Two Models for Inferring Network Structure from Cascades

Dakan Wang
Yu Wu

1 Introduction

In many real world situations, the edges of a network are invisible. However, edges are very fundamental parts of a network to investigate the network structure. It is lucky that although we do not know the edges, we can get some information about the edges. These information can help ups infer the latent edges in the network. Cascade is a such a very useful information source to infer edges. Cascade is a fundamental process in social network. In general, a cascade describes how a contagion spreads or propagates across a network. Research on cascade will enable us to gain insights about how people influence each other. However, sometimes it is very difficult to track a cascade because of its complexity. Even worse, we may not even know the structure of the network over which the cascade happens. In most cases, the information available is very limited. We may only know when the cascade propagates to a given node. For instance, in study of virus contagion, the time when a person was infected may be much easier to track than the source of the infection. In viral marketing, we can easily record the time when a customer made a purchase, but it is almost impossible to know who had influenced him to make the decision. Therefore, the challenging task is to infer the latent network structure from the limited given information. Specifically, we may only know the nodes, and sets of time data from a few cascades, but not the edges in the network. Our goal is to infer the hidden edges based on these limited information. In this report, we proposed two models for inferring edges from cascades.

2 Related Work

There are a few recent works focusing on inferring networks from cascades. [3] conducted an analysis of cascades in blogs. [2] converted the edge inference problem to be combinatorial optimization problem and proposed some very useful heuristics to enable the algorithm to handle large scale data. [5] modeled edge inference to be a efficient convex optimization problem to accurately predict edges in the network. [4] discussed the situation when we only have a fraction

of the complete cascade. [6] tried to infer the influence of a node from cascades.

3 Problem Formulation

We have a graph of n nodes $G = (V, E)$. The edge set E is unknown. We also have a bunch of cascades. Each cascade is represented to be a vector $\{\tau_1, .., \tau_n\}$, where τ_i indicates that node i is infected at time τ_i. If $\tau_i = \infty$, node i is not infected in the cascade. The problem is to infer the edges E from these cascades. The idea of solving that problem is based on two assumptions: 1. Earlier infected nodes may have edges to nodes infected later. 2. Infected nodes are not very likely to have edges to uninfected nodes.

4 Model

4.1 Model A

Before proposing the model, let's talk about our assumptions first. [5] assumes the time period each infected node takes to transmit the disease follows a random distribution. However, we notice there is some inconsistency between the model assumption and the objective function. Inspired by the maximum likelihood estimation in [5], we propose another assumption that an uninfected person i is not always susceptible to infection, and will only be infected when he shows up in the network. For instance, for cascades in blog sphere, one will not be influenced by other persons' blog-posts if he constantly stays off-line. The already infected person, say j, will decide whether or not to infect person i with probability $A_{i,j}$. If j chooses to infect i, he will wait until person i shows up and then try to infect him. The probability of i being infected when he shows up can be defined in various ways to finish model assumption.

There are two points we want to point out about our assumption: 1. Before getting infected at time τ_i, i can also show up at $\tau_i' < \tau_i$(e.g., access the Internet in the blog case). It is not clear whether he was infected at that time. Either infected or not infected can be possible. It may be the case that he was infected , but we did not observe his infection at that time. Thus to avoid ambiguity, we do not take a person's history before his infection into account. 2. For node i, which is never infected, we assume that he will not get infected whenever he shows up. To satisfy this, other infected node must not choose to infect node i, otherwise there will be a probability i being infected when shows up. Hence the probability of i never infected is $\prod_{\text{j infected}} (1 - A_{ij})$.

Moreover, instead of assuming infected people transmit the virus to a susceptible one independently, we assume they impose a collective influence. This assumption is very common in real world. For instance, when one wants to purchase a product online, he may first check its online reviews and then make the decision. In other words, the other people who have already bought the product influence the person collectively through reviews. We still maintain the

assumption that a susceptible person will only get infected when he shows up. Different from the above model, we do not keep the restriction that A_{ij} ranges between 0 and 1, but assume it can take any real value. Large positive value for A_{ij} means that the person j is very likely to infect person i, while negative value indicates that the person j may prevent the person i from being infected (can be interpreted as i distrusts j very much). When i shows up at time τ_i, the influence of j on i is $w(\tau_i - \tau_j)A_{ij}$, where $w(t)$ is the distribution defined in [5]. The collective influence on person i at τ_i is $\sum_{\tau_j < \tau_i} (w(\tau_i - \tau_j)A_{ij})$. Finally, we assume that the collective influence on person i who has never been infected is $\sum_{\text{j infected}} A_{i'j}$, as explained in the above section.

