Learning Influence Diffusion Probabilities under the Linear Threshold Model

Sharan Vaswani and Nayantara Duttachoudhury

Computer Science Department, University of British Columbia, Vancouver, BC, Canada
{sharanv,nduttac}@cs.ubc.ca

Abstract. Influence maximization techniques assume the knowledge of influence diffusion probabilities which are however not known in most real world networks. In this paper, we address the problem of learning these probabilities from available cascade diffusion information. In particular, we learn these diffusion probabilities for influence diffusion under the Linear Threshold model. We use 3 optimization strategies namely gradient descent, interior point methods and expectation maximization. We evaluate the efficacy and feasibility of our approach on synthetic and real world networks. Our preliminary results are encouraging and demonstrate that the proposed methods work well.

1 Introduction

The growth of the internet in the recent years led to the formation of a large scale social networks. The spread of information in social networks has been studied by many researchers [5,7,11,14,15, 18]. In social networks, users affect other users to which they are connected to and information propagates through the network. Influence propagation is the diffusion of information through the network. For example, if a user buys a certain product after one of his 'friends' buys the same product, he is said to be influenced by the friend. The more impact a user has on the people he is connected to, the more is his influence. In social networks, the weight (or diffusion probability) of an edge is a measure of the influence from one user to another. The higher the influence, greater is the value of the diffusion probability.

Probabilistic models such as Independent Cascade (IC) and the Linear Threshold (LT) [10] have been used to study the spread of influence. Under these models, a user is said to be active if he adopts a certain product or technology. In the *Independent Cascade Model* [11], each active neighbor v of u gets exactly one chance at activating u and succeeds with a probability of $P_{v,u}$. The active neighbors can try to activate u in any order. When no new nodes can be activated, the diffusion process ends. In the *Linear Threshold Model* [22], all the nodes are randomly assigned threshold values $\theta \in [0, 1]$. A node can get activated only if the sum of all incoming edge weights is greater than its node threshold. Like the IC Model, the process stops when no new nodes can be activated.

The concept of influence diffusion can be exploited by marketers. Viral marketing [5, 15, 19] uses social networks by encouraging consumers to share product information with their friends. Marketers employ influence maximization techniques to target a small number of influential individuals and trigger a cascade of influence by which friends recommend the product, promote the innovation, or propagate the topic to other friends [14, 19]. Influence maximization is the problem of finding a small subset of nodes (seed nodes) in a network that can maximize the spread of influence [12]. Influence maximization under both the IC and LT models requires diffusion probabilities to be known in advance. This information may not be available for real networks. Diffusion probabilities can to be estimated from a set of information diffusion cascades that are observed as time-sequences of activated nodes. [8, 13, 21]. [3] shows a high sensitivity

of influence maximization on the values of diffusion probabilities. In this paper, we discuss a method of estimating the diffusion probabilities under the LT Model.

The paper is structured as follows. Section 2 discusses related work. We give a mathematical formulation of the problem in section 3 explain the solution methodology in section 4. Section 5 discusses the experimental setup section 6 presents the results. We conclude the paper in section 7 and outline future research directions in section 8.

2 Related Work

The influence maximization described in the previous section is an NP-hard problem under both the Independent cascade (IC) and Linear Threshold (LT)models [10]. The influence function $f(A)$ which maps a set of initially active (seed) nodes to the expected number of nodes that are activated at the end of the diffusion process (spread) is submodular under both models. The submodularity has been exploited to develop a greedy algorithm which iteratively selects a node that gives maximum margin on $f(A)$ [10]. More effective approaches for influence maximization have been proposed in [4,9,16]. But all of these techniques assume the prior knowledge of diffusion probabilities. A more principled approach is to use learn these probabilities from previous cascade information.

Saito et all [20] uses the EM algorithm to predict diffusion probabilities. Although the algorithm does reasonably well in predicting these probabilities, this approach is restricted to the IC model and is difficult to scale for large datasets. Diffusion probabilities can also be learnt from past propagation(action logs). In [8], static and time-dependent models were proposed for capturing influence and algorithms for learning the various parameters of the model were presented. The authors conduct extensive experiments and present results for a large number of real world datasets. However, this paper assumes the availability of action logs. This data is usually held by the social network site and not immediately accessible to outsiders. The problem of predicting diffusion probabilities under the LT model has been studied by Cao et.al in [3]. They defined the problem of finding the model parameters for influence maximization as an active learning problem. A weighted sampling algorithm was developed to solve the active learning problem. Furthermore, [21] address the problem of estimating diffusion probabilities of a probabilistic model as a function of node attributes from the observed diffusion data. Fang et.al [6] developed a locally weighted EM algorithm to predict adoption probabilities on the basis of factors such as social influence, structural equivalence, entity similarity, and other confounding factors.

