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

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. However the second part of the work was also done as a part of my Complex Network Course under Dr. Animesh Mukherjee.

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 :

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:

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

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,

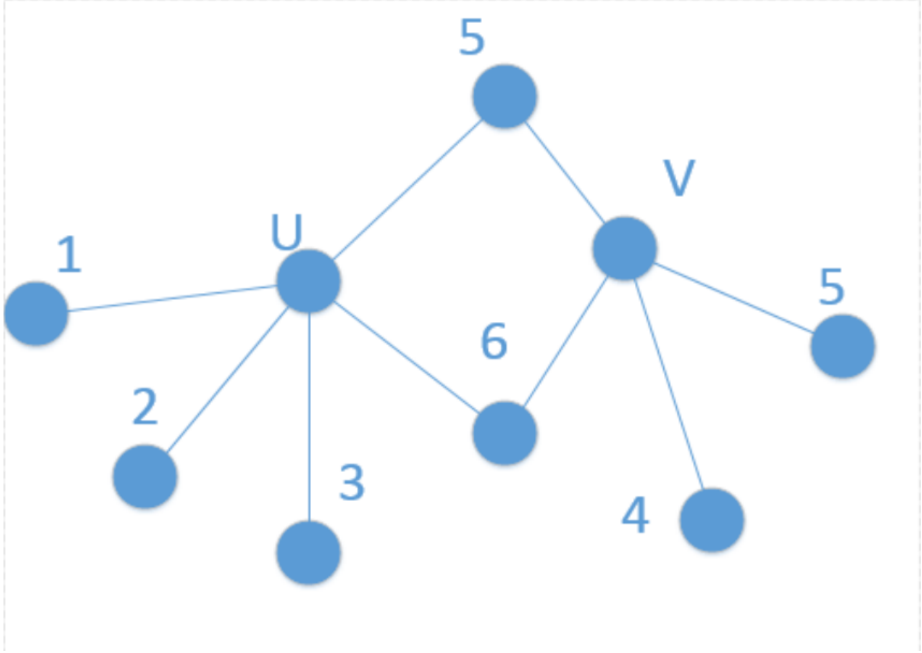

Figure 1: Sample Network

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

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

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

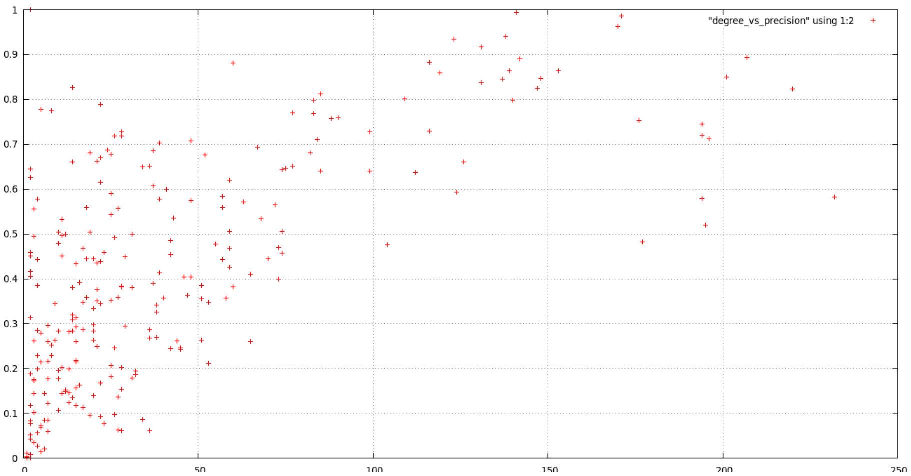

Figure 2: Precision vs Degree

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

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

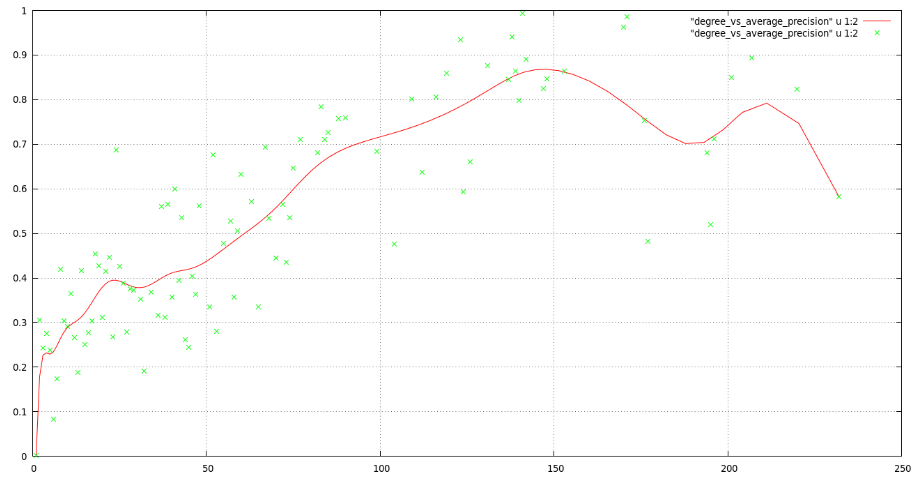

Figure 3: Average Precision vs Degree

matrix H according to the following formula $H = \dot{1}[diag(\widetilde{Z})]^T - \widetilde{Z}$ 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 part distance which we computed using Floyd Warshal Algorithm. For all these methods we have written MATLAB scripts.

Feature coupling study

With our earlier work we have introduced a lot of features. But some features may be good for classification and some may be not. Also if we train using forming vectors using only one dimension features then there is high chance that we might miss the co-relation effect between features which can be a major signal for the outer level classifier. Also as taking all these experiments would have taken a lot of manual effort so we have completely automated the process of linear and quadratic feature coupling study. Our system takes input of a bit vector for linear features and a bit Matrix for co-relation study of features.

Experimental results

In Figure 4 we show the linear feature selection study. Here on average we find that if we take the combination of LL (local learning), CC (co-clustering), KM (Katz Measure), PA (Preferential Attachment) and AA (Adamic Adar) then we get best MAP values in all data sets. In Figure 5 we show the effect of taking the quadratic terms. And we deduce that quadratic combination of the selected features found in linear study gives best performance. Finally in Figure 6 study we study the our methodology against leading algorithm and find that our old framework combined with the LL,CC,KM,PA and AA scores gives superior results.

Tools Used

The initial system for LP and QP based score calculation was built using C++ and Boost Graph Library. The later new featurs and feature combination and arbitration study was done entirely in MATLAB. For LP we used the GLPK library and for QP we used PEGASOS.