We now try to understand the above definition in another way. Each node i is associated with a set of cascades. In some cascades, node i was infected at τ_i, while in others i never got infected. For a cascade in which an node i infected at τ_i , the vector contains non-zero entry $w(\tau_i - \tau_j)$ for every j such that $\tau_j < \tau_i$. For a cascade in which node i never infected , the vector contains entry 1 for every already infected node. The i-th row of A, denoted A_i, is the weight vector characterizing how much node i trust other nodes. The probability of node i infected at τ_i or never infected is a function of the dot product between feature and weight vector. Then we reduce the cascade problem to a classification problem: we have some positive and negative instances, which are respectively different cascades in which node i got infected or not. For each instance we have a feature vector. And we would like to infer the weight vector from the provided training data. Then we may adopt some efficient discriminate model such as logistic regression, SVM. If we want to guarantee sparsity of $|A|$, we may use sparse logistic regression. One possible advantage of this model over [5] is that we do not have an inconvenient product of terms like $\mathbf{1} - \prod_{\tau_j < \tau_i} (1 - w(\tau_i - \tau_j)A_{ij})$.

To summarize, we now mathematically define the above idea. For a certain node i, we have a set of n_i instances $\{x^{(j)}, y^{(j)}\}_{j=1}^{n_i}$. Here $y^{(j)} \in \{0, 1\}$ represents whether node i is infected in cascade j. And x^j is a vector $(x_0^j, .., x_k^j, .., x_n^j)$ with entry x_j^k to be

$$x_k^j = \begin{cases} w(\tau_i^j - \tau_k^j) & : \quad \tau_k^j < \tau_i^j \\ 0 & : \quad \text{else} \end{cases} \tag{1}$$

for $y_i = 1$, and

$$x_k^j = \begin{cases} 1 & : \quad \text{j is infected} \\ 0 & : \quad \text{else} \end{cases} \tag{2}$$

for $y_i = 0$. We would like to learn from the training set our transmission weight $A_i = (A_{1,i}, .., A_{n,i})$ and a hypothesis $h_{A_i} : x \mapsto y$. We use logistic function to

$$h_A(x) = \frac{1}{1 + e^{-(A_i)^T x}} \tag{3}$$

and assume that

$$p(y = 1|x, A_i) = h_A(x)$$
$$p(y = 0|x, A_i) = 1 - h_A(x) \tag{4}$$

The objective function we would like to maximize is

$$F = \sum_{i=1}^{n_i} (y^i log(h_A(x^i)) + (1 - y^i) log(1 - h_A(x^i))) + \lambda \left\| A_i \right\|_1 \tag{5}$$

Finally we use the large scale sparse logistic regression package [1] to solve the above objective function.

4.2 MODEL B

In the above section, we cast the network inference problem to be a binary classification problem and treat the cascades containing node i to be positive instances and those not containing i to be negative instances. We implicitly assume that the cascades not containing i indicate that there are no edges between i and previously infected edges. However, we claim that second assumption is presumptuous. There are lots of examples against this assumption. We take hash tag propagation to be an example. Here each hash tag is a cascade. There are many cascades that one user does not adopt a hashtag and some of the users he follows adopt that hashtag. These cascades are not evidences that there are no edges between node i and already infected nodes. Thus these negative instances are not truly negative. Treating them to be negative maybe harmful for inferring edges. On the other side, those positive instances (cascades containing node i) do indicate that there may be some edges between node i and some previously infected nodes. Those we are facing a problem of only having confident positive instances. It is lucky that there have been many works dealing with problems which only contain only positive instances. Such problems are called one-class classification problem. By definition, one class classification aims istinguish one class of objects from all other possible objects, by learning from a training set containing only the objects of that class. This is exactly the problem we want to solve. It is also worth mentioning that there are also many provide easy-to-use software packages for one-class classification.

The other advantage of modeling network inference to be a one-class classification problem is that training the model will be more efficient. Indeed, in many situations, the proportion of positive instances, or cascades in which node i is infected is very small. Thus our training data size will be significantly reduced if we only consider the positive instances.

The one-class classification problem can be formulated as follows:

$$\begin{aligned} &\min \frac{1}{2}|w|^2 + \frac{1}{vl}\sum_{i=1}^{n}\epsilon_i - \rho \\ &\text{subject to} \\ &w \cdot x_i \geq \rho - \epsilon_i \end{aligned} \tag{6}$$

where $\{x_i\}_{i=1}^{l}$ is our positive training instances. v, ρ are the parameters and w is the weight vector we want to infer.