3 Problem Formulation

In this section, we define the mathematical notation and formulate the problem in terms of a likelihood function. Consider an edge from node v to w in a network. This edge has a diffusion probability $k_{v,w}$ which represents the strength of the link between v and w. Higher the diffusion probability, higher is the probability that node v influences w. As explained in the previous section, we try to infer the diffusion probabilities according to the past cascades which have diffused in the network. We know the precise trace (i.e. which node was influenced at what time) for each past cascade. The total number of such cascades (which we call train cascades) is S. The time when the diffusion process for cascade s ends is represented by $T(s)$. For representing the structure of the network, we use the variables $B(w)$ and $F(w)$. $B(w)$ represents the set of nodes which have an edge to the node w i.e. parent nodes of w whereas $F(w)$ represents the set of nodes to which the node w has an edge i.e. children of the node w. We represent the set of nodes which become active at time t under the cascade s as $D_s(t)$. The cumulative set of nodes

which have become active by the time t are represented by $C_s(t)$. Hence we have the following relation, $C(T) = \bigcup_{t=1}^{t=T} D(t)$ i.e. the nodes active at time T became active before or at time $t = T$.

Let $P_w^s(t+1)$ represent the probability that the node w becomes active at time $t+1$ under the diffusion cascade s. For deriving $P_w^s(t+1)$, we use the live edge model. Under the live edge model, each inactive node w picks at most one incoming edge with a probability equal to the edge weight. An edge is considered to be live only if it is selected. The node w has also a certain non-zero probability of choosing no edge under the live edge model. This results in a subgraph (called a possible world) with all the nodes and only the live edges. In each world, an inactive node w at time t becomes active in the subsequent time step if it is directly connected to an active node v. In this way, the cascade s diffuses in each possible world. The above process is repeated multiple times and results in a distribution over the worlds. Hence the resultant spread is the expected number of nodes reachable from the initial seed set of active nodes S over all the possible worlds. This model was shown to be equivalent to the linear threshold model by Kempe et.al [10].

Since we take an expectation across all possible worlds, the probability that w selects the edge (v, w) as the live edge is exactly equal to the diffusion probability $k_{v,w}$. $P_w^s(t+1)$ is high if it has a greater number of active neighbors i.e. nodes belonging to the set $B(w) \cap C_s(t)$. Hence the probability that the node w becomes active is given by $P_w^s(t+1) = \sum_{v \in B(w) \cap C_s(t)} k_{v,w}$. We can also define the probability that the node w does not become activated at timestep $t+1$ as $P_{\bar{w}}^s(t+1)$. This can be defined as $P_{\bar{w}}^s(t+1) = 1$ - $P_w^s(t+1) = 1 - \sum_{v \in B(w) \cap C_s(t)} k_{v,w}$ which is equal to the probability that the node w selects no edge as the live edge.

Now we derive the likelihood function for explaining the train cascades. Let θ be the vector of all diffusion probabilities. θ is the parameters we need to estimate. The intuition for the likelihood can be explained as: the activation of a node w at time $t+1$ can be explained by the likelihood function as $P_w^s(t+1)$ whereas the fact that it remains inactive at timestep $t+1$ is incorporated into the likelihood by the term $P_{\bar{w}}^s(t+1)$. This needs to be done for all the inactive nodes at timestep t (since once a node becomes active, it does not become inactivate and hence the likelihood function need not explain these nodes). Hence for a particular timestep t for a particular cascade s, the likelihood function can be given by equation 1.

$$L(\theta, t, s) = \prod_{w \in D_s(t+1)} [P_w^s(t+1)] \prod_{w \notin C_s(t+1)} [P_{\bar{w}}^s(t+1)] \tag{1}$$

Since we need the likelihood to explain each cascade s and each time-step t in the cascade, we take the product across all timesteps and cascades to get equation 2.

$$L(\theta) = \prod_{s=1}^{S} \Big[\prod_{t=0}^{T(s)-1} \Big[\prod_{w \in D_s(t+1)} [P_w^s(t+1)] \prod_{w \notin C_s(t+1)} [P_{\bar{w}}^s(t+1)] \Big] \Big] \tag{2}$$

Like always we work with the log-likelihood function, which is given by equation 3.

$$logL(\theta) = \sum_{s=1}^{S} \Big[\sum_{t=0}^{T(s)-1} \Big[\sum_{w \in D_s(t+1)} [log(\sum_{v \in B(w) \cap C_s(t)} K_{v,w})] \Big] + \Big[\sum_{w \notin C_s(t+1)} [log(1 - \sum_{v \in B(w) \cap C_s(t)} K_{v,w}] \Big] \Big] \tag{3}$$

Now that we have formulated the likelihood function, we find the maximum likelihood estimator i.e. we need to find $\theta^* = \text{argmax MLE } \log(L(\theta))$

4 Solution Methodology

We use 3 different methods to for finding the maximum likelihood estimate for the log likelihood derived in equation 3. In this section, we briefly describe each of the optimization strategies.