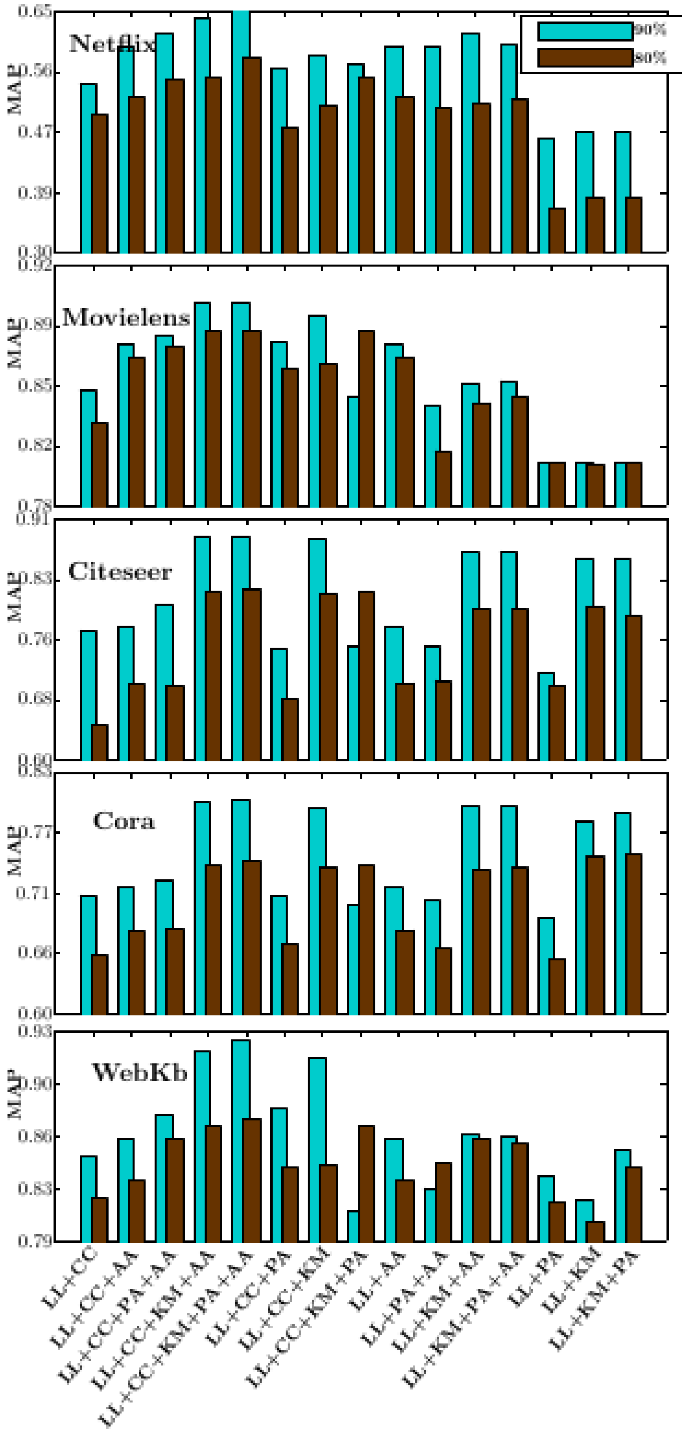

Figure 4: Feature Ablation Study with 90% and 80% Sampling

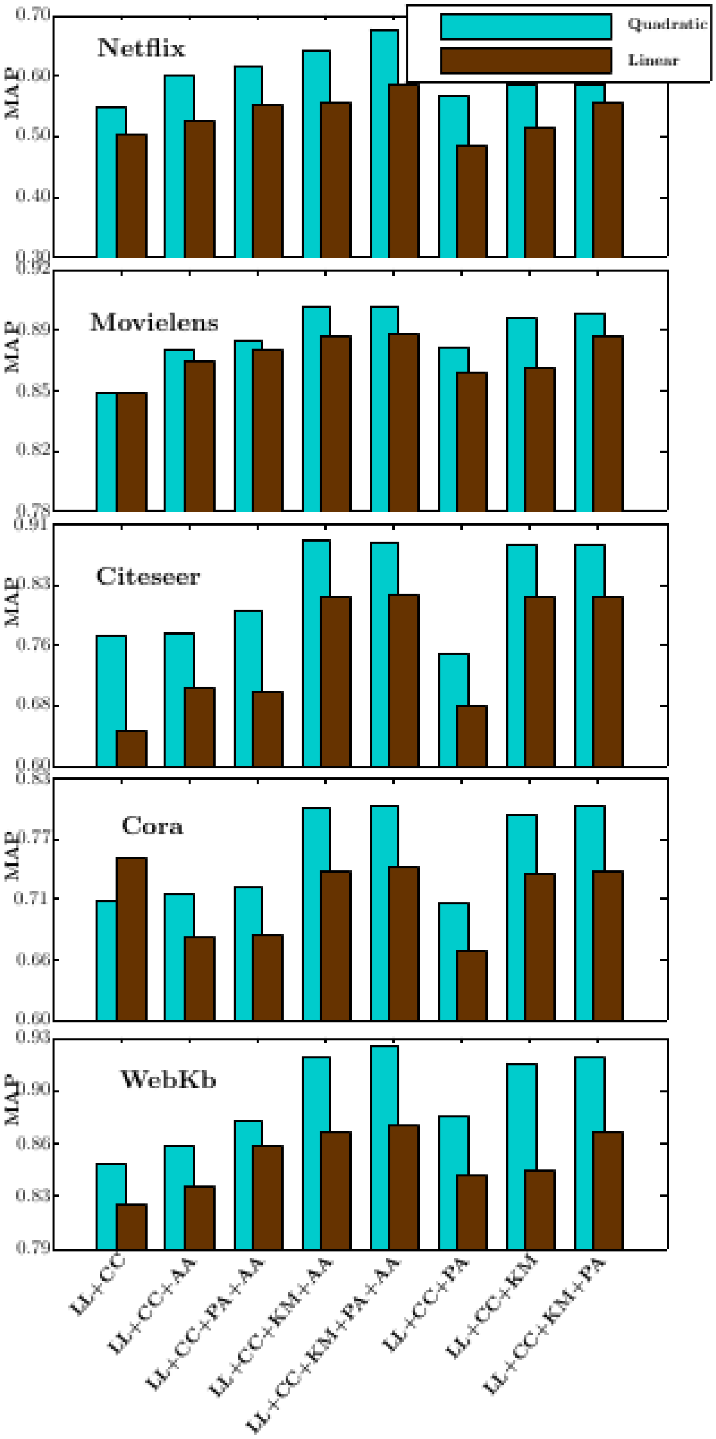

Figure 5: Effect of Quadratic Terms

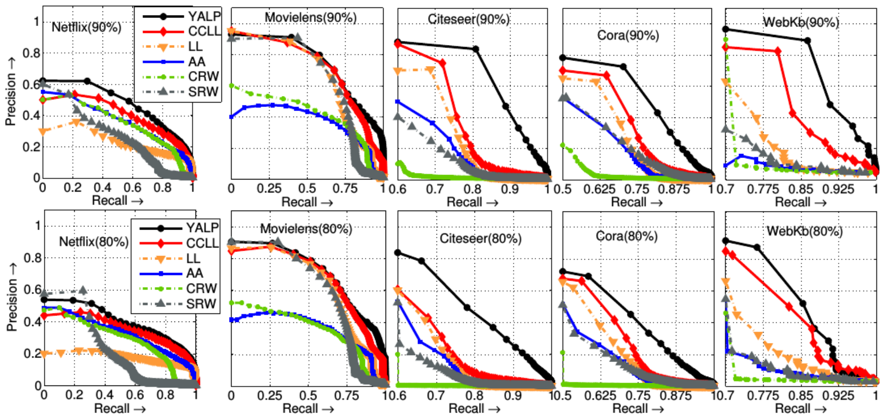

Figure 6: Precision vs. recall curves for all data sets and algorithms

Opinion Dynamics on Social Network

Introduction

Opinion Dynamics means how opinions change with time in a social network. Many social networks that we see nowadays are characterized by opinion. Here opinion means for a given node (say user in Twitter or Reddit) and a given topic (say release of an upcoming movie) we want to study how the opinions vary with time especially say before and after some specific event (like the release of the movie). Also all these social networks have an underlying network between the nodes (like friend and follower in Twitter and friends in Facebook) and opinions usually flow among these networks (like say we get influenced by the review of a movie from a friend of us and we also get the same opinion without even seeing the movie). Now in social networks nodes have many such influence but if we dont have the information about how a particular user likes content of another particular