Link Prediction Algorithms and Opinion Dynamics on Social Netwroks

May 2015

Thesis submitted to the

Indian Institute of Technology Kharagpur

for award of the degree of

Bachelor of Technology

by

Sourav Sarkar

under the guidance of

Dr. Niloy Ganguly

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Link Prediction on Social Graph

Problem Statement

The link prediction (LP) problem is to predict future relationships from a given snapshot of a social network. E.g., one may wish to predict that a user will like a movie or book, or that two researchers will coauthor a paper, a user will endorse another on LinkedIn, or two users will become "friends" on Facebook. Apart from the obvious recommendation motive, LP can be useful in social search, such as Facebook Graph Search, as well as ranking goods or services based on not only real friends' recommendations but also that of imputed social links.

Popular approaches

Jaccard : The Jaccard index, also known as the Jaccard similarity coefficient (originally coined coefficient de communauté by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

(If A and B are both empty, we define J(A,B) = 1.) Clearly, $0 \leq J(A,B) \leq 1$ The MinHash min-wise independent permutations locality sensitive hashing scheme may be used to efficiently compute an accurate estimate of the Jaccard similarity coefficient of pairs of sets, where each set is represented by a constant-sized signature derived from the minimum values of a hash function. The Jaccard distance, which measures dissimilarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union: Adamic Adar:To decide if nodes u and v may get linked, one strong signal is the number of common neighbors they already share. Adamic and Adar (AA) [1] refined this by a weighted counting: common neighbors who have many other neighbors are dialed down in importance:

$$SIM_{i,j}^{AA} = \sum_{k \in \Gamma(i) \cap \Gamma(j)} \frac{1}{\log d(k)} \tag{2}$$

where d(k) is the degree of common neighbor k. The AA and RA predictors both penalize the contribution of high-degree common neighbors.

Our approach

Unlike AA and other node-pair signals, our new approach recognizes that propensity of linkage is not purely a function of node similarity; it changes with neighborhood. Also, the presence or absence of edge (u, v) rarely determined by

nodes far from u and v. So use a edge and triangulated dissimilarity based approach.

Defining edge dissimilarity

Let $\Gamma(u)$ be the (immediate) neighbors of u. We will model the edge dissimilarity between u and v as

$$\Delta_w(u,v) = w_{uv}.|\Theta_u - \Theta_v| \tag{3}$$

w_{uv} is the weight vector fitted locally for u, v.

Triangulated Dissimilarity

The optimization program outputs w^*_{uv}, from which we can compute

$$\delta_{uv} = w^*_{uv}.|\Theta_u - \Theta_v| \tag{4}$$

Using only this may not be useful for learning a global consistent model. Therefore, we also compute the triangulated dissimilarity

$$\Delta^{x,y}_{w_{xy}} = \sum_{x\in Q, y\in Graph} \frac{\sum_{i\in T(x)\cap T(y)} \Delta_{w_{xy}}(i,x) + \Delta_{w_{xy}}(i,y)}{|T(x)\cap T(y)|} \tag{5}$$

Finally we use this as the score metric,

$$f(u,v)[LL] = \Delta_{w^*}(u,v) - \delta_{u,v} \tag{6}$$

Objective function selection

We assume that the value of the dissimilarity function is low for node pairs which have an edge connecting them and relatively higher for node pairs which are not neighbours. Under these impositions, we wish to minimize the sum of the dissimilarity values of x and y with their common neighbours. This is because this minimization would in turn give the maximum possibility of an edge between x and y that may appear in future. So our objective is to find optimal w, so that the triangulated dissimilarity

$$\Delta^{x,y} = min_w \Delta^{xy}_w \tag{7}$$

is minimized.
Apart from that we want to chose the w vectors in such a manner that δ_{uv} is larger than nodes which are known to be connected and also smaller than the nodes. So in both cases we can form our constraints in the following manner

$$\sum_{i\in\Gamma(x)\setminus\Gamma(y)} \Delta_w(i,x) \leq \alpha\Delta^{xy}_w \tag{8}$$

$$\sum_{i \in \Gamma(y) \setminus \Gamma(x)} \Delta_w(i, y) \leq \alpha \Delta_w^{xy} \tag{9}$$

$$\sum_{i \in \Gamma(y) \setminus \Gamma(x)} \Delta_w(i, x) \geq \alpha \Delta_w^{xy} \tag{10}$$

$$\sum_{i \in \Gamma(x) \setminus \Gamma(y)} \Delta_w(i, y) \geq \alpha \Delta_w^{xy} \tag{11}$$

So to find the w for the δ_{uv} computation we can use the constraints as constraints of an SVM. Because as we are trying to minimize the $|w|^2$ which is a classic QP optimization wrt some linear constraints but as our equations can be approximated as SVM constraints (i.e. approximating ≤ 0 as ≤ -1 and ≥ 0 as ≥ 1). So for each query-node pair we can run an SVM and find the required w vector.
Similarly minimizing the triangulated dissimilarity is exactly a classic LP problem wrt to the given constraints. We can then simply compute the actual triangulated dissimilarity for the query-node pair and compute the final score.