4.1 Gradient Descent

Gradient Descent is one of the simplest ways of maximizing a function. We initialize gradient descent to random estimate of diffusion probabilities.To maximize a function, we iteratively find the gradient of the function at the current point and move in the direction of the gradient with a certain step-size. Since at each iteration, we move in the direction in which the function increases the most, we converge to a local maxima. Equation 4 formally presents the notion described above.

$$\theta^{i+1} = \theta^i - \eta g(\theta^i) \tag{4}$$

where i is the current gradient descent iteration, $g(\theta^i)$ represents the gradient at current estimate θ of the diffusion probabilities and η gives the current step size. Equation 5 gives the expression for $g(\theta^i)$. In this expression, $\lambda_{t,s}(v,w) = 1$ iff $v \in C_s(t) \cap B(w)$ i.e. v is an active neighbor of w in the cascade s at time t. Also $\delta_{t,s}(w) = 1$ iff $w \in D_s(t+1)$ i.e w gets activated at time t in the cascade s.

$$g(\theta^i) = \sum_{s=1}^{S} \Bigg[\sum_{t=0}^{T-1} \Big[\frac{\delta_{t,s} - P_w^s(t+1)}{P_w^s(t+1)(1 - P_w^s(t+1))} \lambda_{t,s}(v,w) \Big] \Bigg] \tag{5}$$

The step size can be constant across iterations or can be decreased by a constant factor as the number of iterations increase. More sophisticated ways of choosing the step size include techniques like backtracking line search [24] integrated possibly with BB step [1]. Multiple restarts can be used to alleviate the problem of convergence to local maxima.

4.2 Interior Point methods

We also investigate the use of interior point methods for maximizing the log likelihood. Interior point methods approach the optimal solution from the interior of the feasible set. The feasible set is defined as part of the parameter space which satisfies the constraints. Interior point methods use a barrier function like log-barrier to encode the constraints of the problem. They generally use gradient information and Newton's method to solve the resulting equation. Interior point methods give the theoretical guarantee that the number of iterations of the algorithm is bounded by a polynomial in the dimension and accuracy of the solution. [23] [2] give a detailed explanation of interior point methods.

4.3 Expectation Maximization

Next we describe the technique of Expectation Maximization (EM) as a means of maximizing the log likelihood. This technique was also used in [20] for learning the diffusion probabilities under the independent cascade model. EM is an iterative method for finding the MLE of a likelihood function. EM iteration alternates between performing an expectation (E) step and the maximization (M) step. The E step creates a function for the expectation of the log-likelihood over latent variables. This is evaluated using the current estimate for the parameters. The M step computes parameters maximizing the expected log-likelihood found in the E step. These

parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

In our formulation of the log likelihood, there are no explicit latent variables. Instead we treat the probabilities $P_w^s(t+1)$ as the latent variables and the parameters to be estimated are the diffusion probabilities $k_{v,w}$. Hence the EM algorithm is given by:

1. E step: Calculate the expected value of the log likelihood function $\mathrm{L}(\theta)$, with respect to the conditional distribution of the latent variables $P_w^s(t+1)$ given under the current estimate of the parameters $k_{v,w}$

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \mathrm{E}_{Z|\mathbf{X},\boldsymbol{\theta}^{(t)}}\left[\log L(\boldsymbol{\theta};\mathbf{X},\mathbf{Z})\right] \tag{6}$$

2. M step:

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \tag{7}$$

The E and M steps are alternatively done until a certain maximum number of iterations has been reached or the change in the parameter estimates becomes less than a certain threshold. The EM algorithm is sensitive to the initial estimate and is also prone to converge to a local maxima. Like in gradient descent, multiple restarts are used to alleviate this problem.