influence then they are just binary - influences or not. But in real social network we have preference of opinion among our friends and also this may be both positive and negative (like if we already know that our movie choices does not match with our friends then our opinion on a particular movie is likely to be opposite).

So in this work we take input a network of nodes (with a directed follower relationship) a set of opinions of the particular user on a particular topic in a time window. Now given this data we try to qunatize the influences between the nodes. Also another questions that we may ask are say given just the opinion values can we infer the influence network (completely unsupervised) and also after quantization can we use our model to predict future opinion sentiments of the nodes

So typical usage scenerio can be in Election poll forcasting where we have the opinion values before election and we can predict the results. This can also be used while the campaign is going on to test whether the social media is coming on our side or not. It also has vast application in E-commerce. For example if an ecommerce is trying to launch two products together say Galaxy S6 and Iphone 6 and the ecommerce firms can gather the social data and infer which product is going to lead and restock accordingly and this is possible because we often get influenced by the transactions and reviews made by our peers. Also another optimistic usage can be 'review manipulation'. For example if we want to sell more IPhones then given the model we set our target opinion and use our model to find an initial condition which might lead to this final state. Once we find the initial conditions we can incentivize nodes according to the finding and if it behaves according to our learned model then its a huge gain for such ecommerce firms.

Formal Problem Definition

We have a directed binary influence matrix N i.e. if $N(x, y) = 1 \iff x$is following y and $N(x, y) = 0 \iff x$ is not following y. Also we have a set of opinion values for each node in a time window. So we will divide the opinions in

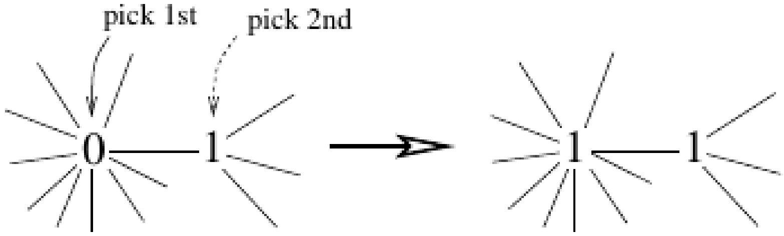

Figure 7: Opinion Dynamics in Voter Model

two sets by fixing a time separator. Opinion before the separator are in ’training’ and after that are in ’test’. So using the training data we will learn the quantified Influence Matrix A and will try to predict opinions in the ’training’ set.