5 Experiments

5.1 Datasets

As in [2, 5], there are two categories of datasets we are considering using: synthetic datasets and real world datasets. The synthetic dataset is mainly used for debugging programs. For this milestone, we only use synthetic datasets. The dataset we used is similar to the one in [5]. That is, we generate a Erdos random graph. For each generated edge, we randomly generate the transmission weight A_{ij}. Then we try to generate a set of cascades. For each cascade, we randomly generate a start node and take it as the newly infected node. Then we iteratively simulate the cascading process. Pick the node i with the earliest infection time from the set of newly infected node. For the picked node, decide whether to infect the other uninfected node based on the transmission weight A_{ij}. If node i choose to infect j, then randomly generate a time t_{ij} according to a specific distribution. If node j is not in the newly infected node set, add it to the set. If it is already in the set and t_{ij} is smaller than its infection time, update the infection time. Finally we remove node i from the newly infected node. In this way we can generate a cascade. As in [5], we also generate enough cascades for a informational training dataset.

5.2 MODEL A

5.2.1 Evaluation

We test whether a latent link can be inferred correctly. We compare the links inferred from the cascades with the links in real network and get the precision and recall of the model. For different choice of the sparsity parameter, precision and recaul are different. Thus we can plot a precision-recall curve. Note that we have connected the first point on the curve to $(0, 1)$ and the last point to $(1, 0)$.

5.2.2 results on synthetical datasets

We tested our algorithm on synthetic datasets. As in [5], we generate a erdos random graph consisting of 512 nodes and 1024 edges. In the graph, the

transmission probability A_{ij} is sampled between 0.05 and 1.

For the time generation function $w(t)$, we also tried three probability distributions detailed as follows

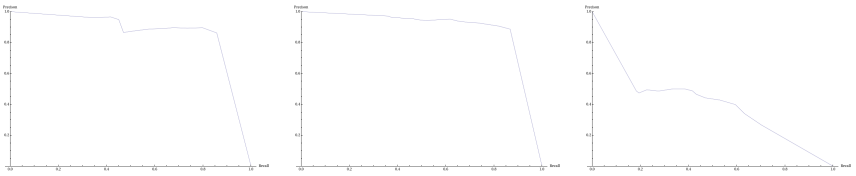
- power law distribution $(\alpha - 1)t^{-\alpha}$, where α is set to 9.5.
- exponential distribution $\frac{1}{\alpha}e^{-\frac{1}{\alpha}t}$, where α is set to 9.5.
- weilbull distribution $\frac{k}{\alpha}(\frac{x}{\alpha})^{k-1}e^{(-\frac{x}{\alpha})^k}$, where α and k are set to 9.5 and 2.3 respectively

Each cascade is generated as specified in 5.1. We generate enough cascades such that at least 95% edges are used to tramit a disease.

We show the experimental results in figure 1. We plot three precision-recall curves for the three generated networks. Each point on the curve corresponds the model trained by setting the sparsity parameter to a certain value. Different points on the curve reflect models with different sparsity parameters. When sparsity parameter decreases, the recall increases and the precision increases.

Our model achieves very good performance comparable with, and even better than that obtained in [5]. However, it is quite weird that our model's performance is bad on the graph whose remission time is generated from a power law distribution. We have not figured the reason yet. Probably our value of α is not the same as that set in [5]. We will try to find the reason in future work.

Apart from the erdos random graph, we also test our model on another random graph, which is generated from the preferential attachment mechanism as taught in class. We report our results in 2. As in erdos random graph, we also tried three different probability density functions for sampling transmission time.



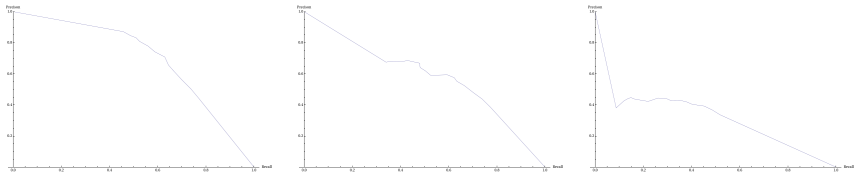
(a) weillbull distribution (b) exponential distribution (c) power law distribution

Fig. 1: PR curves for Erdos Random Graph different traismission time configurations

5.2.3 Results for Real World Datasets

Apart from synthetic dataset, we also tested our model on two real-world datasets, specified as below

- A email network in a research institution consisted of 437 nodes and 2805 edges.