The Q function for the log-likelihood derived in 3 can be given by equation 9.

$$Q(\theta|\hat{\theta}) = \sum_{s=1}^{S}\Bigg[\sum_{t=0}^{T(s)-1}\Bigg[\sum_{v\in C_s(t)}\Big[\sum_{w\in D_s(t+1)\cap w\in F(v)}\frac{\hat{k}_{v,w}}{\hat{P}_w^s(t+1)}log(k_{v,w}) + (1-\frac{\hat{K}_{v,w}}{\hat{P}_w^s(t+1)})log(1-K_{v,w})\Big] \tag{8}$$

$$+\Big[\sum_{w\notin C_s(t+1)\cap w\in F(v)} log(1-k_{v,w})\Big]\Bigg]\Bigg] \tag{9}$$

where $\hat{\theta}$, $\hat{k}_{v,w}$ and $\hat{P}_w^s(t+1)$ represent the current estimates of the quantities θ, $k_{v,w}$ and $P_w^s(t+1)$.

As shown above, the M step consists of maximizing the Q function with respect to the parameters θ. In this case, we can find a closed form solution for the maxima of the Q function. This is given in equation 10.

$$k_{v,w} = \frac{1}{|S_{v,w}^+| + |S_{v,w}^-|}\sum_{s\in S_{v,w}^+}\frac{\hat{k_{v,w}}}{\hat{P_w^s}} \tag{10}$$

where $S^+(v,w)$ refers to the set of cascades for which the condition $v \in C_s(t)$ and $w \in D_s(t+1)$ is satisfied and $S^-(v,w)$ refers to the set of cascades for which $v \in C_s(t)$ and $w \notin C_s(t+1)$. $|S|$ refers to the number of elements in S.

5 Experimental Setup

In this section we describe the experimental setup used to benchmark our results. In particular, we describe each aspect of the setup - description of the datasets used, implementation details, the algorithms considered and the evaluation metrics used.

5.1 Datasets

For our preliminary experiments, we use 3 datasets i.e. 3 kinds of graphs are used to benchmark our results. We generate 2 synthetic graphs and use 1 real dataset.

1. Random graph: Given the number of nodes and the sparsity in the graph, we generate edges between a pair of randomly selected nodes.

2. Forestfire: Forestfire model generates networks that densify and have shrinking diameters. We used Stanford Network Analysis Platform (SNAP) to generate graphs according to the forestfire model. The parameters which need to be specified are the number of nodes, the forward and backward burning probabilities. More details can be found in [17].

3. Facebook: It is available in the SNAP library. The facebook network we use has 347 nodes and 5038 edges.

5.2 Implementation Details

Graph generation: We use the structure of the network from the datasets described above. However to fully specify the ground truth network, we also need the diffusion probability for each edge in the network. Since we don't know the true diffusion probabilities for any of the three graphs described above, we randomly assign the diffusion probabilities in the range (0,1) subject to a constraint. The constraint is that the net influence inflow into any node should be less than 1. Hence if there are 3 edges into a node, each edge is assigned a true diffusion probability between 0 and 0.33.The algorithm aims to recover these diffusion probabilities.

Cascade generation: Since 2 of the graphs are synthetic and the facebook dataset too has no data about the train cascades, we need to artificially generate the cascades. This is done by randomly selecting a set of seed nodes. With these seed nodes and the true diffusion probabilities, the cascades are allowed to diffuse through the network under the linear threshold model. For this, we randomly generate threshold values between (0,1) for each node under each cascade. The probability of selecting a seed nodes is proportional its out-degree. This is to ensure we get cascades which diffuse over a larger number of edges and hence we obtain information about a larger number of diffusion probabilities.

Initialization: We need values of the diffusion probabilities as initial estimates for each of the algorithms described in section 4. These are generated randomly in a similar way to the true diffusion probabilities.

5.3 Algorithms

We use the 3 algorithms described in section 4 namely gradient descent, interior point methods and expectation maximization to evaluate our results. Since the total inflow into any node cannot be greater than 1, this imposes a constraint on the values of diffusion probabilities. For gradient descent and EM, we handle this constraint in a trivial way i.e. as soon as any diffusion probability goes outside the feasible range, we set it to a random value in the feasible region. However, we note that this can be handled in more principled way by incorporating the constraints into the problem formulation. This can be done using a quadratic penalty on the constraints or using a Lagrangian formulation. For interior point methods, the constraint can be modeled explicitly and

taken care by the optimization procedure. For gradient descent, we observe that the likelihood saturates after 10 iterations. Hence we use 20 iterations with a stopping threshold of 1e-3. For both interior point methods and EM, we use 10 iterations and a stopping threshold of 1e-3.

5.4 Evaluation

One of the metrics we use to evaluate the quality of our estimation is the relative error between the true diffusion probabilities and those estimated by the algorithm. Since it is more important to accurately explain future diffusion cascades rather than obtain the exact values of the diffusion probabilities, we use an alternative way to evaluate the effectiveness of our algorithms.