The Voter Model

Many models have been proposed on how opinions vary over time on a social network. One such particular model is Voter Model which is simple to describe. Steps are described in Figure 7. It consists of the following simple steps - (i) pick a random node (a voter),(ii) the voter adopts the state of a random neighbor.

The proposed Model

Now suppose in the train window for node x we have the set of opinions $S_x = \{(O_i, t_i) | 1 \leq i \leq k\}$. Now using voter model for every node we want to maximize $P(S_x|A)$. Now to effectively use voter model need to define the following quantities. Now we define n_t as the opinion vector at time instance t. Now in real social networks nodes dont advertise their opinion at every timestamp. So we also define n_{t-} as the opinion vector of the nodes strictly before the time t. We also define $n_{t,x}$ in the following manner,

$$n_{t,x}(y) = \begin{cases} n_{t-}(y) & \text{if } x \neq y \\ 1 - n_{t-}(y) & \text{if } x = y \end{cases} \tag{20}$$

as our opinion values are discrete and we model them as 1 or 0 so negation with 1 just complements the opinion. Now from the voter model we know that the Probability that a paricular node will change in a given a timestamp is nothing but

$$P(n_{t-} \rightarrow n_{t,x}) = \sum_{y} \frac{N_{x,y}}{N k_x} \xi(x, y, t) \tag{21}$$

where k_x is the number of influences of the node x if we assume a complete uniform voter model. Ansd the $\xi(x, y, t)$ is an indicator function which is defined as follows,

$$\xi(x,y,t) = \begin{cases} 1 & \text{if } n_{t-}(x) \neq n_{t-}(y) \\ 0 & \text{if } n_{t-}(x) = n_{t-}(y) \end{cases} \tag{22}$$

Optimization Problem formation with Maximum Likelihood Estimation

So in our proposed framework to come up with a MLE expression interms of A we can use

$$P(n_{t-} \to n_{t,x}) = \sum_{y} \frac{A_{x,y}}{N} \xi(x,y,t) \tag{23}$$

where $\sum\limits_{y} A_{xy} = 1$. Also note that we are only constrained by the xth row of A while while solving for x and those parameters are in no other MLE as our graph is directed. So we can solve for each of the nodes separately and though this would take up a huge time but it is scalable as each of the opimization problems are tractable and all of them can done parallely accross distributed systems. Also from this we can say

$$P(\text{node } x \text{ does not flip at time } t) = 1 - P(n_{t-} \to n_{t,x}) = 1 - \sum_{y} \frac{A_{x,y}}{N} \xi(x,y,t) \tag{24}$$

Using this definitions we can write

$$P(S_x|A) = P(n_{t_i}(x) = O_{t_i}(x), \forall t_i, 0 \leq i \leq k|A) \tag{25}$$

where $O_{t_i}(x)$ is the observed opinion of node x at time instant t_i. Hence

$$\begin{aligned} P(S_x|A) &\propto \prod_{i}^{k} P(n_{t_1}(x) = O_{t_i}(x)|A, n_{t-}) P(n_{t_0}|A) \\ &\propto \prod_{i}^{k} P(n_{t_1}(x) = O_{t_i}(x)|A, n_{t-}) \\ &= \prod_{i=1}^{k} (1 - \sum_{y} \frac{A_{x,y}}{N} \xi(x,y,t_i))^{I(n_{t_i}(x) = n_{t_i -}(x))} \prod_{i=1}^{k} (\sum_{y} \frac{A_{x,y}}{N} \xi(x,y,t_i))^{I(n_{t_i}(x) \neq n_{t_i -}(x))} \end{aligned}$$

Now by taking logarithm on both sides we find

$$\begin{aligned} L &= ln(P(S_x|G)) \\ &\propto \sum_{i=1}^{k} I(n_{t_i}(x) = n_{t_i -}(x)) ln(1 - \sum_{y} \frac{A_{x,y}}{N} \xi(x,y,t_i)) + \\ &\quad \sum_{i=1}^{k} I(n_{t_i}(x) \neq n_{t_i -}(x)) ln(\sum_{y} \frac{A_{x,y}}{N} \xi(x,y,t_i)) \end{aligned}$$

So our optimization problem is given maximize L given $\sum_y A_{xy} = 1$.