Scoring Metric

We may use only the QP, LP or both approach together. However the meaning of triangulated dissimilarity and edge dissimilarity remains same by same. So the less we get edge dissimilarity the more they link is likely to exist. Also if some nodes triangulated dissimilarity is large then they are well dissimilar with same set of nodes which gives us a signal that they may be similar and we may expect a link indeed. So if the combined score defined as $f(u, v)[LL] = \Delta_{w^*}(u, v) - \delta_{u,v}$ is large then we are very likely to get links in future.

Evaluation

We basically tested on held out data. We sampled the Neighbors and Non Neighbors in .8 or .9 fraction. Then we send the sampled Information to the Recommendation module which analyzes the network based on the given information and comes up with the scores for all query-node pairs. Then for each query we compute the precision and recall and then for all query we compute the MAP.

MAP, Recall, Precision

In LP, a separate ranking is produced for each node q from a set of nodes Q, which are therefore called query nodes. Fix a q and consider the ranking of the other $N - 1$ nodes. Some of these are indeed neighbors (or will end up becoming neighbors). Henceforth, we will call q's neighbors as good nodes $G(q)$ and non-neighbors as bad nodes $B(q)$.Ideally, each good node should rank ahead

of all bad nodes. precision (also called positive predictive value) is the fraction of retrieved instances that are relevant, while recall (also known as sensitivity) is the fraction of relevant instances that are retrieved. In simple terms, high precision means that an algorithm returned substantially more relevant results than irrelevant, while high recall means that an algorithm returned most of the relevant results.

First we define at query node q the quantity

$$Avp(q) = \frac{1}{L} \sum_{k=1}^{N-1} P_q(k) r_q(k) \tag{12}$$

as Average Precision where $N-1$ is the number of nodes excluding the query node itself, L is the number of retrieved relevant items and $r_i(k)$ is an indicator taking value 1 if the item at rank k is a relevant item (actual neighbor) or zero otherwise (non-neighbor). Averaging further over query nodes. Using this we compute the overall Mean Average Precision.

Integrated Single Stage Learning

In both LP and QP based approach though we are using the help of the neighborhood, but in every query-node computation, when we are going to compute both edge and reference dissimilarity we are using their own weight vector i.e. for different query-node pair even in computing the same dissimilarity. One way to encounter this is to use an integrated one step learning approach. Instead of learning for a specific query-node pair we try to learn all query-node pair together. But the main drawback in this approach is problem with scaling. If the graph is in the order of thousands of nodes then depending the query size and feature vector size the QP and LP problems can be in the size (number of variables) of millions. So solving them is not a very practical idea indeed. So we are trying to keep both the essence. Instead of learning the whole graph together we can use a per query based learning or may also define a threshold on number of variables. Whenever the quota is filled we solve the LP and QP problems and then go on solving the rest. Though this decouples the theoretical QP and LP problems still we can do some experimentation.

Implementing the Integrated Approach

To use the integrated approach we need to map the variables due to different each query-node pair in a single variable space. We can formally define the process with an example. Suppose we have,
Undirected Graph: in adjacency Matrix form $G = (V, E), |V| = n$
Feature Data (F): for each node of the graph we have a feature vector of some give dimension.
Query Data (Q): A set of vertexes in $V$for which will run learn together. Now suppose we have the part of the network in Figure 1,