We use a train - test framework to benchmark our results. A train set consist of cascades which were used to estimate the diffusion probabilities whereas the test set consists of future cascades which will diffuse through the network. For each set of cascades, we find the nodes which are active at the end of the diffusion process. The diffusion for each cascade is simulated using both the true and estimated probabilities. For both the train and test cascades, we can compare the spread (spr_{true} and spr_{est}) obtained using the true and estimated probabilities. We measure the relative error between s_{true} and s_{est}.

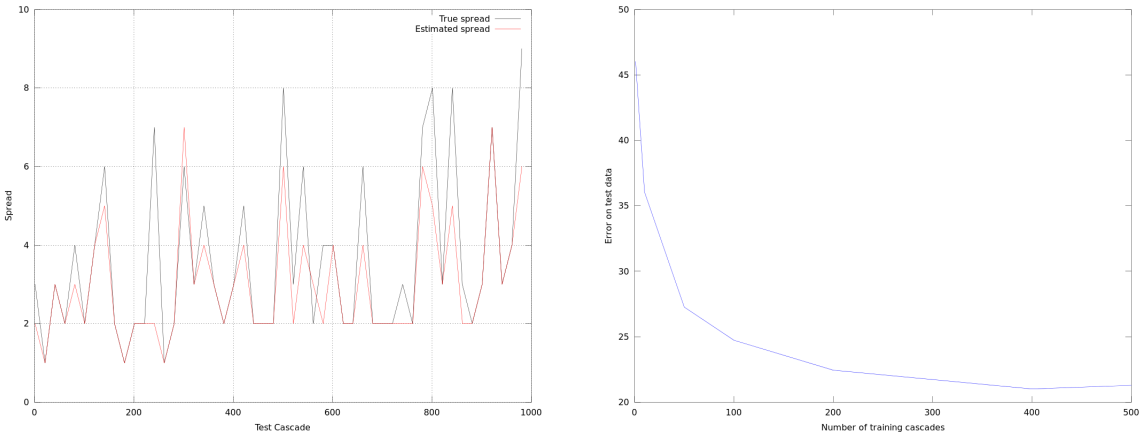
There may exist cascades for which the spread is the same for both cases but the nodes active at the end of the diffusion process are different. To measure this, we generate a vector with length equal to the number of nodes for each cascade. An entry in this vector is 1 if that particular node is active at the end of the diffusion process, else its 0. Such a vector (v_{true} and v_{est}) is generated using the true and estimated probabilities. The hamming distance (HD) between these 2 vectors v_{true} and v_{est} is calculated and is normalized wrt to the true spread. Hence HD lies in the [0,1] range. A HD value of 1 indicates high recall.

6 Results

All of the graphs presented subsequently are on the set of test cascades. The first experiment is for the synthetic random graph with 10 nodes and 45 edges using the EM algorithm. All results have been generated using algorithm parameters as defined in the previous section. Figure 1(a) shows the comparison of spread on test cascades. The X axis represents a particular test cascade whereas the Y axis represents the spread for that cascade. We use 1000 test cascades to estimate the test error in all cases. For each case we report relative error which is the error between the true and estimated probabilities, ,the train error which is the error wrt spread on the train cascades, the test error which is error wrt spread on the test cascades and the time taken to learn the proabilities.

Figure 1(b) shows the decrease in the test error as the number of train cascades increases. This is expected since a greater number of train cascades ensures more edges are covered in the diffusion processes and hence we have data about a larger number of diffusion probabilities. This helps us estimate the diffusion probabilities better. The same trend of decreasing test error with increasing number of train cascades is observed for all graphs and all algorithms. Hence subsequent results use a sufficiently large number of train cascades to minimize the error.

In figure 1(a), we see that the 2 plots are not very close. One of the reasons for this is structure of the graph. Real world social networks follow certain global properties (like power law, higher sparsity) and hence cannot be modeled as as random graphs. Hence diffusion of cascades in a random graph does not give good estimates of influence probabilities. To check whether this hypothesis is correct, we use the forestfire graph as the subject of subsequent experiments. We start with a 10 node forestfire network having 25 edges and use the EM algorithm to infer the probabilities.



(a) (b)

Fig. 1: Graph type = Random, nodes = 10, edges = 45, Algorithm = EM
(a) Comparison of spread on test cascades. Number of train cascades = 200, Relative Error = 36.01 ,Test Error = 21.28, Train Error = 18.43, Time = 415 s (b) Test error with increasing number of cascades.