Concave Maximization

Here we will show that our problem is concave maximization problem. Basically L is the summation of a log-linear function. Now for a function f to be concave we must have $Z^T \nabla^2 f Z \leq 0 \forall Z$.Also if $Z^T \nabla^2 f_1 Z \leq 0 \forall Z$ and $Z^T \nabla^2 f_2 Z \leq 0 \forall Z$, then $Z^T \nabla^2 (f_1 + f_2) Z \leq 0 \forall Z$. So if we can prove the log-linear function

$$f = ln(\sum_i a_i x_i + c)$$

is concave for $a_i, c \in \Re$ then we are done. Let

$$M = \sum_i a_i x_i + c$$

Now its easy to see that

$$\nabla f = [\frac{a_i}{M}]_{1 \times k}$$

and we can also show that

$$\nabla^2 f(i,j) = -\frac{a_i a_j}{M^2}$$

Now if we define $D = [a_1, a_2, ..., a_n]^T$. Then We can write

$$\nabla^2 f = \frac{1}{M^2} D D^T$$

So for $Z \in \Re^n$ we have

$$\begin{aligned} Z^T \nabla^2 f Z &= -\frac{1}{M^2}(Z^T A)(Z^T A)^T \\ &\leq 0 \end{aligned}$$

Hence the given optimization problem is a concave maximization and it can be easily solved numerically.

Implementation

Now our optimization problem can be solved with gradient ascent, there are better methods. So this algorithm is implemented in mosek toolbox. For some experiments we used the given dataset, but we also developed modules to gather tweet stream from Twitter and extract friend follower information from Twitter. We also generated modules to generate synthetic data. We had to do many

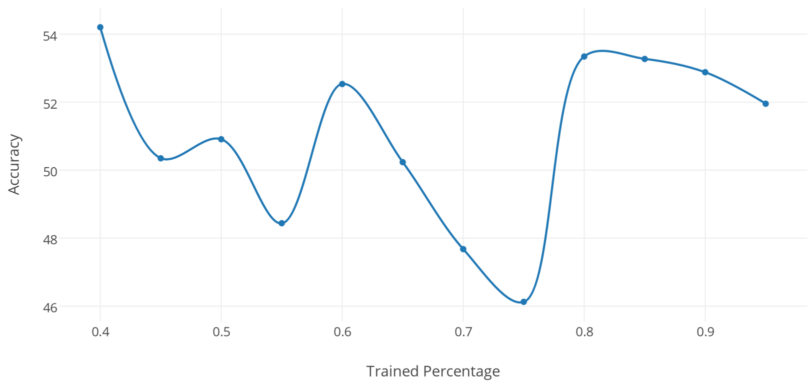

Figure 8: Effect of time window variation on Twitter

experiments by varying the time window that separates the train and test set and we also automated that. Now using the apis provided by the mosek api we developed modules to solve the maximum likelihood problems. For testing part, after learning the influence values we ran stochastic simulation on the test data and computed average output. We define accuracy as the overall fraction of opinion matches accross all the nodes in a particular simulation and we report the average one for several simulation with a particular time window.

Experimental Results

In Figure 10 we show how the accuracy varies accoring to the movement of the time window. In Figure 8 we show it for the same. It seems that if we incrase the training data then the accuracy of our model increases in case of Reddit. But we don't find any recognizable trend in the Twitter Dataset. Which implies our model is not working on the Twitter dataset. This requires further examination.

Drawbacks of the model

Now there may be various drawbacks in the model. One thing we observed that though we get good accuracy in Reddit dataset, the model that we generate is too much sparse unlike a real network. Also as we showed that the optimization problem is a concave maximization, but its not strictly concave one. So there may be various solutions given a test data and we might not get always lucky to pick the model that gives the best test accuracy. For example we may have