(a) weillbull distribution (b) exponential distribution (c) power law distribution

Fig. 2: PR curves for preferenctial attachment graph of different traismission time configurations

- A collaboration network. We extract the largest component of the network. THe number of nodes and edges are 379 and 608 respectively.

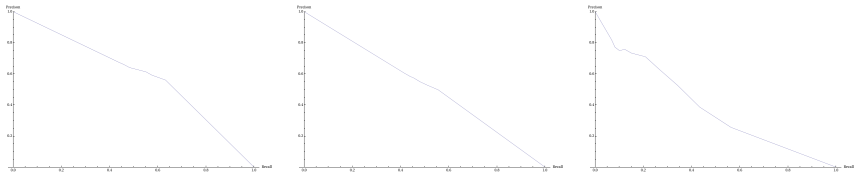
For the collaboration network, we sampled the edge transmission probability A_{ij} uniformly. For the email network, we set $A_ij = 1 - (1 - \phi)(1 - \epsilon)^{m_{ij}}$, where $\phi = 0.05$, $\epsilon = 0.0001$ and m_{ij} is the . This parameter set was suggested in [5] For each network, we also adopted three different probability density function for transmission time as in synthetic datasets. The parameters for those probability density function is

- weilbull distribution $k = 2.5$ $\alpha = 3.5$
- exponential distribution $\mu = 9.5$
- power law distribution $\alpha = 2.5$

We reported our results in figure 3 and figure 4. Our model can also handle edge prediction for real world datasets. It seems that that our model performance is worse than that in [5]. However, we want to point out that our model has large improvement space for two reasons: 1. We do not use enough training data. Indeed, cascades we sampled only covered 90% edges, while those in [5] covered 99% edges. The reason that we did not sample as many cascades as in [5] is that in order to sample cascades to cover 95% edges, we need to sample much more cascades since for some uncovered edges, the transmission probability is very small. Since each cascade will be one additional instance for each node . Our laptop memory cannot hold that many instances . However, sampling more cascades for getting more training instances can improve the performance significantly. We tried sampling cascades to cover 85% edges and 90% edges and tested each setting's performance. And the PR curve in the later setting lifted by 15 points. Thus we expect our model to be much better than its current state if our laptop memory can hold enough training instances.

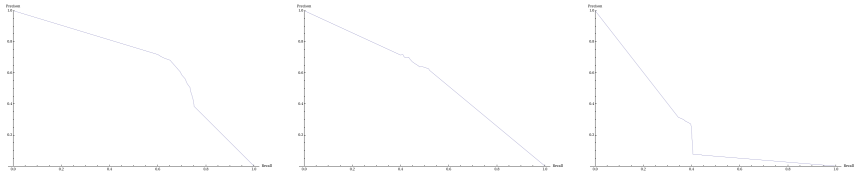
5.3 MODEL B

In this section, we conducted experiments to test our model B. The data were generated as in the above experiments. However, we discarded all negative



(a) weillbull distribution (b) exponential distribution (c) power law distribution

Fig. 3: PR curves for email network of different transmission time configurations



(a) weillbull distribution (b) exponential distribution (c) power law distribution

Fig. 4: PR curves for collaboration network of different transmission time configurations

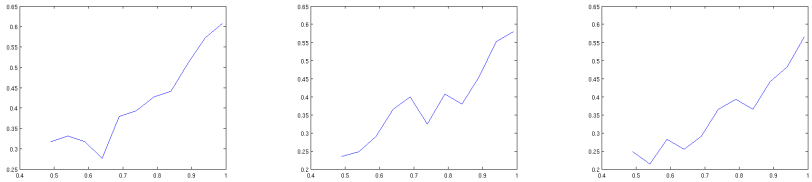
instances and only kept those positive instances for each node. We used one-class SVM component in libSVM to infer the edge weight. The libSVM tool does not provide the functionality of learning a sparse weight vector. Therefore we at this stage cannot get a PR curve with respect to the sparsity parameter as in the experiments in the above section. Thus in this experiment we evaluated our experiment from another perspective: we want to check whether the weights inferred by our model satisfy that the weights of the edges are bigger than the weights of the non-edges. For instance, consider a graph with n nodes. We now want to investigate what nodes have links to node s. Without loss of generality, we suppose that nodes $i_1^s, i_2^s, ..i_m^s$ have links to node s. Then we rank all the nodes by our inferred weight vector w_s. Suppose the top ranked m nodes are j_1^s, j_2^s,..,j_m^s. Then we count how many of our top ranked nodes have edges to node i, or we calculate the cardinality of the set $\{i_1^s, ..i_m^s\} \cap \{j_1^s, .., j_m^s\}$. Suppose that