As we can see that the 2 plots are much closer and the training error is very small (6.49 %). A low training error implies that the estimated diffusion probabilities are able to explain the training cascades well. A high relative error in this case though indicates that we don't have enough number of training cascades or that we have converged to a local maxima. Increasing the number of training cascades or using multiple random restarts as explained in section 3 might help us avoid this problem. The results for the other 2 methods - gradient descent (GD) and interior point method (IP) are similar.

We now scale up our experiments to 100 nodes and show results wrt spread for each of the methods used. Figure 4(a) shows the results for the forestfire graph with 100 nodes and 254 edges. We need to use a large number of cascades (1000) to get a sufficiently low error. We observe that train and test error is quite low and the 2 plots in figure 4(a) are quite close. This shows that although the number of cascades needed is high and this results in a larger time to estimate the probabilities, the EM algorithm scales well wrt quality. Figures 3(a) and 3(b) give similar (even better) results wrt to the train and test errors. We observe that GD is able to find good estimates with just half the time although the relative error for GD is higher than both EM and IP. This suggests an extensive comparison between the various algorithms as part of future work.

In figure 4(b), we present the graph of the normalized hamming distance for the EM algorithm. The graphs for IP and GD are similar. As we can see the values of HD is high (greater than 0.8) for almost all test cascades. This is encouraging since it implies that the algorithms not just find the correct spread but also correctly identify the active nodes at the end of the diffusion process.

The next set of experiments we conducted were for the facebook dataset the characteristics of which are mentioned in the previous section. We show the results for IP and GD algorithm. EM does comparable but worse than both GD and IP for this network. Figures 5(a) and 5(b)

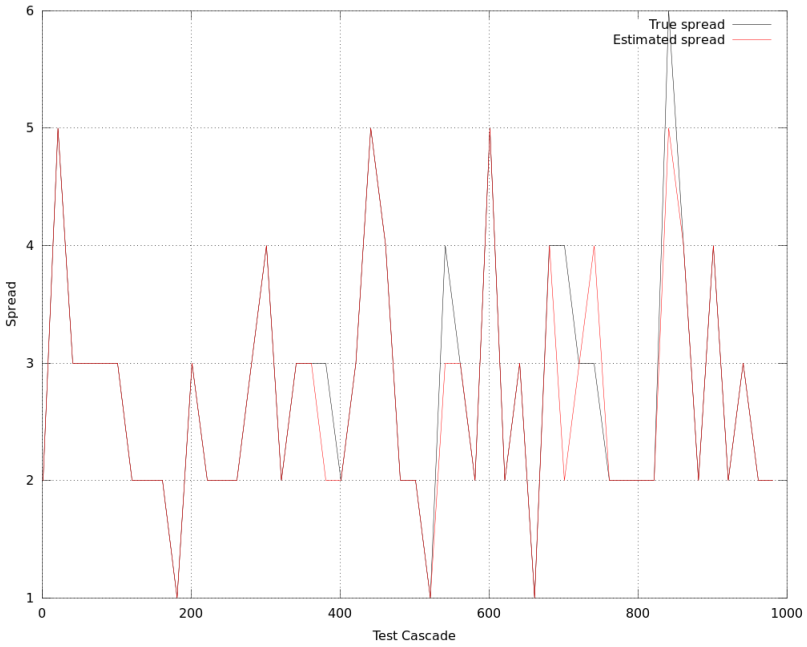
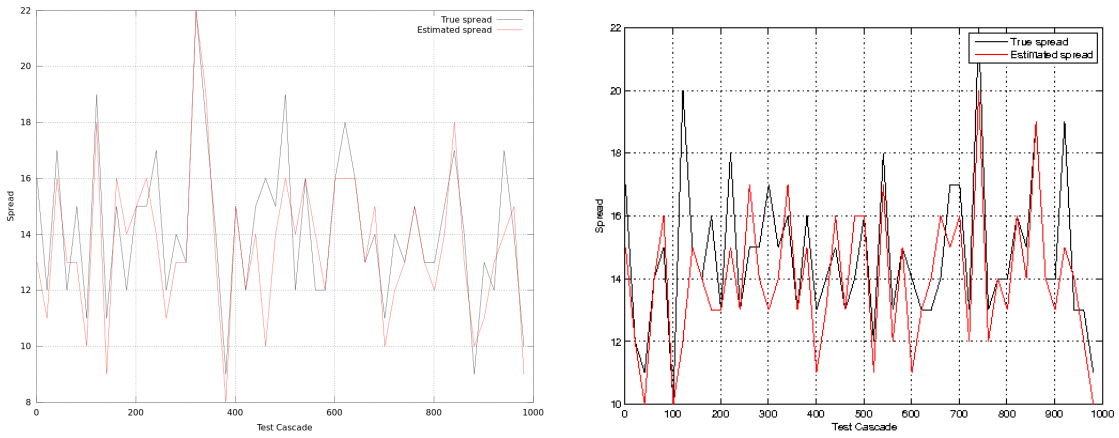
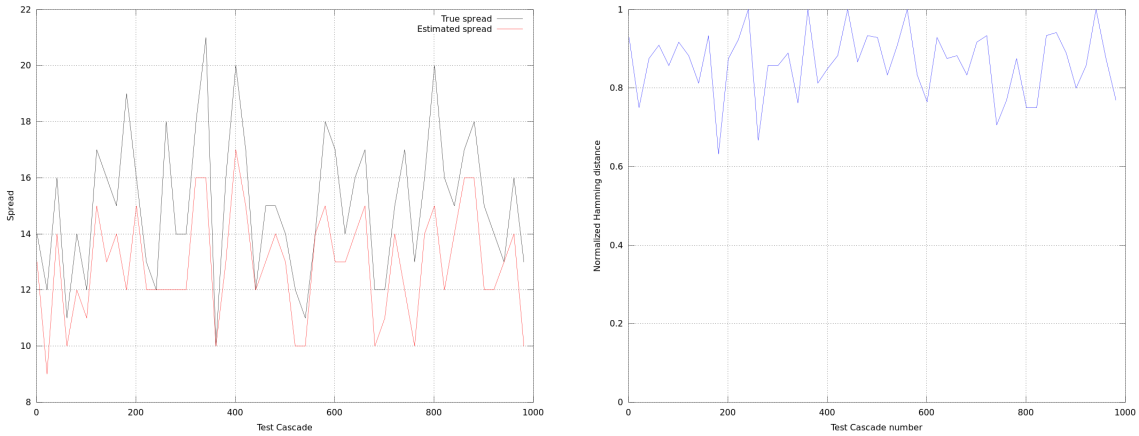