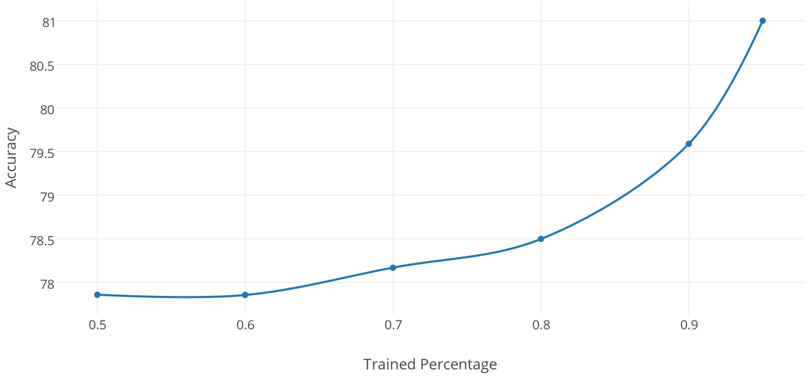

Figure 9: Effect of time window variation on Reddit

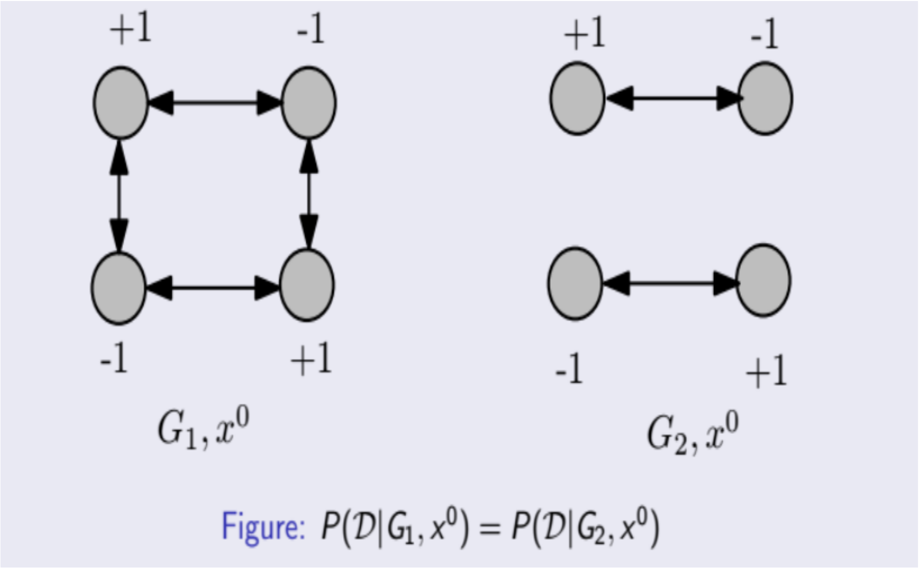

Figure 10: Multiple solutions given same test Data

these two graphs G_1 and G_2. Now let the nodes be a, b, c, d in clock wise sense. Now in G_1 node a can influence by node b, d but only from b in G_2. But if in training we just observe that a has flipped then we can distinguish between the two cases unless we get more observation. However observations can be very sparse as Commercial Social Networking sites like Twitter or Reddit does not give all the opinions while crawling programatically. Also as it shown G_1 is more homogenious but we want heterogenious solutions. But in our model we do not have any extra optimization criteria to ensure this.

Towards a better model

Also as we talked earlier our model has n^2 parameters to learn where number of nodes is n. Also there as we are solving a node by node basis so we might miss the herd mentallity where opinions of a large set of nodes change together. So to tackle these issues one interesting approach might be transforming the problem to a kernel space where we will change $a_{ij} \rightarrow K(x_i, x_j)$. Now if we notice the number of model parameters is n and we can solve the problem in a single turn. Also the most interesting part is that is approach will give us a model value for each node which we can use as a distribution and which if we are lucky might follow some well studied distributions like the power law distribution.So now we frame our optimization problem as

$$\begin{aligned}\text{maximize } L &= log(P(S|A)) \\ &= log(\prod_{i \in V}[\prod_{k \in F_i}[\sum_{j \neq i} a_{ij}\xi(i,j,t_k)] \prod_{k \in \widetilde{F_i}}[1 - \sum_{j \neq i} a_{ij}\xi(i,j,t_k)]]) \\ & \quad \text{subject to } \sum_j a_{ij} \leq 1 \forall i \in V\end{aligned}$$

Now this can be simplified as maximize

$$\begin{aligned}L &= \sum_{i \in V}[\sum_{k \in F_i} log[\sum_{j \neq i} a_{ij}\xi(i,j,t_k)] + \sum_{k \in \widetilde{F_i}} log\lambda_i^k] \\ & \quad \text{with the additonal constraints } \lambda_i^k + \sum_{j \neq i} a_{ij}\xi(i,j,t_k) = 1 \forall k \in \widetilde{F_i} \forall i \in V\end{aligned}$$