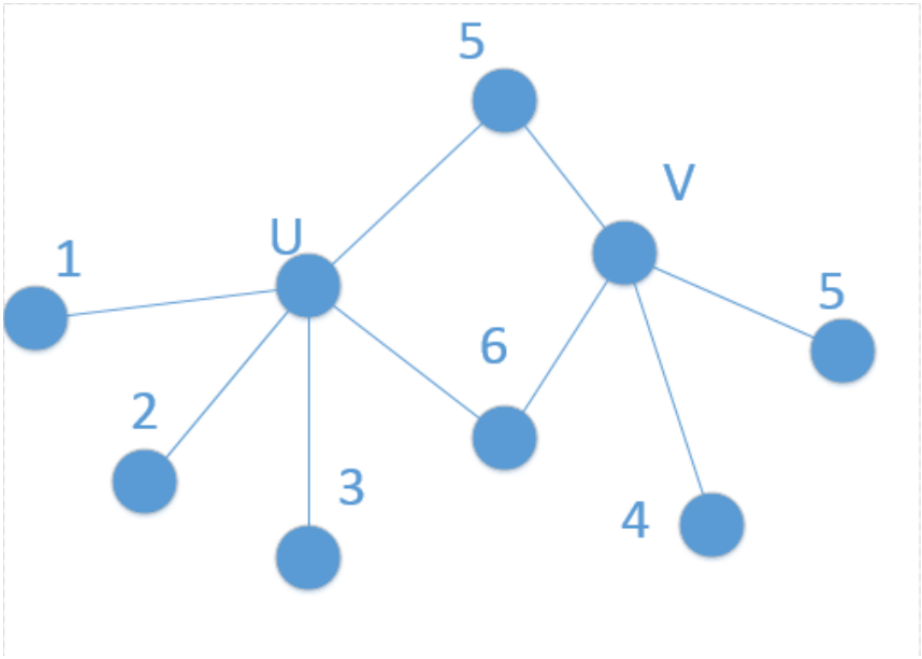

Figure 1: Sample Network

Now according to our problem formulation, the linear constraints after SVM like approximations are,

$$W_{uv}^T \frac{T_{u1} + T_{u2} + T_{u3}}{3} - W_{uv}^T \frac{T_{u5} + T_{u6} + T_{v5} + T_{v6}}{4} \leq -1 \tag{13}$$

$$W_{uv}^T \frac{T_{v4} + T_{v5}}{2} - W_{uv}^T \frac{T_{u5} + T_{u6} + T_{v5} + T_{v6}}{4} \leq -1 \tag{14}$$

$$W_{uv}^T \frac{T_{u4} + T_{u5}}{2} - W_{uv}^T \frac{T_{u5} + T_{u6} + T_{v5} + T_{v6}}{4} \geq 1 \tag{15}$$

$$W_{uv}^T \frac{T_{v1} + T_{v2} + T_{v3}}{3} - W_{uv}^T \frac{T_{u5} + T_{u6} + T_{v5} + T_{v6}}{4} \geq 1 \tag{16}$$

So from these equations (total $|Q| * |V| * 4$ constraints) for each query-node pair we want to find the W_{uv} for all query u and node v together. So we try the bring all the equations in the same variable space and we want to find a W vector of dimension $|F| * |Q| * |V|$, [Where $|F|$ is the feature dimension]
i.e. our

$$W = [W_{q1,1}W_{q1,2}...W_{q1,|V|}W_{q2,1}W_{q2,2}...W_{q2,|V|}W_{q3,1}W_{q3,2}...W_{q3,|V|}...W_{q_{|Q|},1}W_{q_{|Q|},2}...W_{q_{|Q|},|V|}]$$

So we can define the mapping as

$$W_{uv}[k] \rightarrow (i * |V| + v) * |F| + k \tag{17}$$

, where u is the ith query node. So now if we use the QP based approach then our objective function will be

$$\sum_{x \in Q, y \in V} |W|^2_{xy} \tag{18}$$

and for the LP based approach it will be

$$\sum_{x \in Q, y \in Graph} \Delta^{x,y}_{w_{xy}} = \sum_{x \in Q, y \in Graph} \frac{\sum_{i \in T(x) \cap T(y)} \Delta_{w_{xy}}(i,x) + \Delta_{w_{xy}}(i,y)}{|T(x) \cap T(y)|} \tag{19}$$

Now we extract the corresponding weight vector values for each query-node pair and we compute the score with triangulated and edge dissimilarities.

Why use Pegasos and why it did not work ?

As our constraints and objective in finding the edge dissimilarity are similar to SVM equations we tried to use PEGASOS which is a very large scale SVM solver. PEGASOS is a stochastic subgradient descent algorithm for solving the optimization problem cast by Support Vector Machines. When we use a linear kernel the algorithm works on a per training example basis and if the required accuracy is e and regularization is λ, then we get a running time of $O(\frac{d}{\lambda e})$ per example where d is a bound on number of non zero features. So as we have a very large number of equations this algorithm looks particularly suitable for our purpose. But however PEGASOS gives a speedup at the cost of accuracy and we are already approximating our constraints this may not always give a good result. Also in our last approach we used the edge dissimilarity value as the scoring methodology which we may not give good results as described early. so we propose a hybrid variant in next section.