Fig. 2: Comparison of spread on test cascades. Graph type = Forestfire, nodes = 10, edges = 25, Algorithm = EM
Number of training cascades = 500, Relative Error = 27.25, Test Error = 7.60, Train Error = 6.49,Time = 231 s



(a) (b)

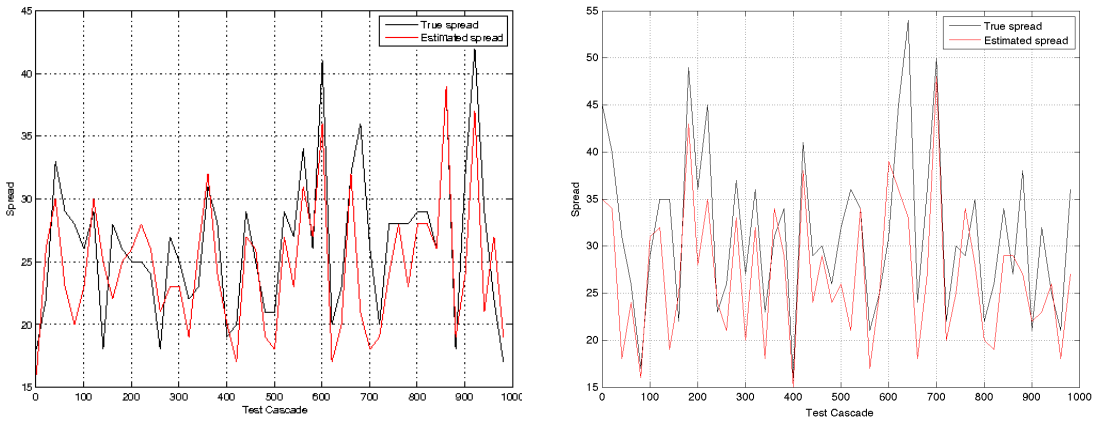
Fig. 3: Graph type = Forestfire, nodes = 100, edges = 254, Number of training cascades = 1000 (a) Comparison of spread on test cascades. Algorithm = GD, Relative Error = 52.14, Test Error = 8.67, Train Error = 8.32, Time = 1534 s (b) Comparison of spread on test cascades.. Algorithm = IP, Relative Error = 44.32, Test Error = 8.36, Train Error = 7.91, Time = 3758 s



(a) (b)

Fig. 4: Graph type = Forestfire, nodes = 100, edges = 254, Algorithm = EM, Number of training cascades = 1000
(a) Comparison of spread on test cascades. Relative Error = 43.36, Test Error = 12.55, Train Error = 11.78, Time = 3344 s (b) Normalized hamming distance on test cascades.

show the results on the facebook dataset for GD and IP. The training and test error are low but the relative error is very high. This indicates that we need a larger number of train cascades. It was not possible to use more train cascades for these experiments because of the prohibitively large time required for learning. We plan to optimize and develop a more scalable algorithm to address this problem and experiment on large networks with greater number of cascades.