Now if we take the kernel function $K(x_i, x_j) = K(x_i)K(x_j)$ where $K(x_i) = e^{-\theta_i}$ and θ_i is the global influence parameter of node i then we can reframe the opimization problem as maximize

$$L = \sum_{i \in V}[\sum_{k \in F_i} log[\sum_{j \neq i} e^{-\theta_i - \theta_j}\xi(i,j,t_k)] + \sum_{k \in \widetilde{F_i}} log\lambda_i^k]$$

which can be further simplified as

$$L = -\sum_{i \in V}\sum_{k \in F_i}\theta_i + \sum_{i \in V}\sum_{k \in F_i} log[\sum_{j \neq i} e^{-\theta_j}\xi(i,j,t_k)] + \sum_{i \in V}\sum_{k \in \widetilde{F_i}} log\lambda_i^k$$

$$\text{with the constraints } \sum_{j \neq i} e^{-\theta_i - \theta_j} \leq 1 \forall i \in V$$

$$\text{and } \lambda_i^k + \sum_{j \neq i} e^{-\theta_i - \theta_j} \xi(i, j, t_k) = 1 \forall k \in \widetilde{F_i} \forall i \in V$$

Now we will show that $f := log(\sum_{i=1}^{n} a_i e^{m_i x_i})$ is convex where $m_i \in \Re$ and $a_i \geq 0$. Define

$$z := \sum_{i=1}^{n} a_i e^{m_i x_i}$$

Hence we can show that

$$\nabla f(i) = \frac{a_i m_i e^{m_i x_i}}{z}$$

Now we can compute the Hessian in the following manner

$$\nabla^2 f(i, j) = \begin{cases} -\frac{a_i^2 m_i^2 e^{2m_i x_i}}{z^2} + \frac{a_i m_i^2 e^{m_i x_i}}{z} & \text{if } i = j \\ -\frac{a_i a_j m_i m_j e^{m_i x_i + m_j x_j}}{z^2} & \text{if } i \neq j \end{cases}$$

Lets define $D = [a_i m_i e^{m_i x_i}]_{n \times 1}$. Now we can write

$$\nabla^2 f = \frac{1}{z^2}[z Diag(a_i^2 m_i^2 e^{m_i x_i}) - DD^T]$$

Now to show the convexivity,

$$\begin{aligned} Z^T \nabla^2 Z &= \frac{1}{z^2}[z(\sum_{i=1}^{n} Z_i^2 a_i m_i^2 e^{m_i x_i}) - (\sum_{i=1}^{n} z_i a_i m_i e^{m_i x_i})] \\ &= \frac{1}{z^2}[(\sum_{i=1}^{n} a_i e^{m_i x_i})(\sum_{i=1}^{n} Z_i^2 a_i m_i^2 e^{m_i x_i}) - (\sum_{i=1}^{n} z_i a_i m_i e^{m_i x_i})] \\ &\geq 0 \text{ (using Cauchy–Schwarz Inequality)} \end{aligned}$$

Hence $f = log(\sum_{i=1}^{n} a_i e^{m_i x_i})$ is convex for all $a_i \geq 0, m_i \in \Re$. Now Lets define $M = -L$. Hence

$$\begin{aligned} M &= \sum_{i \in V} \sum_{k \in F_i} \theta_i - \sum_{i \in V} \sum_{k \in F_i} log[\sum_{j \neq i} e^{-\theta_j} \xi(i, j, t_k)] - \sum_{i \in V} \sum_{k \in \widetilde{F_i}} log \lambda_i^k \\ &= f_1 - f_2 \end{aligned}$$

where f_1 is defined as

$$f_1 = \sum_{i \in V} \sum_{k \in F_i} \theta_i - \sum_{i \in V} \sum_{k \in \widetilde{F_i}} log \lambda_i^k$$

and f_2 is defined as

$$f_2 = \sum_{i\in V}\sum_{k\in F_i} log[\sum_{j\neq i} e^{-\theta_j}\xi(i,j,t_k)]$$

Now clearly both f_1 and f_2 are convex and the constraints are convex. So they can be solved by a iterative algorithm given in [3]. However the solution point is given there need not be the global maxima, it can be a saddle point also. So we need to look for better kernel function which gives us better mathematical behaviour.

Now to take care of our semidefinite problem, we need to add a regularizer.But if we add the entropy regularizer defined as $\lambda\sum_x\sum_y a_{xy}log(a_{xy})$ will not be a good idea because it does not try to make the model heterogenious nor sparse.A very common approach to learn sparse graph is to add an L_1 regularizer.Although addition of such regularizer can preserve the convexity, the problem becomes difficult to solve. Therefore we choose the following regularizer to incorporate sparsity

$$L \quad = \quad \sum_{i\in V}[\sum_{k\in F_i} log[\sum_{j\neq i} a_{ij}\xi(i,j,t_k)] + \sum_{k\in\widetilde{F_i}} log\lambda_i^k] - \lambda\sum_{i\neq j}\frac{1}{1-a_ij}$$

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.