$$\begin{aligned} tt_s &= \#\{i_1^s, ..i_m^s\} \cap \{j_1^s, .., j_m^s\} \\ tf_s &= \{i_1^s, ..i_m^s\} - \{j_1^s, .., j_m^s\} \\ ft_s &= \{j_1^s, ..j_m^s\} - \{i_1^s, .., i_m^s\} \end{aligned}$$

We use the following precision and recall to evaluate our model

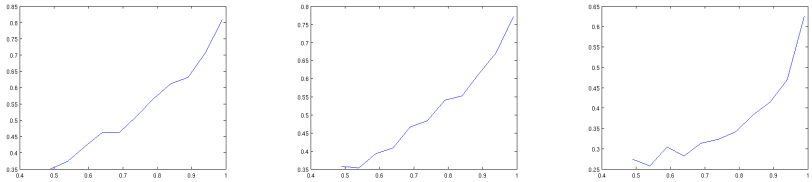
$$\begin{aligned} P &= \frac{tt_s}{tt_s + ft_s} \\ R &= \frac{tt_s}{tt_s + tf_s} \end{aligned}$$

Notice that in this case, $tf_s = ft_s$, leading to $P = R$. We only use precision to evaluate our model. The goal of the experiments is to show that when we have observed enough cascades, the edges in the network can also be inferred even if we discard all the negative instances. We investigated how the performance of our model change with respect to edge cover ratio. Here edge cover ratio means the proportion of edges covered by the generated cascades. The larger the cover ratio, the more cascades we generated. We showed in figure 5 and 6 the precision-cover ratio curve for two networks for different probability density function of transmission time. From the figure, we can see that if the edge cover ratio is high enough, or number of cascades is large, then we may get satisfactory performance even if we do not consider negative instances.



(a) weillbull distribution (b) exponential distribution (c) power law distribution

Fig. 5: Precision-Cover ratio of preferenctial attachment network



(a) weillbull distribution (b) exponential distribution (c) power law distribution

Fig. 6: Precision-Cover ratio of email network

6 future work

In the future, we will test our model on more large scale data. One of the interesting data is the twitter dataset. Indeed, we have explored the twitter dataset. We once wanted to use hashtag propagation to infere the who follows whom links. However, after a cafeful data analysis, we found that the hashtag propagation may not provide very useful information for infering the who follows whom links. If possible, we may try some other evaluations. One possible way is to test whether our inferred latent network structure conveys information that is consistent with our intuition. For instance, from the inferred weight between pairs of nodes, we may discover several communities. Then we may check

whether different communities have different interests(e.g., one community is interested in sports, while the other is interested in entertainment).

7 conclusion

We investigated two models for inferring network edges from cascades. The first model is based on sparse logsitic regression. It is easy to understand and both efficient and effective. Our second model is based on one-class SVM. Experiments showed that even if we discarded all negative training instances, we can also infer network edges accurately.

8 Acknoledgement

We thank Professor Leskovec's insightful and excellent classes. We also thank Prof. Leskovec and Seth Myer for their kindness of providing the datasets. We finally thank all TA's hard work.

9 Contributions of the team members

Dakan Wang: prepared the poster, wrote the milestone and final report, designed and implemented the models, and did all the experiments.

Yu Wu: prepared the data and ploted the graph for experimental results of model A

References

[1] http://www.stanford.edu/ boyd/.

[2] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *KDD '10: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1019–1028, New York, NY, USA, 2010. ACM.

[3] Jure Leskovec, Mary Mcglohon, Christos Faloutsos, Natalie Glance, and Matthew Hurst. Cascading behavior in large blog graphs. In *In SDM*, 2007.

[4] Eldar Sadikov, Montserrat Medina, Jure Leskovec, and Hector Garcia-molina. Correcting for missing data in information cascades, 2010.

[5] S.Myers and J. Leskovec. On the convexity of latent social network. In year = 2010 Massih Amini and Nicolas Usunier and Cyril Goutte, pages = 28–36, editor, *Advances in Neural Information Processing Systems 23*.

[6] Jaewon Yang and Jure Leskovec. Modeling information diffusion in implicit networks. In *IEEE International Conference on Data Mining*. Stanford InfoLab, December 2010.