(a) (b)

Fig. 5: Graph type = Facebook, nodes = 328, edges = 5038, Number of training cascades = 1000 (a) Comparison of spread on test cascades. Algorithm = GD, Relative Error = 55.22, Test Error = 12.16, Train Error = 11.82, Time = 5737s (b) Comparison of spread on test cascades. Algorithm = IP, Relative Error= 53.89, Test Error = 16.17, Train Error = 15.95, Time = 9972 s

Thus we see that our preliminary results are encouraging. They provide various insights on learning diffusion probabilities from cascades. More experiments with higher computational resources and scalable algorithms are needed to justify the correctness of the idea and the efficacy of our approach.

7 Conclusion

In this paper, we addressed the problem of inferring diffusion probabilities under the LT model. In particular,we used 3 different optimization techniques - Gradient Descent, Interior Point Methods and Expectation Maximization(EM) algorithm to learn these probabilities from past cascade information. We validated our approach on 2 synthetic and 1 real world network. A train-test framework was developed to evaluate the precision of our algorithms. Our preliminary results indicate the effectiveness of our approach.

8 Future Work

We plan to conduct extensive experiments to draw conclusions about the strengths and weaknesses of the various optimization algorithms. This includes scaling up the experiments to larger

datasets. In order to scale up our experiments to larger networks, it is necessary to exploit parallelism. We can parallelize across cascades which do not simultaneously affect same edges. Community detection algorithms can be used to detect communities and probabilities can be learnt parallelly in smaller communities across which cascade diffusion is improbable. We plan to explore other optimization techniques such as LBFGS and heuristic methods like stochastic local search. An interesting direction of future work will be to implement an incremental learning framework which refines the solutions as more number of cascades are obtained. Graph sparsification techniques will be useful to reduce the number of edges and hence the number of diffusion probabilities to be learnt, thus increasing scalability.

References

1. Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.
2. Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
3. Tianyu Cao, Xindong Wu, Tony Xiaohua Hu, and Song Wang. Active learning of model parameters for influence maximization. In *Machine Learning and Knowledge Discovery in Databases*, pages 280–295. Springer, 2011.
4. Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1029–1038. ACM, 2010.
5. Pedro Domingos. Mining social networks for viral marketing. *IEEE Intelligent Systems*, 20(1):80–82, 2005.
6. Xiao Fang, Paul Jen-Hwa Hu, Zhepeng Lionel Li, and Weiyu Tsai. Predicting adoption probabilities in social networks. *Information Systems Research*, 24(1):128–145, 2013.
7. Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1019–1028. ACM, 2010.
8. Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM, 2010.
9. Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 211–220. IEEE, 2011.
10. David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.
11. Masahiro Kimura, Kazumi Saito, and Hiroshi Motoda. Blocking links to minimize contamination spread in a social network. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(2):9, 2009.
12. Masahiro Kimura, Kazumi Saito, and Ryohei Nakano. Extracting influential nodes for information diffusion on a social network. In *AAAI*, volume 7, pages 1371–1376, 2007.
13. Masahiro Kimura, Kazumi Saito, Ryohei Nakano, and Hiroshi Motoda. Finding influential nodes in a social network from information diffusion data. In *Social Computing and Behavioral Modeling*, pages 1–8. Springer, 2009.
14. Masahiro Kimura, Kazumi Saito, Ryohei Nakano, and Hiroshi Motoda. Extracting influential nodes on a social network for information diffusion. *Data Mining and Knowledge Discovery*, 20(1):70–97, 2010.
15. Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007.

16. Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.
17. Jure Leskovic. Stanford network analysis project. `http://snap.stanford.edu/`.
18. Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.
19. Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70. ACM, 2002.
20. Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 67–75. Springer, 2008.
21. Kazumi Saito, Kouzou Ohara, Yuki Yamagishi, Masahiro Kimura, and Hiroshi Motoda. Learning diffusion probability based on node attributes in social networks. In *Foundations of Intelligent Systems*, pages 153–162. Springer, 2011.
22. Duncan J Watts and Peter Sheridan Dodds. Influentials, networks, and public opinion formation. *Journal of consumer research*, 34(4):441–458, 2007.
23. Margaret Wright. The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bulletin of the American mathematical society*, 42(1):39–56, 2005.
24. Hongchao Zhang and William W Hager. A nonmonotone line search technique and its application to unconstrained optimization. *SIAM Journal on Optimization*, 14(4):1043–1056, 2004.