Using SGD in Link Prediction Algorithms

November 2015

Master's Thesis - 1 submitted to the

Indian Institute of Technology Kharagpur

by

Sourav Sarkar

under the guidance of

Dr. Niloy Ganguly

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Declaration

This is to certify that

1. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. I have followed the guidelines provided by the Institute in writing the thesis.
4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them in the text of the thesis and giving their details in the references.

Sourav Sarkar,Department of Computer Science and Engineering
IIT Kharagpur
Date :

Introduction

In today's world, online social networks (OSN) are very important media for sharing, organizing and finding contents and contacts. Analysis of such networks not only helps to reveal many interesting properties of these networks, but also it helps to improve the present systems and to design new applications of social networks. The structure of these networks changes from time to time. Moreover a lot of processes goes on over these networks. To analyze various properties of such networks, it is important to understand the underlying dynamics and various factors that drive this dynamics.

Influence between two nodes is an important factor that drive all dynamics in a social network. Therefore in order to understand the dynamics in OSNs, the influence between two nodes needs to be modeled. However, the influence that a person has on another person significantly depends on their relationship and the context in which they are. A professor at an university may not have much influence on who her student connects to, on Facebook. But, the student may be highly influenced by her professor when it comes to which piece of literature to read next. Today's vast proliferation of online social networks (OSN) allows us to study such interactions. Here we present two learning models to address the problems of link prediction and understanding opinion propagation in OSNs.

The problem of link prediction (LP) is stated as follows: given a graph, for every vertex in the graph, find vertices to which the given vertex is most likely to form new edges. The problem of link prediction is very important, in the context of social search and recommendation. Our earlier method used two novel signals to improve accuracy in link prediction. First, we used a co-clustering algorithm to find the underlying communities in the network. We used this information to qualify edges in the graph with a surprise value. This represented how unexpected the edge is in the graph, given the underlying community structure information. Second, we computed a node to node similarity measure which takes into account the local connectivity structure between these nodes. These signals were then used in combination of others as input to a discriminative predictor to estimate likelihoods for future edges. When tested across five diverse datasets, common in link prediction literature, we find our method performs significantly better than standard link prediction methods.

However in recent times we have seen apart from just using feeding the computed node and edge scores to a top level classifier we can learn a Link function which will combine all the features. In this particular work we have mostly studied and verified the work of Aditya Krishna Menon and Charles Elkan on Link Prediction via Matrix Factorization and we propose a methodology about how we can merge both these works for a better Link Prediction Algorithm.

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.

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.

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.

Adding more features

After the previous work instead of going to the hybrid approaches as suggested earlier we tried to introduce new featur - Hitting time, Commute Time, Katz Score, Preferential Attachment, Jaccard Score, Adamic Adar, Common Neighbor score and GraphDistance [1]. Here we give a brief overview of the features. After that we have done a quadratic coupling study of features.

A random walk on G starts at a node x and iteratively moves to a neighbor of x chosen uniformly at random from the set $\Gamma(x)$ (set of direct neighbors of the node x). The hitting time $H_{x,y}$ from x to y is the expected number of steps required for a random walk starting at x to reach y.Because the hitting time is not in general symmetric, it is also natural to consider the commute time $C_{x,y} := H_{x,y} + H_{y,x}$. From [2] we find that we can easily compute hitting time matrix H according to the following formula $H = 1[diag(\widetilde{Z})]^T - \widetilde{Z}$ where $\widetilde{Z}$ is the scaled fundamental matrix as defined in [2].

In this work we have also incorporated two new features - Preferential Attachment and Katz Score. Katz Score is among the methods to refine the notion of shortest paths by considering the set of all paths between two nodes. Katz defines a measure that directly sums over this collection of paths, exponentially damped by length to count short paths more heavily. This notion leads to the measure $Score(x, y) := \sum_{l=1}^{\infty} \beta^l |paths_{x,y}^{<l>}|$ where $paths_{x,y}$ is the set of all length- paths from x to y, and $\beta > 0$ is a parameter of the predictor (A very small β yields predictions much like common neighbors, because paths of length three or more contribute very little to the summation). We can easily find a closed expression of the $Score$ matrix. By definion the $score = \sum_{l=1}^{\infty} \beta^l M^l$ (where M is the adjacency matrix of the graph). Now $Score + I = \sum_{l=0}^{\infty} \beta^l M^l = (I - \beta M)^{-1} \iff Score = (I - \beta M)^{-1} - I$ From the definition of Katz score, as we are considering all possible paths it also captures the notion of the global structure of the Graph.

Our framework also tries to capture the the local effects by introducing the Preferential Attachment feature. For a node x, let $\Gamma(x)$ denote the set of neighbors of x. Now the definition of Preferential Attachment comes from the growth model. We know if we introduce a new node then x will be one of its neighbors with probability $|\Gamma(x)|$. Now in our graphs are not temporal and as both x and y already exists so to capture their local similarity be good approach would be to define $PA(x, y) := |\Gamma(x)|.|\Gamma(y)|$.

