forest and logistic regression in section 7. At the beginning, we think it will at least yield a result better than all of the heuristics. However, it doesn't happen.

However, we find that nodes are extremely less likely to have a link with nodes that are far from with them. We first discover the fact when we perform katz on eight different data set. When we do the grid search, we find the best β tend to very small(ex: 10^-7). Ensemble heuristic let our model have more chance to predict nodes that are far away from each other because we have total neighbors and preferential attachent in the feature vectors. And in fact, we have try to remove these two features and the performance seems to be slightly better.

One of the state of the art method is subgraph sampling which we mentions in the section 2. We think that one of the reason this method performs so good could be that it will not recommend nodes that are far from the source nodes.

9.5 Carefully Check Implmentation Is Very Important

If we don't check the implementation of matrix facotorization, we may think matrix factorization performs really bad. However, matrix factorization is actually not that bad. In fact, considering matrix factorization only need O(k)(k is the size of latent vector) time in the prediction, matrix factorization is actually a very good model in practice. It can achieve a decent performance with a good prediction speed.

References

[1] Ackland, R. Mapping the us political blogsphere: Are conservative bloggers more prominent?

[2] Batagelj, V., and Mrvar, A. Usair data set. http://vlado.fmf.uni-lj.si/pub/networks/data/, 2006.

[3] Bell, R., Koren, Y., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 08 (aug 2009), 30–37.

[4] Chin, W.-S., Yuan, B.-W., Yang, M.-Y., Zhuang, Y., Juan, Y.-C., and Lin, C.-J. Libmf: A library for parallel matrix factorization in shared-memory systems. *Journal of Machine Learning Research 17*, 86 (2016), 1–5.

[5] Leskovec, J., and Krevl, A. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[6] Lichtnwalter, R., and Chawla, N. V. Link prediction: Fair and effective evaluation. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (Aug 2012), pp. 376–383.

[7] Newman, M. E. J. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E 74*, 3 (Sep 2006).

[8] Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence* (Arlington, Virginia, United States, 2009), UAI '09, AUAI Press, pp. 452–461.

[9] Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw. 12*, 1 (Feb. 2004), 2–16.

[10] von Mering, C., Krause, R., Snel, B., Cornell, M., Oliver, S., Fields, S., and Bork, P. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature 417* (06 2002), 399–403.

[11] Watts, D. J. Collective dynamics of small-world networks. *Nature 393*.

[12] Zhang, M., and Chen, Y. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2017), KDD '17, ACM, pp. 575–583.

[13] Zhang, M., and Chen, Y. Link prediction based on graph neural networks. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems* (USA, 2018), NIPS'18, Curran Associates Inc., pp. 5171–5181.