LP Based approach

We used the original constraints on a LP solver (GLPK - GNU Linear Programming toolKit - Simplex algorithm). we ran our implementation on the movielens (3952 nodes) data set and our MAP was .53 (which is still not better than the two step learning approach). Here we also used the same weight values for triangulated and edge dissimilarity value computation.

Results

We ran our system on the Movie lens data set (3952 nodes, 276 queries). Initially we used the one step QP approach. However we got very low MAP. Then we implemented the LP based approach with a limit of number of variables for the problems. Then we got a MAP value of 0.53 which is below the old

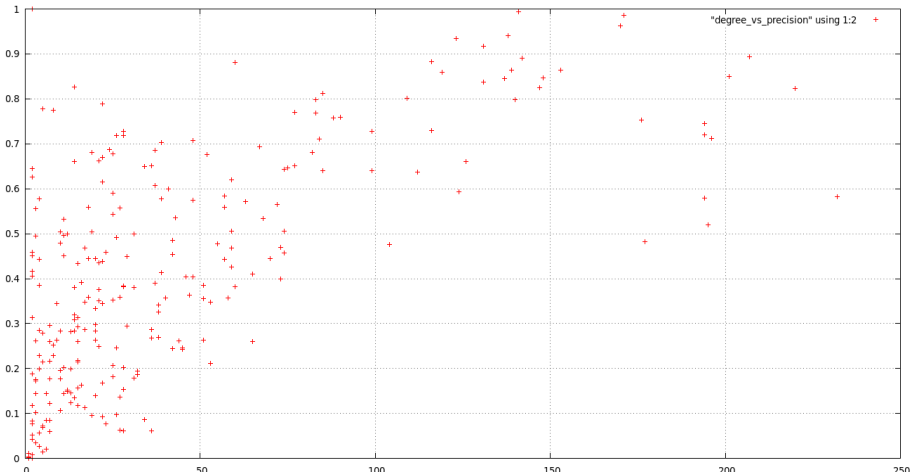

Figure 2: Precision vs Degree

approach (.70). However we also inverted the scoring which resulted in a poor MAP, which implies that our results are not random. We also plotted(Figure 2 and 3) the distribution of the precision values wrt degrees which are promising and shows that it can be improved.

Proposed Hybrid approach for further experimentation

The only QP based approach gave very poor result, but however the LP approach gave comparatively better results. So we can use the LP weights in triangulated weight calculations and QP weights in edge dissimilarity calculations which needs further experimentations.

Adding more features

feature coupling study

Tools Used

C++,Boost Graph Library,PEGASOS,GLPK,Matlab

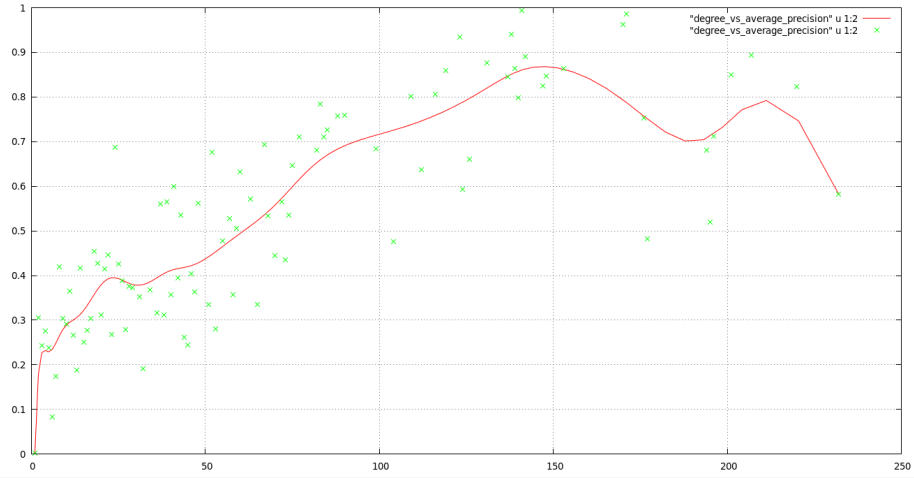

Figure 3: Average Precision vs Degree

Opinion Dynamics on Social Network

Problem Statement

The Voter Model

The proposed Model

Optimization Problem formation with Maximum Likelihood Estimation

Concave Maximization

Implementation

Comparative Results

Drawbacks of the model