The common neighbor score is defined as $score(x, y) := |\Gamma(x) \cap \Gamma(y)|$, the number of neighbors that x and y have in common. And graph distance is basically the all pair shortest path distance which we computed using Floyd Warshall Algorithm. For all these methods we have written MATLAB scripts.

Survey of Matrix Factorization in Link Prediction

We studied matrix factorization techniques for Link Prediction Problems. Till now we have computed features for nodes and node-node pairs. Also using the adjacency information with various techniques we have also learned the pair wise similarity vectors among all pairs. So basically in a we can write the overall link score in a generic manner like this

$$G(i,j) = min_{\theta} L(f_D(z_{ij}; w) + f_M(x_i, x_j, ; v)) \tag{12}$$

Here w and v are weight vectors that we learn from the train data by minimizing some loss function (with some proper regularizer to stop the parameters from blow up). On the contray in the matrix factorization based Latent Feature model we try to factorize the graph $G = L(UAU^T)$. Where L is a linking function applied on each of elements of the matrix. So the optimization problem can be formally posed as

$$G(i,j,U,A) = L(u_i^T A u_j) \tag{13}$$

Now as we have a linking function we can now use it for minimizing a loss function with a suitable regularizer. So this structural node, node-node scores and latent scores can be combined as a single optimization problem in the following manner,

$$U, A, w, v, b = argmin_{U,A,w,v,b} \frac{1}{|O|} \sum_{(i,j)} \in OLoss(G(i,j),$$
$$L(u_i^T A u_j + b_i + b_j + f_D(z_{ij}; w) + f_M(x_i, x_j; v))) + \Omega(U, A, w, v, b) \tag{14}$$

Where Ω is a provided regularizer, and $Loss$ and l are a given loss and regularization function. For example we can use square or logarithmic loss and sigmoid linking function. Also we can try to learn $f_D(z_{ij}; w) = w^T z_{ij}$ and $f_M(x_i, x_j; v) = v^T x_i + v^T x_j$. However to learn non linear features in a better manner we can define the $f_M(x_i, x_j; v) = x_i^T v x_j$ where V is a $d \times d$ matrix where d is the linear node feature dimension. Now v can be symmetric matrix if the given graph is symmetric. We can also approximate v as $D + A^T B$ where D is a diagonal matrix. So its obvious that we can plug our known scores in the top level optimizer problem. However this suffers from the issue of clas imbalance which is predominant in Link Prediction Datasets. So to ovrcome this we directly optimize to maximize the AUC of the ROC. To do this, we begin with the pairwise SVMRank framework . Consider a binary classification scenario with training set $\{(x_i, y_i)\}$, and let$P = \{i : y_i = 1\}$ and $N = \{i : y_i = 0\}$. The empirical AUC of a linear classifier with weight w is

$$A = \frac{1}{|N||P|} \sum_{i \in P} \sum_{j \in N} 1[w^T x_i > w^T x_J] \tag{15}$$

So the problem of optimizing w can be written as

$$w = argmin_w \frac{1}{|N||P|} \sum_{i \in P} \sum_{j \in N} 1[w^T (x_i - x_J) < 0] \tag{16}$$

We can add a suitable regularizer so that w does not overfit. So in our specific case we can reformulate the top level optimization problem as

$$U, A, v, w, b = argmin_{U,A,v,w,b} \frac{1}{|O|} \sum_{i=1}^{n} \sum_{j \in O^+ k \in O^-} Loss(\\ L(u_i^T A(u_j - u_k) + b_i + b_j + f_D(z_{ij}; w) + f_M(x_i, x_j; v)), 1) + \Omega(U, A, w, v, b) \tag{17}$$

Now we could have used gradient descent to optimize this but for a function of $Q(w) = \sum_{i=1}^{n} Q_i(w)$, the update step is $w_n = w_{n-1} - \bigtriangledown Q(w_{n-1}) = w_{n-1} - \sum_{i=1}^{n} \bigtriangledown Q_i(w_{n-1})$. As in our case we have a quadratic number of terms in the optimization function it will be difficult fot standard gradient descent. So we used Stochastic Gradient Descent. In stochastic gradient descent. We randomly shuffle the indexes of the sumation functions and we do gradient descent for that function only ie. $w_n = w_{n-1} - \bigtriangledown Q(w_{n-1})$ and in each epoch each of the sumation is taken exacly once but in a random order. We repeat the epochs until some provided convergence condition is met.

Experimental results

Tools Used

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

References

[1] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. 1993.

[2] Daniel Boley, Gyan Ranjan, and Zhi li Zhang. Commute times for a directed graph using an asymmetric laplacian. 2010.

[3] Bharath K. Sriperumbudur and Gert R. G. Lanckriet. On the convergence of the concave-convex procedure. 2009.

[4] V. Sood, Tibor Antal, and S. Redner. Voter models on heterogeneous networks. 1993.

[5] Abir De, Sourangshu Bhattacharya, Parantapa Bhattacharya, Niloy Ganguly, and Soumen Chakrabarti. Learning a linear influence model from transient opinion dynamics. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 401–410, New York, NY, USA, 2014. ACM.